tor

0.2.2.13-alpha

Generated by Doxygen 1.5.6

Tue May 25 00:30:28 2010

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

address.c (Functions to use and manipulate the **tor_addr_t** (p. 237) structure) . . . 255
address.h (Headers for **address.h** (p. 265)) 265
aes.c (Implements the AES cipher (with 128-bit keys and blocks), and a counter-mode stream cipher on top of AES. This code is taken from the main Rijndael distribution. (We include this because many people are running older versions of OpenSSL without AES support.)) 277
aes.h (Headers for **aes.c** (p. 277)) 281
buffers.c (Implements a generic interface buffer. Buffers are fairly opaque string holders that can read to or flush from: memory, file descriptors, or TLS connections) . 283
circuitbuild.c (The actual details of building circuits) 297
circuitlist.c (Manage the global circuit list) 328
circuituse.c (Launch the right sort of circuits and attach streams to them) 340
command.c (Functions for processing incoming cells) 351
compat.c (Wrappers to make calls more portable. This code defines functions such as tor_malloc, tor_snprintf, get/set various data types, renaming, setting socket options, switching user IDs. It is basically where the non-portable items are conditionally included depending on the platform) 355
compat.h . . . . ??
compat_libevent.c (Wrappers to handle porting between different versions of libevent)370
compat_libevent.h . . . . ??
config.c (Code to parse and interpret configuration files) 375
connection.c (General high-level functions to handle reading and writing on connections)406
connection_edge.c (Handle edge streams) 429
connection_or.c (Functions to handle OR connections, TLS handshaking, and cells on the network) 449
container.c (Implements a smartlist (a resizable array) along with helper functions to use smartlists. Also includes hash table implementations of a string-to-void∗ map, and of a digest-to-void∗ map) 462
container.h . . . . ??
control.c (Implementation for Tor's control-socket interface. See doc/spec/control-spec.txt for full details on protocol) 485
cpuworker.c (Implements a farm of 'CPU worker' processes to perform CPU-intensive tasks in another thread or process, to not interrupt the main thread) 511

Chapter 3

Data Structure Documentation

3.1 addr_policy_t Struct Reference

`#include <or.h>`

Data Fields

- int **refcnt**
- **addr_policy_action_t policy_type**:2
- unsigned int **is_private**:1
- unsigned int **is_canonical**:1
- **maskbits_t maskbits**
- **tor_addr_t addr**
- uint16_t **prt_min**
- uint16_t **prt_max**

3.1.1 Detailed Description

A reference-counted address policy rule.

3.1.2 Field Documentation

3.1.2.1 int addr_policy_t::refcnt

Reference count

Referenced by addr_policy_free().

3.1.2.2 addr_policy_action_t addr_policy_t::policy_type

What to do when the policy matches.

Referenced by cmp_single_addr_policy(), exit_policy_remove_redundancies(), policy_hash(), policy_summary_add_item(), policy_write_item(), router_parse_addr_policy(), and router_parse_addr_policy_private().

3.1.2.3 unsigned int addr_policy_t::is_private

True iff this is the pseudo-address, "private".

Referenced by cmp_single_addr_policy(), policy_expand_private(), policy_hash(), policy_write_item(), and router_parse_addr_policy_private().

3.1.2.4 unsigned int addr_policy_t::is_canonical

True iff this policy is the canonical copy (stored in a hash table to avoid duplication of common policies)

Referenced by addr_policy_free(), load_policy_from_option(), and policy_expand_private().

3.1.2.5 maskbits_t addr_policy_t::maskbits

Accept/reject all addresses **a** such that the first **maskbits** bits of **a** match **addr**.

Referenced by addr_policy_covers(), addr_policy_intersects(), cmp_single_addr_policy(), exit_policy_remove_redundancies(), policy_expand_private(), policy_hash(), policy_summary_add_item(), policy_write_item(), and router_parse_addr_policy().

3.1.2.6 tor_addr_t addr_policy_t::addr

Base address to accept or reject.

Referenced by addr_policy_covers(), addr_policy_intersects(), cmp_single_addr_policy(), policy_expand_private(), policy_hash(), policy_summary_add_item(), policy_write_item(), router_add_exit_policy(), and router_parse_addr_policy().

3.1.2.7 uint16_t addr_policy_t::prt_min

Lowest port number to accept/reject.

Referenced by addr_policy_covers(), addr_policy_intersects(), cmp_single_addr_policy(), exit_policy_remove_redundancies(), load_policy_from_option(), policy_hash(), policy_summary_add_item(), policy_write_item(), router_parse_addr_policy(), and router_parse_addr_policy_private().

3.1.2.8 uint16_t addr_policy_t::prt_max

Highest port number to accept/reject.

Referenced by addr_policy_covers(), addr_policy_intersects(), cmp_single_addr_policy(), exit_policy_remove_redundancies(), load_policy_from_option(), policy_hash(), policy_summary_add_item(), policy_write_item(), router_parse_addr_policy(), and router_parse_addr_policy_private().

The documentation for this struct was generated from the following file:

- **or.h**

3.2 addressmap_entry_t Struct Reference

Data Fields

- char * **new_address**
- time_t **expires**
- **addressmap_entry_source_t** **source**:3
- short **num_resolve_failures**

3.2.1 Detailed Description

A client-side struct to remember requests to rewrite addresses to new addresses. These structs are stored in the hash table "addressmap" below.

There are 5 ways to set an address mapping:

- A MapAddress command from the controller [permanent]
- An AddressMap directive in the torrc [permanent]
- When a TrackHostExits torrc directive is triggered [temporary]
- When a DNS resolve succeeds [temporary]
- When a DNS resolve fails [temporary]

When an addressmap request is made but one is already registered, the new one is replaced only if the currently registered one has no "new_address" (that is, it's in the process of DNS resolve), or if the new one is permanent (expires==0 or 1).

(We overload the 'expires' field, using "0" for mappings set via the configuration file, "1" for mappings set from the control interface, and other values for DNS and TrackHostExit mappings that can expire.)

The documentation for this struct was generated from the following file:

- **connection_edge.c**

3.3 aes_cnt_cipher Struct Reference

Data Fields

- u32 **rk** [4 *(MAXNR+1)]
- int **nr**
- u32 **counter3**
- u32 **counter2**
- u32 **counter1**
- u32 **counter0**
- u8 **buf** [16]
- u8 **pos**

3.3.1 Detailed Description

Implements an AES counter-mode cipher.

3.3.2 Field Documentation

3.3.2.1 u32 aes_cnt_cipher::rk[4 *(MAXNR+1)]

This next element (however it's defined) is the AES key.

Referenced by _aes_fill_buf(), and aes_set_key().

3.3.2.2 u32 aes_cnt_cipher::counter3

These four values, together, implement a 128-bit counter, with counter0 as the low-order word and counter3 as the high-order word.

Referenced by _aes_fill_buf(), aes_set_iv(), and aes_set_key().

3.3.2.3 u8 aes_cnt_cipher::buf[16]

The encrypted value of ctr_buf.

Referenced by _aes_fill_buf(), aes_crypt(), aes_crypt_inplace(), aes_set_iv(), and aes_set_key().

3.3.2.4 u8 aes_cnt_cipher::pos

Our current stream position within buf.

Referenced by aes_crypt(), aes_crypt_inplace(), aes_set_iv(), and aes_set_key().

The documentation for this struct was generated from the following file:

- **aes.c**

3.4 authdir_config_t Struct Reference

Data Fields

- strmap_t * **fp_by_name**
- digestmap_t * **status_by_digest**

3.4.1 Detailed Description

List of nickname->identity fingerprint mappings for all the routers that we name. Used to prevent router impersonation.

3.4.2 Field Documentation

3.4.2.1 strmap_t* authdir_config_t::fp_by_name

Map from lc nickname to fingerprint.

Referenced by add_fingerprint_to_dir(), authdir_config_new(), dirserv_free_fingerprint_list(), dirserv_get_name_status(), and dirserv_get_status_impl().

3.4.2.2 digestmap_t* authdir_config_t::status_by_digest

Map from digest to **router_status_t** (p. 205).

Referenced by add_fingerprint_to_dir(), authdir_config_new(), dirserv_free_fingerprint_list(), dirserv_get_nickname_by_digest(), and dirserv_get_status_impl().

The documentation for this struct was generated from the following file:

- **dirserv.c**

3.5 authority_cert_t Struct Reference

`#include <or.h>`

Data Fields

- **signed_descriptor_t cache_info**
- **crypto_pk_env_t * identity_key**
- **crypto_pk_env_t * signing_key**
- char **signing_key_digest** [DIGEST_LEN]
- time_t **expires**
- uint32_t **addr**
- uint16_t **dir_port**
- uint8_t **is_cross_certified**

3.5.1 Detailed Description

Certificate for v3 directory protocol: binds long-term authority identity keys to medium-term authority signing keys.

3.5.2 Field Documentation

3.5.2.1 signed_descriptor_t authority_cert_t::cache_info

Information relating to caching this cert on disk and looking it up.

Referenced by already_have_cert(), authority_cert_dup(), authority_cert_free(), authority_cert_get_newest_by_id(), authority_cert_parse_from_string(), dirvote_act(), dirvote_add_vote(), dirvote_get_vote(), format_networkstatus_vote(), init_keys(), networkstatus_check_document_signature(), networkstatus_parse_vote_from_string(), trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

3.5.2.2 crypto_pk_env_t* authority_cert_t::identity_key

This authority's long-term authority identity key.

Referenced by authority_cert_dup(), authority_cert_free(), authority_cert_parse_from_string(), dirserv_generate_networkstatus_vote_obj(), and dirvote_compute_consensuses().

3.5.2.3 crypto_pk_env_t* authority_cert_t::signing_key

This authority's medium-term signing key.

Referenced by authority_cert_dup(), authority_cert_free(), authority_cert_parse_from_string(), load_authority_keyset(), networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

3.5.2.4 char authority_cert_t::signing_key_digest[DIGEST_LEN]

The digest of **signing_key**

Referenced by authority_cert_get_by_sk_digest(), authority_cert_parse_from_string(), dirvote_add_vote(), and trusted_dirs_load_certs_from_string().

3.5.2.5 time_t authority_cert_t::expires

The listed expiration time of this certificate.

Referenced by authority_cert_parse_from_string(), authority_certs_fetch_missing(), dirvote_perform_vote(), networkstatus_check_consensus_signature(), and v3_authority_check_key_expiry().

3.5.2.6 uint32_t authority_cert_t::addr

This authority's IPv4 address, in host order.

Referenced by authority_cert_parse_from_string(), and trusted_dirs_load_certs_from_string().

3.5.2.7 uint16_t authority_cert_t::dir_port

This authority's directory port.

Referenced by authority_cert_parse_from_string(), and trusted_dirs_load_certs_from_string().

3.5.2.8 uint8_t authority_cert_t::is_cross_certified

True iff this certificate was cross-certified by signing the identity key with the signing key.

Referenced by authority_cert_parse_from_string().

The documentation for this struct was generated from the following file:

- **or.h**

3.6 bridge_info_t Struct Reference

Data Fields

- **tor_addr_t addr**
- uint16_t **port**
- char **identity** [DIGEST_LEN]
- **download_status_t fetch_status**

3.6.1 Detailed Description

Information about a configured bridge. Currently this just matches the ones in the torrc file, but one day we may be able to learn about new bridges on our own, and remember them in the state file.

3.6.2 Field Documentation

3.6.2.1 tor_addr_t bridge_info_t::addr

Address of the bridge.

Referenced by bridge_add_from_config(), and launch_direct_bridge_descriptor_fetch().

3.6.2.2 uint16_t bridge_info_t::port

TLS port for the bridge.

Referenced by bridge_add_from_config(), and launch_direct_bridge_descriptor_fetch().

3.6.2.3 char bridge_info_t::identity[DIGEST_LEN]

Expected identity digest, or all zero bytes if we don't know what the digest should be.

Referenced by bridge_add_from_config(), and launch_direct_bridge_descriptor_fetch().

3.6.2.4 download_status_t bridge_info_t::fetch_status

When should we next try to fetch a descriptor for this bridge?

Referenced by bridge_add_from_config(), and learned_bridge_descriptor().

The documentation for this struct was generated from the following file:

- **circuitbuild.c**

3.7 buf_pos_t Struct Reference

Data Fields

- const **chunk_t** ∗ **chunk**
- int **pos**
- size_t **chunk_pos**

3.7.1 Detailed Description

Internal structure: represents a position in a buffer.

3.7.2 Field Documentation

3.7.2.1 const chunk_t∗ buf_pos_t::chunk

Which chunk are we pointing to?

Referenced by buf_find_pos_of_char(), buf_matches_at_pos(), buf_pos_inc(), and buf_pos_init().

3.7.2.2 int buf_pos_t::pos

Which character inside the chunk's data are we pointing to?

Referenced by buf_find_pos_of_char(), buf_find_string_offset(), buf_matches_at_pos(), buf_pos_inc(), and buf_pos_init().

3.7.2.3 size_t buf_pos_t::chunk_pos

Total length of all previous chunks.

Referenced by buf_find_pos_of_char(), buf_find_string_offset(), buf_pos_inc(), and buf_pos_init().

The documentation for this struct was generated from the following file:

- **buffers.c**

3.8 buf_t Struct Reference

Data Fields

- uint32_t **magic**
- size_t **datalen**
- size_t **default_chunk_size**
- **chunk_t** ∗ **head**
- **chunk_t** ∗ **tail**

3.8.1 Detailed Description

A resizeable buffer, optimized for reading and writing.

3.8.2 Field Documentation

3.8.2.1 uint32_t buf_t::magic

Magic cookie for debugging: Must be set to BUFFER_MAGIC.

Referenced by assert_buf_ok(), buf_free(), and buf_new().

3.8.2.2 size_t buf_t::datalen

How many bytes is this buffer holding right now?

Referenced by assert_buf_ok(), buf_clear(), buf_datalen(), buf_pullup(), buf_remove_from_front(), fetch_from_buf(), fetch_from_buf_http(), fetch_from_buf_socks(), fetch_from_buf_socks_client(), fetch_var_cell_from_buf(), flush_chunk_tls(), move_buf_to_buf(), peek_buf_has_control0_command(), read_to_chunk(), read_to_chunk_tls(), write_to_buf(), and write_to_buf_zlib().

3.8.2.3 size_t buf_t::default_chunk_size

Don't allocate any chunks smaller than this for this buffer.

Referenced by buf_add_chunk_with_capacity(), buf_new(), and buf_new_with_capacity().

3.8.2.4 chunk_t∗ buf_t::head

First chunk in the list, or NULL for none.

Referenced by assert_buf_ok(), buf_add_chunk_with_capacity(), buf_allocation(), buf_clear(), buf_find_offset_of_char(), buf_pos_init(), buf_pullup(), buf_remove_from_front(), fetch_from_buf_http(), fetch_from_buf_line(), fetch_from_buf_socks(), fetch_from_buf_socks_client(), flush_buf(), flush_buf_tls(), and peek_from_buf().

3.8.2.5 chunk_t∗ buf_t::tail

Last chunk in the list, or NULL for none.

Referenced by assert_buf_ok(), buf_add_chunk_with_capacity(), buf_clear(), buf_pullup(), buf_remove_from_front(), buf_slack(), read_to_buf(), read_to_buf_tls(), write_to_buf(), and write_to_buf_zlib().

The documentation for this struct was generated from the following file:

- **buffers.c**

3.9 bw_array_t Struct Reference

Data Fields

- uint64_t **obs** [NUM_SECS_ROLLING_MEASURE]
- int **cur_obs_idx**
- time_t **cur_obs_time**
- uint64_t **total_obs**
- uint64_t **max_total**
- uint64_t **total_in_period**
- time_t **next_period**
- int **next_max_idx**
- int **num_maxes_set**
- uint64_t **maxima** [NUM_TOTALS]
- uint64_t **totals** [NUM_TOTALS]

3.9.1 Detailed Description

Structure to track bandwidth use, and remember the maxima for a given time period.

3.9.2 Field Documentation

3.9.2.1 uint64_t bw_array_t::obs[NUM_SECS_ROLLING_MEASURE]

Observation array: Total number of bytes transferred in each of the last NUM_SECS_-ROLLING_MEASURE seconds. This is used as a circular array.

Referenced by add_obs(), advance_obs(), and rep_hist_load_state().

3.9.2.2 int bw_array_t::cur_obs_idx

Current position in obs.

Referenced by add_obs(), and advance_obs().

3.9.2.3 time_t bw_array_t::cur_obs_time

Time represented in obs[cur_obs_idx]

Referenced by add_obs(), advance_obs(), bw_array_new(), and rep_hist_load_state().

3.9.2.4 uint64_t bw_array_t::total_obs

Total for all members of obs except obs[cur_obs_idx]

Referenced by advance_obs(), and rep_hist_load_state().

3.9.2.5 uint64_t bw_array_t::max_total

Largest value that total_obs has taken on in the current period.

Referenced by advance_obs(), commit_max(), and rep_hist_load_state().

3.9.2.6 uint64_t bw_array_t::total_in_period

Total bytes transferred in the current period.

Referenced by add_obs(), commit_max(), and rep_hist_update_state().

3.9.2.7 time_t bw_array_t::next_period

When does the next period begin?

Referenced by advance_obs(), bw_array_new(), commit_max(), rep_hist_get_bandwidth_lines(), rep_hist_load_state(), and rep_hist_update_state().

3.9.2.8 int bw_array_t::next_max_idx

Where in 'maxima' should the maximum bandwidth usage for the current period be stored?

Referenced by commit_max(), and rep_hist_fill_bandwidth_history().

3.9.2.9 int bw_array_t::num_maxes_set

How many values in maxima/totals have been set ever?

Referenced by commit_max(), and rep_hist_fill_bandwidth_history().

3.9.2.10 uint64_t bw_array_t::maxima[NUM_TOTALS]

Circular array of the maximum bandwidth-per-NUM_SECS_ROLLING_MEASURE usage for the last NUM_TOTALS periods

Referenced by commit_max(), find_largest_max(), and rep_hist_load_state().

3.9.2.11 uint64_t bw_array_t::totals[NUM_TOTALS]

Circular array of the total bandwidth usage for the last NUM_TOTALS periods

Referenced by commit_max(), and rep_hist_fill_bandwidth_history().

The documentation for this struct was generated from the following file:

- **rephist.c**

3.10 c_hist_t Struct Reference

Data Fields

- char **country** [3]
- unsigned **total**

3.10.1 Detailed Description

Helper type: used to sort per-country totals by value.

3.10.2 Field Documentation

3.10.2.1 char c_hist_t::country[3]

Two-letter country code.

Referenced by _c_hist_compare(), and geoip_get_client_history().

3.10.2.2 unsigned c_hist_t::total

Total IP addresses seen in this country.

Referenced by _c_hist_compare(), and geoip_get_client_history().

The documentation for this struct was generated from the following file:

- **geoip.c**

3.11 cached_dir_t Struct Reference

`#include <or.h>`

Data Fields

- char * **dir**
- char * **dir_z**
- size_t **dir_len**
- size_t **dir_z_len**
- time_t **published**
- **digests_t digests**
- int **refcnt**

3.11.1 Detailed Description

A **cached_dir_t** (p. 23) represents a cacheable directory object, along with its compressed form.

3.11.2 Field Documentation

3.11.2.1 char* cached_dir_t::dir

Contents of this object, NUL-terminated.

Referenced by clear_cached_dir(), directory_handle_command_get(), directory_set_dirty(), dirserv_regenerate_directory(), dirvote_perform_vote(), new_cached_dir(), and set_cached_dir().

3.11.2.2 char* cached_dir_t::dir_z

Compressed contents of this object.

Referenced by clear_cached_dir(), connection_dirserv_add_dir_bytes_to_outbuf(), directory_handle_command_get(), new_cached_dir(), and set_cached_dir().

3.11.2.3 size_t cached_dir_t::dir_len

Length of **dir** (not counting its NUL).

Referenced by directory_handle_command_get(), dirserv_estimate_data_size(), dirserv_regenerate_directory(), dirvote_perform_vote(), new_cached_dir(), and set_cached_dir().

3.11.2.4 size_t cached_dir_t::dir_z_len

Length of **dir_z**.

Referenced by connection_dirserv_add_dir_bytes_to_outbuf(), directory_handle_command_get(), dirserv_estimate_data_size(), new_cached_dir(), and set_cached_dir().

3.11.2.5 time_t cached_dir_t::published

When was this object published.

Referenced by directory_handle_command_get(), directory_set_dirty(), dirserv_clear_old_v1_info(), dirserv_remove_old_statuses(), dirserv_set_cached_networkstatus_v2(), networkstatus_set_current_consensus(), new_cached_dir(), and set_cached_dir().

3.11.2.6 digests_t cached_dir_t::digests

Digests of this object (networkstatus only)

Referenced by dirserv_set_cached_consensus_networkstatus(), and networkstatus_set_current_consensus().

3.11.2.7 int cached_dir_t::refcnt

Reference count for this **cached_dir_t** (p. 23).

Referenced by cached_dir_decref(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), directory_handle_command_get(), and new_cached_dir().

The documentation for this struct was generated from the following file:

- **or.h**

3.12 cached_resolve_t Struct Reference

Public Member Functions

- **HT_ENTRY** (**cached_resolve_t**)

Data Fields

- **result**
- uint8_t **state**
- uint8_t **is_reverse**
- time_t **expire**
- uint32_t **ttl**
- **pending_connection_t** ∗ **pending_connections**
- int **minheap_idx**

3.12.1 Detailed Description

A DNS request: possibly completed, possibly pending; cached_resolve structs are stored at the OR side in a hash table, and as a linked list from oldest to newest.

3.12.2 Member Function Documentation

3.12.2.1 cached_resolve_t::HT_ENTRY (cached_resolve_t) `[inline]`

< IPv6 addr for **address**.

< IPv4 addr for **address**.

< Hostname for **address** (if a reverse lookup)

3.12.3 Field Documentation

3.12.3.1 uint8_t cached_resolve_t::state

Is this cached entry pending/done/valid/failed?

Referenced by add_answer_to_cache(), assert_resolve_ok(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), and purge_expired_resolves().

3.12.3.2 uint8_t cached_resolve_t::is_reverse

Is this a reverse (addr-to-hostname) lookup?

Referenced by _free_cached_resolve(), add_answer_to_cache(), assert_resolve_ok(), dns_resolve_impl(), and purge_expired_resolves().

3.12.3.3 time_t cached_resolve_t::expire

Remove items from cache after this time.

Referenced by _compare_cached_resolves_by_expiry(), dns_resolve_impl(), purge_expired_resolves(), and set_expiry().

3.12.3.4 uint32_t cached_resolve_t::ttl

What TTL did the nameserver tell us?

Referenced by add_answer_to_cache(), assert_resolve_ok(), and dns_resolve_impl().

3.12.3.5 pending_connection_t* cached_resolve_t::pending_connections

Connections that want to know when we get an answer for this resolve.

Referenced by _free_cached_resolve(), assert_connection_edge_not_dns_pending(), assert_resolve_ok(), connection_dns_remove(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), and purge_expired_resolves().

3.12.3.6 int cached_resolve_t::minheap_idx

Position of this element in the heap

Referenced by dns_resolve_impl().

The documentation for this struct was generated from the following file:

- **dns.c**

3.13 cell_ewma_t Struct Reference

`#include <or.h>`

Data Fields

- unsigned **last_adjusted_tick**
- double **cell_count**
- unsigned int **is_for_p_conn**: 1
- int **heap_index**

3.13.1 Detailed Description

The **cell_ewma_t** (p. 27) structure keeps track of how many cells a circuit has transferred recently. It keeps an EWMA (exponentially weighted moving average) of the number of cells flushed from the circuit queue onto a connection in **connection_or_flush_from_first_active_circuit()** (p. 877).

3.13.2 Field Documentation

3.13.2.1 unsigned cell_ewma_t::last_adjusted_tick

The last 'tick' at which we recalibrated cell_count.

A cell sent at exactly the start of this tick has weight 1.0. Cells sent since the start of this tick have weight greater than 1.0; ones sent earlier have less weight.

Referenced by init_circuit_base(), or_circuit_new(), and scale_single_cell_ewma().

3.13.2.2 double cell_ewma_t::cell_count

The EWMA of the cell count.

Referenced by compare_cell_ewma_counts(), init_circuit_base(), or_circuit_new(), and scale_single_cell_ewma().

3.13.2.3 unsigned int cell_ewma_t::is_for_p_conn

True iff this is the cell count for a circuit's previous connection.

Referenced by assert_active_circuits_ok(), cell_ewma_to_circuit(), init_circuit_base(), and or_circuit_new().

3.13.2.4 int cell_ewma_t::heap_index

The position of the circuit within the OR connection's priority queue.

Referenced by add_cell_ewma_to_conn(), assert_active_circuits_ok(), init_circuit_base(), or_circuit_new(), and remove_cell_ewma_from_conn().

The documentation for this struct was generated from the following file:

- **or.h**

3.14 cell_queue_t Struct Reference

`#include <or.h>`

Data Fields

- **packed_cell_t** ∗ **head**
- **packed_cell_t** ∗ **tail**
- int **n**
- **insertion_time_queue_t** ∗ **insertion_times**

3.14.1 Detailed Description

A queue of cells on a circuit, waiting to be added to the or_connection_t's outbuf.

3.14.2 Field Documentation

3.14.2.1 packed_cell_t∗ cell_queue_t::head

The first cell, or NULL if the queue is empty.

Referenced by cell_queue_append(), cell_queue_clear(), cell_queue_pop(), and connection_or_flush_from_first_active_circuit().

3.14.2.2 packed_cell_t∗ cell_queue_t::tail

The last cell, or NULL if the queue is empty.

Referenced by cell_queue_append(), cell_queue_clear(), and cell_queue_pop().

3.14.2.3 int cell_queue_t::n

The number of cells in the queue.

Referenced by append_cell_to_circuit_queue(), cell_queue_append(), cell_queue_clear(), cell_queue_pop(), circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), connection_or_flush_from_first_active_circuit(), dump_cell_pool_usage(), and HT_HEAD().

3.14.2.4 insertion_time_queue_t∗ cell_queue_t::insertion_times

Insertion times of cells.

Referenced by cell_queue_append_packed_copy(), cell_queue_clear(), and connection_or_flush_from_first_active_circuit().

The documentation for this struct was generated from the following file:

- **or.h**

3.15 cell_t Struct Reference

`#include <or.h>`

Data Fields

- **circid_t circ_id**
- uint8_t **command**
- char **payload** [CELL_PAYLOAD_SIZE]

3.15.1 Detailed Description

Parsed onion routing cell. All communication between nodes is via cells.

3.15.2 Field Documentation

3.15.2.1 circid_t cell_t::circ_id

Circuit which received the cell.

Referenced by cell_pack(), cell_unpack(), circuit_deliver_create_cell(), circuit_receive_relay_cell(), command_process_create_cell(), command_process_created_cell(), command_process_destroy_cell(), command_process_relay_cell(), connection_or_send_destroy(), onionskin_answer(), and relay_send_command_from_edge().

3.15.2.2 uint8_t cell_t::command

Type of the cell: one of CELL_PADDING, CELL_CREATE, CELL_DESTROY, etc

Referenced by append_cell_to_circuit_queue(), cell_pack(), cell_unpack(), circuit_deliver_create_cell(), circuit_receive_relay_cell(), command_process_cell(), command_process_create_cell(), command_process_created_cell(), command_process_relay_cell(), connection_or_send_destroy(), connection_or_send_netinfo(), connection_or_write_cell_to_buf(), onionskin_answer(), relay_send_command_from_edge(), and run_connection_housekeeping().

3.15.2.3 char cell_t::payload[CELL_PAYLOAD_SIZE]

Cell body.

Referenced by cell_pack(), cell_unpack(), circuit_deliver_create_cell(), circuit_extend(), circuit_package_relay_cell(), command_process_create_cell(), command_process_created_cell(), command_process_destroy_cell(), command_process_netinfo_cell(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_or_send_destroy(), connection_or_send_netinfo(), onionskin_answer(), relay_crypt(), relay_digest_matches(), relay_lookup_conn(), relay_send_command_from_edge(), and relay_set_digest().

The documentation for this struct was generated from the following file:

- **or.h**

3.16 cert_list_t Struct Reference

Data Fields

- **download_status_t dl_status**
- **smartlist_t * certs**

3.16.1 Detailed Description

List of for a given authority, and download status for latest certificate.

The documentation for this struct was generated from the following file:

- **routerlist.c**

3.17 chunk_freelist_t Struct Reference

Data Fields

- size_t **alloc_size**
- int **max_length**
- int **slack**
- int **cur_length**
- int **lowest_length**
- uint64_t **n_alloc**
- uint64_t **n_free**
- uint64_t **n_hit**
- **chunk_t** * **head**

3.17.1 Detailed Description

A freelist of chunks.

3.17.2 Field Documentation

3.17.2.1 size_t chunk_freelist_t::alloc_size

What size chunks does this freelist hold?

Referenced by assert_freelist_ok(), buf_dump_freelist_sizes(), buf_shrink_freelists(), and get_freelist().

3.17.2.2 int chunk_freelist_t::max_length

Never allow more than this number of chunks in the freelist.

Referenced by chunk_free_unchecked().

3.17.2.3 int chunk_freelist_t::slack

When trimming the freelist, leave this number of extra chunks beyond lowest_length.

Referenced by buf_shrink_freelists().

3.17.2.4 int chunk_freelist_t::cur_length

How many chunks on the freelist now?

Referenced by assert_freelist_ok(), buf_shrink_freelists(), chunk_free_unchecked(), and chunk_new_with_alloc_size().

3.17.2.5 int chunk_freelist_t::lowest_length

What's the smallest value of cur_length since the last time we cleaned this freelist?

Referenced by assert_freelist_ok(), buf_shrink_freelists(), and chunk_new_with_alloc_size().

3.17.2.6 chunk_t* chunk_freelist_t::head

First chunk on the freelist.

Referenced by assert_freelist_ok(), buf_shrink_freelists(), chunk_free_unchecked(), and chunk_new_with_alloc_size().

The documentation for this struct was generated from the following file:

- **buffers.c**

3.18 chunk_t Struct Reference

Data Fields

- struct **chunk_t** ∗ **next**
- size_t **datalen**
- size_t **memlen**
- char ∗ **data**
- char **mem** [1]

3.18.1 Detailed Description

A single chunk on a buffer or in a freelist.

3.18.2 Field Documentation

3.18.2.1 struct chunk_t∗ chunk_t::next `[read]`

The next chunk on the buffer or freelist.

Referenced by assert_buf_ok(), assert_freelist_ok(), buf_add_chunk_with_capacity(), buf_allocation(), buf_clear(), buf_find_offset_of_char(), buf_find_pos_of_char(), buf_pos_inc(), buf_pullup(), buf_remove_from_front(), buf_shrink_freelists(), chunk_free_unchecked(), chunk_new_with_alloc_size(), flush_chunk(), peek_from_buf(), and read_to_chunk().

3.18.2.2 size_t chunk_t::datalen

The number of bytes stored in this chunk

Referenced by assert_buf_ok(), buf_find_offset_of_char(), buf_find_pos_of_char(), buf_pos_inc(), buf_pullup(), buf_remove_from_front(), chunk_new_with_alloc_size(), CHUNK_REMAINING_CAPACITY(), chunk_repack(), CHUNK_WRITE_PTR(), fetch_from_buf_http(), fetch_from_buf_socks(), fetch_from_buf_socks_client(), flush_buf(), flush_buf_tls(), flush_chunk(), peek_from_buf(), read_to_chunk(), read_to_chunk_tls(), write_to_buf(), and write_to_buf_zlib().

3.18.2.3 size_t chunk_t::memlen

The number of usable bytes of storage in **mem**.

Referenced by assert_buf_ok(), assert_freelist_ok(), buf_allocation(), buf_pullup(), chunk_free_unchecked(), chunk_grow(), chunk_new_with_alloc_size(), CHUNK_REMAINING_CAPACITY(), read_to_buf(), and read_to_buf_tls().

3.18.2.4 char∗ chunk_t::data

A pointer to the first byte of data stored in **mem**.

Referenced by assert_buf_ok(), buf_find_offset_of_char(), buf_find_pos_of_char(), buf_matches_at_pos(), buf_pullup(), buf_remove_from_front(), chunk_grow(), chunk_new_with_alloc_size(), CHUNK_REMAINING_CAPACITY(), chunk_repack(), CHUNK_-

WRITE_PTR(), fetch_from_buf_http(), fetch_from_buf_socks(), fetch_from_buf_socks_client(), flush_chunk(), flush_chunk_tls(), and peek_from_buf().

3.18.2.5 char chunk_t::mem[1]

The actual memory used for storage in this chunk. May be more than one byte long.

Referenced by assert_buf_ok(), chunk_grow(), chunk_new_with_alloc_size(), CHUNK_REMAINING_CAPACITY(), and chunk_repack().

The documentation for this struct was generated from the following file:

- **buffers.c**

3.19 cipher_info_t Struct Reference

Data Fields

- unsigned **id**
- const char ∗ **name**

3.19.1 Detailed Description

Holds a cipher that we want to advertise, and its 2-byte ID.

The documentation for this struct was generated from the following file:

- **tortls.c**

3.20 circuit_build_times_t Struct Reference

`#include <or.h>`

Data Fields

- **build_time_t circuit_build_times** [CBT_NCIRCUITS_TO_OBSERVE]
- int **build_times_idx**
- int **total_build_times**
- **network_liveness_t liveness**
- time_t **last_circ_at**
- int **pre_timeouts**
- **build_time_t Xm**
- double **alpha**
- int **have_computed_timeout**
- double **timeout_ms**

3.20.1 Detailed Description

Structure for circuit build times history

3.20.2 Field Documentation

3.20.2.1 build_time_t circuit_build_times_t::circuit_build_times[CBT_NCIRCUITS_TO_OBSERVE]

The circular array of recorded build times in milliseconds

Referenced by circuit_build_times_add_time(), circuit_build_times_create_histogram(), circuit_build_times_max(), circuit_build_times_parse_state(), circuit_build_times_reset(), circuit_build_times_rewind_history(), and circuit_build_times_update_alpha().

3.20.2.2 int circuit_build_times_t::build_times_idx

Current index in the circuit_build_times circular array

Referenced by circuit_build_times_add_time(), circuit_build_times_reset(), and circuit_build_times_rewind_history().

3.20.2.3 int circuit_build_times_t::total_build_times

Total number of build times accumulated. Max CBT_NCIRCUITS_TO_OBSERVE

Referenced by circuit_build_times_add_time(), circuit_build_times_add_timeout(), circuit_build_times_count_pretimeouts(), circuit_build_times_needs_circuits(), circuit_build_times_network_check_changed(), circuit_build_times_parse_state(), circuit_build_times_reset(), circuit_build_times_rewind_history(), circuit_build_times_set_timeout(), circuit_build_times_update_alpha(), circuit_build_times_update_state(), and control_event_buildtimeout_set().

3.20.2.4 network_liveness_t circuit_build_times_t::liveness

Information about the state of our local network connection

Referenced by circuit_build_times_init(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), circuit_build_times_network_circ_success(), circuit_build_times_network_is_live(), circuit_build_times_network_timeout(), and circuit_build_times_new_consensus_params().

3.20.2.5 time_t circuit_build_times_t::last_circ_at

Last time we built a circuit. Used to decide to build new test circs

Referenced by circuit_build_times_needs_circuits_now(), and origin_circuit_new().

3.20.2.6 int circuit_build_times_t::pre_timeouts

Number of timeouts that have happened before estimating pareto parameters

Referenced by circuit_build_times_add_timeout(), circuit_build_times_count_pretimeouts(), circuit_build_times_reset(), and circuit_build_times_rewind_history().

3.20.2.7 build_time_t circuit_build_times_t::Xm

"Minimum" value of our pareto distribution (actually mode)

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_calculate_-timeout(), circuit_build_times_cdf(), circuit_build_times_count_pretimeouts(), circuit_-build_times_initial_alpha(), circuit_build_times_set_timeout(), circuit_build_times_-update_alpha(), and control_event_buildtimeout_set().

3.20.2.8 double circuit_build_times_t::alpha

alpha exponent for pareto dist.

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_calculate_-timeout(), circuit_build_times_cdf(), circuit_build_times_initial_alpha(), circuit_build_-times_set_timeout(), circuit_build_times_update_alpha(), and control_event_buildtimeout_-set().

3.20.2.9 int circuit_build_times_t::have_computed_timeout

Have we computed a timeout?

Referenced by circuit_build_times_add_timeout(), circuit_build_times_reset(), and circuit_-build_times_set_timeout().

3.20.2.10 double circuit_build_times_t::timeout_ms

The exact value for that timeout in milliseconds

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_count_-pretimeouts(), circuit_build_times_init(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), circuit_build_times_network_timeout(), circuit_build_times_set_timeout(), circuit_expire_building(), and control_event_buildtimeout_set().

The documentation for this struct was generated from the following file:

- **or.h**

3.21 circuit_t Struct Reference

`#include <or.h>`

Data Fields

- uint32_t **magic**
- **cell_queue_t n_conn_cells**
- **or_connection_t** ∗ **n_conn**
- **circid_t n_circ_id**
- **extend_info_t** ∗ **n_hop**
- unsigned int **streams_blocked_on_n_conn**: 1
- unsigned int **streams_blocked_on_p_conn**: 1
- uint8_t **state**
- uint8_t **purpose**
- int **package_window**
- int **deliver_window**
- char ∗ **n_conn_onionskin**
- time_t **timestamp_created**
- time_t **timestamp_dirty**
- struct timeval **highres_created**
- uint16_t **marked_for_close**
- const char ∗ **marked_for_close_file**
- struct **circuit_t** ∗ **next_active_on_n_conn**
- struct **circuit_t** ∗ **prev_active_on_n_conn**
- struct **circuit_t** ∗ **next**
- uint64_t **dirreq_id**
- **cell_ewma_t n_cell_ewma**

3.21.1 Detailed Description

A circuit is a path over the onion routing network. Applications can connect to one end of the circuit, and can create exit connections at the other end of the circuit. AP and exit connections have only one circuit associated with them (and thus these connection types are closed when the circuit is closed), whereas OR connections multiplex many circuits at once, and stay standing even when there are no circuits running over them.

A **circuit_t** (p. 40) structure can fill one of two roles. First, a **or_circuit_t** (p. 135) links two connections together: either an edge connection and an OR connection, or two OR connections. (When joined to an OR connection, a **circuit_t** (p. 40) affects only cells sent to a particular circID on that connection. When joined to an edge connection, a **circuit_t** (p. 40) affects all data.)

Second, an **origin_circuit_t** (p. 181) holds the cipher keys and state for sending data along a given circuit. At the OP, it has a sequence of ciphers, each of which is shared with a single OR along the circuit. Separate ciphers are used for data going "forward" (away from the OP) and "backward" (towards the OP). At the OR, a circuit has only two stream ciphers: one for data going forward, and one for data going backward.

3.21.2 Field Documentation

3.21.2.1 uint32_t circuit_t::magic

For memory and type debugging: must equal ORIGIN_CIRCUIT_MAGIC or OR_CIRCUIT_MAGIC.

Referenced by assert_circuit_ok(), circuit_free(), circuit_get_by_edge_conn(), or_circuit_new(), origin_circuit_new(), TO_OR_CIRCUIT(), and TO_ORIGIN_CIRCUIT().

3.21.2.2 cell_queue_t circuit_t::n_conn_cells

Queue of cells waiting to be transmitted on n_conn.

Referenced by append_cell_to_circuit_queue(), circuit_free(), circuit_set_n_circid_orconn(), connection_or_flush_from_first_active_circuit(), and dump_cell_pool_usage().

3.21.2.3 or_connection_t∗ circuit_t::n_conn

The OR connection that is next in this circuit.

Referenced by _circuit_mark_for_close(), assert_active_circuits_ok(), assert_circuit_ok(), circuit_build_failed(), circuit_deliver_create_cell(), circuit_dump_by_conn(), circuit_expire_building(), circuit_expire_old_circuits_serverside(), circuit_extend(), circuit_get_by_circid_orconn_impl(), circuit_handle_first_hop(), circuit_is_acceptable(), circuit_package_relay_cell(), circuit_receive_relay_cell(), circuit_send_next_onion_skin(), circuit_unlink_all_from_or_conn(), connection_edge_process_relay_cell(), connection_or_flush_from_first_active_circuit(), make_circuit_active_on_conn(), make_circuit_inactive_on_conn(), next_circ_on_conn_p(), prev_circ_on_conn_p(), relay_send_command_from_edge(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), and set_streams_blocked_on_circ().

3.21.2.4 circid_t circuit_t::n_circ_id

The circuit_id used in the next (forward) hop of this circuit.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_deliver_create_cell(), circuit_dump_by_conn(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_get_by_circid_orconn_impl(), circuit_receive_relay_cell(), command_process_created_cell(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_process_relay_cell(), link_apconn_to_circ(), relay_send_command_from_edge(), rend_client_introduction_acked(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

3.21.2.5 extend_info_t∗ circuit_t::n_hop

The hop to which we want to extend this circuit. Should be NULL if the circuit has attached to a connection.

Referenced by assert_circuit_ok(), circuit_dump_by_conn(), circuit_extend(), circuit_free(), and circuit_handle_first_hop().

3.21.2.6 unsigned int circuit_t::streams_blocked_on_n_conn

True iff we are waiting for n_conn_cells to become less full before allowing p_streams to add any more cells. (Origin circuit only.)

Referenced by append_cell_to_circuit_queue(), connection_or_flush_from_first_active_circuit(), and set_streams_blocked_on_circ().

3.21.2.7 unsigned int circuit_t::streams_blocked_on_p_conn

True iff we are waiting for p_conn_cells to become less full before allowing n_streams to add any more cells. (OR circuit only.)

Referenced by append_cell_to_circuit_queue(), connection_or_flush_from_first_active_circuit(), and set_streams_blocked_on_circ().

3.21.2.8 uint8_t circuit_t::state

Current status of this circuit.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_build_failed(), circuit_dump_by_conn(), circuit_dump_details(), circuit_enough_testing_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_find_to_cannibalize(), circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_list_path_impl(), circuit_send_next_onion_skin(), circuit_set_state(), command_process_relay_cell(), connection_ap_handshake_attach_chosen_circuit(), count_established_intro_points(), getinfo_helper_events(), handle_control_attachstream(), handle_control_extendcircuit(), rend_service_dump_stats(), rend_service_launch_establish_intro(), and router_perform_bandwidth_test().

3.21.2.9 uint8_t circuit_t::purpose

Why are we creating this circuit?

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_build_failed(), circuit_enough_testing_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_find_to_cannibalize(), circuit_get_by_rend_query_and_purpose(), circuit_get_by_rend_token_and_purpose(), circuit_get_next_by_pk_and_purpose(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_is_better(), circuit_launch_by_extend_info(), circuit_predict_and_launch_new(), circuit_receive_relay_cell(), circuit_stream_is_being_handled(), command_process_create_cell(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_end(), connection_edge_process_relay_cell(), connection_exit_begin_conn(), control_event_circuit_status(), count_established_intro_points(), getinfo_helper_events(), handle_control_setcircuitpurpose(), onion_extend_cpath(), onion_pick_cpath_exit(), origin_circuit_init(), relay_send_command_from_edge(), relay_send_end_cell_from_edge(), rend_client_introcirc_has_opened(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_rendcirc_has_opened(), rend_client_rendezvous_acked(), rend_client_send_establish_rendezvous(), rend_client_send_introduction(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_relaunch_rendezvous(), rend_service_rendezvous_has_opened(), rend_service_-

set_connection_addr_port(), rend_service_update_descriptor(), and warn_if_last_router_excluded().

3.21.2.10 int circuit_t::package_window

How many relay data cells can we package (read from edge streams) on this circuit before we receive a circuit-level sendme cell asking for more?

Referenced by assert_circuit_ok(), circuit_consider_stop_edge_reading(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), and init_circuit_base().

3.21.2.11 int circuit_t::deliver_window

How many relay data cells will we deliver (write to edge streams) on this circuit? When deliver_window gets low, we send some circuit-level sendme cells to indicate that we're willing to accept more.

Referenced by assert_circuit_ok(), circuit_consider_sending_sendme(), connection_edge_process_relay_cell(), and init_circuit_base().

3.21.2.12 char∗ circuit_t::n_conn_onionskin

For storage while n_conn is pending (state CIRCUIT_STATE_OR_WAIT). When defined, it is always length ONIONSKIN_CHALLENGE_LEN.

Referenced by assert_circuit_ok(), circuit_extend(), circuit_free(), and circuit_set_state().

3.21.2.13 time_t circuit_t::timestamp_created

When was this circuit created?

Referenced by circuit_dump_details(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_get_best(), circuit_is_better(), circuit_launch_by_extend_info(), init_circuit_base(), and rep_hist_buffer_stats_add_circ().

3.21.2.14 time_t circuit_t::timestamp_dirty

When the circuit was first used, or 0 if the circuit is clean.

Referenced by circuit_expire_all_dirty_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_find_to_cannibalize(), circuit_is_acceptable(), circuit_is_better(), circuit_mark_all_unused_circs(), circuit_predict_and_launch_new(), circuit_stream_is_being_handled(), connection_ap_expire_beginning(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_attach_circuit(), connection_ap_process_end_not_open(), and router_perform_bandwidth_test().

3.21.2.15 struct timeval circuit_t::highres_created `[read]`

When exactly was the circuit created?

Referenced by circuit_send_next_onion_skin(), and init_circuit_base().

3.21.2.16 uint16_t circuit_t::marked_for_close

Should we close this circuit at the end of the main loop? (If true, holds the line number where this circuit was marked.)

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_close_all_marked(), circuit_dump_by_conn(), circuit_enough_testing_circs(), circuit_expire_all_dirty_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_expire_old_circuits_serverside(), circuit_find_to_cannibalize(), circuit_get_by_circid_orconn(), circuit_get_by_global_id(), circuit_get_by_rend_query_and_purpose(), circuit_get_by_rend_token_and_purpose(), circuit_get_next_by_pk_and_purpose(), circuit_is_acceptable(), circuit_mark_all_unused_circs(), circuit_predict_and_launch_new(), circuit_receive_relay_cell(), circuit_stream_is_being_handled(), circuit_unlink_all_from_or_conn(), connection_ap_handshake_attach_circuit(), connection_edge_end(), count_established_intro_points(), and getinfo_helper_events().

3.21.2.17 const char* circuit_t::marked_for_close_file

For debugging: in which file was this circuit marked for close?

Referenced by _circuit_mark_for_close().

3.21.2.18 struct circuit_t* circuit_t::next_active_on_n_conn [read]

Next circuit in the doubly-linked ring of circuits waiting to add cells to n_conn. NULL if we have no cells pending, or if we're not linked to an OR connection.

Referenced by circuit_set_n_circid_orconn(), and next_circ_on_conn_p().

3.21.2.19 struct circuit_t* circuit_t::prev_active_on_n_conn [read]

Previous circuit in the doubly-linked ring of circuits waiting to add cells to n_conn. NULL if we have no cells pending, or if we're not linked to an OR connection.

Referenced by prev_circ_on_conn_p().

3.21.2.20 struct circuit_t* circuit_t::next [read]

Next circuit in linked list of all circuits.

Referenced by circuit_add(), circuit_close_all_marked(), circuit_dump_by_conn(), circuit_enough_testing_circs(), circuit_expire_all_dirty_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_expire_old_circuits_serverside(), circuit_find_to_cannibalize(), circuit_free_all(), circuit_get_best(), circuit_get_by_circid_orconn_impl(), circuit_get_by_global_id(), circuit_get_by_rend_query_and_purpose(), circuit_get_by_rend_token_and_purpose(), circuit_get_next_by_pk_and_purpose(), circuit_mark_all_unused_circs(), circuit_predict_and_launch_new(), circuit_stream_is_being_handled(), circuit_unlink_all_from_or_conn(), connection_ap_handshake_attach_circuit(), count_established_intro_points(), dump_cell_pool_usage(), getinfo_helper_events(), rend_config_services(), and rep_hist_buffer_stats_write().

3.21.2.21 uint64_t circuit_t::dirreq_id

Unique ID for measuring tunneled network status requests.

Referenced by connection_edge_process_relay_cell(), connection_exit_begin_conn(), and relay_send_command_from_edge().

3.21.2.22 cell_ewma_t circuit_t::n_cell_ewma

The EWMA count for the number of cells flushed from the n_conn_cells queue. Used to determine which circuit to flush from next.

Referenced by assert_active_circuits_ok(), init_circuit_base(), make_circuit_active_on_conn(), and make_circuit_inactive_on_conn().

The documentation for this struct was generated from the following file:

- **or.h**

3.22 clientmap_entry_t Struct Reference

3.22.1 Detailed Description

Entry in a map from IP address to the last time we've seen an incoming connection from that IP address. Used by bridges only, to track which countries have them blocked.

The documentation for this struct was generated from the following file:

- **geoip.c**

3.23 config_abbrev_t Struct Reference

Data Fields

- const char * **abbreviated**
- const char * **full**
- int **commandline_only**
- int **warn**

3.23.1 Detailed Description

An abbreviation for a configuration option allowed on the command line.

The documentation for this struct was generated from the following file:

- **config.c**

3.24 config_format_t Struct Reference

Data Fields

- size_t **size**
- uint32_t **magic**
- off_t **magic_offset**
- **config_abbrev_t** ∗ **abbrevs**
- **config_var_t** ∗ **vars**
- **validate_fn_t** **validate_fn**
- **config_var_t** ∗ **extra**

3.24.1 Detailed Description

Information on the keys, value types, key-to-struct-member mappings, variable descriptions, validation functions, and abbreviations for a configuration or storage format.

3.24.2 Field Documentation

3.24.2.1 size_t config_format_t::size

Size of the struct that everything gets parsed into.

Referenced by config_alloc().

3.24.2.2 uint32_t config_format_t::magic

Required 'magic value' to make sure we have a struct of the right type.

Referenced by config_alloc().

3.24.2.3 off_t config_format_t::magic_offset

Offset of the magic value within the struct.

Referenced by config_alloc().

3.24.2.4 config_abbrev_t∗ config_format_t::abbrevs

List of abbreviations that we expand when parsing this format.

Referenced by expand_abbrev().

3.24.2.5 config_var_t∗ config_format_t::vars

List of variables we recognize, their default values, and where we stick them in the structure.

Referenced by config_dump(), config_find_option(), config_free(), config_init(), and options_dup().

3.24.2.6 validate_fn_t config_format_t::validate_fn

Function to validate config.

Referenced by config_dump().

3.24.2.7 config_var_t∗ config_format_t::extra

If present, extra is a LINELIST variable for unrecognized lines. Otherwise, unrecognized lines are an error.

Referenced by config_assign_line(), config_dump(), and config_free().

The documentation for this struct was generated from the following file:

- **config.c**

3.25 config_line_t Struct Reference

`#include <or.h>`

Data Fields

- char * **key**
- char * **value**
- struct **config_line_t** * **next**

3.25.1 Detailed Description

A linked list of lines in a config file.

The documentation for this struct was generated from the following file:

- **or.h**

3.26 config_var_description_t Struct Reference

Data Fields

- const char * **name**
- const char * **description**

3.26.1 Detailed Description

Represents an English description of a configuration variable; used when generating configuration file comments.

The documentation for this struct was generated from the following file:

- **config.c**

3.27 config_var_t Struct Reference

Data Fields

- const char * **name**
- **config_type_t type**
- off_t **var_offset**
- const char * **initvalue**

3.27.1 Detailed Description

A variable allowed in the configuration file or on the command line.

3.27.2 Field Documentation

3.27.2.1 const char* config_var_t::name

The full keyword (case insensitive).

Referenced by config_assign_line(), config_dump(), config_find_option(), config_free(), config_init(), get_assigned_option(), getinfo_helper_config(), list_torrc_options(), option_get_canonical_name(), option_reset(), options_dup(), and options_init_from_string().

3.27.2.2 config_type_t config_var_t::type

How to interpret the type and turn it into a value.

Referenced by config_assign_line(), config_assign_value(), config_dump(), get_assigned_option(), getinfo_helper_config(), list_torrc_options(), option_clear(), and options_dup().

3.27.2.3 off_t config_var_t::var_offset

Offset of the corresponding member of **or_options_t** (p. 148).

Referenced by config_assign_line(), config_assign_value(), config_dump(), config_free(), get_assigned_option(), and option_clear().

3.27.2.4 const char* config_var_t::initvalue

String (or null) describing initial value.

Referenced by config_init(), option_reset(), and options_init_from_string().

The documentation for this struct was generated from the following file:

- **config.c**

3.28 connection_t Struct Reference

`#include <or.h>`

Data Fields

- uint32_t **magic**
- uint8_t **state**
- unsigned int **type**:4
- unsigned int **purpose**:5
- unsigned int **read_blocked_on_bw**:1
- unsigned int **write_blocked_on_bw**:1
- unsigned int **hold_open_until_flushed**:1
- unsigned int **inbuf_reached_eof**:1
- unsigned int **in_flushed_some**:1
- unsigned int **linked**:1
- unsigned int **reading_from_linked_conn**:1
- unsigned int **writing_to_linked_conn**:1
- unsigned int **active_on_link**:1
- unsigned int **linked_conn_is_closed**:1
- unsigned int **proxy_state**:4
- evutil_socket_t **s**
- int **conn_array_index**
- struct event * **read_event**
- struct event * **write_event**
- **buf_t** * **inbuf**
- **buf_t** * **outbuf**
- size_t **outbuf_flushlen**
- time_t **timestamp_lastread**
- time_t **timestamp_lastwritten**
- time_t **timestamp_created**
- int **socket_family**
- **tor_addr_t** **addr**
- uint16_t **port**
- uint16_t **marked_for_close**
- const char * **marked_for_close_file**
- char * **address**
- struct **connection_t** * **linked_conn**
- uint64_t **global_identifier**
- struct evdns_server_port * **dns_server_port**
- uint64_t **dirreq_id**

3.28.1 Detailed Description

Description of a connection to another host or process, and associated data.

A connection is named based on what it's connected to – an "OR connection" has a Tor node on the other end, an "exit connection" has a website or other server on the other end, and an "AP connection" has an application proxy (and thus a user) on the other end.

Every connection has a type and a state. Connections never change their type, but can go through many state changes in their lifetime.

Every connection has two associated input and output buffers. Listeners don't use them. For non-listener connections, incoming data is appended to conn->inbuf, and outgoing data is taken from conn->outbuf. Connections differ primarily in the functions called to fill and drain these buffers.

3.28.2 Field Documentation

3.28.2.1 uint32_t connection_t::magic

For memory debugging: must equal one of *_CONNECTION_MAGIC.

Referenced by _connection_free(), assert_connection_ok(), connection_init(), connection_or_get_for_extend(), HT_HEAD(), TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), and TO_OR_CONN().

3.28.2.2 uint8_t connection_t::state

Current state of this connection.

Referenced by _connection_free(), ap_stream_wants_exit_attention(), assert_connection_ok(), assign_onionskin_to_cpuworker(), circuit_get_open_circ_or_launch(), command_process_cell(), command_process_netinfo_cell(), command_process_var_cell(), command_process_versions_cell(), conn_close_if_marked(), connection_about_to_close_connection(), connection_ap_detach_retriable(), connection_ap_expire_beginning(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_bucket_read_limit(), connection_bucket_should_increase(), connection_bucket_write_limit(), connection_buckets_decrement(), connection_close_immediate(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_control_process_inbuf(), connection_cpu_process_inbuf(), connection_cpu_reached_eof(), connection_create_listener(), connection_dir_finished_connecting(), connection_dir_finished_flushing(), connection_dir_process_inbuf(), connection_dir_reached_eof(), connection_dirserv_flushed_some(), connection_dns_remove(), connection_edge_finished_connecting(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect(), connection_exit_connect_dir(), connection_flushed_some(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init_accepted_conn(), connection_or_connect(), connection_or_finished_connecting(), connection_or_finished_flushing(), connection_or_get_for_extend(), connection_or_group_set_badness(), connection_or_process_inbuf(), connection_or_set_state_open(), connection_read_to_buf(), connection_state_is_connecting(), connection_state_is_open(), connection_tls_continue_handshake(), connection_tls_finish_handshake(), connection_tls_start_handshake(), directory_handle_command_get(), directory_handle_command_post(), directory_initiate_command_rend(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve(), dnsserv_launch_request(), evdns_server_callback(), getinfo_helper_events(), handle_control_attachstream(), handle_control_authenticate(), is_valid_initial_command(), rend_client_desc_trynow(), rend_client_remove_intro_point(), run_connection_housekeeping(), and spawn_cpuworker().

3.28.2.3 unsigned int connection_t::type

What kind of connection is this?

Referenced by _connection_free(), _connection_mark_unattached_ap(), _connection_write_to_buf_impl(), ap_stream_wants_exit_attention(), assert_circuit_ok(), assert_connection_ok(), check_sockaddr_family_match(), circuit_dump_by_conn(), conn_close_if_marked(), conn_read_callback(), conn_write_callback(), connection_about_to_close_connection(), connection_add(), connection_ap_can_use_exit(), connection_ap_handshake_process_socks(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_bucket_read_limit(), connection_bucket_write_limit(), connection_buckets_decrement(), connection_close_immediate(), connection_counts_as_relayed_traffic(), connection_cpu_finished_flushing(), connection_cpu_process_inbuf(), connection_dir_finished_connecting(), connection_dir_finished_flushing(), connection_dir_process_inbuf(), connection_dns_remove(), connection_edge_destroy(), connection_edge_finished_connecting(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_edge_send_command(), connection_exit_connect_dir(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_free(), connection_handle_listener_read(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init(), connection_init_accepted_conn(), connection_is_listener(), connection_or_get_for_extend(), connection_or_nonopen_was_started_here(), connection_process_inbuf(), connection_reached_eof(), connection_read_to_buf(), connection_remove(), connection_state_is_connecting(), connection_state_is_open(), connection_unlink(), control_event_stream_status(), directory_handle_command(), directory_send_command(), dnsserv_close_listener(), dnsserv_configure_listener(), get_stream(), hibernate_begin(), hibernate_go_dormant(), retry_listeners(), and run_connection_housekeeping().

3.28.2.4 unsigned int connection_t::purpose

Only used for DIR and EXIT types currently.

Referenced by assert_connection_ok(), conn_close_if_marked(), connection_about_to_close_connection(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), connection_dir_download_routerdesc_failed(), connection_dir_request_failed(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect_dir(), connection_init_accepted_conn(), control_event_stream_status(), directory_initiate_command_rend(), dns_found_answer(), dns_resolve(), and run_connection_housekeeping().

3.28.2.5 unsigned int connection_t::read_blocked_on_bw

Boolean: should we start reading again once the bandwidth throttler allows it?

Referenced by connection_consider_empty_read_buckets().

3.28.2.6 unsigned int connection_t::write_blocked_on_bw

Boolean: should we start writing again once the bandwidth throttler allows writes?

Referenced by assert_connection_ok(), connection_consider_empty_write_buckets(), and connection_handle_write_impl().

3.28.2.7 unsigned int connection_t::hold_open_until_flushed

Despite this connection's being marked for close, do we flush it before closing it?

Referenced by _connection_mark_unattached_ap(), _connection_write_to_buf_impl(), assert_connection_ok(), conn_close_if_marked(), connection_about_to_close_connection(), connection_control_process_inbuf(), connection_edge_destroy(), connection_edge_process_relay_cell(), and run_connection_housekeeping().

3.28.2.8 unsigned int connection_t::inbuf_reached_eof

Boolean: did read() return 0 on this conn?

Referenced by connection_dir_process_inbuf(), connection_handle_read_impl(), and connection_read_to_buf().

3.28.2.9 unsigned int connection_t::in_flushed_some

Set to 1 when we're inside connection_flushed_some to keep us from calling connection_handle_write() recursively.

Referenced by _connection_write_to_buf_impl(), connection_flushed_some(), and connection_handle_write_impl().

3.28.2.10 unsigned int connection_t::linked

True if there is, or has been, a linked_conn.

Referenced by _connection_free(), assert_connection_ok(), connection_add(), connection_ap_make_link(), connection_close_immediate(), connection_is_rate_limited(), connection_link_connections(), connection_read_to_buf(), connection_should_read_from_linked_conn(), connection_start_reading(), connection_start_reading_from_linked_conn(), connection_start_writing(), connection_stop_reading(), connection_stop_reading_from_linked_conn(), and connection_stop_writing().

3.28.2.11 unsigned int connection_t::reading_from_linked_conn

True iff we'd like to be notified about read events from the linked conn.

Referenced by connection_free(), connection_is_reading(), connection_should_read_from_linked_conn(), connection_start_reading(), connection_stop_reading(), and connection_unlink().

3.28.2.12 unsigned int connection_t::writing_to_linked_conn

True iff we're willing to write to the linked conn.

Referenced by connection_is_writing(), connection_should_read_from_linked_conn(), connection_start_writing(), and connection_stop_writing().

3.28.2.13 unsigned int connection_t::active_on_link

True iff we're currently able to read on the linked conn, and our read_event should be made active with libevent.

Referenced by connection_handle_read_impl(), connection_start_reading_from_linked_conn(), and connection_stop_reading_from_linked_conn().

3.28.2.14 unsigned int connection_t::linked_conn_is_closed

True iff we've called **connection_close_immediate()** (p. 414) on this linked connection.

Referenced by connection_close_immediate().

3.28.2.15 unsigned int connection_t::proxy_state

CONNECT/SOCKS proxy client handshake state (for outgoing connections).

Referenced by connection_proxy_connect(), connection_read_proxy_handshake(), and connection_send_socks5_connect().

3.28.2.16 evutil_socket_t connection_t::s

Our socket; -1 if this connection is closed, or has no socket.

Referenced by _connection_free(), _connection_write_to_buf_impl(), assert_all_pending_dns_resolves_ok(), assert_connection_ok(), conn_close_if_marked(), conn_read_callback(), conn_write_callback(), connection_add(), connection_ap_get_original_destination(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_close_immediate(), connection_connect(), connection_create_listener(), connection_dns_remove(), connection_edge_end(), connection_edge_end_errno(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), connection_edge_reached_eof(), connection_handle_listener_read(), connection_handle_write_impl(), connection_init(), connection_link_connections(), connection_or_group_set_badness(), connection_or_process_cells_from_inbuf(), connection_read_to_buf(), connection_remove(), connection_start_reading(), connection_start_writing(), connection_stop_reading(), connection_stop_writing(), connection_tls_start_handshake(), connection_unregister_events(), dns_cancel_pending_resolve(), dns_resolve_impl(), dnsserv_configure_listener(), handle_control_authenticate(), purge_expired_resolves(), run_connection_housekeeping(), and spawn_cpuworker().

3.28.2.17 int connection_t::conn_array_index

Index into the global connection array.

Referenced by circuit_dump_by_conn(), close_closeable_connections(), connection_add(), connection_init(), connection_remove(), and connection_unlink().

3.28.2.18 struct event∗ connection_t::read_event [read]

Libevent event structure.

Referenced by _connection_free(), connection_add(), connection_is_reading(), connection_start_reading(), connection_stop_reading(), connection_unregister_events(), and set_streams_blocked_on_circ().

3.28.2.19 struct event∗ connection_t::write_event [read]

Libevent event structure.

Referenced by _connection_free(), connection_add(), connection_is_writing(), connection_start_writing(), connection_stop_writing(), and connection_unregister_events().

3.28.2.20 buf_t∗ connection_t::inbuf

Buffer holding data read over this connection.

Referenced by _connection_free(), assert_connection_ok(), conn_close_if_marked(), connection_ap_handshake_process_socks(), connection_ap_process_natd(), connection_control_process_inbuf(), connection_cpu_process_inbuf(), connection_dir_client_reached_eof(), connection_dir_process_inbuf(), connection_edge_package_raw_inbuf(), connection_edge_reached_eof(), connection_fetch_from_buf(), connection_fetch_var_cell_from_buf(), connection_handle_read_impl(), connection_init(), connection_or_process_cells_from_inbuf(), connection_read_https_proxy_response(), connection_read_proxy_handshake(), connection_read_to_buf(), directory_handle_command(), and run_connection_housekeeping().

3.28.2.21 buf_t∗ connection_t::outbuf

Buffer holding data to write over this connection.

Referenced by _connection_free(), _connection_write_to_buf_impl(), append_cell_to_circuit_queue(), assert_connection_ok(), conn_close_if_marked(), connection_close_immediate(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dirserv_flushed_some(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init(), connection_or_flushed_some(), connection_read_to_buf(), connection_should_read_from_linked_conn(), and run_connection_housekeeping().

3.28.2.22 size_t connection_t::outbuf_flushlen

How much data should we try to flush from the outbuf?

Referenced by _connection_write_to_buf_impl(), assert_connection_ok(), conn_close_if_marked(), connection_bucket_write_limit(), connection_close_immediate(), connection_edge_consider_sending_sendme(), connection_handle_write_impl(), connection_outbuf_too_full(), connection_read_to_buf(), and connection_wants_to_flush().

3.28.2.23 time_t connection_t::timestamp_lastread

When was the last time libevent said we could read?

Referenced by connection_ap_detach_retriable(), connection_ap_expire_beginning(), connection_edge_process_relay_cell_not_open(), connection_handle_read_impl(), connection_init(), link_apconn_to_circ(), rend_client_desc_trynow(), and run_connection_housekeeping().

3.28.2.24 time_t connection_t::timestamp_lastwritten

When was the last time libevent said we could write?

Referenced by _connection_mark_for_close(), assign_onionskin_to_cpuworker(), conn_close_if_marked(), connection_dir_client_reached_eof(), connection_handle_write_impl(), connection_init(), rend_client_desc_trynow(), and run_connection_housekeeping().

3.28.2.25 time_t connection_t::timestamp_created

When was this **connection_t** (p. 53) created?

Referenced by command_process_create_cell(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_cpu_process_inbuf(), connection_init(), connection_or_group_set_badness(), connection_or_is_better(), and rend_client_desc_trynow().

3.28.2.26 int connection_t::socket_family

Address family of this connection's socket. Usually AF_INET, but it can also be AF_UNIX, or in the future AF_INET6

Referenced by _connection_free(), check_sockaddr_family_match(), connection_create_listener(), connection_handle_listener_read(), connection_init(), and retry_listeners().

3.28.2.27 tor_addr_t connection_t::addr

IP of the other side of the connection; used to identify routers, along with port.

Referenced by circuit_dump_by_conn(), circuit_get_all_pending_on_or_conn(), circuit_n_conn_done(), connection_ap_expire_beginning(), connection_ap_get_original_destination(), connection_ap_handshake_attach_circuit(), connection_ap_make_link(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_begin_conn(), connection_exit_connect(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_is_rate_limited(), connection_or_init_conn_from_address(), connection_or_send_netinfo(), connection_proxy_connect(), connection_send_socks5_connect(), connection_tls_finish_handshake(), directory_conn_is_self_reachability_test(), directory_handle_command_get(), directory_initiate_command_rend(), dns_found_answer(), dns_resolve_impl(), onionskin_answer(), router_compare_to_my_exit_policy(), router_new_address_suggestion(), send_resolved_cell(), and write_http_response_header_impl().

3.28.2.28 uint16_t connection_t::port

If non-zero, port on the other end of the connection.

Referenced by circuit_build_failed(), circuit_dump_by_conn(), circuit_expire_building(), circuit_extend(), circuit_get_all_pending_on_or_conn(), circuit_n_conn_done(), command_process_create_cell(), command_process_destroy_cell(), command_process_netinfo_cell(), command_process_relay_cell(), command_process_versions_cell(), connection_ap_expire_beginning(), connection_ap_get_original_destination(), connection_ap_handshake_attach_circuit(), connection_ap_make_link(), connection_buckets_decrement(), connection_create_listener(), connection_dir_client_reached_eof(), connection_dir_finished_connecting(), connection_edge_finished_connecting(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_or_check_valid_tls_handshake(), connection_or_finished_connecting(), connection_or_group_set_badness(), connection_or_init_conn_from_address(), connection_proxy_connect(), connection_read_proxy_handshake(), connection_send_socks5_connect(), connection_tls_finish_handshake(), directory_conn_is_self_reachability_test(), directory_initiate_command_rend(), directory_send_command(), orconn_target_get_name(), rend_service_set_connection_addr_port(), retry_listeners(), router_compare_to_my_exit_policy(), and run_connection_housekeeping().

3.28.2.29 uint16_t connection_t::marked_for_close

Should we close this conn on the next iteration of the main loop? (If true, holds the line number where this connection was marked.)

Referenced by _circuit_mark_for_close(), _connection_mark_for_close(), _connection_mark_unattached_ap(), _connection_write_to_buf_impl(), add_connection_to_closeable_list(), ap_stream_wants_exit_attention(), assert_connection_ok(), circuit_resume_edge_reading_helper(), conn_close_if_marked(), conn_read_callback(), conn_write_callback(), connection_about_to_close_connection(), connection_ap_attach_pending(), connection_ap_expire_beginning(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_cpu_process_inbuf(), connection_edge_destroy(), connection_edge_end(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), connection_edge_reached_eof(), connection_edge_send_command(), connection_exit_begin_resolve(), connection_free(), connection_handle_read_impl(), connection_handle_write_impl(), connection_or_get_for_extend(), connection_or_group_set_badness(), connection_read_to_buf(), connection_state_is_connecting(), connection_state_is_open(), connection_unlink(), dns_cancel_pending_resolve(), dns_resolve(), get_stream(), relay_lookup_conn(), rend_client_desc_trynow(), retry_listeners(), and run_connection_housekeeping().

3.28.2.30 const char∗ connection_t::marked_for_close_file

For debugging: in which file were we marked for close?

Referenced by _connection_mark_for_close(), conn_close_if_marked(), connection_about_to_close_connection(), connection_edge_end(), connection_edge_package_raw_inbuf(), and connection_edge_send_command().

3.28.2.31 char∗ connection_t::address

FQDN (or IP) of the guy on the other end. strdup into this, because free_connection() frees it.

Referenced by _connection_free(), assert_connection_edge_not_dns_pending(), assert_connection_ok(), circuit_build_failed(), circuit_expire_building(), circuit_extend(), circuit_n_conn_done(), command_process_create_cell(), command_process_destroy_cell(), command_process_netinfo_cell(), command_process_relay_cell(), command_process_versions_cell(), conn_close_if_marked(), connection_add(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_create_listener(), connection_dir_client_reached_eof(), connection_dir_finished_connecting(), connection_dir_request_failed(), connection_dns_remove(), connection_edge_finished_connecting(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_finished_connecting(), connection_or_group_set_badness(), connection_or_init_conn_from_address(), connection_read_https_proxy_response(), connection_read_proxy_handshake(), connection_read_to_buf(), connection_tls_finish_handshake(), connection_tls_start_handshake(), directory_handle_command(), directory_handle_command_post(), directory_initiate_command_rend(), directory_send_command(), dns_resolve(), dns_resolve_impl(), http_set_address_origin(), launch_resolve(), orconn_target_get_name(), retry_listeners(), router_new_address_suggestion(), run_connection_housekeeping(), spawn_cpuworker(), and write_http_response_header_impl().

3.28.2.32 struct connection_t∗ connection_t::linked_conn `[read]`

Another connection that's connected to this one in lieu of a socket.

Referenced by assert_connection_ok(), conn_close_if_marked(), connection_free(), connection_handle_read_impl(), connection_link_connections(), connection_read_to_buf(), connection_should_read_from_linked_conn(), connection_start_writing(), connection_stop_writing(), and connection_unlink().

3.28.2.33 uint64_t connection_t::global_identifier

Unique identifier for this connection on this Tor instance.

Referenced by assign_onionskin_to_cpuworker(), connection_init(), control_event_stream_bandwidth(), control_event_stream_bandwidth_used(), control_event_stream_status(), and getinfo_helper_events().

3.28.2.34 struct evdns_server_port∗ connection_t::dns_server_port `[read]`

If the connection is a CONN_TYPE_AP_DNS_LISTENER, this field points to the evdns_server_port is uses to listen to and answer connections.

Referenced by connection_unregister_events(), dnsserv_close_listener(), and dnsserv_configure_listener().

3.28.2.35 uint64_t connection_t::dirreq_id

Unique ID for measuring tunneled network status requests.

Referenced by connection_handle_write_impl().

The documentation for this struct was generated from the following file:

- **or.h**

3.29 consensus_waiting_for_certs_t Struct Reference

Data Fields

- **networkstatus_t** ∗ **consensus**
- char ∗ **body**
- time_t **set_at**
- int **dl_failed**

3.29.1 Detailed Description

A v3 consensus networkstatus that we've received, but which we don't have enough certificates to be happy about.

3.29.2 Field Documentation

3.29.2.1 networkstatus_t∗ consensus_waiting_for_certs_t::consensus

The consensus itself.

Referenced by consensus_is_waiting_for_certs(), networkstatus_free_all(), networkstatus_note_certs_arrived(), networkstatus_set_current_consensus(), and update_consensus_networkstatus_downloads().

3.29.2.2 char∗ consensus_waiting_for_certs_t::body

The encoded version of the consensus, nul-terminated.

Referenced by networkstatus_free_all(), networkstatus_note_certs_arrived(), and networkstatus_set_current_consensus().

3.29.2.3 time_t consensus_waiting_for_certs_t::set_at

When did we set the current value of consensus_waiting_for_certs? If this is too recent, we shouldn't try to fetch a new consensus for a little while, to give ourselves time to get certificates for this one.

Referenced by networkstatus_set_current_consensus(), and update_consensus_networkstatus_downloads().

3.29.2.4 int consensus_waiting_for_certs_t::dl_failed

Set to 1 if we've been holding on to it for so long we should maybe treat it as being bad.

Referenced by networkstatus_set_current_consensus(), and update_consensus_networkstatus_downloads().

The documentation for this struct was generated from the following file:

- **networkstatus.c**

3.30 control_connection_t Struct Reference

`#include <or.h>`

Data Fields

- **connection_t _base**
- uint32_t **event_mask**
- unsigned int **have_sent_protocolinfo**:1
- uint32_t **incoming_cmd_len**
- uint32_t **incoming_cmd_cur_len**
- char * **incoming_cmd**

3.30.1 Detailed Description

Subtype of **connection_t** (p. 53) for an connection to a controller.

3.30.2 Field Documentation

3.30.2.1 uint32_t control_connection_t::event_mask

Bitfield: which events does this controller care about?

Referenced by connection_free(), handle_control_resolve(), handle_control_setevents(), and send_control_event_string().

3.30.2.2 unsigned int control_connection_t::have_sent_protocolinfo

True if we have sent a protocolinfo reply on this connection.

Referenced by handle_control_protocolinfo(), and is_valid_initial_command().

3.30.2.3 uint32_t control_connection_t::incoming_cmd_len

Amount of space allocated in incoming_cmd.

Referenced by connection_control_process_inbuf().

3.30.2.4 uint32_t control_connection_t::incoming_cmd_cur_len

Number of bytes currently stored in incoming_cmd.

Referenced by connection_control_process_inbuf().

3.30.2.5 char* control_connection_t::incoming_cmd

A control command that we're reading from the inbuf, but which has not yet arrived completely.

Referenced by _connection_free(), and connection_control_process_inbuf().

The documentation for this struct was generated from the following file:

- **or.h**

3.31 cpath_build_state_t Struct Reference

`#include <or.h>`

Data Fields

- int **desired_path_len**
- **extend_info_t** ∗ **chosen_exit**
- int **need_uptime**
- int **need_capacity**
- int **is_internal**
- int **onehop_tunnel**
- **crypt_path_t** ∗ **pending_final_cpath**
- int **failure_count**
- time_t **expiry_time**

3.31.1 Detailed Description

Information used to build a circuit.

3.31.2 Field Documentation

3.31.2.1 int cpath_build_state_t::desired_path_len

Intended length of the final circuit.

Referenced by circuit_append_new_exit(), circuit_list_path_impl(), onion_extend_cpath(), and onion_pick_cpath_exit().

3.31.2.2 extend_info_t∗ cpath_build_state_t::chosen_exit

How to extend to the planned exit node.

Referenced by _circuit_mark_for_close(), build_state_get_exit_nickname(), build_state_get_exit_router(), circuit_append_new_exit(), circuit_free(), circuit_is_acceptable(), connection_ap_fail_onehop(), connection_ap_process_end_not_open(), find_intro_circuit(), handle_control_attachstream(), onion_extend_cpath(), onion_pick_cpath_exit(), rend_client_introduction_acked(), rend_client_send_introduction(), rend_service_launch_establish_intro(), and rend_service_relaunch_rendezvous().

3.31.2.3 int cpath_build_state_t::need_uptime

Whether every node in the circ must have adequate uptime.

Referenced by choose_good_middle_server(), choose_random_entry(), circuit_find_to_cannibalize(), circuit_is_acceptable(), circuit_list_path_impl(), circuit_predict_and_launch_new(), circuit_stream_is_being_handled(), onion_pick_cpath_exit(), and origin_circuit_init().

3.31.2.4 int cpath_build_state_t::need_capacity

Whether every node in the circ must have adequate capacity.

Referenced by choose_good_middle_server(), choose_random_entry(), circuit_find_to_cannibalize(), onion_pick_cpath_exit(), and origin_circuit_init().

3.31.2.5 int cpath_build_state_t::is_internal

Whether the last hop was picked with exiting in mind.

Referenced by circuit_find_to_cannibalize(), circuit_is_acceptable(), circuit_list_path_impl(), circuit_predict_and_launch_new(), circuit_stream_is_being_handled(), onion_pick_cpath_exit(), origin_circuit_init(), and warn_if_last_router_excluded().

3.31.2.6 int cpath_build_state_t::onehop_tunnel

Did we pick this as a one-hop tunnel (not safe for other conns)? These are for encrypted connections that exit to this router, not for arbitrary exits from the circuit.

Referenced by circuit_expire_building(), circuit_handle_first_hop(), circuit_is_acceptable(), circuit_predict_and_launch_new(), circuit_send_next_onion_skin(), circuit_stream_is_being_handled(), connection_ap_handshake_send_begin(), onion_pick_cpath_exit(), origin_circuit_init(), and warn_if_last_router_excluded().

3.31.2.7 crypt_path_t* cpath_build_state_t::pending_final_cpath

The **crypt_path_t** (p. 67) to append after rendezvous: used for rendezvous.

Referenced by circuit_free(), rend_client_receive_rendezvous(), rend_service_introduce(), rend_service_relaunch_rendezvous(), and rend_service_rendezvous_has_opened().

3.31.2.8 int cpath_build_state_t::failure_count

How many times has building a circuit for this task failed?

Referenced by rend_service_relaunch_rendezvous().

3.31.2.9 time_t cpath_build_state_t::expiry_time

At what time should we give up on this task?

Referenced by rend_service_introduce(), and rend_service_relaunch_rendezvous().

The documentation for this struct was generated from the following file:

- **or.h**

3.32 crypt_path_t Struct Reference

`#include <or.h>`

Data Fields

- uint32_t **magic**
- **crypto_cipher_env_t** ∗ **f_crypto**
- **crypto_cipher_env_t** ∗ **b_crypto**
- **crypto_digest_env_t** ∗ **f_digest**
- **crypto_digest_env_t** ∗ **b_digest**
- **crypto_dh_env_t** ∗ **dh_handshake_state**
- char **fast_handshake_state** [DIGEST_LEN]
- char **handshake_digest** [DIGEST_LEN]
- **extend_info_t** ∗ **extend_info**
- uint8_t **state**
- struct **crypt_path_t** ∗ **next**
- struct **crypt_path_t** ∗ **prev**
- int **package_window**
- int **deliver_window**

3.32.1 Detailed Description

Holds accounting information for a single step in the layered encryption performed by a circuit. Used only at the client edge of a circuit.

3.32.2 Field Documentation

3.32.2.1 crypto_cipher_env_t∗ crypt_path_t::f_crypto

Encryption key and counter for cells heading towards the OR at this step.

Referenced by assert_cpath_layer_ok(), circuit_free_cpath_node(), circuit_init_cpath_crypto(), circuit_package_relay_cell(), and onionskin_answer().

3.32.2.2 crypto_cipher_env_t∗ crypt_path_t::b_crypto

Encryption key and counter for cells heading back from the OR at this step.

Referenced by assert_cpath_layer_ok(), circuit_free_cpath_node(), circuit_init_cpath_crypto(), onionskin_answer(), and relay_crypt().

3.32.2.3 crypto_digest_env_t∗ crypt_path_t::f_digest

Digest state for cells heading towards the OR at this step.

Referenced by circuit_free_cpath_node(), circuit_init_cpath_crypto(), circuit_package_relay_cell(), and onionskin_answer().

3.32.2.4 crypto_digest_env_t* crypt_path_t::b_digest

Digest state for cells heading away from the OR at this step.

Referenced by circuit_free_cpath_node(), circuit_init_cpath_crypto(), onionskin_answer(), and relay_crypt().

3.32.2.5 crypto_dh_env_t* crypt_path_t::dh_handshake_state

Current state of Diffie-Hellman key negotiation with the OR at this step.

Referenced by assert_cpath_layer_ok(), circuit_finish_handshake(), circuit_free_cpath_node(), circuit_send_next_onion_skin(), rend_client_receive_rendezvous(), rend_service_introduce(), and rend_service_rendezvous_has_opened().

3.32.2.6 char crypt_path_t::fast_handshake_state[DIGEST_LEN]

Current state of 'fast' (non-PK) key negotiation with the OR at this step. Used to save CPU when TLS is already providing all the authentication, secrecy, and integrity we need, and we're already distinguishable from an OR.

Referenced by circuit_finish_handshake(), and circuit_send_next_onion_skin().

3.32.2.7 char crypt_path_t::handshake_digest[DIGEST_LEN]

Negotiated key material shared with the OR at this step.

Referenced by circuit_finish_handshake(), rend_service_intro_has_opened(), rend_service_introduce(), and rend_service_rendezvous_has_opened().

3.32.2.8 extend_info_t* crypt_path_t::extend_info

Information to extend to the OR at this step.

Referenced by circuit_build_failed(), circuit_conforms_to_options(), circuit_find_to_cannibalize(), circuit_free_cpath_node(), circuit_handle_first_hop(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), connection_ap_expire_beginning(), onion_append_hop(), and should_use_create_fast_for_circuit().

3.32.2.9 uint8_t crypt_path_t::state

Is the circuit built to this step? Must be one of:

- CPATH_STATE_CLOSED (The circuit has not been extended to this step)
- CPATH_STATE_AWAITING_KEYS (We have sent an EXTEND/CREATE to this step and not received an EXTENDED/CREATED)
- CPATH_STATE_OPEN (The circuit has been extended to this step)

Referenced by assert_cpath_layer_ok(), assert_cpath_ok(), circuit_build_failed(), circuit_expire_building(), circuit_finish_handshake(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), link_apconn_to_circ(), onion_append_hop(), onion_next_hop_in_cpath(), relay_crypt(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

3.32.2.10 struct crypt_path_t* crypt_path_t::next [read]

Link to next **crypt_path_t** (p. 67) in the circuit. (The list is circular, so the last node links to the first.)

Referenced by assert_cpath_ok(), choose_good_middle_server(), circuit_conforms_to_options(), circuit_find_to_cannibalize(), circuit_free_cpath(), circuit_get_cpath_hop(), circuit_get_cpath_len(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_truncated(), cpath_is_on_circuit(), onion_append_to_cpath(), onion_next_hop_in_cpath(), and relay_crypt().

3.32.2.11 struct crypt_path_t* crypt_path_t::prev [read]

Link to previous **crypt_path_t** (p. 67) in the circuit.

Referenced by assert_cpath_ok(), circuit_build_failed(), circuit_package_relay_cell(), circuit_send_next_onion_skin(), connection_exit_begin_conn(), link_apconn_to_circ(), onion_append_to_cpath(), rend_client_send_establish_rendezvous(), rend_process_relay_cell(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), and router_perform_bandwidth_test().

3.32.2.12 int crypt_path_t::package_window

How many cells are we allowed to originate ending at this step?

Referenced by assert_cpath_layer_ok(), circuit_consider_stop_edge_reading(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), onion_append_hop(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

3.32.2.13 int crypt_path_t::deliver_window

How many cells are we willing to deliver originating at this step?

Referenced by assert_cpath_layer_ok(), circuit_consider_sending_sendme(), connection_edge_process_relay_cell(), onion_append_hop(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

The documentation for this struct was generated from the following file:

- **or.h**

3.33 crypto_cipher_env_t Struct Reference

Data Fields

- char **key** [CIPHER_KEY_LEN]
- **aes_cnt_cipher_t ∗ cipher**

3.33.1 Detailed Description

Key and stream information for a stream cipher.

The documentation for this struct was generated from the following file:

- **crypto.c**

3.34 crypto_dh_env_t Struct Reference

Data Fields

- DH * **dh**

3.34.1 Detailed Description

A structure to hold the first half (x, g^x) of a Diffie-Hellman handshake while we're waiting for the second.

The documentation for this struct was generated from the following file:

- **crypto.c**

3.35 crypto_digest_env_t Struct Reference

Data Fields

- union {
  SHA_CTX **sha1**
  SHA256_CTX **sha2**
  } **d**

- digest_algorithm_t **algorithm**: 8

3.35.1 Detailed Description

Intermediate information about the digest of a stream of data.

The documentation for this struct was generated from the following file:

- **crypto.c**

3.36 CRYPTO_dynlock_value Struct Reference

Data Fields

- **tor_mutex_t** ∗ **lock**

3.36.1 Detailed Description

OpenSSL helper type: wraps a Tor mutex so that OpenSSL can use it as a lock.

The documentation for this struct was generated from the following file:

- **crypto.c**

3.37 crypto_pk_env_t Struct Reference

Data Fields

- int **refs**
- RSA * **key**

3.37.1 Detailed Description

A public key, or a public/private key-pair.

The documentation for this struct was generated from the following file:

- **crypto.c**

3.38 desc_store_t Struct Reference

`#include <or.h>`

Data Fields

- const char * **fname_base**
- const char * **fname_alt_base**
- const char * **description**
- **tor_mmap_t** * **mmap**
- **store_type_t** **type**
- size_t **journal_len**
- size_t **store_len**
- size_t **bytes_dropped**

3.38.1 Detailed Description

A 'store' is a set of descriptors saved on disk, with accompanying journal, mmaped as needed, rebuilt as needed.

3.38.2 Field Documentation

3.38.2.1 const char* desc_store_t::fname_base

Filename (within DataDir) for the store. We append .tmp to this filename for a temporary file when rebuilding the store, and .new to this filename for the journal.

Referenced by router_get_routerlist(), router_rebuild_store(), router_reload_router_list_impl(), and signed_desc_append_to_journal().

3.38.2.2 const char* desc_store_t::fname_alt_base

Alternative (obsolete) value for fname_base: if the file named by fname_base isn't present, we read from here instead, but we never write here.

Referenced by router_get_routerlist(), and router_reload_router_list_impl().

3.38.2.3 const char* desc_store_t::description

Human-readable description of what this store contains.

Referenced by router_get_routerlist(), and router_rebuild_store().

3.38.2.4 tor_mmap_t* desc_store_t::mmap

A mmap for the main file in the store.

Referenced by router_rebuild_store(), router_reload_router_list_impl(), routerlist_free(), and signed_descriptor_get_body_impl().

3.38.2.5 store_type_t desc_store_t::type

What's stored in this store?

Referenced by router_get_routerlist(), router_rebuild_store(), and router_reload_router_list_impl().

3.38.2.6 size_t desc_store_t::journal_len

The size of the router log, in bytes.

Referenced by router_reload_router_list_impl(), router_should_rebuild_store(), and signed_desc_append_to_journal().

3.38.2.7 size_t desc_store_t::store_len

The size of the router store, in bytes.

Referenced by router_reload_router_list_impl(), and router_should_rebuild_store().

3.38.2.8 size_t desc_store_t::bytes_dropped

Total bytes dropped since last rebuild: this is space currently used in the cache and the journal that could be freed by a rebuild.

Referenced by extrainfo_insert(), router_should_rebuild_store(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

The documentation for this struct was generated from the following file:

- **or.h**

3.39 digests_t Struct Reference

`#include <crypto.h>`

Data Fields

- char **d** [N_DIGEST_ALGORITHMS][DIGEST256_LEN]

3.39.1 Detailed Description

A set of all the digests we know how to compute, taken on a single string. Any digests that are shorter than 256 bits are right-padded with 0 bits.

Note that this representation wastes 12 bytes for the SHA1 case, so don't use it for anything where we need to allocate a whole bunch at once.

The documentation for this struct was generated from the following file:

- **crypto.h**

3.40 digestset_t Struct Reference

`#include <container.h>`

Data Fields

- int **mask**
- bitarray_t * **ba**

3.40.1 Detailed Description

A set of digests, implemented as a Bloom filter.

The documentation for this struct was generated from the following file:

- container.h

3.41 dir_connection_t Struct Reference

`#include <or.h>`

Public Types

- enum {

  DIR_SPOOL_NONE = 0, **DIR_SPOOL_SERVER_BY_DIGEST**, **DIR_SPOOL_SERVER_BY_FP**, **DIR_SPOOL_EXTRA_BY_DIGEST**,

  DIR_SPOOL_EXTRA_BY_FP, **DIR_SPOOL_CACHED_DIR**, **DIR_SPOOL_NETWORKSTATUS**, **DIR_SPOOL_MICRODESC** }

Data Fields

- **connection_t _base**
- char ∗ **requested_resource**
- unsigned int **dirconn_direct**:1
- enum dir_connection_t:: { ... } **dir_spool_src**
- uint8_t **router_purpose**
- **smartlist_t** ∗ **fingerprint_stack**
- struct **cached_dir_t** ∗ **cached_dir**
- off_t **cached_dir_offset**
- **tor_zlib_state_t** ∗ **zlib_state**
- **rend_data_t** ∗ **rend_data**
- char **identity_digest** [DIGEST_LEN]

3.41.1 Detailed Description

Subtype of **connection_t** (p. 53) for an "directory connection" – that is, an HTTP connection to retrieve or serve directory material.

3.41.2 Member Enumeration Documentation

3.41.2.1 anonymous enum

What exactly are we spooling right now?

3.41.3 Field Documentation

3.41.3.1 char∗ dir_connection_t::requested_resource

Which 'resource' did we ask the directory for?

Referenced by _connection_free(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), directory_conn_is_self_reachability_test(), and directory_send_command().

3.41.3.2 unsigned int dir_connection_t::dirconn_direct

Is this dirconn direct, or via Tor?

Referenced by connection_dir_client_reached_eof(), and directory_initiate_command_rend().

3.41.3.3 enum { ... } dir_connection_t::dir_spool_src

What exactly are we spooling right now?

Referenced by connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dirserv_finish_spooling(), connection_dirserv_flushed_some(), and directory_handle_command_get().

3.41.3.4 uint8_t dir_connection_t::router_purpose

If we're fetching descriptors, what router purpose shall we assign to them?

Referenced by connection_dir_client_reached_eof(), connection_dir_download_networkstatus_failed(), and directory_initiate_command_rend().

3.41.3.5 smartlist_t∗ dir_connection_t::fingerprint_stack

List of fingerprints for networkstatuses or descriptors to be spooled.

Referenced by _connection_free(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), and directory_handle_command_get().

3.41.3.6 struct cached_dir_t∗ dir_connection_t::cached_dir `[read]`

A **cached_dir_t** (p. 23) object that we're currently spooling out

Referenced by _connection_free(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), and directory_handle_command_get().

3.41.3.7 off_t dir_connection_t::cached_dir_offset

The current offset into cached_dir.

Referenced by connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), and directory_handle_command_get().

3.41.3.8 tor_zlib_state_t∗ dir_connection_t::zlib_state

The zlib object doing on-the-fly compression for spooled data.

Referenced by _connection_free(), _connection_write_to_buf_impl(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dirserv_finish_spooling(), and directory_handle_command_get().

3.41.3.9 rend_data_t* dir_connection_t::rend_data

What rendezvous service are we querying for?

Referenced by _connection_free(), connection_about_to_close_connection(), connection_dir_client_reached_eof(), and directory_initiate_command_rend().

3.41.3.10 char dir_connection_t::identity_digest[DIGEST_LEN]

Hash of the public RSA key for the directory server's signing key.

Referenced by connection_dir_client_reached_eof(), connection_dir_request_failed(), directory_conn_is_self_reachability_test(), and directory_initiate_command_rend().

The documentation for this struct was generated from the following file:

- **or.h**

3.42 dir_src_ent_t Struct Reference

Data Fields

- **networkstatus_t** ∗ **v**
- const char ∗ **digest**
- int **is_legacy**

3.42.1 Detailed Description

Temporary structure used in constructing a list of dir-source entries for a consensus. One of these is generated for every vote, and one more for every legacy key in each vote.

The documentation for this struct was generated from the following file:

- **dirvote.c**

3.43 directory_token_t Struct Reference

Data Fields

- **directory_keyword tp**
- int **n_args**:30
- char ** **args**
- char * **object_type**
- size_t **object_size**
- char * **object_body**
- **crypto_pk_env_t** * **key**
- char * **error**

3.43.1 Detailed Description

Structure to hold a single directory token.

We parse a directory by breaking it into "tokens", each consisting of a keyword, a line full of arguments, and a binary object. The arguments and object are both optional, depending on the keyword type.

This structure is only allocated in memareas; do not allocate it on the heap, or **token_clear()** (p. 1201) won't work.

3.43.2 Field Documentation

3.43.2.1 directory_keyword directory_token_t::tp

Type of the token.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), find_dir_signing_key(), get_next_token(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), router_add_exit_policy(), router_parse_addr_policy(), router_parse_addr_policy_item_from_string(), router_parse_addr_policy_private(), router_parse_directory(), router_parse_runningrouters(), and tokenize_string().

3.43.2.2 int directory_token_t::n_args

Number of elements in args

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), get_next_token(), get_token_arguments(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), rend_parse_client_keys(), rend_parse_v2_service_descriptor(), router_parse_addr_policy(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

3.43.2.3 char** directory_token_t::args

Array of arguments from keyword line.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), get_next_token(), get_token_arguments(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), router_parse_addr_policy(), router_parse_addr_policy_private(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

3.43.2.4 char∗ directory_token_t::object_type

—–BEGIN [object_type]—–

Referenced by check_signature_token(), extrainfo_parse_entry_from_string(), get_next_token(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), and rend_parse_v2_service_descriptor().

3.43.2.5 size_t directory_token_t::object_size

Bytes in object_body

Referenced by check_signature_token(), extrainfo_parse_entry_from_string(), get_next_token(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), and rend_parse_v2_service_descriptor().

3.43.2.6 char∗ directory_token_t::object_body

Contents of object, base64-decoded.

Referenced by check_signature_token(), extrainfo_parse_entry_from_string(), get_next_token(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), rend_parse_v2_service_descriptor(), and token_check_object().

3.43.2.7 crypto_pk_env_t∗ directory_token_t::key

For public keys only. Heap-allocated.

Referenced by authority_cert_parse_from_string(), find_dir_signing_key(), get_next_token(), microdescs_parse_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), router_parse_entry_from_string(), token_check_object(), and token_clear().

3.43.2.8 char∗ directory_token_t::error

For _ERR tokens only.

Referenced by router_parse_addr_policy_item_from_string(), and tokenize_string().

The documentation for this struct was generated from the following file:

- **routerparse.c**

3.44 dirreq_map_entry_t Struct Reference

3.44.1 Detailed Description

Entry in a map from either conn->global_identifier for direct requests or a unique circuit identifier for tunneled requests to request time, response size, and completion time of a network status request. Used to measure download times of requests to derive average client bandwidths.

The documentation for this struct was generated from the following file:

- **geoip.c**

3.45 document_signature_t Struct Reference

`#include <or.h>`

Data Fields

- char **identity_digest** [DIGEST_LEN]
- char **signing_key_digest** [DIGEST_LEN]
- digest_algorithm_t **alg**
- char * **signature**
- int **signature_len**
- unsigned int **bad_signature**: 1
- unsigned int **good_signature**: 1

3.45.1 Detailed Description

A signature of some document by an authority.

3.45.2 Field Documentation

3.45.2.1 char document_signature_t::identity_digest[DIGEST_LEN]

Declared SHA-1 digest of this voter's identity key

Referenced by networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

3.45.2.2 char document_signature_t::signing_key_digest[DIGEST_LEN]

Declared SHA-1 digest of signing key used by this voter.

Referenced by networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

3.45.2.3 digest_algorithm_t document_signature_t::alg

Algorithm used to compute the digest of the document.

Referenced by networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

3.45.2.4 char* document_signature_t::signature

Signature of the signed thing.

Referenced by document_signature_dup(), document_signature_free(), networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

3.45.2.5 int document_signature_t::signature_len

Length of **signature**

Referenced by document_signature_dup(), networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

3.45.2.6 unsigned int document_signature_t::bad_signature

Set to true if we've tried to verify the sig, and we know it's bad.

Referenced by networkstatus_add_detached_signatures(), and networkstatus_check_document_signature().

3.45.2.7 unsigned int document_signature_t::good_signature

Set to true if we've verified the sig as good.

Referenced by networkstatus_add_detached_signatures(), networkstatus_check_document_signature(), and networkstatus_parse_vote_from_string().

The documentation for this struct was generated from the following file:

- **or.h**

3.46 download_status_t Struct Reference

`#include <or.h>`

Data Fields

- time_t **next_attempt_at**
- uint8_t **n_download_failures**
- **download_schedule_t schedule**: 8

3.46.1 Detailed Description

Information about our plans for retrying downloads for a downloadable object.

3.46.2 Field Documentation

3.46.2.1 time_t download_status_t::next_attempt_at

When should we try downloading this descriptor again?

Referenced by download_status_increment_failure(), download_status_is_ready(), and download_status_reset().

3.46.2.2 uint8_t download_status_t::n_download_failures

Number of failures trying to download the most recent descriptor.

Referenced by dir_routerdesc_download_failed(), download_status_increment_failure(), download_status_is_ready(), download_status_mark_impossible(), and download_status_reset().

The documentation for this struct was generated from the following file:

- **or.h**

3.47 duration_idx_t Struct Reference

Data Fields

- int **duration**
- int **idx**
- int **old**

3.47.1 Detailed Description

Internal type used to represent how long an old descriptor was valid, where it appeared in the list of old descriptors, and whether it's extra old. Used only by **routerlist_remove_old_cached_routers_with_id()** (p. 1167).

The documentation for this struct was generated from the following file:

- **routerlist.c**

3.48 edge_connection_t Struct Reference

`#include <or.h>`

Data Fields

- **connection_t _base**
- struct **edge_connection_t** ∗ **next_stream**
- struct **crypt_path_t** ∗ **cpath_layer**
- int **package_window**
- int **deliver_window**
- char ∗ **chosen_exit_name**
- **socks_request_t** ∗ **socks_request**
- struct **circuit_t** ∗ **on_circuit**
- uint32_t **address_ttl**
- **streamid_t stream_id**
- uint16_t **end_reason**
- uint32_t **n_read**
- uint32_t **n_written**
- **rend_data_t** ∗ **rend_data**
- uint8_t **num_socks_retries**
- unsigned int **is_dns_request**:1
- unsigned int **want_onehop**:1
- unsigned int **use_begindir**:1
- unsigned int **edge_has_sent_end**:1
- unsigned int **edge_blocked_on_circ**:1
- unsigned int **chosen_exit_optional**:1
- unsigned int **chosen_exit_retries**:3
- struct evdns_server_request ∗ **dns_server_request**

3.48.1 Detailed Description

Subtype of **connection_t** (p. 53) for an "edge connection" – that is, a socks (ap) connection, or an exit.

3.48.2 Field Documentation

3.48.2.1 struct edge_connection_t∗ edge_connection_t::next_stream `[read]`

Points to the next stream at this edge, if any

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_consider_stop_edge_reading(), circuit_detach_stream(), circuit_dump_by_conn(), circuit_free_all(), circuit_resume_edge_reading_helper(), circuit_truncated(), connection_exit_begin_conn(), connection_exit_connect_dir(), dns_found_answer(), dns_resolve(), get_unique_stream_id_by_circ(), link_apconn_to_circ(), relay_lookup_conn(), and set_streams_blocked_on_circ().

3.48.2.2 struct crypt_path_t* edge_connection_t::cpath_layer [read]

A pointer to which node in the circ this conn exits at.

Referenced by assert_connection_ok(), circuit_consider_stop_edge_reading(), circuit_detach_stream(), circuit_resume_edge_reading_helper(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_edge_consider_sending_sendme(), connection_edge_destroy(), connection_edge_package_raw_inbuf(), connection_edge_send_command(), connection_exit_begin_conn(), link_apconn_to_circ(), and relay_lookup_conn().

3.48.2.3 int edge_connection_t::package_window

How many more relay cells can I send into the circuit?

Referenced by circuit_resume_edge_reading_helper(), connection_ap_handshake_send_begin(), connection_edge_finished_connecting(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), and connection_exit_begin_conn().

3.48.2.4 int edge_connection_t::deliver_window

How many more relay cells can end at me?

Referenced by connection_ap_handshake_send_begin(), connection_edge_consider_sending_sendme(), connection_edge_process_relay_cell(), and connection_exit_begin_conn().

3.48.2.5 char* edge_connection_t::chosen_exit_name

Nickname of planned exit node – used with .exit support.

Referenced by _connection_free(), assert_connection_ok(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), connection_ap_can_use_exit(), connection_ap_fail_onehop(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_socks_resolved(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell_not_open(), handle_control_attachstream(), and write_stream_target_to_buf().

3.48.2.6 socks_request_t* edge_connection_t::socks_request

SOCKS structure describing request (AP only.)

Referenced by _connection_free(), _connection_mark_unattached_ap(), assert_connection_ok(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_stream_is_being_handled(), connection_about_to_close_connection(), connection_ap_can_use_exit(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_reached_eof(), connection_handle_read_impl(), consider_recording_trackhost(), control_event_stream_status(), directory_all_unreachable(), dnsserv_launch_request(), dnsserv_resolved(), edge_connection_new(), evdns_server_callback(), getinfo_helper_events(), handle_control_redirectstream(), remap_event_helper(), tell_controller_about_resolved_result(), and write_stream_target_to_buf().

3.48.2.7 struct circuit_t* edge_connection_t::on_circuit `[read]`

The circuit (if any) that this edge connection is using.

Referenced by _circuit_mark_for_close(), circuit_detach_stream(), circuit_get_by_edge_conn(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_destroy(), connection_edge_send_command(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect_dir(), dns_found_answer(), dns_resolve(), and link_apconn_to_circ().

3.48.2.8 uint32_t edge_connection_t::address_ttl

TTL for address-to-addr mapping on exit connection. Exit connections only.

Referenced by connection_edge_end(), connection_edge_finished_connecting(), connection_exit_connect(), dns_found_answer(), dns_resolve_impl(), send_resolved_cell(), and send_resolved_hostname_cell().

3.48.2.9 streamid_t edge_connection_t::stream_id

The stream ID used for this edge connection on its circuit

Referenced by connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_process_relay_cell(), connection_edge_send_command(), connection_exit_begin_conn(), connection_exit_begin_resolve(), get_unique_stream_id_by_circ(), and relay_lookup_conn().

3.48.2.10 uint16_t edge_connection_t::end_reason

The reason why this connection is closing; passed to the controller.

Referenced by _circuit_mark_for_close(), _connection_mark_unattached_ap(), conn_write_callback(), connection_about_to_close_connection(), connection_ap_expire_beginning(), connection_edge_destroy(), connection_edge_end(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_send_command(), and handle_control_attachstream().

3.48.2.11 uint32_t edge_connection_t::n_read

Bytes read since last call to **control_event_stream_bandwidth_used()** (p. 497)

Referenced by connection_read_to_buf(), control_event_stream_bandwidth(), and control_event_stream_bandwidth_used().

3.48.2.12 uint32_t edge_connection_t::n_written

Bytes written since last call to **control_event_stream_bandwidth_used()** (p. 497)

Referenced by connection_handle_write_impl(), control_event_stream_bandwidth(), and control_event_stream_bandwidth_used().

3.48.2.13 rend_data_t* edge_connection_t::rend_data

What rendezvous service are we querying for? (AP only)

Referenced by _connection_free(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), connection_ap_handshake_attach_circuit(), connection_edge_is_rendezvous_stream(), connection_exit_begin_conn(), and rend_client_desc_trynow().

3.48.2.14 uint8_t edge_connection_t::num_socks_retries

Number of times we've reassigned this application connection to a new circuit. We keep track because the timeout is longer if we've already retried several times.

Referenced by compute_retry_timeout(), and connection_ap_expire_beginning().

3.48.2.15 unsigned int edge_connection_t::is_dns_request

True iff this connection is for a DNS request only.

Referenced by connection_ap_handshake_socks_resolved(), control_event_stream_status(), dnsserv_launch_request(), and evdns_server_callback().

3.48.2.16 unsigned int edge_connection_t::want_onehop

True iff this stream must attach to a one-hop circuit (e.g. for begin_dir).

Referenced by circuit_get_open_circ_or_launch(), circuit_is_acceptable(), connection_ap_fail_onehop(), connection_ap_handshake_attach_circuit(), and connection_ap_make_link().

3.48.2.17 unsigned int edge_connection_t::use_begindir

True iff this stream should use a BEGIN_DIR relay command to establish itself rather than BEGIN (either via onehop or via a whole circuit).

Referenced by circuit_get_open_circ_or_launch(), connection_ap_can_use_exit(), connection_ap_detach_retriable(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_send_begin(), connection_ap_make_link(), and control_event_stream_status().

3.48.2.18 unsigned int edge_connection_t::edge_has_sent_end

For debugging; only used on edge connections. Set once we've set the stream end, and check in **connection_about_to_close_connection()** (p. 412).

Referenced by _circuit_mark_for_close(), _connection_mark_unattached_ap(), conn_write_callback(), connection_about_to_close_connection(), connection_ap_expire_beginning(), connection_edge_destroy(), connection_edge_end(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_send_command(), and handle_control_attachstream().

3.48.2.19 unsigned int edge_connection_t::edge_blocked_on_circ

True iff we've blocked reading until the circuit has fewer queued cells.

Referenced by set_streams_blocked_on_circ().

3.48.2.20 unsigned int edge_connection_t::chosen_exit_optional

For AP connections only. If 1, and we fail to reach the chosen exit, stop requiring it.

Referenced by assert_connection_ok(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), connection_ap_handshake_attach_circuit(), and connection_ap_process_end_not_open().

3.48.2.21 unsigned int edge_connection_t::chosen_exit_retries

For AP connections only. If non-zero, this exit node was picked as a result of the TrackHostExit, and the value decrements every time we fail to complete a circuit to our chosen exit – if it reaches zero, abandon the associated mapaddress.

Referenced by assert_connection_ok(), circuit_discard_optional_exit_enclaves(), and connection_ap_process_end_not_open().

3.48.2.22 struct evdns_server_request∗ edge_connection_t::dns_server_request `[read]`

If this is a DNSPort connection, this field holds the pending DNS request that we're going to try to answer.

Referenced by connection_about_to_close_connection(), connection_ap_handshake_socks_resolved(), dnsserv_reject_request(), dnsserv_resolved(), and evdns_server_callback().

The documentation for this struct was generated from the following file:

- **or.h**

3.49 entry_guard_t Struct Reference

Data Fields

- char **nickname** [MAX_NICKNAME_LEN+1]
- char **identity** [DIGEST_LEN]
- time_t **chosen_on_date**
- char ∗ **chosen_by_version**
- unsigned int **made_contact**: 1
- unsigned int **can_retry**: 1
- time_t **bad_since**
- time_t **unreachable_since**
- time_t **last_attempted**

3.49.1 Detailed Description

An **entry_guard_t** (p. 95) represents our information about a chosen long-term first hop, known as a "helper" node in the literature. We can't just use a **routerinfo_t** (p. 206), since we want to remember these even when we don't have a directory.

3.49.2 Field Documentation

3.49.2.1 time_t entry_guard_t::chosen_on_date

Approximately when was this guard added? "0" if we don't know.

Referenced by add_an_entry_guard(), and remove_obsolete_entry_guards().

3.49.2.2 char∗ entry_guard_t::chosen_by_version

What tor version added this guard? NULL if we don't know.

Referenced by add_an_entry_guard(), entry_guard_free(), and remove_obsolete_entry_guards().

3.49.2.3 unsigned int entry_guard_t::made_contact

0 if we have never connected to this router, 1 if we have.

Referenced by entry_guards_parse_state().

3.49.2.4 unsigned int entry_guard_t::can_retry

Should we retry connecting to this entry, in spite of having it marked as unreachable?

Referenced by add_an_entry_guard(), and entry_is_live().

3.49.2.5 time_t entry_guard_t::bad_since

0 if this guard is currently usable, or the time at which it was observed to become (according to the directory or the user configuration) unusable.

Referenced by add_an_entry_guard(), entry_guard_set_status(), entry_guards_parse_state(), entry_is_live(), and remove_dead_entry_guards().

3.49.2.6 time_t entry_guard_t::unreachable_since

0 if we can connect to this guard, or the time at which we first noticed we couldn't connect to it.

Referenced by entry_guards_parse_state(), entry_is_live(), and entry_is_time_to_retry().

3.49.2.7 time_t entry_guard_t::last_attempted

0 if we can connect to this guard, or the time at which we last failed to connect to it.

Referenced by entry_guards_parse_state(), and entry_is_time_to_retry().

The documentation for this struct was generated from the following file:

- **circuitbuild.c**

3.50 extend_info_t Struct Reference

`#include <or.h>`

Data Fields

- char **nickname** [MAX_HEX_NICKNAME_LEN+1]
- char **identity_digest** [DIGEST_LEN]
- uint16_t **port**
- **tor_addr_t addr**
- **crypto_pk_env_t ∗ onion_key**

3.50.1 Detailed Description

Information on router used when extending a circuit. We don't need a full **routerinfo_t** (p. 206) to extend: we only need addr:port:keyid to build an OR connection, and onion_key to create the onionskin. Note that for onehop general-purpose tunnels, the onion_key is NULL.

3.50.2 Field Documentation

3.50.2.1 char extend_info_t::nickname[MAX_HEX_NICKNAME_LEN+1]

This router's nickname for display.

Referenced by build_state_get_exit_nickname(), circuit_discard_optional_exit_enclaves(), circuit_extend_to_new_exit(), circuit_get_open_circ_or_launch(), circuit_handle_first_hop(), circuit_list_path_impl(), connection_ap_expire_beginning(), extend_info_alloc(), onion_extend_cpath(), onion_pick_cpath_exit(), rend_client_get_random_intro(), rend_client_introduction_acked(), rend_parse_introduction_points(), rend_parse_service_descriptor(), rend_service_dump_stats(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_relaunch_rendezvous(), rend_services_introduce(), routerset_contains_extendinfo(), and warn_if_last_router_excluded().

3.50.2.2 char extend_info_t::identity_digest[DIGEST_LEN]

Hash of this router's identity key.

Referenced by build_state_get_exit_router(), circuit_build_failed(), circuit_dump_by_conn(), circuit_find_to_cannibalize(), circuit_handle_first_hop(), circuit_is_acceptable(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), connection_ap_process_end_not_open(), extend_info_alloc(), find_intro_circuit(), handle_control_attachstream(), rend_client_remove_intro_point(), rend_client_send_introduction(), rend_encode_v2_intro_points(), rend_parse_introduction_points(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_services_introduce(), and routerset_contains_extendinfo().

3.50.2.3 uint16_t extend_info_t::port

OR port.

Referenced by circuit_dump_by_conn(), circuit_handle_first_hop(), circuit_is_acceptable(), circuit_send_next_onion_skin(), connection_ap_fail_onehop(), extend_info_alloc(), rend_encode_v2_intro_points(), rend_parse_introduction_points(), rend_service_introduce(), and routerset_contains_extendinfo().

3.50.2.4 tor_addr_t extend_info_t::addr

IP address.

Referenced by circuit_dump_by_conn(), circuit_handle_first_hop(), circuit_is_acceptable(), circuit_send_next_onion_skin(), connection_ap_fail_onehop(), extend_info_alloc(), rend_encode_v2_intro_points(), rend_parse_introduction_points(), rend_service_introduce(), and routerset_contains_extendinfo().

3.50.2.5 crypto_pk_env_t∗ extend_info_t::onion_key

Current onionskin key.

Referenced by circuit_send_next_onion_skin(), extend_info_alloc(), extend_info_dup(), extend_info_free(), rend_client_get_random_intro(), rend_encode_v2_intro_points(), rend_parse_introduction_points(), rend_service_introduce(), and should_use_create_fast_for_circuit().

The documentation for this struct was generated from the following file:

- **or.h**

3.51 extrainfo_t Struct Reference

`#include <or.h>`

Data Fields

- **signed_descriptor_t cache_info**
- char **nickname** [MAX_NICKNAME_LEN+1]
- unsigned int **bad_sig**: 1
- char * **pending_sig**
- size_t **pending_sig_len**

3.51.1 Detailed Description

Information needed to keep and cache a signed extra-info document.

3.51.2 Field Documentation

3.51.2.1 char extrainfo_t::nickname[MAX_NICKNAME_LEN+1]

The router's nickname.

Referenced by extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), and router_rebuild_descriptor().

3.51.2.2 unsigned int extrainfo_t::bad_sig

True iff we found the right key for this extra-info, verified the signature, and found it to be bad.

Referenced by routerinfo_incompatible_with_extrainfo().

3.51.2.3 char* extrainfo_t::pending_sig

If present, we didn't have the right key to verify this extra-info, so this is a copy of the signature in the document.

Referenced by extrainfo_free(), extrainfo_parse_entry_from_string(), and routerinfo_incompatible_with_extrainfo().

3.51.2.4 size_t extrainfo_t::pending_sig_len

Length of pending_sig.

Referenced by extrainfo_parse_entry_from_string(), and routerinfo_incompatible_with_extrainfo().

The documentation for this struct was generated from the following file:

- **or.h**

3.52 fp_pair_t Struct Reference

`#include <or.h>`

Data Fields

- char **first** [DIGEST_LEN]
- char **second** [DIGEST_LEN]

3.52.1 Detailed Description

A pair of digests created by dir_split_resource_info_fingerprint_pairs()

The documentation for this struct was generated from the following file:

- **or.h**

3.53 geoip_country_t Struct Reference

Data Fields

- char **countrycode** [3]
- uint32_t **n_v2_ns_requests** [REQUEST_HIST_LEN]
- uint32_t **n_v3_ns_requests** [REQUEST_HIST_LEN]

3.53.1 Detailed Description

A per-country record for GeoIP request history.

The documentation for this struct was generated from the following file:

- **geoip.c**

3.54 geoip_entry_t Struct Reference

Data Fields

- uint32_t **ip_low**
- uint32_t **ip_high**
- intptr_t **country**

3.54.1 Detailed Description

An entry from the GeoIP file: maps an IP range to a country.

3.54.2 Field Documentation

3.54.2.1 uint32_t geoip_entry_t::ip_low

The lowest IP in the range, in host order

Referenced by _geoip_compare_entries(), and geoip_add_entry().

3.54.2.2 uint32_t geoip_entry_t::ip_high

The highest IP in the range, in host order

Referenced by _geoip_compare_key_to_entry(), and geoip_add_entry().

3.54.2.3 intptr_t geoip_entry_t::country

An index into geoip_countries

Referenced by geoip_add_entry(), and geoip_get_country_by_ip().

The documentation for this struct was generated from the following file:

- **geoip.c**

3.55 getinfo_item_t Struct Reference

Data Fields

- const char * **varname**
- **getinfo_helper_t fn**
- const char * **desc**
- int **is_prefix**

3.55.1 Detailed Description

A single item for the GETINFO question-to-answer-function table.

3.55.2 Field Documentation

3.55.2.1 const char* getinfo_item_t::varname

The value (or prefix) of the question.

Referenced by handle_getinfo_helper(), and list_getinfo_options().

3.55.2.2 getinfo_helper_t getinfo_item_t::fn

The function that knows the answer: NULL if this entry is documentation-only.

Referenced by handle_getinfo_helper().

3.55.2.3 const char* getinfo_item_t::desc

Description of the variable.

The documentation for this struct was generated from the following file:

- **control.c**

3.56 insertion_time_elem_t Struct Reference

`#include <or.h>`

Data Fields

- struct **insertion_time_elem_t** ∗ **next**
- uint32_t **insertion_time**
- unsigned **counter**

3.56.1 Detailed Description

Number of cells added to a circuit queue including their insertion time on 10 millisecond detail; used for buffer statistics.

3.56.2 Field Documentation

3.56.2.1 struct insertion_time_elem_t∗ insertion_time_elem_t::next `[read]`

Next element in queue.

Referenced by cell_queue_append_packed_copy(), cell_queue_clear(), and connection_or_flush_from_first_active_circuit().

3.56.2.2 uint32_t insertion_time_elem_t::insertion_time

When were cells inserted (in 10 ms steps starting at 0:00 of the current day)?

Referenced by cell_queue_append_packed_copy(), and connection_or_flush_from_first_active_circuit().

3.56.2.3 unsigned insertion_time_elem_t::counter

How many cells were inserted?

Referenced by cell_queue_append_packed_copy(), and connection_or_flush_from_first_active_circuit().

The documentation for this struct was generated from the following file:

- **or.h**

3.57 insertion_time_queue_t Struct Reference

`#include <or.h>`

Data Fields

- struct **insertion_time_elem_t** ∗ **first**
- struct **insertion_time_elem_t** ∗ **last**

3.57.1 Detailed Description

Queue of insertion times.

3.57.2 Field Documentation

3.57.2.1 struct insertion_time_elem_t∗ insertion_time_queue_t::first `[read]`

First element in queue.

Referenced by cell_queue_append_packed_copy(), cell_queue_clear(), and connection_or_flush_from_first_active_circuit().

3.57.2.2 struct insertion_time_elem_t∗ insertion_time_queue_t::last `[read]`

Last element in queue.

Referenced by cell_queue_append_packed_copy(), and connection_or_flush_from_first_active_circuit().

The documentation for this struct was generated from the following file:

- **or.h**

3.58 link_history_t Struct Reference

Data Fields

- time_t **since**
- time_t **changed**
- unsigned long **n_extend_ok**
- unsigned long **n_extend_fail**

3.58.1 Detailed Description

History of an OR->OR link.

3.58.2 Field Documentation

3.58.2.1 time_t link_history_t::since

When did we start tracking this list?

Referenced by get_link_history().

3.58.2.2 time_t link_history_t::changed

When did we most recently note a change to this link

Referenced by get_link_history(), rep_hist_note_extend_failed(), rep_hist_note_extend_succeeded(), and rep_history_clean().

3.58.2.3 unsigned long link_history_t::n_extend_ok

How many times did extending from OR1 to OR2 succeed?

Referenced by rep_hist_dump_stats(), and rep_hist_note_extend_succeeded().

3.58.2.4 unsigned long link_history_t::n_extend_fail

How many times did extending from OR1 to OR2 fail?

Referenced by rep_hist_dump_stats(), and rep_hist_note_extend_failed().

The documentation for this struct was generated from the following file:

- **rephist.c**

3.59 log_severity_list_t Struct Reference

`#include <log.h>`

Data Fields

- log_domain_mask_t **masks** [LOG_DEBUG-LOG_ERR+1]

3.59.1 Detailed Description

Configures which severities are logged for each logging domain for a given log target.

3.59.2 Field Documentation

3.59.2.1 log_domain_mask_t log_severity_list_t::masks[LOG_DEBUG-LOG_ERR+1]

For each log severity, a bitmask of which domains a given logger is logging.

Referenced by get_min_log_level(), logv(), parse_log_severity_config(), set_log_severity_config(), and switch_logs_debug().

The documentation for this struct was generated from the following file:

- **log.h**

3.60 logfile_t Struct Reference

Data Fields

- struct **logfile_t** ∗ **next**
- char ∗ **filename**
- int **fd**
- int **seems_dead**
- int **needs_close**
- int **is_temporary**
- int **is_syslog**
- **log_callback callback**
- **log_severity_list_t** ∗ **severities**

3.60.1 Detailed Description

Information for a single logfile; only used in **log.c** (p. 668)

3.60.2 Field Documentation

3.60.2.1 struct logfile_t∗ logfile_t::next `[read]`

Next **logfile_t** (p. 108) in the linked list.

Referenced by add_callback_log(), add_stream_log_impl(), add_syslog_log(), change_callback_log_severity(), close_temp_logs(), delete_log(), get_min_log_level(), logs_free_all(), logv(), mark_logs_temp(), rollback_log_changes(), and switch_logs_debug().

3.60.2.2 char∗ logfile_t::filename

Filename to open.

Referenced by add_callback_log(), add_stream_log_impl(), add_syslog_log(), and log_free().

3.60.2.3 int logfile_t::fd

fd to receive log messages, or -1 for none.

Referenced by add_callback_log(), add_stream_log_impl(), add_syslog_log(), close_log(), log_tor_version(), and logv().

3.60.2.4 int logfile_t::seems_dead

Boolean: true if the stream seems to be kaput.

Referenced by logv().

3.60.2.5 int logfile_t::needs_close

Boolean: true if the stream gets closed on shutdown.

Referenced by add_file_log(), close_log(), and log_tor_version().

3.60.2.6 int logfile_t::is_temporary

Boolean: close after initializing logging subsystem.

Referenced by add_temp_log(), log_tor_version(), mark_logs_temp(), and rollback_log_changes().

3.60.2.7 int logfile_t::is_syslog

Boolean: send messages to syslog.

Referenced by add_syslog_log(), close_log(), and logv().

3.60.2.8 log_callback logfile_t::callback

If not NULL, send messages to this function.

Referenced by add_callback_log(), change_callback_log_severity(), and logv().

3.60.2.9 log_severity_list_t∗ logfile_t::severities

Which severity of messages should we log for each log domain?

Referenced by add_callback_log(), add_stream_log_impl(), add_syslog_log(), change_callback_log_severity(), get_min_log_level(), log_free(), logv(), and switch_logs_debug().

The documentation for this struct was generated from the following file:

- **log.c**

3.61 memarea_chunk_t Struct Reference

Data Fields

- struct **memarea_chunk_t** ∗ **next_chunk**
- size_t **mem_size**
- char ∗ **next_mem**
- union {
  char **mem** [1]
  void ∗ **_void_for_alignment**
  } **u**

3.61.1 Detailed Description

Implements part of a memarea. New memory is carved off from chunk->mem in increasing order until a request is too big, at which point a new chunk is allocated.

3.61.2 Field Documentation

3.61.2.1 struct memarea_chunk_t∗ memarea_chunk_t::next_chunk `[read]`

Next chunk in this area. Only kept around so we can free it.

Referenced by alloc_chunk(), chunk_free_unchecked(), memarea_alloc(), memarea_assert_ok(), memarea_clear(), memarea_clear_freelist(), memarea_drop_all(), memarea_get_stats(), and memarea_owns_ptr().

3.61.2.2 size_t memarea_chunk_t::mem_size

How much RAM is available in u.mem, total?

Referenced by alloc_chunk(), memarea_alloc(), memarea_assert_ok(), and memarea_get_stats().

3.61.2.3 char∗ memarea_chunk_t::next_mem

Next position in u.mem to allocate data at. If it's greater than or equal to mem+mem_size, this chunk is full.

Referenced by alloc_chunk(), chunk_free_unchecked(), memarea_alloc(), memarea_assert_ok(), memarea_clear(), and memarea_get_stats().

3.61.2.4 char memarea_chunk_t::mem[1]

Memory space in this chunk.

Referenced by alloc_chunk(), chunk_free_unchecked(), memarea_alloc(), memarea_assert_ok(), memarea_clear(), memarea_get_stats(), and memarea_owns_ptr().

3.61.2.5 void* memarea_chunk_t::_void_for_alignment

Dummy; used to make sure mem is aligned.

The documentation for this struct was generated from the following file:

- **memarea.c**

3.62 memarea_t Struct Reference

Data Fields

- **memarea_chunk_t ∗ first**

3.62.1 Detailed Description

A **memarea_t** (p. 112) is an allocation region for a set of small memory requests that will all be freed at once.

3.62.2 Field Documentation

3.62.2.1 memarea_chunk_t∗ memarea_t::first

Top of the chunk stack: never NULL.

Referenced by memarea_alloc(), memarea_assert_ok(), memarea_clear(), memarea_drop_all(), memarea_get_stats(), memarea_new(), and memarea_owns_ptr().

The documentation for this struct was generated from the following file:

- **memarea.c**

3.63 microdesc_cache_t Struct Reference

3.63.1 Detailed Description

A data structure to hold a bunch of cached microdescriptors. There are two active files in the cache: a "cache file" that we mmap, and a "journal file" that we append to. Periodically, we rebuild the cache file to hold only the microdescriptors that we want to keep

The documentation for this struct was generated from the following file:

- microdesc.c

3.64 microdesc_t Struct Reference

`#include <or.h>`

Public Member Functions

- **HT_ENTRY** (**microdesc_t**) node

Data Fields

- time_t **last_listed**
- **saved_location_t saved_location**: 3
- unsigned int **no_save**: 1
- off_t **off**
- char ∗ **body**
- size_t **bodylen**
- char **digest** [DIGEST256_LEN]
- **crypto_pk_env_t** ∗ **onion_pkey**
- **smartlist_t** ∗ **family**
- char ∗ **exitsummary**

3.64.1 Detailed Description

A microdescriptor is the smallest amount of information needed to build a circuit through a router. They are generated by the directory authorities, using information from the uploaded routerinfo documents. They are not self-signed, but are rather authenticated by having their hash in a signed networkstatus document.

3.64.2 Member Function Documentation

3.64.2.1 microdesc_t::HT_ENTRY (microdesc_t)

Hashtable node, used to look up the microdesc by its digest.

3.64.3 Field Documentation

3.64.3.1 time_t microdesc_t::last_listed

When was this microdescriptor last listed in a consensus document? Once a microdesc has been unlisted long enough, we can drop it.

Referenced by microdescs_parse_from_string().

3.64.3.2 saved_location_t microdesc_t::saved_location

Where is this microdescriptor currently stored?

Referenced by microdesc_cache_rebuild(), and microdesc_free().

3.64.3.3 unsigned int microdesc_t::no_save

If true, do not attempt to cache this microdescriptor on disk.

Referenced by microdesc_cache_rebuild().

3.64.3.4 off_t microdesc_t::off

If saved_location == SAVED_IN_CACHE, this field holds the offset of the microdescriptor in the cache.

Referenced by microdesc_cache_rebuild(), and microdescs_parse_from_string().

3.64.3.5 char∗ microdesc_t::body

A pointer to the encoded body of the microdescriptor. If the saved_location is SAVED_IN_CACHE, then the body is a pointer into an mmap'd region. Otherwise, it is a malloc'd string. The string might not be NUL-terminated; take the length from **bodylen**.

Referenced by connection_dirserv_add_microdescs_to_outbuf(), microdesc_cache_rebuild(), microdesc_free(), and microdescs_parse_from_string().

3.64.3.6 size_t microdesc_t::bodylen

The length of the microdescriptor in **body**.

Referenced by connection_dirserv_add_microdescs_to_outbuf(), and microdescs_parse_from_string().

3.64.3.7 char microdesc_t::digest[DIGEST256_LEN]

A SHA256-digest of the microdescriptor.

Referenced by dirvote_format_microdesc_vote_line(), microdesc_cache_lookup_by_digest256(), and microdescs_parse_from_string().

3.64.3.8 crypto_pk_env_t∗ microdesc_t::onion_pkey

As **routerinfo_t.onion_pkey** (p. 208)

Referenced by microdesc_free(), and microdescs_parse_from_string().

3.64.3.9 smartlist_t∗ microdesc_t::family

As routerinfo_t.family

Referenced by microdesc_free(), and microdescs_parse_from_string().

3.64.3.10 char∗ microdesc_t::exitsummary

Encoded exit policy summary exit policy summary - XXX this probably should not stay a string.

Referenced by microdesc_free(), and microdescs_parse_from_string().

The documentation for this struct was generated from the following file:

- **or.h**

3.65 mp_allocated_t Struct Reference

Data Fields

- **mp_chunk_t ∗ in_chunk**
- union {
  mp_allocated_t ∗ next_free
  char **mem** [1]
  ALIGNMENT_TYPE **_dummy**
  } **u**

3.65.1 Detailed Description

Holds a single allocated item, allocated as part of a chunk.

3.65.2 Field Documentation

3.65.2.1 mp_chunk_t∗ mp_allocated_t::in_chunk

The chunk that this item is allocated in. This adds overhead to each allocated item, thus making this implementation inappropriate for very small items.

Referenced by mp_pool_get(), and mp_pool_release().

3.65.2.2 mp_allocated_t∗ mp_allocated_t::next_free

If this item is free, the next item on the free list.

Referenced by mp_pool_get(), and mp_pool_release().

3.65.2.3 char mp_allocated_t::mem[1]

If this item is not free, the actual memory contents of this item. (Not actual size.)

3.65.2.4 ALIGNMENT_TYPE mp_allocated_t::_dummy

An extra element to the union to insure correct alignment.

The documentation for this struct was generated from the following file:

- mempool.c

3.66 mp_chunk_t Struct Reference

Data Fields

- unsigned long **magic**
- **mp_chunk_t** ∗ **next**
- **mp_chunk_t** ∗ **prev**
- **mp_pool_t** ∗ **pool**
- **mp_allocated_t** ∗ **first_free**
- int **n_allocated**
- int **capacity**
- size_t **mem_size**
- char ∗ **next_mem**
- char **mem** [1]

3.66.1 Detailed Description

A chunk of memory. Chunks come from malloc; we use them

3.66.2 Field Documentation

3.66.2.1 unsigned long mp_chunk_t::magic

Must be MP_CHUNK_MAGIC if this chunk is valid.

Referenced by mp_pool_clean(), and mp_pool_release().

3.66.2.2 mp_chunk_t∗ mp_chunk_t::next

The next free, used, or full chunk in sequence.

Referenced by mp_pool_clean(), mp_pool_get(), mp_pool_log_status(), and mp_pool_release().

3.66.2.3 mp_chunk_t∗ mp_chunk_t::prev

The previous free, used, or full chunk in sequence.

Referenced by mp_pool_get(), and mp_pool_release().

3.66.2.4 mp_pool_t∗ mp_chunk_t::pool

The pool that this chunk is part of.

Referenced by mp_pool_release().

3.66.2.5 mp_allocated_t∗ mp_chunk_t::first_free

First free item in the freelist for this chunk. Note that this may be NULL even if this chunk is not at capacity: if so, the free memory at next_mem has not yet been carved into items.

Referenced by mp_pool_get(), and mp_pool_release().

3.66.2.6 int mp_chunk_t::n_allocated

Number of currently allocated items in this chunk.

Referenced by mp_pool_get(), mp_pool_log_status(), and mp_pool_release().

3.66.2.7 int mp_chunk_t::capacity

Number of items that can be fit into this chunk.

Referenced by mp_pool_get(), and mp_pool_release().

3.66.2.8 size_t mp_chunk_t::mem_size

Number of usable bytes in mem.

Referenced by mp_pool_get(), and mp_pool_log_status().

3.66.2.9 char* mp_chunk_t::next_mem

Pointer into part of **mem** not yet carved up.

Referenced by mp_pool_get(), and mp_pool_release().

3.66.2.10 char mp_chunk_t::mem[1]

Storage for this chunk. (Not actual size.)

Referenced by mp_pool_get(), and mp_pool_release().

The documentation for this struct was generated from the following file:

- mempool.c

3.67 network_liveness_t Struct Reference

`#include <or.h>`

Data Fields

- time_t **network_last_live**
- int **nonlive_timeouts**
- int **nonlive_discarded**
- int8_t * **timeouts_after_firsthop**
- int **num_recent_circs**
- int **after_firsthop_idx**
- int **net_suspended**

3.67.1 Detailed Description

Information about the state of our local network connection

3.67.2 Field Documentation

3.67.2.1 time_t network_liveness_t::network_last_live

The timestamp we last completed a TLS handshake or received a cell

Referenced by circuit_build_times_network_check_live(), circuit_build_times_network_is_live(), and circuit_build_times_network_timeout().

3.67.2.2 int network_liveness_t::nonlive_timeouts

If the network is not live, how many timeouts has this caused?

Referenced by circuit_build_times_network_check_live(), circuit_build_times_network_is_live(), and circuit_build_times_network_timeout().

3.67.2.3 int network_liveness_t::nonlive_discarded

If the network is not live, have we yet discarded our history?

Referenced by circuit_build_times_network_check_live(), and circuit_build_times_network_is_live().

3.67.2.4 int8_t* network_liveness_t::timeouts_after_firsthop

Circular array of circuits that have made it to the first hop. Slot is 1 if circuit timed out, 0 if circuit succeeded

Referenced by circuit_build_times_init(), circuit_build_times_network_check_changed(), circuit_build_times_network_circ_success(), circuit_build_times_network_timeout(), and circuit_build_times_new_consensus_params().

3.67.2.5 int network_liveness_t::num_recent_circs

Number of elements allocated for the above array

Referenced by circuit_build_times_init(), circuit_build_times_network_check_changed(), circuit_build_times_network_circ_success(), circuit_build_times_network_timeout(), and circuit_build_times_new_consensus_params().

3.67.2.6 int network_liveness_t::after_firsthop_idx

Index into circular array.

Referenced by circuit_build_times_network_check_changed(), circuit_build_times_network_circ_success(), circuit_build_times_network_timeout(), and circuit_build_times_new_consensus_params().

3.67.2.7 int network_liveness_t::net_suspended

The network is not live. Timeout gathering is suspended

Referenced by circuit_build_times_network_check_live().

The documentation for this struct was generated from the following file:

- **or.h**

3.68 networkstatus_t Struct Reference

`#include <or.h>`

Data Fields

- **networkstatus_type_t type**: 8
- **consensus_flavor_t flavor**: 8
- time_t **published**
- time_t **valid_after**
- time_t **fresh_until**
- time_t **valid_until**
- int **consensus_method**
- **smartlist_t** ∗ **supported_methods**
- int **vote_seconds**
- int **dist_seconds**
- char ∗ **client_versions**
- char ∗ **server_versions**
- **smartlist_t** ∗ **known_flags**
- **smartlist_t** ∗ **net_params**
- **smartlist_t** ∗ **weight_params**
- **smartlist_t** ∗ **voters**
- struct **authority_cert_t** ∗ **cert**
- **digests_t digests**
- **smartlist_t** ∗ **routerstatus_list**
- digestmap_t ∗ **desc_digest_map**

3.68.1 Detailed Description

A common structure to hold a v3 network status vote, or a v3 network status consensus.

3.68.2 Field Documentation

3.68.2.1 networkstatus_type_t networkstatus_t::type

Vote, consensus, or opinion?

Referenced by format_networkstatus_vote(), get_voter(), networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), networkstatus_parse_vote_from_string(), and networkstatus_vote_free().

3.68.2.2 consensus_flavor_t networkstatus_t::flavor

If a consensus, what kind?

Referenced by dirvote_add_signatures_to_pending_consensus(), networkstatus_add_detached_signatures(), networkstatus_format_signatures(), networkstatus_parse_vote_from_string(), and networkstatus_set_current_consensus().

3.68.2.3 time_t networkstatus_t::published

Vote only: Time when vote was written.

Referenced by dirvote_add_vote(), format_networkstatus_vote(), and networkstatus_parse_vote_from_string().

3.68.2.4 time_t networkstatus_t::valid_after

Time after which this vote or consensus applies.

Referenced by directory_get_from_dirserver(), dirvote_add_vote(), dirvote_recalculate_timing(), format_networkstatus_vote(), networkstatus_add_detached_signatures(), networkstatus_get_detached_signatures(), networkstatus_get_live_consensus(), networkstatus_get_reasonably_live_consensus(), networkstatus_parse_vote_from_string(), networkstatus_set_current_consensus(), router_reload_consensus_networkstatus(), and update_consensus_networkstatus_fetch_time().

3.68.2.5 time_t networkstatus_t::fresh_until

Time before which this is the most recent vote or consensus.

Referenced by directory_handle_command_get(), dirvote_recalculate_timing(), format_networkstatus_vote(), networkstatus_add_detached_signatures(), networkstatus_get_detached_signatures(), networkstatus_parse_vote_from_string(), and update_consensus_networkstatus_fetch_time().

3.68.2.6 time_t networkstatus_t::valid_until

Time after which this vote or consensus should not be used.

Referenced by format_networkstatus_vote(), networkstatus_add_detached_signatures(), networkstatus_get_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_set_current_consensus(), run_scheduled_events(), signed_descs_update_status_from_consensus_networkstatus(), and update_consensus_networkstatus_fetch_time().

3.68.2.7 int networkstatus_t::consensus_method

Consensus only: what method was used to produce this consensus?

Referenced by networkstatus_parse_vote_from_string().

3.68.2.8 smartlist_t* networkstatus_t::supported_methods

Vote only: what methods is this voter willing to use?

Referenced by networkstatus_parse_vote_from_string(), and networkstatus_vote_free().

3.68.2.9 int networkstatus_t::vote_seconds

How long does this vote/consensus claim that authorities take to distribute their votes to one another?

Referenced by dirvote_recalculate_timing(), format_networkstatus_vote(), and networkstatus_parse_vote_from_string().

3.68.2.10 int networkstatus_t::dist_seconds

How long does this vote/consensus claim that authorities take to distribute their consensus signatures to one another?

Referenced by dirvote_recalculate_timing(), format_networkstatus_vote(), and networkstatus_parse_vote_from_string().

3.68.2.11 char* networkstatus_t::client_versions

Comma-separated list of recommended client software, or NULL if this voter has no opinion.

Referenced by format_networkstatus_vote(), getinfo_helper_events(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), and routers_update_all_from_networkstatus().

3.68.2.12 smartlist_t* networkstatus_t::known_flags

List of flags that this vote/consensus applies to routers. If a flag is not listed here, the voter has no opinion on what its value should be.

Referenced by format_networkstatus_vote(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), and routerstatus_parse_entry_from_string().

3.68.2.13 smartlist_t* networkstatus_t::net_params

List of key=value strings for the parameters in this vote or consensus, sorted by key.

Referenced by format_networkstatus_vote(), networkstatus_get_param(), networkstatus_parse_vote_from_string(), and networkstatus_vote_free().

3.68.2.14 smartlist_t* networkstatus_t::weight_params

List of key=value strings for the bw weight parameters in the consensus.

Referenced by networkstatus_get_bw_weight(), and networkstatus_parse_vote_from_string().

3.68.2.15 smartlist_t* networkstatus_t::voters

List of **networkstatus_voter_info_t** (p. 129). For a vote, only one element is included. For a consensus, one element is included for every voter whose vote contributed to the consensus.

Referenced by authority_certs_fetch_missing(), client_likes_consensus(), dirvote_add_vote(), format_networkstatus_vote(), get_voter(), networkstatus_check_consensus_signature(), networkstatus_format_signatures(), networkstatus_get_voter_by_id(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), and update_consensus_router_descriptor_downloads().

3.68.2.16 struct authority_cert_t* networkstatus_t::cert [read]

Vote only: the voter's certificate.

Referenced by dirvote_add_vote(), format_networkstatus_vote(), networkstatus_parse_vote_from_string(), and networkstatus_vote_free().

3.68.2.17 digests_t networkstatus_t::digests

Digests of this document, as signed.

Referenced by networkstatus_add_detached_signatures(), networkstatus_check_document_signature(), networkstatus_get_detached_signatures(), networkstatus_parse_vote_from_string(), and networkstatus_set_current_consensus().

3.68.2.18 smartlist_t* networkstatus_t::routerstatus_list

List of router statuses, sorted by identity digest. For a vote, the elements are **vote_routerstatus_t** (p. 252); for a consensus, the elements are **routerstatus_t** (p. 218).

Referenced by control_event_newconsensus(), count_usable_descriptors(), format_networkstatus_vote(), getinfo_helper_networkstatus(), hid_serv_get_responsible_directories(), networkstatus_copy_old_consensus_info(), networkstatus_parse_vote_from_string(), networkstatus_reset_warnings(), networkstatus_verify_bw_weights(), networkstatus_vote_find_entry(), networkstatus_vote_find_entry_idx(), networkstatus_vote_free(), notify_control_networkstatus_changed(), router_get_consensus_status_by_descriptor_digest(), router_get_consensus_status_by_id(), router_get_consensus_status_by_nickname(), router_pick_directory_server_impl(), routerlist_remove_old_routers(), routers_update_status_from_consensus_networkstatus(), routerstatus_list_update_named_server_map(), update_consensus_router_descriptor_downloads(), and upload_service_descriptor().

3.68.2.19 digestmap_t* networkstatus_t::desc_digest_map

If present, a map from descriptor digest to elements of routerstatus_list.

Referenced by networkstatus_vote_free(), router_get_consensus_status_by_descriptor_digest(), and signed_descs_update_status_from_consensus_networkstatus().

The documentation for this struct was generated from the following file:

- **or.h**

3.69 networkstatus_v2_t Struct Reference

`#include <or.h>`

Data Fields

- time_t **received_on**
- char **networkstatus_digest** [DIGEST_LEN]
- time_t **published_on**
- char * **source_address**
- uint32_t **source_addr**
- uint16_t **source_dirport**
- unsigned int **binds_names**:1
- unsigned int **recommends_versions**:1
- unsigned int **lists_bad_exits**:1
- unsigned int **lists_bad_directories**:1
- char **identity_digest** [DIGEST_LEN]
- char * **contact**
- **crypto_pk_env_t** * **signing_key**
- char * **client_versions**
- char * **server_versions**
- **smartlist_t** * **entries**

3.69.1 Detailed Description

Contents of a v2 (non-consensus, non-vote) network status object.

3.69.2 Field Documentation

3.69.2.1 time_t networkstatus_v2_t::received_on

When did we receive the network-status document?

Referenced by router_set_networkstatus_v2().

3.69.2.2 char networkstatus_v2_t::networkstatus_digest[DIGEST_LEN]

What was the digest of the document?

Referenced by networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

3.69.2.3 time_t networkstatus_v2_t::published_on

Declared publication date.

Referenced by _compare_networkstatus_v2_published_on(), add_networkstatus_to_cache(), networkstatus_v2_list_clean(), networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

3.69.2.4 char∗ networkstatus_v2_t::source_address

Canonical directory server hostname.

Referenced by networkstatus_v2_free(), networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

3.69.2.5 uint32_t networkstatus_v2_t::source_addr

Canonical directory server IP.

Referenced by networkstatus_v2_parse_from_string().

3.69.2.6 uint16_t networkstatus_v2_t::source_dirport

Canonical directory server dirport.

Referenced by networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

3.69.2.7 unsigned int networkstatus_v2_t::binds_names

True iff this directory server binds names.

Referenced by networkstatus_v2_parse_from_string().

3.69.2.8 unsigned int networkstatus_v2_t::recommends_versions

True iff this directory server recommends client and server software versions.

Referenced by networkstatus_v2_parse_from_string().

3.69.2.9 unsigned int networkstatus_v2_t::lists_bad_exits

True iff this directory server marks malfunctioning exits as bad.

Referenced by networkstatus_v2_parse_from_string().

3.69.2.10 unsigned int networkstatus_v2_t::lists_bad_directories

True iff this directory server marks malfunctioning directories as bad.

Referenced by networkstatus_v2_parse_from_string().

3.69.2.11 char networkstatus_v2_t::identity_digest[DIGEST_LEN]

Digest of signing key.

Referenced by add_networkstatus_to_cache(), networkstatus_v2_list_clean(), networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

3.69.2.12 char∗ networkstatus_v2_t::contact

How to contact directory admin? (may be NULL).

Referenced by networkstatus_v2_free(), and networkstatus_v2_parse_from_string().

3.69.2.13 crypto_pk_env_t* networkstatus_v2_t::signing_key

Key used to sign this directory.

Referenced by networkstatus_v2_free(), and networkstatus_v2_parse_from_string().

3.69.2.14 char* networkstatus_v2_t::client_versions

comma-separated list of recommended client versions.

Referenced by networkstatus_v2_free(), and networkstatus_v2_parse_from_string().

3.69.2.15 char* networkstatus_v2_t::server_versions

comma-separated list of recommended server versions.

Referenced by networkstatus_v2_free(), and networkstatus_v2_parse_from_string().

3.69.2.16 smartlist_t* networkstatus_v2_t::entries

List of routerstatus_t*. This list is kept sorted by identity_digest.

Referenced by networkstatus_v2_find_entry(), networkstatus_v2_free(), networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

The documentation for this struct was generated from the following file:

- **or.h**

3.70 networkstatus_voter_info_t Struct Reference

`#include <or.h>`

Data Fields

- char **identity_digest** [DIGEST_LEN]
- char ∗ **nickname**
- char **legacy_id_digest** [DIGEST_LEN]
- char ∗ **address**
- uint32_t **addr**
- uint16_t **dir_port**
- uint16_t **or_port**
- char ∗ **contact**
- char **vote_digest** [DIGEST_LEN]
- **smartlist_t** ∗ **sigs**

3.70.1 Detailed Description

Information about a single voter in a vote or a consensus.

3.70.2 Field Documentation

3.70.2.1 char networkstatus_voter_info_t::identity_digest[DIGEST_LEN]

Declared SHA-1 digest of this voter's identity key

Referenced by _compare_dir_src_ents_by_authority_id(), dirvote_add_vote(), dirvote_get_vote(), networkstatus_compute_consensus(), networkstatus_parse_vote_from_string(), and update_consensus_router_descriptor_downloads().

3.70.2.2 char∗ networkstatus_voter_info_t::nickname

Nickname of this voter

Referenced by dirvote_add_vote(), format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.3 char networkstatus_voter_info_t::legacy_id_digest[DIGEST_LEN]

Digest of this voter's "legacy" identity key, if any. In vote only; for consensuses, we treat legacy keys as additional signers.

Referenced by _compare_dir_src_ents_by_authority_id(), format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.4 char∗ networkstatus_voter_info_t::address

Address of this voter, in string format.

Referenced by dirvote_add_vote(), format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.5 uint32_t networkstatus_voter_info_t::addr

Address of this voter, in IPv4, in host order.

Referenced by format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.6 uint16_t networkstatus_voter_info_t::dir_port

Directory port of this voter

Referenced by format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.7 uint16_t networkstatus_voter_info_t::or_port

OR port of this voter

Referenced by format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.8 char* networkstatus_voter_info_t::contact

Contact information for this voter.

Referenced by format_networkstatus_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.9 char networkstatus_voter_info_t::vote_digest[DIGEST_LEN]

Digest of this voter's vote, as signed.

Referenced by dirvote_add_vote(), networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

3.70.2.10 smartlist_t* networkstatus_voter_info_t::sigs

The signature of the document and the signature's status.

Referenced by dirvote_add_vote(), networkstatus_add_detached_signatures(), networkstatus_parse_vote_from_string(), and voter_get_sig_by_algorithm().

The documentation for this struct was generated from the following file:

- **or.h**

3.71 ns_detached_signatures_t Struct Reference

`#include <or.h>`

Data Fields

- time_t **valid_after**
- time_t **fresh_until**
- time_t **valid_until**
- strmap_t * **digests**
- strmap_t * **signatures**

3.71.1 Detailed Description

A set of signatures for a networkstatus consensus. Unless otherwise noted, all fields are as for **networkstatus_t** (p. 122).

3.71.2 Field Documentation

3.71.2.1 strmap_t* ns_detached_signatures_t::digests

Map from flavor name to **digestset_t** (p. 78)

Referenced by detached_get_digests(), networkstatus_add_detached_signatures(), networkstatus_parse_detached_signatures(), and ns_detached_signatures_free().

3.71.2.2 strmap_t* ns_detached_signatures_t::signatures

Map from flavor name to list of **document_signature_t** (p. 86)

Referenced by detached_get_signatures(), dirvote_add_signatures_to_pending_consensus(), networkstatus_add_detached_signatures(), networkstatus_parse_detached_signatures(), and ns_detached_signatures_free().

The documentation for this struct was generated from the following file:

- **or.h**

3.72 onion_queue_t Struct Reference

Data Fields

- **or_circuit_t** ∗ **circ**
- char ∗ **onionskin**
- time_t **when_added**
- struct **onion_queue_t** ∗ **next**

3.72.1 Detailed Description

Type for a linked list of circuits that are waiting for a free CPU worker to process a waiting onion handshake.

The documentation for this struct was generated from the following file:

- **onion.c**

3.73 open_file_t Struct Reference

Data Fields

- char * **tempname**
- char * **filename**
- unsigned **rename_on_close**:1
- unsigned **binary**:1
- int **fd**
- FILE * **stdio_file**

3.73.1 Detailed Description

Represents a file that we're writing to, with support for atomic commit: we can write into a temporary file, and either remove the file on failure, or replace the original file on success.

3.73.2 Field Documentation

3.73.2.1 char* open_file_t::tempname

Name of the temporary file.

Referenced by finish_writing_to_file_impl(), and start_writing_to_file().

3.73.2.2 char* open_file_t::filename

Name of the original file.

Referenced by fdopen_file(), finish_writing_to_file_impl(), and start_writing_to_file().

3.73.2.3 unsigned open_file_t::rename_on_close

Are we using the temporary file or not?

Referenced by finish_writing_to_file_impl(), and start_writing_to_file().

3.73.2.4 unsigned open_file_t::binary

Did we open in binary mode?

Referenced by fdopen_file(), and start_writing_to_file().

3.73.2.5 int open_file_t::fd

fd for the open file.

Referenced by fdopen_file(), finish_writing_to_file_impl(), and start_writing_to_file().

3.73.2.6 FILE∗ open_file_t::stdio_file

stdio wrapper for **fd**.

Referenced by fdopen_file(), and finish_writing_to_file_impl().

The documentation for this struct was generated from the following file:

- **util.c**

3.74 or_circuit_t Struct Reference

`#include <or.h>`

Data Fields

- **circuit_t _base**
- struct **circuit_t** ∗ **next_active_on_p_conn**
- struct **circuit_t** ∗ **prev_active_on_p_conn**
- **circid_t p_circ_id**
- **cell_queue_t p_conn_cells**
- **or_connection_t** ∗ **p_conn**
- **edge_connection_t** ∗ **n_streams**
- **edge_connection_t** ∗ **resolving_streams**
- **crypto_cipher_env_t** ∗ **p_crypto**
- **crypto_cipher_env_t** ∗ **n_crypto**
- **crypto_digest_env_t** ∗ **p_digest**
- **crypto_digest_env_t** ∗ **n_digest**
- struct **or_circuit_t** ∗ **rend_splice**
- char **rend_token** [REND_TOKEN_LEN]
- char **handshake_digest** [DIGEST_LEN]
- unsigned int **remaining_relay_early_cells**: 4
- unsigned int **is_first_hop**: 1
- uint32_t **processed_cells**
- uint64_t **total_cell_waiting_time**
- **cell_ewma_t p_cell_ewma**

3.74.1 Detailed Description

An **or_circuit_t** (p. 135) holds information needed to implement a circuit at an OR.

3.74.2 Field Documentation

3.74.2.1 struct circuit_t∗ or_circuit_t::next_active_on_p_conn `[read]`

Next circuit in the doubly-linked ring of circuits waiting to add cells to p_conn. NULL if we have no cells pending, or if we're not linked to an OR connection.

Referenced by circuit_set_p_circid_orconn(), HT_HEAD(), and next_circ_on_conn_p().

3.74.2.2 struct circuit_t∗ or_circuit_t::prev_active_on_p_conn `[read]`

Previous circuit in the doubly-linked ring of circuits waiting to add cells to p_conn. NULL if we have no cells pending, or if we're not linked to an OR connection.

Referenced by prev_circ_on_conn_p().

3.74.2.3 circid_t or_circuit_t::p_circ_id

The circuit_id used in the previous (backward) hop of this circuit.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), assign_onionskin_to_cpuworker(), circuit_dump_by_conn(), circuit_expire_old_circuits_serverside(), circuit_get_by_circid_orconn_impl(), circuit_receive_relay_cell(), HT_HEAD(), onion_pending_remove(), onionskin_answer(), relay_send_command_from_edge(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), and rend_mid_rendezvous().

3.74.2.4 cell_queue_t or_circuit_t::p_conn_cells

Queue of cells waiting to be transmitted on p_conn.

Referenced by append_cell_to_circuit_queue(), circuit_free(), circuit_set_p_circid_orconn(), connection_or_flush_from_first_active_circuit(), dump_cell_pool_usage(), and HT_HEAD().

3.74.2.5 or_connection_t* or_circuit_t::p_conn

The OR connection that is previous in this circuit.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), assign_onionskin_to_cpuworker(), circuit_expire_old_circuits_serverside(), circuit_get_by_circid_orconn_impl(), circuit_package_relay_cell(), circuit_receive_relay_cell(), circuit_unlink_all_from_or_conn(), connection_exit_begin_conn(), HT_HEAD(), make_circuit_active_on_conn(), make_circuit_inactive_on_conn(), next_circ_on_conn_p(), onion_next_task(), onionskin_answer(), and prev_circ_on_conn_p().

3.74.2.6 edge_connection_t* or_circuit_t::n_streams

Linked list of Exit streams associated with this circuit.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_consider_stop_edge_reading(), circuit_detach_stream(), circuit_expire_old_circuits_serverside(), connection_exit_connect_dir(), dns_found_answer(), dns_resolve(), and set_streams_blocked_on_circ().

3.74.2.7 edge_connection_t* or_circuit_t::resolving_streams

Linked list of Exit streams associated with this circuit that are still being resolved.

Referenced by _circuit_mark_for_close(), circuit_detach_stream(), circuit_expire_old_circuits_serverside(), circuit_free_all(), and dns_resolve().

3.74.2.8 crypto_cipher_env_t* or_circuit_t::p_crypto

The cipher used by intermediate hops for cells heading toward the OP.

Referenced by assert_circuit_ok(), circuit_free(), circuit_package_relay_cell(), and onionskin_answer().

3.74.2.9 crypto_cipher_env_t* or_circuit_t::n_crypto

The cipher used by intermediate hops for cells heading away from the OP.

Referenced by assert_circuit_ok(), circuit_free(), and onionskin_answer().

3.74.2.10 crypto_digest_env_t* or_circuit_t::p_digest

The integrity-checking digest used by intermediate hops, for cells packaged here and heading towards the OP.

Referenced by assert_circuit_ok(), circuit_free(), circuit_package_relay_cell(), and onionskin_answer().

3.74.2.11 crypto_digest_env_t* or_circuit_t::n_digest

The integrity-checking digest used by intermediate hops, for cells packaged at the OP and arriving here.

Referenced by assert_circuit_ok(), circuit_free(), and onionskin_answer().

3.74.2.12 struct or_circuit_t* or_circuit_t::rend_splice `[read]`

Points to spliced circuit if purpose is REND_ESTABLISHED, and circuit is not marked for close.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_free(), circuit_receive_relay_cell(), and rend_mid_rendezvous().

3.74.2.13 char or_circuit_t::rend_token[REND_TOKEN_LEN]

A hash of location-hidden service's PK if purpose is INTRO_POINT, or a rendezvous cookie if purpose is REND_POINT_WAITING. Filled with zeroes otherwise. ???? move to a subtype or adjunct structure? Wastes 20 bytes. -NM

Referenced by rend_mid_establish_intro(), rend_mid_establish_rendezvous(), and rend_mid_rendezvous().

3.74.2.14 char or_circuit_t::handshake_digest[DIGEST_LEN]

Stores KH for the handshake.

Referenced by onionskin_answer(), and rend_mid_establish_intro().

3.74.2.15 unsigned int or_circuit_t::remaining_relay_early_cells

How many more relay_early cells can we send on this circuit, according to the specification?

Referenced by command_process_relay_cell(), and or_circuit_new().

3.74.2.16 unsigned int or_circuit_t::is_first_hop

True iff this circuit was made with a CREATE_FAST cell.

Referenced by circuit_expire_old_circuits_serverside(), connection_exit_begin_conn(), and onionskin_answer().

3.74.2.17 uint32_t or_circuit_t::processed_cells

Number of cells that were removed from circuit queue; reset every time when writing buffer stats to disk.

Referenced by connection_or_flush_from_first_active_circuit(), and rep_hist_buffer_stats_add_circ().

3.74.2.18 uint64_t or_circuit_t::total_cell_waiting_time

Total time in milliseconds that cells spent in both app-ward and exit-ward queues of this circuit; reset every time when writing buffer stats to disk.

Referenced by connection_or_flush_from_first_active_circuit(), and rep_hist_buffer_stats_add_circ().

3.74.2.19 cell_ewma_t or_circuit_t::p_cell_ewma

The EWMA count for the number of cells flushed from the p_conn_cells queue.

Referenced by assert_active_circuits_ok(), make_circuit_active_on_conn(), make_circuit_inactive_on_conn(), and or_circuit_new().

The documentation for this struct was generated from the following file:

- **or.h**

3.75 or_connection_t Struct Reference

`#include <or.h>`

Data Fields

- **connection_t _base**
- char **identity_digest** [DIGEST_LEN]
- char * **nickname**
- **tor_tls_t * tls**
- int **tls_error**
- time_t **client_used**
- **tor_addr_t real_addr**
- **circ_id_type_t circ_id_type**:2
- unsigned int **is_canonical**:1
- unsigned int **is_bad_for_new_circs**:1
- uint8_t **link_proto**
- **circid_t next_circ_id**
- **or_handshake_state_t * handshake_state**
- time_t **timestamp_lastempty**
- time_t **timestamp_last_added_nonpadding**
- int **bandwidthrate**
- int **bandwidthburst**
- int **read_bucket**
- int **write_bucket**
- int **n_circuits**
- struct **circuit_t * active_circuits**
- **smartlist_t * active_circuit_pqueue**
- unsigned **active_circuit_pqueue_last_recalibrated**
- struct **or_connection_t * next_with_same_id**

3.75.1 Detailed Description

Subtype of **connection_t** (p. 53) for an "OR connection" – that is, one that speaks cells over TLS.

3.75.2 Field Documentation

3.75.2.1 char or_connection_t::identity_digest[DIGEST_LEN]

Hash of the public RSA key for the other side's identity key, or zeroes if the other side hasn't shown us a valid identity key.

Referenced by circuit_get_all_pending_on_or_conn(), circuit_n_conn_done(), circuit_send_next_onion_skin(), command_process_create_cell(), command_process_netinfo_cell(), connection_about_to_close_connection(), connection_exit_begin_conn(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_get_for_extend(), connection_or_init_conn_from_address(), connection_or_remove_from_identity_map(), connection_or_set_identity_digest(), connection_or_set_state_open(), and orconn_target_get_name().

3.75.2.2 char∗ or_connection_t::nickname

Nickname of OR on other side (if any).

Referenced by _connection_free(), circuit_count_pending_on_or_conn(), circuit_n_conn_done(), connection_or_check_valid_tls_handshake(), connection_or_init_conn_from_address(), connection_or_remove_from_identity_map(), connection_read_to_buf(), and getinfo_helper_events().

3.75.2.3 tor_tls_t∗ or_connection_t::tls

TLS connection state.

Referenced by _connection_free(), assert_connection_ok(), connection_handle_write_impl(), connection_or_check_valid_tls_handshake(), connection_or_nonopen_was_started_here(), connection_or_process_cells_from_inbuf(), connection_read_to_buf(), connection_tls_continue_handshake(), connection_tls_finish_handshake(), connection_tls_start_handshake(), and dumpstats().

3.75.2.4 int or_connection_t::tls_error

Last tor_tls error code.

Referenced by connection_about_to_close_connection(), and connection_read_to_buf().

3.75.2.5 time_t or_connection_t::client_used

When we last used this conn for any client traffic. If not recent, we can rate limit it further.

Referenced by circuit_deliver_create_cell(), command_process_relay_cell(), and relay_send_command_from_edge().

3.75.2.6 tor_addr_t or_connection_t::real_addr

The actual address that this connection came from or went to. The **addr** field is prone to getting overridden by the address from the router descriptor matching **identity_digest**.

Referenced by command_process_netinfo_cell(), connection_exit_begin_conn(), connection_or_get_for_extend(), connection_or_group_set_badness(), and connection_or_init_conn_from_address().

3.75.2.7 circ_id_type_t or_connection_t::circ_id_type

When we send CREATE cells along this connection, which half of the space should we use?

Referenced by command_process_create_cell(), connection_or_check_valid_tls_handshake(), and get_unique_circ_id_by_conn().

3.75.2.8 unsigned int or_connection_t::is_canonical

Should this connection be used for extending circuits to the server matching the **identity_digest** field? Set to true if we're pretty sure we aren't getting MITMed, either because we're connected to an address listed in a server descriptor, or because an authenticated NETINFO cell listed the address we're connected to as recognized.

Referenced by command_process_netinfo_cell(), connection_or_get_for_extend(), connection_or_group_set_badness(), connection_or_init_conn_from_address(), and connection_or_is_better().

3.75.2.9 unsigned int or_connection_t::is_bad_for_new_circs

True iff this connection shouldn't get any new circs attached to it, because the connection is too old, or because there's a better one, etc.

Referenced by circuit_build_failed(), connection_or_get_for_extend(), connection_or_group_set_badness(), and run_connection_housekeeping().

3.75.2.10 uint8_t or_connection_t::link_proto

What protocol version are we using? 0 for "none negotiated yet."

Referenced by append_cell_to_circuit_queue(), command_process_netinfo_cell(), command_process_versions_cell(), connection_fetch_var_cell_from_buf(), connection_or_get_for_extend(), and connection_tls_finish_handshake().

3.75.2.11 circid_t or_connection_t::next_circ_id

Which circ_id do we try to use next on this connection? This is always in the range 0..1<<15-1.

Referenced by get_unique_circ_id_by_conn(), and or_connection_new().

3.75.2.12 or_handshake_state_t* or_connection_t::handshake_state

If we are setting this connection up, state information to do so.

Referenced by _connection_free(), command_process_netinfo_cell(), command_process_versions_cell(), connection_init_or_handshake_state(), connection_or_nonopen_was_started_here(), connection_or_send_versions(), and connection_or_set_state_open().

3.75.2.13 time_t or_connection_t::timestamp_lastempty

When was the outbuf last completely empty?

Referenced by run_connection_housekeeping().

3.75.2.14 int or_connection_t::bandwidthrate

When did we last add a non-padding cell to the outbuf? Bytes/s added to the bucket. (OPEN ORs only.)

Referenced by connection_bucket_refill(), and connection_or_init_conn_from_address().

3.75.2.15 int or_connection_t::bandwidthburst

Max bucket size for this conn. (OPEN ORs only.)

Referenced by connection_bucket_refill(), connection_bucket_should_increase(), and connection_or_init_conn_from_address().

3.75.2.16 int or_connection_t::read_bucket

When this hits 0, stop receiving. Every second we add 'bandwidthrate' to this, capping it at bandwidthburst. (OPEN ORs only)

Referenced by connection_bucket_read_limit(), connection_bucket_refill(), connection_buckets_decrement(), and connection_or_init_conn_from_address().

3.75.2.17 int or_connection_t::write_bucket

When this hits 0, stop writing. Like read_bucket.

Referenced by connection_bucket_refill(), connection_bucket_write_limit(), connection_buckets_decrement(), and connection_or_init_conn_from_address().

3.75.2.18 int or_connection_t::n_circuits

How many circuits use this connection as p_conn or n_conn ?

Referenced by connection_or_is_better(), control_event_or_conn_status(), HT_HEAD(), and run_connection_housekeeping().

3.75.2.19 struct circuit_t* or_connection_t::active_circuits `[read]`

Double-linked ring of circuits with queued cells waiting for room to free up on this connection's outbuf. Every time we pull cells from a circuit, we advance this pointer to the next circuit in the ring.

Referenced by assert_active_circuits_ok(), connection_or_flush_from_first_active_circuit(), connection_or_flushed_some(), connection_or_unlink_all_active_circs(), make_circuit_active_on_conn(), and make_circuit_inactive_on_conn().

3.75.2.20 smartlist_t* or_connection_t::active_circuit_pqueue

Priority queue of **cell_ewma_t** (p. 27) for circuits with queued cells waiting for room to free up on this connection's outbuf. Kept in heap order according to EWMA.

This is redundant with active_circuits; if we ever decide only to use the cell_ewma algorithm for choosing circuits, we can remove active_circuits.

Referenced by _connection_free(), add_cell_ewma_to_conn(), assert_active_circuits_ok(), connection_or_flush_from_first_active_circuit(), connection_or_unlink_all_active_circs(), or_connection_new(), pop_first_cell_ewma_from_conn(), remove_cell_ewma_from_conn(), and scale_active_circuits().

3.75.2.21 unsigned or_connection_t::active_circuit_pqueue_last_recalibrated

The tick on which the cell_ewma_ts in active_circuit_pqueue last had their ewma values rescaled.

Referenced by add_cell_ewma_to_conn(), connection_or_flush_from_first_active_circuit(), or_connection_new(), and scale_active_circuits().

3.75.2.22 struct or_connection_t* or_connection_t::next_with_same_id `[read]`

Next connection with same identity digest as this one.

Referenced by connection_or_get_for_extend(), connection_or_group_set_badness(), connection_or_remove_from_identity_map(), and connection_or_set_identity_digest().

The documentation for this struct was generated from the following file:

- **or.h**

3.76 or_handshake_state_t Struct Reference

`#include <or.h>`

Data Fields

- time_t **sent_versions_at**
- unsigned int **started_here**: 1
- unsigned int **received_versions**: 1

3.76.1 Detailed Description

Stores flags and information related to the portion of a v2 Tor OR connection handshake that happens after the TLS handshake is finished.

3.76.2 Field Documentation

3.76.2.1 time_t or_handshake_state_t::sent_versions_at

When was the VERSIONS cell sent on this connection? Used to get an estimate of the skew in the returning NETINFO reply.

Referenced by command_process_netinfo_cell(), and connection_or_send_versions().

3.76.2.2 unsigned int or_handshake_state_t::started_here

True iff we originated this connection

Referenced by connection_init_or_handshake_state(), and connection_or_nonopen_was_started_here().

3.76.2.3 unsigned int or_handshake_state_t::received_versions

True iff we have received and processed a VERSIONS cell.

Referenced by command_process_netinfo_cell(), and command_process_versions_cell().

The documentation for this struct was generated from the following file:

- **or.h**

3.77 or_history_t Struct Reference

Data Fields

- time_t **since**
- time_t **changed**
- unsigned long **n_conn_ok**
- unsigned long **n_conn_fail**
- unsigned long **uptime**
- unsigned long **downtime**
- time_t **up_since**
- time_t **down_since**
- unsigned long **weighted_run_length**
- time_t **start_of_run**
- double **total_run_weights**
- time_t **start_of_downtime**
- unsigned long **weighted_uptime**
- unsigned long **total_weighted_time**
- digestmap_t * **link_history_map**

3.77.1 Detailed Description

History of an OR.

3.77.2 Field Documentation

3.77.2.1 time_t or_history_t::since

When did we start tracking this OR?

Referenced by get_or_history().

3.77.2.2 time_t or_history_t::changed

When did we most recently note a change to this OR?

Referenced by get_or_history(), rep_hist_note_connect_failed(), rep_hist_note_connect_succeeded(), rep_hist_note_connection_died(), rep_hist_note_disconnect(), and rep_history_clean().

3.77.2.3 unsigned long or_history_t::n_conn_ok

How many times did we successfully connect?

Referenced by rep_hist_dump_stats(), and rep_hist_note_connect_succeeded().

3.77.2.4 unsigned long or_history_t::n_conn_fail

How many times did we try to connect and fail?

Referenced by rep_hist_dump_stats(), and rep_hist_note_connect_failed().

3.77.2.5 unsigned long or_history_t::uptime

How many seconds have we been connected to this OR before 'up_since'?

Referenced by mark_or_down(), rep_hist_dump_stats(), and update_or_history().

3.77.2.6 unsigned long or_history_t::downtime

How many seconds have we been unable to connect to this OR before 'down_since'?

Referenced by mark_or_up(), rep_hist_dump_stats(), and update_or_history().

3.77.2.7 time_t or_history_t::up_since

If nonzero, we have been connected since this time.

Referenced by mark_or_down(), mark_or_up(), and update_or_history().

3.77.2.8 time_t or_history_t::down_since

If nonzero, we have been unable to connect since this time.

Referenced by mark_or_down(), mark_or_up(), and update_or_history().

3.77.2.9 unsigned long or_history_t::weighted_run_length

Weighted sum total of all times that this router has been online.

Referenced by get_stability(), rep_hist_downrate_old_runs(), rep_hist_format_router_status(), rep_hist_load_mtbf_data(), rep_hist_note_router_unreachable(), and rep_hist_record_mtbf_data().

3.77.2.10 time_t or_history_t::start_of_run

If the router is now online (according to stability-checking rules), when did it come online?

Referenced by get_stability(), get_total_weighted_time(), get_weighted_fractional_uptime(), rep_hist_format_router_status(), rep_hist_load_mtbf_data(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), rep_hist_record_mtbf_data(), and rep_history_clean().

3.77.2.11 double or_history_t::total_run_weights

Sum of weights for runs in weighted_run_length.

Referenced by get_stability(), rep_hist_downrate_old_runs(), rep_hist_format_router_status(), rep_hist_load_mtbf_data(), rep_hist_note_router_unreachable(), rep_hist_record_mtbf_data(), and rep_history_clean().

3.77.2.12 digestmap_t∗ or_history_t::link_history_map

Map from hex OR2 identity digest to a **link_history_t** (p. 106) for the link from this OR to OR2.

Referenced by free_or_history(), get_link_history(), get_or_history(), rep_hist_dump_stats(), and rep_history_clean().

The documentation for this struct was generated from the following file:

- **rephist.c**

3.78 or_options_t Struct Reference

`#include <or.h>`

Public Types

- enum {
  CMD_RUN_TOR = 0, **CMD_LIST_FINGERPRINT**, **CMD_HASH_PASSWORD**, **CMD_VERIFY_CONFIG**,
  CMD_RUN_UNITTESTS }
- enum { **SAFELOG_SCRUB_ALL**, **SAFELOG_SCRUB_RELAY**, **SAFELOG_SCRUB_NONE** }

Data Fields

- uint32_t **_magic**
- enum or_options_t:: { ... } **command**
- const char ∗ **command_arg**
- **config_line_t** ∗ **Logs**
- char ∗ **DebugLogFile**
- char ∗ **DataDirectory**
- char ∗ **Nickname**
- char ∗ **Address**
- char ∗ **PidFile**
- **routerset_t** ∗ **ExitNodes**
- **routerset_t** ∗ **EntryNodes**
- int **StrictNodes**
- **routerset_t** ∗ **ExcludeNodes**
- **routerset_t** ∗ **ExcludeExitNodes**
- struct **routerset_t** ∗ **_ExcludeExitNodesUnion**
- int **DisableAllSwap**
- **smartlist_t** ∗ **AllowInvalidNodes**
- **invalid_router_usage_t** **_AllowInvalid**
- **config_line_t** ∗ **ExitPolicy**
- int **ExitPolicyRejectPrivate**
- **config_line_t** ∗ **SocksPolicy**
- **config_line_t** ∗ **DirPolicy**
- **config_line_t** ∗ **SocksListenAddress**
- **config_line_t** ∗ **TransListenAddress**
- **config_line_t** ∗ **NatdListenAddress**
- **config_line_t** ∗ **DNSListenAddress**
- **config_line_t** ∗ **ORListenAddress**
- **config_line_t** ∗ **DirListenAddress**
- **config_line_t** ∗ **ControlListenAddress**
- char ∗ **OutboundBindAddress**
- **config_line_t** ∗ **RecommendedVersions**
- **config_line_t** ∗ **RecommendedClientVersions**
- **config_line_t** ∗ **RecommendedServerVersions**
- int **DirAllowPrivateAddresses**

- char * **User**
- char * **Group**
- int **ORPort**
- int **SocksPort**
- int **TransPort**
- int **NatdPort**
- int **ControlPort**
- **config_line_t** * **ControlSocket**
- int **DirPort**
- int **DNSPort**
- int **AssumeReachable**
- int **AuthoritativeDir**
- int **V1AuthoritativeDir**
- int **V2AuthoritativeDir**
- int **V3AuthoritativeDir**
- int **HSAuthoritativeDir**
- int **NamingAuthoritativeDir**
- int **VersioningAuthoritativeDir**
- int **BridgeAuthoritativeDir**
- char * **BridgePassword**
- int **UseBridges**
- **config_line_t** * **Bridges**
- int **BridgeRelay**
- int **UpdateBridgesFromAuthority**
- int **AvoidDiskWrites**
- int **ClientOnly**
- int **NoPublish**
- **smartlist_t** * **PublishServerDescriptor**
- **authority_type_t** **_PublishServerDescriptor**
- int **PublishHidServDescriptors**
- int **FetchServerDescriptors**
- int **FetchHidServDescriptors**
- int **HidServDirectoryV2**
- int **MinUptimeHidServDirectoryV2**
- int **FetchUselessDescriptors**
- int **AllDirActionsPrivate**
- int **ConnLimit**
- int **_ConnLimit**
- int **RunAsDaemon**
- int **FascistFirewall**
- **smartlist_t** * **FirewallPorts**
- **config_line_t** * **ReachableAddresses**
- **config_line_t** * **ReachableORAddresses**
- **config_line_t** * **ReachableDirAddresses**
- int **ConstrainedSockets**
- uint64_t **ConstrainedSockSize**
- int **RefuseUnknownExits**
- **smartlist_t** * **LongLivedPorts**
- **smartlist_t** * **RejectPlaintextPorts**
- **smartlist_t** * **WarnPlaintextPorts**

- **smartlist_t** ∗ **TrackHostExits**
- int **TrackHostExitsExpire**
- **config_line_t** ∗ **AddressMap**
- int **AutomapHostsOnResolve**
- **smartlist_t** ∗ **AutomapHostsSuffixes**
- int **RendPostPeriod**
- int **KeepalivePeriod**
- int **SocksTimeout**
- int **CircuitBuildTimeout**
- int **CircuitIdleTimeout**
- int **CircuitStreamTimeout**
- int **MaxOnionsPending**
- int **NewCircuitPeriod**
- int **MaxCircuitDirtiness**
- uint64_t **BandwidthRate**
- uint64_t **BandwidthBurst**
- uint64_t **MaxAdvertisedBandwidth**
- uint64_t **RelayBandwidthRate**
- uint64_t **RelayBandwidthBurst**
- uint64_t **PerConnBWRate**
- uint64_t **PerConnBWBurst**
- int **NumCpus**
- int **RunTesting**
- **config_line_t** ∗ **RendConfigLines**
- **config_line_t** ∗ **HidServAuth**
- char ∗ **ContactInfo**
- char ∗ **HttpProxy**
- **tor_addr_t** **HttpProxyAddr**
- uint16_t **HttpProxyPort**
- char ∗ **HttpProxyAuthenticator**
- char ∗ **HttpsProxy**
- **tor_addr_t** **HttpsProxyAddr**
- uint16_t **HttpsProxyPort**
- char ∗ **HttpsProxyAuthenticator**
- char ∗ **Socks4Proxy**
- **tor_addr_t** **Socks4ProxyAddr**
- uint16_t **Socks4ProxyPort**
- char ∗ **Socks5Proxy**
- **tor_addr_t** **Socks5ProxyAddr**
- uint16_t **Socks5ProxyPort**
- char ∗ **Socks5ProxyUsername**
- char ∗ **Socks5ProxyPassword**
- **config_line_t** ∗ **DirServers**
- **config_line_t** ∗ **AlternateDirAuthority**
- **config_line_t** ∗ **AlternateBridgeAuthority**
- **config_line_t** ∗ **AlternateHSAuthority**
- char ∗ **MyFamily**
- **config_line_t** ∗ **NodeFamilies**
- **config_line_t** ∗ **AuthDirBadDir**
- **config_line_t** ∗ **AuthDirBadExit**

- **config_line_t** ∗ **AuthDirReject**
- **config_line_t** ∗ **AuthDirInvalid**
- int **AuthDirListBadDirs**
- int **AuthDirListBadExits**
- int **AuthDirRejectUnlisted**
- int **AuthDirMaxServersPerAddr**
- int **AuthDirMaxServersPerAuthAddr**
- char ∗ **AccountingStart**
- uint64_t **AccountingMax**
- **config_line_t** ∗ **HashedControlPassword**
- **config_line_t** ∗ **HashedControlSessionPassword**
- int **CookieAuthentication**
- char ∗ **CookieAuthFile**
- int **CookieAuthFileGroupReadable**
- int **LeaveStreamsUnattached**
- int **DisablePredictedCircuits**
- int **ShutdownWaitLength**
- char ∗ **SafeLogging**
- enum or_options_t:: { ... } **_SafeLogging**
- int **SafeSocks**
- int **ProtocolWarnings**
- int **TestSocks**
- int **HardwareAccel**
- char ∗ **AccelName**
- char ∗ **AccelDir**
- int **UseEntryGuards**
- int **NumEntryGuards**
- int **RephistTrackTime**
- int **FastFirstHopPK**
- int **FetchDirInfoEarly**
- int **FetchDirInfoExtraEarly**
- char ∗ **VirtualAddrNetwork**
- int **ServerDNSSearchDomains**
- int **ServerDNSDetectHijacking**
- int **ServerDNSRandomizeCase**
- char ∗ **ServerDNSResolvConfFile**
- char ∗ **DirPortFrontPage**
- int **ServerDNSAllowBrokenConfig**
- **smartlist_t** ∗ **ServerDNSTestAddresses**
- int **EnforceDistinctSubnets**
- int **TunnelDirConns**
- int **PreferTunneledDirConns**
- int **AllowNonRFC953Hostnames**
- int **ServerDNSAllowNonRFC953Hostnames**
- int **DownloadExtraInfo**
- int **AllowSingleHopExits**
- int **ExcludeSingleHopRelays**
- int **AllowSingleHopCircuits**
- int **AllowDotExit**
- int **DirReqStatistics**

- int **ExitPortStatistics**
- int **CellStatistics**
- int **EntryStatistics**
- int **ExtraInfoStatistics**
- int **ClientDNSRejectInternalAddresses**
- int **V3AuthVotingInterval**
- int **V3AuthVoteDelay**
- int **V3AuthDistDelay**
- int **V3AuthNIntervalsValid**
- int **V3AuthUseLegacyKey**
- char * **V3BandwidthsFile**
- char * **ConsensusParams**
- int **TestingV3AuthInitialVotingInterval**
- int **TestingV3AuthInitialVoteDelay**
- int **TestingV3AuthInitialDistDelay**
- int **TestingAuthDirTimeToLearnReachability**
- int **TestingEstimatedDescriptorPropagationTime**
- int **TestingTorNetwork**
- char * **FallbackNetworkstatusFile**
- int **BridgeRecordUsageByCountry**
- char * **GeoIPFile**
- int **ReloadTorrcOnSIGHUP**
- double **CircuitPriorityHalflife**

3.78.1 Detailed Description

Configuration options for a Tor process.

3.78.2 Member Enumeration Documentation

3.78.2.1 anonymous enum

What should the tor process actually do?

3.78.3 Field Documentation

3.78.3.1 enum { ... } or_options_t::command

What should the tor process actually do?

Referenced by init_keys(), options_act(), options_act_reversible(), options_init_from_string(), and tor_cleanup().

3.78.3.2 const char* or_options_t::command_arg

Argument for command-line option.

Referenced by options_init_from_string().

3.78.3.3 config_line_t* or_options_t::Logs

New-style list of configuration lines for logs

Referenced by options_init_logs(), options_transition_affects_workers(), and options_validate().

3.78.3.4 char* or_options_t::DebugLogFile

Where to send verbose log messages.

3.78.3.5 char* or_options_t::DataDirectory

OR only: where to store long-term data.

Referenced by init_keys(), networkstatus_dump_bridge_status_to_file(), normalize_data_directory(), options_act(), options_act_reversible(), options_get_datadir_fname2_suffix(), options_transition_affects_descriptor(), options_transition_affects_workers(), options_transition_allowed(), and validate_data_directory().

3.78.3.6 char* or_options_t::Nickname

OR only: nickname of this onion router.

Referenced by do_list_fingerprint(), init_keys(), options_transition_affects_descriptor(), options_validate(), and router_rebuild_descriptor().

3.78.3.7 char* or_options_t::Address

OR only: configured address for this onion router.

Referenced by directory_command_should_use_begindir(), options_transition_affects_descriptor(), resolve_my_address(), and update_router_descriptor_downloads().

3.78.3.8 char* or_options_t::PidFile

Where to store PID of Tor process.

Referenced by options_act(), options_transition_allowed(), and tor_cleanup().

3.78.3.9 routerset_t* or_options_t::ExitNodes

Structure containing nicknames, digests, country codes and IP address patterns of ORs to consider as exits.

Referenced by choose_good_exit_server_general(), options_act(), options_need_geoip_info(), and refresh_all_country_info().

3.78.3.10 routerset_t* or_options_t::EntryNodes

Structure containing nicknames, digests, country codes and IP address patterns of ORs to consider as entry points.

Referenced by choose_random_entry(), entry_guard_set_status(), entry_guards_compute_status(), entry_guards_prepend_from_config(), entry_is_live(), entry_list_is_constrained(), options_act(), options_need_geoip_info(), options_validate(), and refresh_all_country_info().

3.78.3.11 int or_options_t::StrictNodes

Boolean: When none of our EntryNodes or ExitNodes are up, or we need to access a node in ExcludeNodes, do we just fail instead?

Referenced by choose_good_exit_server_general(), choose_random_entry(), connection_ap_can_use_exit(), options_act(), and options_validate().

3.78.3.12 routerset_t∗ or_options_t::ExcludeNodes

Structure containing nicknames, digests, country codes and IP address patterns of ORs not to use in circuits. But see StrictNodes above.

Referenced by choose_good_exit_server(), choose_good_middle_server(), circuit_conforms_to_options(), entry_guard_set_status(), options_act(), options_need_geoip_info(), options_validate(), refresh_all_country_info(), and warn_if_last_router_excluded().

3.78.3.13 routerset_t∗ or_options_t::ExcludeExitNodes

Structure containing nicknames, digests, country codes and IP address patterns of ORs not to consider as exits.

Referenced by circuit_conforms_to_options(), options_act(), options_need_geoip_info(), options_validate(), and refresh_all_country_info().

3.78.3.14 struct routerset_t∗ or_options_t::_ExcludeExitNodesUnion `[read]`

Union of ExcludeNodes and ExcludeExitNodes

Referenced by choose_good_exit_server_general(), connection_ap_can_use_exit(), options_validate(), or_options_free(), refresh_all_country_info(), and warn_if_last_router_excluded().

3.78.3.15 int or_options_t::DisableAllSwap

Boolean: Attempt to call mlockall() on our process for all current and future memory.

Referenced by options_act_reversible(), and options_transition_allowed().

3.78.3.16 smartlist_t∗ or_options_t::AllowInvalidNodes

List of "entry", "middle", "exit", "introduction", "rendezvous".

3.78.3.17 invalid_router_usage_t or_options_t::_AllowInvalid

Bitmask; derived from AllowInvalidNodes.

Referenced by choose_good_exit_server(), choose_good_exit_server_general(), and choose_good_middle_server().

3.78.3.18 config_line_t∗ or_options_t::ExitPolicy

Lists of exit policy components.

Referenced by options_transition_affects_descriptor(), router_rebuild_descriptor(), and validate_addr_policies().

3.78.3.19 int or_options_t::ExitPolicyRejectPrivate

Should we not exit to local addresses?

Referenced by options_transition_affects_descriptor(), router_rebuild_descriptor(), and validate_addr_policies().

3.78.3.20 config_line_t∗ or_options_t::SocksPolicy

Lists of socks policy components

Referenced by policies_parse_from_options(), and validate_addr_policies().

3.78.3.21 config_line_t∗ or_options_t::DirPolicy

Lists of dir policy components

Referenced by policies_parse_from_options(), and validate_addr_policies().

3.78.3.22 config_line_t∗ or_options_t::SocksListenAddress

Addresses to bind for listening for SOCKS connections.

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.23 config_line_t∗ or_options_t::TransListenAddress

Addresses to bind for listening for transparent pf/netfilter connections.

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.24 config_line_t∗ or_options_t::NatdListenAddress

Addresses to bind for listening for transparent natd connections

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.25 config_line_t∗ or_options_t::DNSListenAddress

Addresses to bind for listening for SOCKS connections.

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.26 config_line_t∗ or_options_t::ORListenAddress

Addresses to bind for listening for OR connections.

Referenced by options_validate(), retry_all_listeners(), and server_mode().

3.78.3.27 config_line_t∗ or_options_t::DirListenAddress

Addresses to bind for listening for directory connections.

Referenced by options_validate(), and retry_all_listeners().

3.78.3.28 config_line_t∗ or_options_t::ControlListenAddress

Addresses to bind for listening for control connections.

Referenced by options_validate(), and retry_all_listeners().

3.78.3.29 char∗ or_options_t::OutboundBindAddress

Local address to bind outbound sockets

Referenced by configure_nameservers(), and connection_connect().

3.78.3.30 config_line_t∗ or_options_t::RecommendedVersions

Directory server only: which versions of Tor should we tell users to run?

Referenced by options_validate().

3.78.3.31 int or_options_t::DirAllowPrivateAddresses

Whether dirservers refuse router descriptors with private IPs.

3.78.3.32 char∗ or_options_t::User

Name of user to run Tor as.

Referenced by options_act_reversible(), and options_transition_allowed().

3.78.3.33 char∗ or_options_t::Group

Name of group to run Tor as.

Referenced by options_transition_allowed().

3.78.3.34 int or_options_t::ORPort

Port to listen on for OR connections.

Referenced by init_keys(), options_transition_affects_descriptor(), options_transition_affects_workers(), options_validate(), retry_all_listeners(), router_rebuild_descriptor(), and server_mode().

3.78.3.35 int or_options_t::SocksPort

Port to listen on for SOCKS connections.

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.36 int or_options_t::TransPort

Port to listen on for transparent pf/netfilter connections.

Referenced by options_act_reversible(), options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.37 int or_options_t::NatdPort

Port to listen on for transparent natd connections.

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.38 int or_options_t::ControlPort

Port to listen on for control connections.

Referenced by options_validate(), and retry_all_listeners().

3.78.3.39 config_line_t* or_options_t::ControlSocket

List of Unix Domain Sockets to listen on for control connections.

Referenced by options_act_reversible(), and retry_all_listeners().

3.78.3.40 int or_options_t::DirPort

Port to listen on for directory connections.

Referenced by check_whether_dirport_reachable(), directory_caches_dir_info(), directory_caches_v2_dir_info(), directory_fetches_from_authorities(), directory_permits_begindir_requests(), directory_permits_controller_requests(), generate_v2_networkstatus_opinion(), init_keys(), options_transition_affects_descriptor(), options_validate(), retry_all_listeners(), and router_rebuild_descriptor().

3.78.3.41 int or_options_t::DNSPort

Port to listen on for DNS requests.

Referenced by options_validate(), proxy_mode(), and retry_all_listeners().

3.78.3.42 int or_options_t::AssumeReachable

Whether to publish our descriptor regardless.

Referenced by check_whether_dirport_reachable(), check_whether_orport_reachable(), and dirserv_set_router_is_running().

3.78.3.43 int or_options_t::AuthoritativeDir

Boolean: is this an authoritative directory?

Referenced by authdir_mode(), and options_validate().

3.78.3.44 int or_options_t::V1AuthoritativeDir

Boolean: is this an authoritative directory for version 1 directories?

Referenced by authdir_mode_any_main(), authdir_mode_v1(), init_keys(), and options_validate().

3.78.3.45 int or_options_t::V2AuthoritativeDir

Boolean: is this an authoritative directory for version 2 directories?

Referenced by authdir_mode_any_main(), authdir_mode_v2(), init_keys(), and options_validate().

3.78.3.46 int or_options_t::V3AuthoritativeDir

Boolean: is this an authoritative directory for version 3 directories?

Referenced by authdir_mode_any_main(), authdir_mode_v3(), init_keys(), options_act(), and options_validate().

3.78.3.47 int or_options_t::HSAuthoritativeDir

Boolean: does this an authoritative directory handle hidden service requests?

Referenced by directory_handle_command_get(), directory_handle_command_post(), init_keys(), and options_validate().

3.78.3.48 int or_options_t::NamingAuthoritativeDir

Boolean: is this an authoritative directory that's willing to bind names?

Referenced by dirserv_generate_networkstatus_vote_obj(), dirserv_load_fingerprint_file(), generate_v2_networkstatus_opinion(), and routers_update_status_from_consensus_networkstatus().

3.78.3.49 int or_options_t::VersioningAuthoritativeDir

Boolean: is this an authoritative directory that's willing to recommend versions?

Referenced by dirserv_generate_networkstatus_vote_obj(), generate_v2_networkstatus_opinion(), and options_validate().

3.78.3.50 int or_options_t::BridgeAuthoritativeDir

Boolean: is this an authoritative directory that aggregates bridge descriptors?

Referenced by authdir_mode_any_nonhidserv(), authdir_mode_bridge(), authdir_mode_handles_descs(), directory_handle_command_get(), geoip_note_client_seen(), init_keys(), options_validate(), and run_scheduled_events().

3.78.3.51 char∗ or_options_t::BridgePassword

If set on a bridge authority, it will answer requests on its dirport for bridge statuses – but only if the requests use this password. If set on a bridge user, request bridge statuses, and use this password when doing so.

Referenced by directory_handle_command_get().

3.78.3.52 int or_options_t::UseBridges

Boolean: should we start all circuits with a bridge?

Referenced by circuit_get_open_circ_or_launch(), directory_fetches_dir_info_later(), directory_get_from_dirserver(), entry_guard_set_status(), entry_list_is_constrained(), run_scheduled_events(), and should_delay_dir_fetches().

3.78.3.53 config_line_t∗ or_options_t::Bridges

List of bootstrap bridge addresses.

Referenced by options_act().

3.78.3.54 int or_options_t::BridgeRelay

Boolean: are we acting as a bridge relay? We make this explicit so we can change how we behave in the future.

Referenced by choose_good_entry_server(), compute_publishserverdescriptor(), directory_caches_dir_info(), directory_fetches_from_authorities(), directory_handle_command_get(), directory_permits_begindir_requests(), geoip_note_client_seen(), options_act(), options_need_geoip_info(), router_rebuild_descriptor(), should_record_bridge_info(), and validate_addr_policies().

3.78.3.55 int or_options_t::UpdateBridgesFromAuthority

Boolean: if we know the bridge's digest, should we get new descriptors from the bridge authorities or from the bridge itself?

Referenced by fetch_bridge_descriptors().

3.78.3.56 int or_options_t::AvoidDiskWrites

Boolean: should we never cache things to disk? Not used yet.

Referenced by circuit_build_times_add_time(), circuit_build_times_update_state(), entry_guards_changed(), entry_guards_update_state(), init_keys(), and rep_hist_update_state().

3.78.3.57 int or_options_t::ClientOnly

Boolean: should we never evolve into a server role?

Referenced by decide_if_publishable_server(), options_transition_affects_descriptor(), options_transition_affects_workers(), options_validate(), retry_all_listeners(), and server_mode().

3.78.3.58 int or_options_t::NoPublish

Boolean: should we never publish a descriptor? Deprecated.

Referenced by options_transition_affects_descriptor(), and options_validate().

3.78.3.59 smartlist_t* or_options_t::PublishServerDescriptor

To what authority types do we publish our descriptor? Choices are "v1", "v2", "v3", "bridge", or "".

Referenced by compute_publishserverdescriptor(), and options_validate().

3.78.3.60 authority_type_t or_options_t::_PublishServerDescriptor

An authority type, derived from PublishServerDescriptor.

Referenced by compute_publishserverdescriptor(), decide_if_publishable_server(), directories_have_accepted_server_descriptor(), options_transition_affects_descriptor(), resolve_my_address(), and router_upload_dir_desc_to_dirservers().

3.78.3.61 int or_options_t::PublishHidServDescriptors

Boolean: do we publish hidden service descriptors to the HS auths?

3.78.3.62 int or_options_t::FetchServerDescriptors

Do we fetch server descriptors as normal?

Referenced by directory_get_from_dirserver().

3.78.3.63 int or_options_t::HidServDirectoryV2

and hidden service descriptors? Do we participate in the HS DHT?

Referenced by directory_handle_command_get(), directory_handle_command_post(), and router_dump_router_to_string().

3.78.3.64 int or_options_t::MinUptimeHidServDirectoryV2

As directory authority, accept hidden service directories after what time?

3.78.3.65 int or_options_t::FetchUselessDescriptors

Do we fetch non-running descriptors too?

Referenced by client_would_use_router(), and directory_too_idle_to_fetch_descriptors().

3.78.3.66 int or_options_t::AllDirActionsPrivate

Should every directory action be sent through a Tor circuit?

3.78.3.67 int or_options_t::ConnLimit

Demanded minimum number of simultaneous connections.

Referenced by options_act_reversible(), and options_validate().

3.78.3.68 int or_options_t::_ConnLimit

Maximum allowed number of simultaneous connections.

Referenced by options_act_reversible().

3.78.3.69 int or_options_t::RunAsDaemon

If true, run in the background. (Unix only)

Referenced by options_act(), options_act_reversible(), options_init_logs(), options_transition_allowed(), and options_validate().

3.78.3.70 int or_options_t::FascistFirewall

Whether to prefer ORs reachable on open ports.

Referenced by options_validate().

3.78.3.71 smartlist_t* or_options_t::FirewallPorts

Which ports our firewall allows (strings).

Referenced by options_validate().

3.78.3.72 config_line_t* or_options_t::ReachableAddresses

IP:ports our firewall allows.

Referenced by options_validate(), parse_reachable_addresses(), and validate_addr_policies().

3.78.3.73 config_line_t* or_options_t::ReachableORAddresses

IP:ports for OR conns.

Referenced by options_validate(), parse_reachable_addresses(), and validate_addr_policies().

3.78.3.74 config_line_t* or_options_t::ReachableDirAddresses

IP:ports for Dir conns.

Referenced by options_validate(), parse_reachable_addresses(), and validate_addr_policies().

3.78.3.75 int or_options_t::ConstrainedSockets

Shrink xmit and recv socket buffers.

Referenced by connection_connect(), and connection_handle_listener_read().

3.78.3.76 uint64_t or_options_t::ConstrainedSockSize

Size of constrained buffers.

Referenced by connection_connect(), and connection_handle_listener_read().

3.78.3.77 int or_options_t::RefuseUnknownExits

Whether we should drop exit streams from Tors that we don't know are relays. XXX022 In here for 0.2.2.11 as a temporary test before we switch over to putting it in consensusparams. -RD

Referenced by directory_caches_dir_info(), and directory_fetches_from_authorities().

3.78.3.78 smartlist_t* or_options_t::LongLivedPorts

Application ports that require all nodes in circ to have sufficient uptime.

Referenced by circuit_all_predicted_ports_handled(), circuit_get_open_circ_or_launch(), and options_validate().

3.78.3.79 smartlist_t* or_options_t::RejectPlaintextPorts

Application ports that are likely to be unencrypted and unauthenticated; we reject requests for them to prevent the user from screwing up and leaking plaintext secrets to an observer somewhere on the Internet.

Referenced by consider_plaintext_ports(), and options_validate().

3.78.3.80 smartlist_t* or_options_t::WarnPlaintextPorts

Related to RejectPlaintextPorts above, except this config option controls whether we warn (in the log and via a controller status event) every time a risky connection is attempted.

Referenced by consider_plaintext_ports(), and options_validate().

3.78.3.81 smartlist_t* or_options_t::TrackHostExits

Should we try to reuse the same exit node for a given host

Referenced by consider_recording_trackhost().

3.78.3.82 int or_options_t::TrackHostExitsExpire

Number of seconds until we expire an addressmap

Referenced by consider_recording_trackhost().

3.78.3.83 config_line_t* or_options_t::AddressMap

List of address map directives.

Referenced by config_register_addressmaps().

3.78.3.84 int or_options_t::AutomapHostsOnResolve

If true, when we get a resolve request for a hostname ending with one of the suffixes in **AutomapHostsSuffixes**, map it to a virtual address.

Referenced by connection_ap_handshake_rewrite_and_attach().

3.78.3.85 smartlist_t* or_options_t::AutomapHostsSuffixes

List of suffixes for **AutomapHostsOnResolve**.

Referenced by connection_ap_handshake_rewrite_and_attach().

3.78.3.86 int or_options_t::RendPostPeriod

How often do we post each rendezvous service descriptor? Remember to publish them independently.

Referenced by rend_consider_services_upload(), and upload_service_descriptor().

3.78.3.87 int or_options_t::KeepalivePeriod

How often do we send padding cells to keep connections alive?

Referenced by run_connection_housekeeping().

3.78.3.88 int or_options_t::SocksTimeout

How long do we let a socks connection wait unattached before we fail it?

Referenced by connection_ap_expire_beginning(), and connection_ap_handshake_attach_-circuit().

3.78.3.89 int or_options_t::CircuitBuildTimeout

If non-zero, cull non-open circuits that were born at least this many seconds ago. If zero, use the internal adaptive algorithm.

Referenced by circuit_build_times_get_initial_timeout().

3.78.3.90 int or_options_t::CircuitIdleTimeout

Cull open clean circuits that were born at least this many seconds ago.

Referenced by circuit_expire_old_circuits_clientside().

3.78.3.91 int or_options_t::CircuitStreamTimeout

If non-zero, detach streams from circuits and try a new circuit if the stream has been waiting for this many seconds. If zero, use our default internal timeout schedule.

Referenced by compute_retry_timeout().

3.78.3.92 int or_options_t::MaxOnionsPending

How many circuit CREATE requests do we allow to wait simultaneously before we start dropping them?

3.78.3.93 int or_options_t::NewCircuitPeriod

How long do we use a circuit before building a new one?

Referenced by circuit_build_needed_circs().

3.78.3.94 int or_options_t::MaxCircuitDirtiness

Never use circs that were first used more than this interval ago.

Referenced by circuit_expire_all_dirty_circs(), connection_ap_expire_beginning(), and connection_ap_process_end_not_open().

3.78.3.95 uint64_t or_options_t::BandwidthRate

How much bandwidth, on average, are we willing to use in a second?

Referenced by connection_bucket_refill(), connection_or_init_conn_from_address(), decide_to_advertise_dirport(), get_effective_bwrate(), global_write_bucket_low(), router_perform_bandwidth_test(), and update_expected_bandwidth().

3.78.3.96 uint64_t or_options_t::BandwidthBurst

How much bandwidth, at maximum, are we willing to use in a second?

Referenced by connection_bucket_init(), connection_bucket_refill(), connection_or_init_conn_from_address(), and get_effective_bwburst().

3.78.3.97 uint64_t or_options_t::MaxAdvertisedBandwidth

How much bandwidth are we willing to tell people we have?

Referenced by get_effective_bwrate().

3.78.3.98 uint64_t or_options_t::RelayBandwidthRate

How much bandwidth, on average, are we willing to use for all relayed conns?

Referenced by connection_bucket_init(), connection_bucket_refill(), decide_to_advertise_dirport(), get_effective_bwrate(), global_write_bucket_low(), and rep_hist_fill_bandwidth_history().

3.78.3.99 uint64_t or_options_t::RelayBandwidthBurst

How much bandwidth, at maximum, will we use in a second for all relayed conns?

Referenced by connection_bucket_init(), connection_bucket_refill(), and get_effective_bwburst().

3.78.3.100 uint64_t or_options_t::PerConnBWRate

Long-term bw on a single TLS conn, if set.

Referenced by connection_or_init_conn_from_address().

3.78.3.101 uint64_t or_options_t::PerConnBWBurst

Allowed burst on a single TLS conn, if set.

Referenced by connection_or_init_conn_from_address().

3.78.3.102 int or_options_t::NumCpus

How many CPUs should we try to use?

Referenced by options_transition_affects_workers(), and spawn_enough_cpuworkers().

3.78.3.103 int or_options_t::RunTesting

If true, create testing circuits to measure how well the other ORs are running.

3.78.3.104 config_line_t* or_options_t::RendConfigLines

List of configuration lines for rendezvous services.

Referenced by options_validate(), and rend_config_services().

3.78.3.105 config_line_t* or_options_t::HidServAuth

List of configuration lines for client-side authorizations for hidden services

Referenced by rend_parse_service_authorization().

3.78.3.106 char* or_options_t::ContactInfo

Contact info to be published in the directory.

Referenced by dirserv_generate_networkstatus_vote_obj(), generate_v2_networkstatus_opinion(), options_transition_affects_descriptor(), options_validate(), and router_dump_router_to_string().

3.78.3.107 char∗ or_options_t::HttpProxy

hostname[:port] to use as http proxy, if any.

Referenced by directory_initiate_command_rend().

3.78.3.108 tor_addr_t or_options_t::HttpProxyAddr

Parsed IPv4 addr for http proxy, if any.

Referenced by directory_initiate_command_rend().

3.78.3.109 uint16_t or_options_t::HttpProxyPort

Parsed port for http proxy, if any.

Referenced by directory_initiate_command_rend().

3.78.3.110 char∗ or_options_t::HttpProxyAuthenticator

username:password string, if any.

Referenced by directory_send_command().

3.78.3.111 char∗ or_options_t::HttpsProxy

hostname[:port] to use as https proxy, if any.

Referenced by connection_about_to_close_connection(), and connection_or_connect().

3.78.3.112 tor_addr_t or_options_t::HttpsProxyAddr

Parsed addr for https proxy, if any.

Referenced by connection_or_connect().

3.78.3.113 uint16_t or_options_t::HttpsProxyPort

Parsed port for https proxy, if any.

Referenced by connection_or_connect().

3.78.3.114 char∗ or_options_t::HttpsProxyAuthenticator

username:password string, if any.

Referenced by connection_proxy_connect().

3.78.3.115 char* or_options_t::Socks4Proxy

hostname:port to use as a SOCKS4 proxy, if any.

Referenced by connection_or_connect(), and directory_initiate_command_rend().

3.78.3.116 tor_addr_t or_options_t::Socks4ProxyAddr

Derived from Socks4Proxy.

Referenced by connection_or_connect().

3.78.3.117 uint16_t or_options_t::Socks4ProxyPort

Derived from Socks4Proxy.

Referenced by connection_or_connect().

3.78.3.118 char* or_options_t::Socks5Proxy

hostname:port to use as a SOCKS5 proxy, if any.

Referenced by connection_or_connect(), and directory_initiate_command_rend().

3.78.3.119 tor_addr_t or_options_t::Socks5ProxyAddr

Derived from Sock5Proxy.

Referenced by connection_or_connect().

3.78.3.120 uint16_t or_options_t::Socks5ProxyPort

Derived from Socks5Proxy.

Referenced by connection_or_connect().

3.78.3.121 char* or_options_t::Socks5ProxyUsername

Username for SOCKS5 authentication, if any

Referenced by connection_proxy_connect(), and connection_read_proxy_handshake().

3.78.3.122 char* or_options_t::Socks5ProxyPassword

Password for SOCKS5 authentication, if any

Referenced by connection_read_proxy_handshake().

3.78.3.123 config_line_t* or_options_t::DirServers

List of configuration lines for replacement directory authorities. If you just want to replace one class of authority at a time, use the "Alternate*Authority" options below instead.

Referenced by consider_adding_dir_authorities(), resolve_my_address(), and validate_dir_authorities().

3.78.3.124 config_line_t* or_options_t::AlternateDirAuthority

If set, use these main (currently v3) directory authorities and not the default ones.

Referenced by consider_adding_dir_authorities(), resolve_my_address(), and validate_dir_authorities().

3.78.3.125 config_line_t* or_options_t::AlternateBridgeAuthority

If set, use these bridge authorities and not the default one.

Referenced by consider_adding_dir_authorities(), and validate_dir_authorities().

3.78.3.126 config_line_t* or_options_t::AlternateHSAuthority

If set, use these HS authorities and not the default ones.

Referenced by consider_adding_dir_authorities(), and validate_dir_authorities().

3.78.3.127 char* or_options_t::MyFamily

Declared family for this OR.

Referenced by options_transition_affects_descriptor(), and router_rebuild_descriptor().

3.78.3.128 config_line_t* or_options_t::NodeFamilies

List of config lines for node families

Referenced by routers_in_same_family().

3.78.3.129 config_line_t* or_options_t::AuthDirBadDir

Address policy for descriptors to mark as bad dir mirrors.

Referenced by policies_parse_from_options(), and validate_addr_policies().

3.78.3.130 config_line_t* or_options_t::AuthDirBadExit

Address policy for descriptors to mark as bad exits.

Referenced by policies_parse_from_options(), and validate_addr_policies().

3.78.3.131 config_line_t* or_options_t::AuthDirReject

Address policy for descriptors to reject.

Referenced by policies_parse_from_options(), and validate_addr_policies().

3.78.3.132 config_line_t* or_options_t::AuthDirInvalid

Address policy for descriptors to never mark as valid.

Referenced by policies_parse_from_options(), and validate_addr_policies().

3.78.3.133 int or_options_t::AuthDirListBadDirs

True iff we should list bad dirs, and vote for all other dir mirrors as good.

Referenced by dirserv_generate_networkstatus_vote_obj(), and generate_v2_networkstatus_opinion().

3.78.3.134 int or_options_t::AuthDirListBadExits

True iff we should list bad exits, and vote for all other exits as good.

Referenced by dirserv_generate_networkstatus_vote_obj(), and generate_v2_networkstatus_opinion().

3.78.3.135 int or_options_t::AuthDirRejectUnlisted

Boolean: do we reject all routers that aren't named in our fingerprint file?

Referenced by dirserv_get_status_impl().

3.78.3.136 int or_options_t::AuthDirMaxServersPerAddr

Do not permit more than this number of servers per IP address.

Referenced by get_possible_sybil_list().

3.78.3.137 int or_options_t::AuthDirMaxServersPerAuthAddr

Do not permit more than this number of servers per IP address shared with an authority.

Referenced by get_possible_sybil_list().

3.78.3.138 char* or_options_t::AccountingStart

How long is the accounting interval, and when does it start?

Referenced by accounting_parse_options(), and options_transition_affects_descriptor().

3.78.3.139 uint64_t or_options_t::AccountingMax

How many bytes do we allow per accounting interval before hibernation? 0 for "never hibernate."

Referenced by accounting_is_enabled(), accounting_set_wakeup_time(), consider_hibernation(), getinfo_helper_accounting(), hibernate_hard_limit_reached(), options_transition_affects_descriptor(), and options_validate().

3.78.3.140 config_line_t* or_options_t::HashedControlPassword

Base64-encoded hash of accepted passwords for the control system.

Referenced by handle_control_authenticate(), and handle_control_protocolinfo().

3.78.3.141 config_line_t* or_options_t::HashedControlSessionPassword

As HashedControlPassword, but not saved.

Referenced by handle_control_authenticate(), and handle_control_protocolinfo().

3.78.3.142 int or_options_t::CookieAuthentication

Boolean: do we enable cookie-based auth for the control system?

Referenced by handle_control_authenticate(), handle_control_protocolinfo(), and options_-act().

3.78.3.143 char* or_options_t::CookieAuthFile

Location of a cookie authentication file.

Referenced by get_cookie_file().

3.78.3.144 int or_options_t::CookieAuthFileGroupReadable

Boolean: Is the CookieAuthFile g+r?

3.78.3.145 int or_options_t::LeaveStreamsUnattached

Boolean: Does Tor attach new streams to circuits itself (0), or does it expect a controller to cope? (1)

Referenced by connection_ap_handshake_process_socks(), connection_ap_process_natd(), and connection_ap_process_transparent().

3.78.3.146 int or_options_t::DisablePredictedCircuits

Boolean: does Tor preemptively make circuits in the background (0), or not (1)?

Referenced by circuit_build_needed_circs().

3.78.3.147 int or_options_t::ShutdownWaitLength

When we get a SIGINT and we're a server, how long do we wait before exiting?

Referenced by hibernate_begin().

3.78.3.148 char* or_options_t::SafeLogging

Contains "relay", "1", "0" (meaning no scrubbing).

Referenced by options_transition_affects_workers().

3.78.3.149 int or_options_t::SafeSocks

Boolean: should we outright refuse application connections that use socks4 or socks5-with-local-dns?

Referenced by connection_ap_handshake_process_socks().

3.78.3.150 int or_options_t::ProtocolWarnings

Boolean: when other parties screw up the Tor protocol, is it a warn or an info in our logs?

3.78.3.151 int or_options_t::TestSocks

Boolean: when we get a socks connection, do we loudly log whether it was DNS-leaking or not?

Referenced by connection_ap_handshake_process_socks().

3.78.3.152 int or_options_t::HardwareAccel

Boolean: Should we enable OpenSSL hardware acceleration where available?

Referenced by options_transition_allowed().

3.78.3.153 char∗ or_options_t::AccelName

Optional hardware acceleration engine name.

Referenced by options_transition_allowed().

3.78.3.154 char∗ or_options_t::AccelDir

Optional hardware acceleration engine search dir.

Referenced by options_transition_allowed().

3.78.3.155 int or_options_t::UseEntryGuards

Boolean: Do we try to enter from a smallish number of fixed nodes?

Referenced by choose_good_entry_server(), options_act(), and options_validate().

3.78.3.156 int or_options_t::NumEntryGuards

How many entry guards do we try to establish?

Referenced by choose_random_entry(), control_event_guard_deferred(), and pick_entry_guards().

3.78.3.157 int or_options_t::RephistTrackTime

How many seconds do we keep rephist info?

Referenced by run_scheduled_events().

3.78.3.158 int or_options_t::FastFirstHopPK

If Tor believes it is safe, should we save a third of our PK time by sending CREATE_FAST cells?

Referenced by should_use_create_fast_for_circuit().

3.78.3.159 int or_options_t::FetchDirInfoEarly

Should we always fetch our dir info on the mirror schedule (which means directly from the authorities) no matter our other config?

Referenced by directory_fetches_from_authorities(), and options_validate().

3.78.3.160 int or_options_t::FetchDirInfoExtraEarly

Should we fetch our dir info at the start of the consensus period?

Referenced by options_validate(), and update_consensus_networkstatus_fetch_time().

3.78.3.161 char∗ or_options_t::VirtualAddrNetwork

Address and mask to hand out for virtual MAPADDRESS requests.

Referenced by options_act().

3.78.3.162 int or_options_t::ServerDNSSearchDomains

Boolean: If set, we don't force exit addresses to be FQDNs, but rather search for them in the local domains.

Referenced by launch_resolve(), and options_transition_affects_workers().

3.78.3.163 int or_options_t::ServerDNSDetectHijacking

Boolean: If true, check for DNS failure hijacking.

3.78.3.164 int or_options_t::ServerDNSRandomizeCase

Boolean: Use the 0x20-hack to prevent DNS poisoning attacks.

Referenced by configure_nameservers().

3.78.3.165 char∗ or_options_t::ServerDNSResolvConfFile

If provided, we configure our internal resolver from the file here rather than from /etc/resolv.conf (Unix) or the registry (Windows).

Referenced by configure_nameservers().

3.78.3.166 char∗ or_options_t::DirPortFrontPage

This is a full path to a file with an html disclaimer. This allows a server administrator to show that they're running Tor and anyone visiting their server will know this without any specialized knowledge.

Referenced by options_act().

3.78.3.167 int or_options_t::ServerDNSAllowBrokenConfig

Boolean: if set, we start even if our resolv.conf file is missing or broken.

3.78.3.168 smartlist_t∗ or_options_t::ServerDNSTestAddresses

A list of addresses that definitely should be resolvable. Used for testing our DNS server.

Referenced by add_wildcarded_test_address(), is_test_address(), and launch_test_addresses().

3.78.3.169 int or_options_t::EnforceDistinctSubnets

If true, don't allow multiple routers in the same network zone in the same circuit.

Referenced by routerlist_add_family(), and routers_in_same_family().

3.78.3.170 int or_options_t::TunnelDirConns

If true, use BEGIN_DIR rather than BEGIN when possible.

Referenced by directory_command_should_use_begindir(), and hid_serv_get_responsible_directories().

3.78.3.171 int or_options_t::PreferTunneledDirConns

If true, avoid dirservers that don't support BEGIN_DIR, when possible.

3.78.3.172 int or_options_t::AllowNonRFC953Hostnames

If true, we allow connections to hostnames with weird characters.

3.78.3.173 int or_options_t::ServerDNSAllowNonRFC953Hostnames

If true, we try resolving hostnames with weird characters.

3.78.3.174 int or_options_t::DownloadExtraInfo

If true, we try to download extra-info documents (and we serve them, if we are a cache). For authorities, this is always true.

Referenced by directory_info_has_arrived(), options_validate(), router_dump_router_to_string(), and update_extrainfo_downloads().

3.78.3.175 int or_options_t::AllowSingleHopExits

If true, and we are acting as a relay, allow exit circuits even when we are the first hop of a circuit.

Referenced by router_dump_router_to_string().

3.78.3.176 int or_options_t::ExcludeSingleHopRelays

If true, don't allow relays with AllowSingleHopExits=1 to be used in circuits that we build.

Referenced by choose_good_exit_server_general(), and router_choose_random_node().

3.78.3.177 int or_options_t::AllowSingleHopCircuits

If true, and the controller tells us to use a one-hop circuit, and the exit allows it, we use it.

3.78.3.178 int or_options_t::AllowDotExit

If true, we convert "www.google.com.foo.exit" addresses on the socks/trans/natd ports into "www.google.com" addresses that exit from the node "foo". Disabled by default since attacking websites and exit relays can use it to manipulate your path selection.

3.78.3.179 int or_options_t::DirReqStatistics

If true, the user wants us to collect statistics on clients requesting network statuses from us as directory.

Referenced by extrainfo_dump_to_string(), geoip_note_client_seen(), options_act(), options_transition_allowed(), and run_scheduled_events().

3.78.3.180 int or_options_t::ExitPortStatistics

If true, the user wants us to collect statistics on port usage.

Referenced by extrainfo_dump_to_string(), options_transition_allowed(), and run_scheduled_events().

3.78.3.181 int or_options_t::CellStatistics

If true, the user wants us to collect cell statistics.

Referenced by extrainfo_dump_to_string(), options_transition_allowed(), and run_scheduled_events().

3.78.3.182 int or_options_t::EntryStatistics

If true, the user wants us to collect statistics as entry node.

Referenced by extrainfo_dump_to_string(), geoip_note_client_seen(), options_act(), options_transition_allowed(), and run_scheduled_events().

3.78.3.183 int or_options_t::ExtraInfoStatistics

If true, include statistics file contents in extra-info documents.

Referenced by extrainfo_dump_to_string(), and router_rebuild_descriptor().

3.78.3.184 int or_options_t::ClientDNSRejectInternalAddresses

If true, do not believe anybody who tells us that a domain resolves to an internal address, or that an internal address has a PTR mapping. Helps avoid some cross-site attacks.

Referenced by connection_ap_process_end_not_open().

3.78.3.185 int or_options_t::V3AuthVotingInterval

The length of time that we think a consensus should be fresh.

Referenced by dirvote_get_preferred_voting_intervals().

3.78.3.186 int or_options_t::V3AuthVoteDelay

The length of time we think it will take to distribute votes.

Referenced by dirvote_get_preferred_voting_intervals().

3.78.3.187 int or_options_t::V3AuthDistDelay

The length of time we think it will take to distribute signatures.

Referenced by dirvote_get_preferred_voting_intervals().

3.78.3.188 int or_options_t::V3AuthNIntervalsValid

The number of intervals we think a consensus should be valid.

Referenced by dirvote_get_preferred_voting_intervals().

3.78.3.189 int or_options_t::V3AuthUseLegacyKey

Should advertise and sign consensuses with a legacy key, for key migration purposes?

3.78.3.190 char∗ or_options_t::V3BandwidthsFile

Location of bandwidth measurement file

Referenced by options_validate().

3.78.3.191 char∗ or_options_t::ConsensusParams

Authority only: key=value pairs that we add to our networkstatus consensus vote on the 'params' line.

3.78.3.192 int or_options_t::TestingV3AuthInitialVotingInterval

The length of time that we think an initial consensus should be fresh. Only altered on testing networks.

Referenced by dirvote_recalculate_timing().

3.78.3.193 int or_options_t::TestingV3AuthInitialVoteDelay

The length of time we think it will take to distribute initial votes. Only altered on testing networks.

Referenced by dirvote_recalculate_timing().

3.78.3.194 int or_options_t::TestingV3AuthInitialDistDelay

The length of time we think it will take to distribute initial signatures. Only altered on testing networks.

Referenced by dirvote_recalculate_timing().

3.78.3.195 int or_options_t::TestingAuthDirTimeToLearnReachability

If an authority has been around for less than this amount of time, it does not believe its reachability information is accurate. Only altered on testing networks.

3.78.3.196 int or_options_t::TestingEstimatedDescriptorPropagationTime

Clients don't download any descriptor this recent, since it will probably not have propagated to enough caches. Only altered on testing networks.

Referenced by client_would_use_router().

3.78.3.197 int or_options_t::TestingTorNetwork

If true, we take part in a testing network. Change the defaults of a couple of other configuration options and allow to change the values of certain configuration options.

Referenced by options_init_from_string(), options_transition_allowed(), and options_validate().

3.78.3.198 char∗ or_options_t::FallbackNetworkstatusFile

File to check for a consensus networkstatus, if we don't have one cached.

3.78.3.199 int or_options_t::BridgeRecordUsageByCountry

If true, and we have GeoIP data, and we're a bridge, keep a per-country count of how many client addresses have contacted us so that we can help the bridge authority guess which countries have blocked access to us.

Referenced by geoip_note_client_seen(), options_need_geoip_info(), and should_record_-bridge_info().

3.78.3.200 char∗ or_options_t::GeoIPFile

Optionally, a file with GeoIP data.

Referenced by options_act().

3.78.3.201 int or_options_t::ReloadTorrcOnSIGHUP

If true, SIGHUP should reload the torrc. Sometimes controllers want to make this false.

Referenced by do_hup().

The documentation for this struct was generated from the following file:

- **or.h**

3.79 or_state_t Struct Reference

`#include <or.h>`

Data Fields

- uint32_t **_magic**
- time_t **next_write**
- time_t **LastWritten**
- time_t **AccountingIntervalStart**
- uint64_t **AccountingBytesReadInInterval**
- uint64_t **AccountingBytesWrittenInInterval**
- int **AccountingSecondsActive**
- uint64_t **AccountingExpectedUsage**
- **config_line_t** ∗ **EntryGuards**
- time_t **BWHistoryReadEnds**
- int **BWHistoryReadInterval**
- **smartlist_t** ∗ **BWHistoryReadValues**
- time_t **BWHistoryWriteEnds**
- int **BWHistoryWriteInterval**
- **smartlist_t** ∗ **BWHistoryWriteValues**
- **config_line_t** ∗ **BuildtimeHistogram**
- uint16_t **TotalBuildTimes**
- char ∗ **TorVersion**
- **config_line_t** ∗ **ExtraLines**
- time_t **LastRotatedOnionKey**

3.79.1 Detailed Description

Persistent state for an onion router, as saved to disk.

3.79.2 Field Documentation

3.79.2.1 time_t or_state_t::next_write

The time at which we next plan to write the state to the disk. Equal to TIME_MAX if there are no savable changes, 0 if there are changes that should be saved right away.

Referenced by or_state_load(), or_state_mark_dirty(), and or_state_save().

3.79.2.2 time_t or_state_t::LastWritten

When was the state last written to disk?

Referenced by or_state_save(), and read_bandwidth_usage().

3.79.2.3 time_t or_state_t::AccountingIntervalStart

Fields for accounting bandwidth use.

Referenced by accounting_record_bandwidth_usage(), and read_bandwidth_usage().

3.79.2.4 config_line_t∗ or_state_t::EntryGuards

A list of Entry Guard-related configuration lines.

Referenced by entry_guards_parse_state(), and entry_guards_update_state().

3.79.2.5 time_t or_state_t::BWHistoryReadEnds

These fields hold information on the history of bandwidth usage for servers. The "Ends" fields hold the time when we last updated the bandwidth usage. The "Interval" fields hold the granularity, in seconds, of the entries of Values. The "Values" lists hold decimal string representations of the number of bytes read or written in each interval.

Referenced by rep_hist_load_state(), and rep_hist_update_state().

3.79.2.6 config_line_t∗ or_state_t::BuildtimeHistogram

Build time histogram

Referenced by circuit_build_times_parse_state(), and circuit_build_times_update_state().

3.79.2.7 char∗ or_state_t::TorVersion

What version of Tor wrote this state file?

Referenced by entry_guards_parse_state(), and or_state_save().

3.79.2.8 config_line_t∗ or_state_t::ExtraLines

Holds any unrecognized values we found in the state file, in the order in which we found them.

3.79.2.9 time_t or_state_t::LastRotatedOnionKey

When did we last rotate our onion key? "0" for 'no idea'.

Referenced by init_keys(), and rotate_onion_key().

The documentation for this struct was generated from the following file:

- **or.h**

3.80 orconn_circid_circuit_map_t Struct Reference

3.80.1 Detailed Description

A map from OR connection and circuit ID to circuit. (Lookup performance is very important here, since we need to do it every time a cell arrives.)

The documentation for this struct was generated from the following file:

- **circuitlist.c**

3.81 origin_circuit_t Struct Reference

`#include <or.h>`

Data Fields

- **circuit_t _base**
- **edge_connection_t * p_streams**
- **cpath_build_state_t * build_state**
- **crypt_path_t * cpath**
- **rend_data_t * rend_data**
- unsigned int **remaining_relay_early_cells**: 4
- uint8_t **relay_early_commands** [MAX_RELAY_EARLY_CELLS_PER_CIRCUIT]
- int **relay_early_cells_sent**
- **streamid_t next_stream_id**
- **crypto_pk_env_t * intro_key**
- uint32_t **global_identifier**

3.81.1 Detailed Description

An **origin_circuit_t** (p. 181) holds data necessary to build and use a circuit.

3.81.2 Field Documentation

3.81.2.1 edge_connection_t* origin_circuit_t::p_streams

Linked list of AP streams (or EXIT streams if hidden service) associated with this circuit.

Referenced by _circuit_mark_for_close(), circuit_detach_stream(), circuit_truncated(), connection_exit_begin_conn(), get_unique_stream_id_by_circ(), handle_control_closecircuit(), link_apconn_to_circ(), and set_streams_blocked_on_circ().

3.81.2.2 cpath_build_state_t* origin_circuit_t::build_state

Build state for this circuit. It includes the intended path length, the chosen exit router, rendezvous information, etc.

Referenced by _circuit_mark_for_close(), circuit_append_new_exit(), circuit_build_failed(), circuit_expire_building(), circuit_find_to_cannibalize(), circuit_free(), circuit_handle_first_hop(), circuit_is_acceptable(), circuit_launch_by_extend_info(), circuit_list_path_impl(), circuit_predict_and_launch_new(), circuit_send_next_onion_skin(), circuit_stream_is_being_handled(), connection_ap_handshake_send_begin(), connection_ap_process_end_not_open(), find_intro_circuit(), handle_control_attachstream(), onion_extend_cpath(), onion_pick_cpath_exit(), origin_circuit_init(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_send_introduction(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_relaunch_rendezvous(), rend_service_rendezvous_has_opened(), and warn_if_last_router_excluded().

3.81.2.3 crypt_path_t∗ origin_circuit_t::cpath

The doubly-linked list of **crypt_path_t** (p. 67) entries, one per hop, for this circuit. This includes ciphers for each hop, integrity-checking digests for each hop, and package/delivery windows for each hop.

Referenced by assert_circuit_ok(), circuit_append_new_exit(), circuit_build_failed(), circuit_conforms_to_options(), circuit_expire_building(), circuit_find_to_cannibalize(), circuit_finish_handshake(), circuit_free(), circuit_get_cpath_hop(), circuit_get_cpath_len(), circuit_handle_first_hop(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), circuit_truncated(), connection_exit_begin_conn(), cpath_is_on_circuit(), link_apconn_to_circ(), onion_extend_cpath(), relay_crypt(), relay_send_command_from_edge(), rend_client_introcirc_has_opened(), rend_client_receive_rendezvous(), rend_client_send_establish_rendezvous(), rend_process_relay_cell(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), router_perform_bandwidth_test(), and should_use_create_fast_for_circuit().

3.81.2.4 rend_data_t∗ origin_circuit_t::rend_data

Holds all rendezvous data on either client or service side.

Referenced by _circuit_mark_for_close(), circuit_free(), circuit_get_by_rend_query_and_purpose(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), connection_ap_handshake_attach_circuit(), connection_exit_begin_conn(), count_established_intro_points(), find_intro_circuit(), rend_client_introduction_acked(), rend_client_send_establish_rendezvous(), rend_client_send_introduction(), rend_config_services(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_relaunch_rendezvous(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

3.81.2.5 unsigned int origin_circuit_t::remaining_relay_early_cells

How many more relay_early cells can we send on this circuit, according to the specification?

Referenced by circuit_find_to_cannibalize(), origin_circuit_new(), relay_send_command_from_edge(), and rend_client_introduction_acked().

3.81.2.6 uint8_t origin_circuit_t::relay_early_commands[MAX_RELAY_EARLY_CELLS_PER_CIRCUIT]

What commands were sent over this circuit that decremented the RELAY_EARLY counter? This is for debugging task 878.

Referenced by relay_send_command_from_edge().

3.81.2.7 int origin_circuit_t::relay_early_cells_sent

How many RELAY_EARLY cells have been sent over this circuit? This is for debugging task 878, too.

Referenced by relay_send_command_from_edge().

3.81.2.8 streamid_t origin_circuit_t::next_stream_id

The next stream_id that will be tried when we're attempting to construct a new AP stream originating at this circuit.

Referenced by get_unique_stream_id_by_circ(), and origin_circuit_new().

3.81.2.9 uint32_t origin_circuit_t::global_identifier

Quasi-global identifier for this circuit; used for **control.c** (p. 485)

Referenced by control_event_circuit_status(), control_event_stream_status(), getinfo_helper_events(), handle_control_extendcircuit(), and origin_circuit_new().

The documentation for this struct was generated from the following file:

- **or.h**

3.82 packed_cell_t Struct Reference

`#include <or.h>`

Data Fields

- struct **packed_cell_t** * **next**
- char **body** [CELL_NETWORK_SIZE]

3.82.1 Detailed Description

A cell as packed for writing to the network.

3.82.2 Field Documentation

3.82.2.1 struct packed_cell_t* packed_cell_t::next `[read]`

Next cell queued on this circuit.

Referenced by cell_queue_append(), cell_queue_clear(), cell_queue_pop(), and packed_cell_copy().

3.82.2.2 char packed_cell_t::body[CELL_NETWORK_SIZE]

Cell as packed for network.

Referenced by cell_pack(), connection_or_flush_from_first_active_circuit(), and connection_or_write_cell_to_buf().

The documentation for this struct was generated from the following file:

- **or.h**

3.83 pending_connection_t Struct Reference

Data Fields

- **edge_connection_t** ∗ **conn**
- struct **pending_connection_t** ∗ **next**

3.83.1 Detailed Description

Linked list of connections waiting for a DNS answer.

The documentation for this struct was generated from the following file:

- **dns.c**

3.84 pending_consensus_t Struct Reference

Data Fields

- char * **body**
- **networkstatus_t** * **consensus**

3.84.1 Detailed Description

A consensus that we have built and are appending signatures to. Once it's time to publish it, it will become an active consensus if it accumulates enough signatures.

3.84.2 Field Documentation

3.84.2.1 char* pending_consensus_t::body

The body of the consensus that we're currently building. Once we have it built, it goes into **dirserv.c** (p. 583)

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_clear_pending_consensuses(), dirvote_compute_consensuses(), dirvote_get_pending_consensus(), and dirvote_publish_consensus().

3.84.2.2 networkstatus_t* pending_consensus_t::consensus

The parsed in-progress consensus document.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_clear_pending_consensuses(), dirvote_compute_consensuses(), dirvote_fetch_missing_signatures(), and dirvote_publish_consensus().

The documentation for this struct was generated from the following file:

- **dirvote.c**

3.85 pending_vote_t Struct Reference

Data Fields

- **cached_dir_t ∗ vote_body**
- **networkstatus_t ∗ vote**

3.85.1 Detailed Description

A vote **networkstatus_t** (p. 122) and its unparsed body: held around so we can use it to generate a consensus (at voting_ends) and so we can serve it to other authorities that might want it.

The documentation for this struct was generated from the following file:

- **dirvote.c**

3.86 policy_map_ent_t Struct Reference

3.86.1 Detailed Description

Node in hashtable used to store address policy entries.

The documentation for this struct was generated from the following file:

- **policies.c**

3.87 policy_summary_item_t Struct Reference

Data Fields

- uint16_t **prt_min**
- uint16_t **prt_max**
- uint64_t **reject_count**
- int **accepted**:1

3.87.1 Detailed Description

Element of an exit policy summary

3.87.2 Field Documentation

3.87.2.1 uint16_t policy_summary_item_t::prt_min

Lowest port number to accept/reject.

Referenced by policy_summary_create(), and policy_summary_item_split().

3.87.2.2 uint16_t policy_summary_item_t::prt_max

Highest port number to accept/reject.

Referenced by policy_summary_create(), and policy_summary_item_split().

3.87.2.3 uint64_t policy_summary_item_t::reject_count

Number of IP-Addresses that are rejected to this port range.

Referenced by policy_summary_create(), and policy_summary_item_split().

The documentation for this struct was generated from the following file:

- **policies.c**

3.88 relay_header_t Struct Reference

`#include <or.h>`

Data Fields

- uint8_t **command**
- uint16_t **recognized**
- **streamid_t** **stream_id**
- char **integrity** [4]
- uint16_t **length**

3.88.1 Detailed Description

Beginning of a RELAY cell payload.

3.88.2 Field Documentation

3.88.2.1 uint8_t relay_header_t::command

The end-to-end relay command.

Referenced by connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), relay_header_pack(), relay_header_unpack(), and relay_send_command_from_edge().

3.88.2.2 uint16_t relay_header_t::recognized

Used to tell whether cell is for us.

Referenced by relay_crypt(), relay_header_pack(), and relay_header_unpack().

3.88.2.3 streamid_t relay_header_t::stream_id

Which stream is this cell associated with?

Referenced by connection_edge_process_relay_cell(), connection_exit_begin_conn(), connection_exit_begin_resolve(), relay_header_pack(), relay_header_unpack(), relay_lookup_conn(), and relay_send_command_from_edge().

3.88.2.4 char relay_header_t::integrity[4]

Used to tell whether cell is corrupted.

Referenced by relay_digest_matches(), relay_header_pack(), relay_header_unpack(), and relay_set_digest().

3.88.2.5 uint16_t relay_header_t::length

How long is the payload body?

Referenced by circuit_extend(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_exit_begin_resolve(), relay_header_pack(), relay_header_unpack(), and relay_send_command_from_edge().

The documentation for this struct was generated from the following file:

- **or.h**

3.89 rend_authorized_client_t Struct Reference

`#include <or.h>`

Data Fields

- char * **client_name**
- char **descriptor_cookie** [REND_DESC_COOKIE_LEN]
- **crypto_pk_env_t** * **client_key**

3.89.1 Detailed Description

Hidden-service side configuration of client authorization.

The documentation for this struct was generated from the following file:

- **or.h**

3.90 rend_cache_entry_t Struct Reference

`#include <or.h>`

Data Fields

- size_t **len**
- time_t **received**
- char ∗ **desc**
- **rend_service_descriptor_t** ∗ **parsed**

3.90.1 Detailed Description

A cached rendezvous descriptor.

3.90.2 Field Documentation

3.90.2.1 size_t rend_cache_entry_t::len

Length of **desc**

Referenced by rend_cache_lookup_desc(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), and rend_cache_store_v2_desc_as_dir().

3.90.2.2 time_t rend_cache_entry_t::received

When was the descriptor received?

Referenced by rend_cache_store(), rend_cache_store_v2_desc_as_client(), and rend_cache_store_v2_desc_as_dir().

3.90.2.3 char∗ rend_cache_entry_t::desc

Service descriptor

Referenced by rend_cache_entry_free(), rend_cache_lookup_desc(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), and rend_cache_store_v2_desc_as_dir().

3.90.2.4 rend_service_descriptor_t∗ rend_cache_entry_t::parsed

Parsed value of 'desc'

Referenced by rend_cache_clean(), rend_cache_clean_v2_descs_as_dir(), rend_cache_entry_free(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_remove_intro_point(), and rend_client_send_introduction().

The documentation for this struct was generated from the following file:

- **or.h**

3.91 rend_data_t Struct Reference

`#include <or.h>`

Data Fields

- char **onion_address** [REND_SERVICE_ID_LEN_BASE32+1]
- char **descriptor_cookie** [REND_DESC_COOKIE_LEN]
- **rend_auth_type_t auth_type**
- char **rend_pk_digest** [DIGEST_LEN]
- char **rend_cookie** [REND_COOKIE_LEN]

3.91.1 Detailed Description

Client- and server-side data that is used for hidden service connection establishment. Not all fields contain data depending on where this struct is used.

3.91.2 Field Documentation

3.91.2.1 char rend_data_t::onion_address[REND_SERVICE_ID_LEN_BASE32+1]

Onion address (without the .onion part) that a client requests.

Referenced by _circuit_mark_for_close(), circuit_get_by_rend_query_and_purpose(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), connection_about_to_close_connection(), connection_ap_handshake_attach_circuit(), connection_dir_client_reached_eof(), count_established_intro_points(), directory_get_from_hs_dir(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_introduction_acked(), rend_client_refetch_v2_renddesc(), rend_client_remove_intro_point(), rend_client_send_introduction(), rend_service_introduce(), and rend_service_launch_establish_intro().

3.91.2.2 char rend_data_t::descriptor_cookie[REND_DESC_COOKIE_LEN]

(Optional) descriptor cookie that is used by a client.

Referenced by directory_get_from_hs_dir(), rend_cache_store_v2_desc_as_client(), and rend_client_refetch_v2_renddesc().

3.91.2.3 rend_auth_type_t rend_data_t::auth_type

Authorization type for accessing a service used by a client.

Referenced by directory_get_from_hs_dir(), rend_cache_store_v2_desc_as_client(), and rend_client_refetch_v2_renddesc().

3.91.2.4 char rend_data_t::rend_pk_digest[DIGEST_LEN]

Hash of the hidden service's PK used by a service.

Referenced by rend_config_services(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

3.91.2.5 char rend_data_t::rend_cookie[REND_COOKIE_LEN]

Rendezvous cookie used by both, client and service.

Referenced by rend_client_send_establish_rendezvous(), rend_service_introduce(), and rend_service_rendezvous_has_opened().

The documentation for this struct was generated from the following file:

- **or.h**

3.92 rend_encoded_v2_service_descriptor_t Struct Reference

`#include <or.h>`

Data Fields

- char **desc_id** [DIGEST_LEN]
- char * **desc_str**

3.92.1 Detailed Description

ASCII-encoded v2 hidden service descriptor.

3.92.2 Field Documentation

3.92.2.1 char rend_encoded_v2_service_descriptor_t::desc_id[DIGEST_LEN]

Descriptor ID.

Referenced by directory_post_to_hs_dir(), and rend_encode_v2_descriptors().

3.92.2.2 char* rend_encoded_v2_service_descriptor_t::desc_str

Descriptor string.

Referenced by directory_post_to_hs_dir(), rend_desc_v2_is_parsable(), rend_encode_v2_descriptors(), and rend_encoded_v2_service_descriptor_free().

The documentation for this struct was generated from the following file:

- **or.h**

3.93 rend_intro_point_t Struct Reference

`#include <or.h>`

Data Fields

- **extend_info_t * extend_info**
- **crypto_pk_env_t * intro_key**

3.93.1 Detailed Description

Introduction point information.

3.93.2 Field Documentation

3.93.2.1 extend_info_t* rend_intro_point_t::extend_info

Extend info of this introduction point.

Referenced by find_intro_circuit(), rend_client_get_random_intro(), rend_client_remove_intro_point(), rend_encode_v2_intro_points(), rend_intro_point_free(), rend_parse_introduction_points(), rend_parse_service_descriptor(), rend_service_dump_stats(), rend_service_launch_establish_intro(), rend_service_update_descriptor(), and rend_services_introduce().

3.93.2.2 crypto_pk_env_t* rend_intro_point_t::intro_key

Introduction key that replaces the service key, if this descriptor is V2.

Referenced by rend_encode_v2_intro_points(), rend_intro_point_free(), rend_parse_introduction_points(), rend_service_launch_establish_intro(), and rend_service_update_descriptor().

The documentation for this struct was generated from the following file:

- **or.h**

3.94 rend_service_authorization_t Struct Reference

`#include <or.h>`

Data Fields

- char **descriptor_cookie** [REND_DESC_COOKIE_LEN]
- char **onion_address** [REND_SERVICE_ADDRESS_LEN+1]
- **rend_auth_type_t auth_type**

3.94.1 Detailed Description

Client-side configuration of authorization for a hidden service.

The documentation for this struct was generated from the following file:

- **or.h**

3.95 rend_service_descriptor_t Struct Reference

`#include <or.h>`

Data Fields

- **crypto_pk_env_t * pk**
- int **version**
- time_t **timestamp**
- uint16_t **protocols**
- **smartlist_t * intro_nodes**
- int **all_uploads_performed**
- **smartlist_t * successful_uploads**

3.95.1 Detailed Description

Information used to connect to a hidden service.

3.95.2 Field Documentation

3.95.2.1 crypto_pk_env_t* rend_service_descriptor_t::pk

This service's public key.

Referenced by rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_encode_v2_descriptors(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), rend_service_update_descriptor(), and upload_service_descriptor().

3.95.2.2 int rend_service_descriptor_t::version

Version of the descriptor format: 0 or 2.

Referenced by rend_parse_service_descriptor(), and rend_parse_v2_service_descriptor().

3.95.2.3 time_t rend_service_descriptor_t::timestamp

Time when the descriptor was generated.

Referenced by rend_cache_clean(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_encode_v2_descriptors(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), and rend_service_update_descriptor().

3.95.2.4 uint16_t rend_service_descriptor_t::protocols

Bitmask: which rendezvous protocols are supported? (We allow bits '0', '1', and '2' to be set.)

Referenced by rend_encode_v2_descriptors(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), and rend_service_update_descriptor().

3.95.2.5 smartlist_t∗ rend_service_descriptor_t::intro_nodes

List of the service's introduction points. Elements are removed if introduction attempts fail.

Referenced by rend_cache_store_v2_desc_as_client(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_remove_intro_point(), rend_client_send_introduction(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_parse_introduction_points(), rend_parse_service_descriptor(), rend_service_descriptor_free(), rend_service_update_descriptor(), and rend_services_introduce().

3.95.2.6 int rend_service_descriptor_t::all_uploads_performed

Has descriptor been uploaded to all hidden service directories?

Referenced by directory_post_to_hs_dir(), and rend_consider_descriptor_republication().

3.95.2.7 smartlist_t∗ rend_service_descriptor_t::successful_uploads

List of hidden service directories to which an upload request for this descriptor could be sent. Smartlist exists only when at least one of the previous upload requests failed (otherwise it's not important to know which uploads succeeded and which not).

Referenced by directory_post_to_hs_dir(), and rend_service_descriptor_free().

The documentation for this struct was generated from the following file:

- **or.h**

3.96 rend_service_port_config_t Struct Reference

Data Fields

- uint16_t **virtual_port**
- uint16_t **real_port**
- **tor_addr_t real_addr**

3.96.1 Detailed Description

Represents the mapping from a virtual port of a rendezvous service to a real port on some IP.

The documentation for this struct was generated from the following file:

- **rendservice.c**

3.97 rend_service_t Struct Reference

Data Fields

- char ∗ **directory**
- **smartlist_t** ∗ **ports**
- **rend_auth_type_t** **auth_type**
- **smartlist_t** ∗ **clients**
- **crypto_pk_env_t** ∗ **private_key**
- char **service_id** [REND_SERVICE_ID_LEN_BASE32+1]
- char **pk_digest** [DIGEST_LEN]
- **smartlist_t** ∗ **intro_nodes**
- time_t **intro_period_started**
- int **n_intro_circuits_launched**
- **rend_service_descriptor_t** ∗ **desc**
- time_t **desc_is_dirty**
- time_t **next_upload_time**
- digestmap_t ∗ **accepted_intros**
- time_t **last_cleaned_accepted_intros**

3.97.1 Detailed Description

Represents a single hidden service running at this OP.

3.97.2 Field Documentation

3.97.2.1 char∗ rend_service_t::directory

where in the filesystem it stores it

Referenced by rend_add_service(), rend_config_services(), rend_service_dump_stats(), and rend_service_free().

3.97.2.2 smartlist_t∗ rend_service_t::ports

List of **rend_service_port_config_t** (p. 201)

Referenced by rend_add_service(), rend_config_services(), rend_service_free(), rend_service_requires_uptime(), and rend_service_set_connection_addr_port().

3.97.2.3 rend_auth_type_t rend_service_t::auth_type

Client authorization type or 0 if no client authorization is performed.

Referenced by rend_add_service(), rend_config_services(), and upload_service_descriptor().

3.97.2.4 smartlist_t∗ rend_service_t::clients

List of rend_authorized_client_t's of clients that may access our service. Can be NULL if no client authorization is performed.

Referenced by rend_add_service(), rend_check_authorization(), rend_config_services(), rend_service_free(), rend_service_introduce(), and upload_service_descriptor().

3.97.2.5 crypto_pk_env_t∗ rend_service_t::private_key

Permanent hidden-service key.

Referenced by rend_service_free(), and rend_service_update_descriptor().

3.97.2.6 char rend_service_t::service_id[REND_SERVICE_ID_LEN_BASE32+1]

Onion address without '.onion'

Referenced by rend_service_introduce(), rend_service_launch_establish_intro(), and rend_services_introduce().

3.97.2.7 char rend_service_t::pk_digest[DIGEST_LEN]

Hash of permanent hidden-service key.

Referenced by rend_service_dump_stats(), rend_service_launch_establish_intro(), rend_service_update_descriptor(), and rend_services_introduce().

3.97.2.8 smartlist_t∗ rend_service_t::intro_nodes

List of rend_intro_point_t's we have, or are trying to establish.

Referenced by rend_add_service(), rend_service_dump_stats(), rend_service_free(), rend_service_update_descriptor(), and rend_services_introduce().

3.97.2.9 time_t rend_service_t::intro_period_started

Start of the current period to build introduction points.

Referenced by rend_config_services(), and rend_services_introduce().

3.97.2.10 int rend_service_t::n_intro_circuits_launched

Count of intro circuits we have established in this period.

Referenced by rend_service_launch_establish_intro(), and rend_services_introduce().

3.97.2.11 rend_service_descriptor_t∗ rend_service_t::desc

Current hidden service descriptor.

Referenced by rend_consider_descriptor_republication(), rend_service_free(), rend_service_update_descriptor(), rend_services_introduce(), and upload_service_descriptor().

3.97.2.12 time_t rend_service_t::desc_is_dirty

Time at which changes to the hidden service descriptor content occurred, or 0 if it's up-to-date.

Referenced by rend_consider_services_upload(), rend_service_intro_established(), rend_services_introduce(), and upload_service_descriptor().

3.97.2.13 time_t rend_service_t::next_upload_time

Scheduled next hidden service descriptor upload time.

Referenced by rend_consider_services_upload(), and upload_service_descriptor().

3.97.2.14 digestmap_t* rend_service_t::accepted_intros

Map from digests of Diffie-Hellman values INTRODUCE2 to time_t of when they were received; used to prevent replays.

Referenced by clean_accepted_intros(), rend_service_free(), and rend_service_introduce().

3.97.2.15 time_t rend_service_t::last_cleaned_accepted_intros

Time at which we last removed expired values from accepted_intros.

Referenced by clean_accepted_intros(), and rend_service_introduce().

The documentation for this struct was generated from the following file:

- **rendservice.c**

3.98 router_status_t Struct Reference

Data Fields

- char **nickname** [MAX_NICKNAME_LEN+1]
- uint32_t **status**

3.98.1 Detailed Description

Encapsulate a nickname and an FP_* status; target of status_by_digest map.

The documentation for this struct was generated from the following file:

- **dirserv.c**

3.99 routerinfo_t Struct Reference

`#include <or.h>`

Data Fields

- **signed_descriptor_t cache_info**
- char * **address**
- char * **nickname**
- uint32_t **addr**
- uint16_t **or_port**
- uint16_t **dir_port**
- **crypto_pk_env_t** * **onion_pkey**
- **crypto_pk_env_t** * **identity_pkey**
- char * **platform**
- uint32_t **bandwidthrate**
- uint32_t **bandwidthburst**
- uint32_t **bandwidthcapacity**
- **smartlist_t** * **exit_policy**
- long **uptime**
- **smartlist_t** * **declared_family**
- char * **contact_info**
- unsigned int **is_hibernating**:1
- unsigned int **has_old_dnsworkers**:1
- unsigned int **caches_extra_info**:1
- unsigned int **allow_single_hop_exits**:1
- unsigned int **is_running**:1
- unsigned int **is_valid**:1
- unsigned int **is_named**:1
- unsigned int **is_fast**:1
- unsigned int **is_stable**:1
- unsigned int **is_possible_guard**:1
- unsigned int **is_exit**:1
- unsigned int **is_bad_exit**:1
- unsigned int **is_bad_directory**:1
- unsigned int **wants_to_be_hs_dir**:1
- unsigned int **is_hs_dir**:1
- unsigned int **policy_is_reject_star**:1
- uint8_t **purpose**
- time_t **last_reachable**
- time_t **testing_since**
- **country_t country**

3.99.1 Detailed Description

Information about another onion router in the network.

3.99.2 Field Documentation

3.99.2.1 char* routerinfo_t::address

Location of OR: either a hostname or an IP address.

Referenced by authdir_wants_to_reject_router(), connection_or_init_conn_from_address(), consider_testing_reachability(), directory_get_from_dirserver(), directory_initiate_command_routerstatus_rend(), dirserv_router_get_status(), dirserv_router_has_valid_address(), dirserv_single_reachability_test(), inform_testing_reachability(), router_differences_are_cosmetic(), router_dirport_found_reachable(), router_dump_router_to_string(), router_orport_found_reachable(), router_parse_entry_from_string(), router_rebuild_descriptor(), router_set_status(), routerinfo_free(), and second_elapsed_callback().

3.99.2.2 char* routerinfo_t::nickname

Human-readable OR name.

Referenced by add_an_entry_guard(), authdir_wants_to_reject_router(), choose_good_exit_server_general(), circuit_send_next_onion_skin(), command_process_create_cell(), connection_or_init_conn_from_address(), directory_remove_invalid(), dirserv_add_descriptor(), dirserv_router_get_status(), dirserv_router_has_valid_address(), dirserv_single_reachability_test(), extend_info_from_router(), learned_bridge_descriptor(), list_single_server_status(), rep_hist_dump_stats(), rep_hist_get_router_stability_doc(), router_add_to_routerlist(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_get_by_hexdigest(), router_get_verbose_nickname(), router_hex_digest_matches(), router_nickname_matches(), router_parse_entry_from_string(), router_parse_list_from_string(), router_rebuild_descriptor(), router_set_status(), routerinfo_free(), routerlist_remove_old_routers(), routerset_contains_router(), and set_routerstatus_from_routerinfo().

3.99.2.3 uint32_t routerinfo_t::addr

IPv4 address of OR, in host order.

Referenced by _compare_routerinfo_by_ip_and_bw(), check_descriptor_ipaddress_changed(), connection_or_init_conn_from_address(), connection_or_send_netinfo(), consider_testing_reachability(), directory_conn_is_self_reachability_test(), directory_get_from_dirserver(), dirserv_router_get_status(), dirserv_single_reachability_test(), extend_info_from_router(), fascist_firewall_allows_or(), rep_hist_get_router_stability_doc(), router_add_to_routerlist(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_get_configured_bridge(), routerinfo_set_country(), routers_in_same_network_family(), routerset_contains_router(), and set_routerstatus_from_routerinfo().

3.99.2.4 uint16_t routerinfo_t::or_port

Port for TLS connections.

Referenced by connection_or_init_conn_from_address(), consider_testing_reachability(), directory_get_from_dirserver(), dirserv_router_get_status(), dirserv_single_reachability_test(), extend_info_from_router(), fascist_firewall_allows_or(), inform_testing_reachability(), router_add_to_routerlist(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_orport_found_reachable(), router_parse_entry_from_string(), router_-

rebuild_descriptor(), routerinfo_get_configured_bridge(), routerset_contains_router(), second_elapsed_callback(), and set_routerstatus_from_routerinfo().

3.99.2.5 uint16_t routerinfo_t::dir_port

Port for HTTP directory connections.

Referenced by consider_testing_reachability(), directory_conn_is_self_reachability_test(), directory_fetches_from_authorities(), inform_testing_reachability(), router_differences_are_cosmetic(), router_dirport_found_reachable(), router_dump_router_to_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), second_elapsed_callback(), set_routerstatus_from_routerinfo(), and smartlist_choose_by_bandwidth_weights().

3.99.2.6 crypto_pk_env_t∗ routerinfo_t::onion_pkey

Public RSA key for onions.

Referenced by dirvote_create_microdescriptor(), extend_info_from_router(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), and routerinfo_free().

3.99.2.7 crypto_pk_env_t∗ routerinfo_t::identity_pkey

Public RSA key for signing.

Referenced by dirserv_router_get_status(), extrainfo_parse_entry_from_string(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_free(), and routerinfo_incompatible_with_extrainfo().

3.99.2.8 char∗ routerinfo_t::platform

What software/operating system is this OR using?

Referenced by command_process_create_cell(), dirserv_router_get_status(), esc_router_info(), router_differences_are_cosmetic(), router_digest_version_as_new_as(), router_dump_router_to_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), router_supports_extrainfo(), routerinfo_free(), and set_routerstatus_from_routerinfo().

3.99.2.9 uint32_t routerinfo_t::bandwidthrate

How many bytes does this OR add to its token bucket per second?

Referenced by router_differences_are_cosmetic(), router_dump_router_to_string(), router_get_advertised_bandwidth(), router_get_advertised_bandwidth_capped(), router_parse_entry_from_string(), router_rebuild_descriptor(), and run_scheduled_events().

3.99.2.10 uint32_t routerinfo_t::bandwidthburst

How large is this OR's token bucket?

Referenced by router_differences_are_cosmetic(), router_dump_router_to_string(), router_parse_entry_from_string(), and router_rebuild_descriptor().

3.99.2.11 uint32_t routerinfo_t::bandwidthcapacity

How many bytes/s is this router known to handle?

Referenced by check_descriptor_bandwidth_changed(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_get_advertised_bandwidth(), router_get_advertised_bandwidth_capped(), router_parse_entry_from_string(), router_rebuild_descriptor(), and run_scheduled_events().

3.99.2.12 smartlist_t∗ routerinfo_t::exit_policy

What streams will this OR permit to exit? NULL for 'reject ∗:∗'.

Referenced by circuit_stream_is_being_handled(), connection_ap_can_use_exit(), dirvote_create_microdescriptor(), dns_resolve_impl(), policies_set_router_exitpolicy_to_reject_all(), router_add_exit_policy(), router_compare_to_my_exit_policy(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_handles_some_port(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_free(), and routerstatus_format_entry().

3.99.2.13 long routerinfo_t::uptime

How many seconds the router claims to have been up

Referenced by real_uptime(), rep_hist_get_router_stability_doc(), router_differences_are_cosmetic(), and router_parse_entry_from_string().

3.99.2.14 smartlist_t∗ routerinfo_t::declared_family

Nicknames of router which this router claims are its family.

Referenced by dirvote_create_microdescriptor(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_free(), routerlist_add_family(), and routers_in_same_family().

3.99.2.15 char∗ routerinfo_t::contact_info

Declared contact info for this router.

Referenced by authdir_wants_to_reject_router(), dirserv_router_get_status(), esc_router_info(), router_differences_are_cosmetic(), router_parse_entry_from_string(), and routerinfo_free().

3.99.2.16 unsigned int routerinfo_t::is_hibernating

Whether the router claims to be hibernating

Referenced by rep_hist_get_router_stability_doc(), router_differences_are_cosmetic(), router_is_active(), and router_parse_entry_from_string().

3.99.2.17 unsigned int routerinfo_t::has_old_dnsworkers

Whether the router is using dnsworker code.

Referenced by connection_ap_can_use_exit(), router_differences_are_cosmetic(), and router_parse_entry_from_string().

3.99.2.18 unsigned int routerinfo_t::caches_extra_info

Whether the router caches and serves extrainfo documents.

Referenced by router_parse_entry_from_string(), and router_supports_extrainfo().

3.99.2.19 unsigned int routerinfo_t::allow_single_hop_exits

Whether the router allows single hop exits.

Referenced by choose_good_exit_server_general(), handle_control_attachstream(), and router_parse_entry_from_string().

3.99.2.20 unsigned int routerinfo_t::is_running

As far as we know, is this OR currently running?

Referenced by _compare_routerinfo_by_ip_and_bw(), add_nickname_list_to_smartlist(), bridges_retry_helper(), choose_good_exit_server_general(), count_acceptable_routers(), dirserv_set_router_is_running(), dirserv_thinks_router_is_hs_dir(), entry_guard_set_status(), learned_bridge_descriptor(), rep_hist_get_router_stability_doc(), router_clear_status_flags(), router_is_active(), router_rebuild_descriptor(), router_set_status(), and set_routerstatus_from_routerinfo().

3.99.2.21 unsigned int routerinfo_t::is_valid

Has a trusted dirserver validated this OR? (For Authdir: Have we validated this OR?)

Referenced by authdir_wants_to_reject_router(), choose_good_exit_server_general(), count_acceptable_routers(), directory_remove_invalid(), dirserv_add_descriptor(), list_single_server_status(), rep_hist_get_router_stability_doc(), router_add_to_routerlist(), router_clear_status_flags(), router_is_active(), router_rebuild_descriptor(), and set_routerstatus_from_routerinfo().

3.99.2.22 unsigned int routerinfo_t::is_named

Do we believe the nickname that this OR gives us?

Referenced by authdir_wants_to_reject_router(), circuit_list_path_impl(), directory_remove_invalid(), router_get_by_hexdigest(), router_get_verbose_nickname(), router_hex_digest_matches(), router_rebuild_descriptor(), and routerset_contains_router().

3.99.2.23 unsigned int routerinfo_t::is_stable

Do we think this is a fast OR?

Referenced by router_clear_status_flags(), router_is_unreliable(), and set_routerstatus_from_routerinfo().

3.99.2.24 unsigned int routerinfo_t::is_possible_guard

Do we think this is a stable OR? Do we think this is an OK guard?

Referenced by entry_guard_set_status(), router_clear_status_flags(), router_is_unreliable(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

3.99.2.25 unsigned int routerinfo_t::is_exit

Do we think this is an OK exit?

Referenced by router_clear_status_flags(), set_routerstatus_from_routerinfo(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

3.99.2.26 unsigned int routerinfo_t::is_bad_exit

Do we think this exit is censored, borked, or otherwise nasty?

Referenced by authdir_wants_to_reject_router(), choose_good_exit_server_general(), directory_remove_invalid(), router_clear_status_flags(), and set_routerstatus_from_routerinfo().

3.99.2.27 unsigned int routerinfo_t::is_bad_directory

Do we think this directory is junky, underpowered, or otherwise useless?

Referenced by authdir_wants_to_reject_router(), directory_remove_invalid(), router_clear_status_flags(), and set_routerstatus_from_routerinfo().

3.99.2.28 unsigned int routerinfo_t::wants_to_be_hs_dir

True iff this router claims to be a hidden service directory.

Referenced by dirserv_thinks_router_is_hs_dir(), and router_parse_entry_from_string().

3.99.2.29 unsigned int routerinfo_t::is_hs_dir

True iff this router is a hidden service directory according to the authorities.

Referenced by router_clear_status_flags(), and set_routerstatus_from_routerinfo().

3.99.2.30 unsigned int routerinfo_t::policy_is_reject_star

True iff the exit policy for this router rejects everything.

Referenced by router_exit_policy_rejects_all(), and router_parse_entry_from_string().

3.99.2.31 time_t routerinfo_t::last_reachable

What positions in a circuit is this router good for? When was the last time we could reach this OR?

Referenced by dirserv_set_router_is_running(), and router_add_to_routerlist().

3.99.2.32 time_t routerinfo_t::testing_since

When did we start testing reachability for this OR?

Referenced by dirserv_single_reachability_test(), and router_add_to_routerlist().

3.99.2.33 country_t routerinfo_t::country

According to the geoip db what country is this router in?

Referenced by router_parse_entry_from_string(), routerinfo_set_country(), and routerset_contains_router().

The documentation for this struct was generated from the following file:

- **or.h**

3.100 routerlist_t Struct Reference

`#include <or.h>`

Data Fields

- struct digest_ri_map_t * **identity_map**
- struct digest_sd_map_t * **desc_digest_map**
- struct digest_ei_map_t * **extra_info_map**
- struct digest_sd_map_t * **desc_by_eid_map**
- **smartlist_t** * **routers**
- **smartlist_t** * **old_routers**
- **desc_store_t desc_store**
- **desc_store_t extrainfo_store**

3.100.1 Detailed Description

Contents of a directory of onion routers.

3.100.2 Field Documentation

3.100.2.1 struct digest_ri_map_t* routerlist_t::identity_map `[read]`

Map from server identity digest to a member of routers.

Referenced by extrainfo_insert(), router_add_to_routerlist(), router_get_by_digest(), router_get_by_nickname(), router_get_routerlist(), router_parse_list_from_string(), routerlist_assert_ok(), routerlist_free(), routerlist_insert(), routerlist_remove(), and routerlist_replace().

3.100.2.2 struct digest_sd_map_t* routerlist_t::desc_digest_map `[read]`

Map from server descriptor digest to a **signed_descriptor_t** (p. 227) from routers or old_routers.

Referenced by router_add_to_routerlist(), router_get_by_descriptor_digest(), router_get_routerlist(), routerlist_assert_ok(), routerlist_free(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

3.100.2.3 struct digest_ei_map_t* routerlist_t::extra_info_map `[read]`

Map from extra-info digest to an **extrainfo_t** (p. 99). Only exists for routers in routers or old_routers.

Referenced by extrainfo_get_by_descriptor_digest(), extrainfo_insert(), router_get_routerlist(), router_rebuild_store(), routerlist_free(), routerlist_remove(), routerlist_remove_old(), routerlist_replace(), and update_extrainfo_downloads().

3.100.2.4 struct digest_sd_map_t∗ routerlist_t::desc_by_eid_map [read]

Map from extra-info digests to a **signed_descriptor_t** (p. 227) for a router descriptor having that extra-info digest. Only exists for routers in routers or old_routers.

Referenced by extrainfo_insert(), router_get_by_extrainfo_digest(), router_get_routerlist(), routerlist_free(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

3.100.2.5 smartlist_t∗ routerlist_t::routers

List of **routerinfo_t** (p. 206) for all currently live routers we know.

Referenced by choose_good_entry_server(), choose_good_exit_server_general(), directory_remove_invalid(), dirserv_compute_performance_thresholds(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirserv_orconn_tls_done(), dirserv_test_reachability(), dump_routerlist_mem_usage(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), getinfo_helper_dir(), mark_all_trusteddirservers_up(), networkstatus_getinfo_by_purpose(), onion_pick_cpath_exit(), router_add_running_routers_to_smartlist(), router_choose_random_node(), router_exit_policy_all_routers_reject(), router_find_exact_exit_enclave(), router_get_by_nickname(), router_get_routerlist(), router_rebuild_store(), router_reset_descriptor_download_failures(), routerlist_add_network_family(), routerlist_assert_ok(), routerlist_find_my_routerinfo(), routerlist_free(), routerlist_insert(), routerlist_refresh_countries(), routerlist_remove(), routerlist_remove_old_routers(), routerlist_replace(), routers_update_all_from_networkstatus(), and update_extrainfo_downloads().

3.100.2.6 smartlist_t∗ routerlist_t::old_routers

List of **signed_descriptor_t** (p. 227) for older router descriptors we're caching.

Referenced by dump_routerlist_mem_usage(), router_get_routerlist(), router_rebuild_store(), routerlist_free(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), routerlist_replace(), routers_update_all_from_networkstatus(), and update_extrainfo_downloads().

3.100.2.7 desc_store_t routerlist_t::desc_store

Store holding server descriptors. If present, any router whose cache_info.saved_location == SAVED_IN_CACHE is stored in this file starting at cache_info.saved_offset

Referenced by desc_get_store(), router_add_to_routerlist(), router_get_routerlist(), router_load_routers_from_string(), router_reload_router_list(), routerlist_free(), routerlist_remove(), routerlist_remove_old_routers(), and routerlist_replace().

3.100.2.8 desc_store_t routerlist_t::extrainfo_store

Store holding extra-info documents.

Referenced by desc_get_store(), extrainfo_insert(), router_add_extrainfo_to_routerlist(), router_get_routerlist(), router_reload_router_list(), routerlist_free(), routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_routers(), and routerlist_replace().

The documentation for this struct was generated from the following file:

- **or.h**

3.101 routerset_t Struct Reference

Data Fields

- **smartlist_t * list**
- strmap_t * **names**
- digestmap_t * **digests**
- **smartlist_t * policies**
- char * **description**
- **smartlist_t * country_names**
- int **n_countries**
- bitarray_t * **countries**

3.101.1 Detailed Description

A routerset specifies constraints on a set of possible routerinfos, based on their names, identities, or addresses. It is optimized for determining whether a router is a member or not, in O(1+P) time, where P is the number of address policy constraints.

3.101.2 Field Documentation

3.101.2.1 smartlist_t* routerset_t::list

A list of strings for the elements of the policy. Each string is either a nickname, a hexadecimal identity fingerprint, or an address policy. A router belongs to the set if its nickname OR its identity OR its address matches an entry here.

Referenced by routerset_equal(), routerset_free(), routerset_get_all_routers(), routerset_new(), routerset_parse(), and routerset_union().

3.101.2.2 strmap_t* routerset_t::names

A map from lowercase nicknames of routers in the set to (void*)1

Referenced by routerset_free(), routerset_new(), and routerset_parse().

3.101.2.3 digestmap_t* routerset_t::digests

A map from identity digests routers in the set to (void*)1

Referenced by routerset_free(), routerset_new(), and routerset_parse().

3.101.2.4 smartlist_t* routerset_t::policies

An address policy for routers in the set. For implementation reasons, a router belongs to the set if it is _rejected_ by this policy.

Referenced by routerset_free(), routerset_new(), and routerset_parse().

3.101.2.5 char∗ routerset_t::description

A human-readable description of what this routerset is for. Used in log messages.

3.101.2.6 smartlist_t∗ routerset_t::country_names

A list of the country codes in this set.

Referenced by routerset_free(), routerset_new(), routerset_parse(), and routerset_refresh_countries().

3.101.2.7 int routerset_t::n_countries

Total number of countries we knew about when we built **countries**.

Referenced by routerset_refresh_countries().

3.101.2.8 bitarray_t∗ routerset_t::countries

Bit array mapping the return value of **geoip_get_country()** (p. 652) to 1 iff the country is a member of this routerset. Note that we MUST call **routerset_refresh_countries()** (p. 1011) whenever the geoip country list is reloaded.

Referenced by routerset_free(), and routerset_refresh_countries().

The documentation for this struct was generated from the following file:

- **routerlist.c**

3.102 routerstatus_t Struct Reference

`#include <or.h>`

Data Fields

- time_t **published_on**
- char **nickname** [MAX_NICKNAME_LEN+1]
- char **identity_digest** [DIGEST_LEN]
- char **descriptor_digest** [DIGEST_LEN]
- uint32_t **addr**
- uint16_t **or_port**
- uint16_t **dir_port**
- unsigned int **is_authority**:1
- unsigned int **is_exit**:1
- unsigned int **is_stable**:1
- unsigned int **is_fast**:1
- unsigned int **is_running**:1
- unsigned int **is_named**:1
- unsigned int **is_unnamed**:1
- unsigned int **is_valid**:1
- unsigned int **is_v2_dir**:1
- unsigned int **is_possible_guard**:1
- unsigned int **is_bad_exit**:1
- unsigned int **is_bad_directory**:1
- unsigned int **is_hs_dir**:1
- unsigned int **version_known**:1
- unsigned int **version_supports_begindir**:1
- unsigned int **version_supports_conditional_consensus**:1
- unsigned int **version_supports_extrainfo_upload**:1
- unsigned int **version_supports_v3_dir**:1
- unsigned int **has_bandwidth**:1
- unsigned int **has_exitsummary**:1
- unsigned int **has_measured_bw**:1
- uint32_t **measured_bw**
- uint32_t **bandwidth**
- char * **exitsummary**
- unsigned int **need_to_mirror**:1
- unsigned int **name_lookup_warned**:1
- time_t **last_dir_503_at**
- **download_status_t dl_status**

3.102.1 Detailed Description

Contents of a single router entry in a network status object.

3.102.2 Field Documentation

3.102.2.1 time_t routerstatus_t::published_on

When was this router published?

Referenced by client_would_use_router(), compare_vote_rs(), compute_routerstatus_consensus(), routerstatus_format_entry(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.2 char routerstatus_t::nickname[MAX_NICKNAME_LEN+1]

The nickname this router says it has.

Referenced by add_trusted_dir_server(), compare_vote_rs(), directory_initiate_command_routerstatus_rend(), directory_post_to_hs_dir(), dirserv_would_reject_router(), hid_serv_get_responsible_directories(), routerset_contains_routerstatus(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.3 char routerstatus_t::identity_digest[DIGEST_LEN]

Digest of the router's identity key.

Referenced by add_trusted_dir_server(), compare_digest_to_routerstatus_entry(), compare_routerstatus_entries(), compare_vote_rs(), directory_initiate_command_routerstatus_rend(), directory_post_to_hs_dir(), dirserv_would_reject_router(), hid_serv_responsible_for_desc_id(), lookup_last_hid_serv_request(), networkstatus_parse_vote_from_string(), routerset_contains_routerstatus(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), set_routerstatus_from_routerinfo(), and smartlist_choose_by_bandwidth().

3.102.2.4 char routerstatus_t::descriptor_digest[DIGEST_LEN]

Digest of the router's most recent descriptor.

Referenced by compare_vote_rs(), router_add_to_routerlist(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), set_routerstatus_from_routerinfo(), and signed_desc_digest_is_recognized().

3.102.2.5 uint32_t routerstatus_t::addr

IPv4 address for this router.

Referenced by add_trusted_dir_server(), compare_vote_rs(), directory_initiate_command_routerstatus_rend(), directory_post_to_hs_dir(), dirserv_would_reject_router(), routerset_contains_routerstatus(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.6 uint16_t routerstatus_t::or_port

OR port for this router.

Referenced by add_trusted_dir_server(), compare_vote_rs(), directory_initiate_command_routerstatus_rend(), directory_post_to_hs_dir(), dirserv_would_reject_router(), routerset_contains_routerstatus(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.7 uint16_t routerstatus_t::dir_port

Directory port for this router.

Referenced by add_trusted_dir_server(), compare_vote_rs(), directory_initiate_command_routerstatus_rend(), hid_serv_get_responsible_directories(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), set_routerstatus_from_routerinfo(), and smartlist_choose_by_bandwidth_weights().

3.102.2.8 unsigned int routerstatus_t::is_authority

True iff this router is an authority.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.9 unsigned int routerstatus_t::is_exit

True iff this router is a good exit.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), set_routerstatus_from_routerinfo(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

3.102.2.10 unsigned int routerstatus_t::is_stable

True iff this router stays up a long time.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.11 unsigned int routerstatus_t::is_fast

True iff this router has good bandwidth.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), set_routerstatus_from_routerinfo(), and smartlist_choose_by_bandwidth().

3.102.2.12 unsigned int routerstatus_t::is_running

True iff this router is up.

Referenced by clear_status_flags_on_sybil(), client_would_use_router(), router_set_status(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.13 unsigned int routerstatus_t::is_named

True iff "nickname" belongs to this router.

Referenced by clear_status_flags_on_sybil(), routerset_contains_routerstatus(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.14 unsigned int routerstatus_t::is_unnamed

True iff "nickname" belongs to another router.

Referenced by routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.15 unsigned int routerstatus_t::is_valid

True iff this router isn't invalid.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.16 unsigned int routerstatus_t::is_v2_dir

True iff this router can serve directory information with v2 of the directory protocol. (All directory caches cache v1 directories.)

Referenced by clear_status_flags_on_sybil(), router_get_my_share_of_directory_requests(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.17 unsigned int routerstatus_t::is_possible_guard

True iff this router would be a good choice as an entry guard.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), set_routerstatus_from_routerinfo(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

3.102.2.18 unsigned int routerstatus_t::is_bad_exit

True iff this node is a bad choice for an exit node.

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.19 unsigned int routerstatus_t::is_bad_directory

Do we think this directory is junky, underpowered, or otherwise useless?

Referenced by clear_status_flags_on_sybil(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.20 unsigned int routerstatus_t::is_hs_dir

True iff this router is a v2-or-later hidden service directory.

Referenced by clear_status_flags_on_sybil(), hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

3.102.2.21 unsigned int routerstatus_t::version_known

True iff we know version info for this router. (i.e., a "v" entry was included.) We'll replace all these with a big **tor_version_t** (p. 245) or a char[] if the number of traits we care about ever becomes incredibly big.

Referenced by routerstatus_has_changed(), and routerstatus_parse_entry_from_string().

3.102.2.22 unsigned int routerstatus_t::version_supports_begindir

True iff this router is a version that supports BEGIN_DIR cells.

Referenced by add_trusted_dir_server(), directory_initiate_command_routerstatus_rend(), routerstatus_has_changed(), and routerstatus_parse_entry_from_string().

3.102.2.23 unsigned int routerstatus_t::version_supports_conditional_consensus

True iff this router is a version that supports conditional consensus downloads (signed by list of authorities).

Referenced by add_trusted_dir_server(), directory_initiate_command_routerstatus_rend(), routerstatus_has_changed(), and routerstatus_parse_entry_from_string().

3.102.2.24 unsigned int routerstatus_t::version_supports_extrainfo_upload

True iff this router is a version that we can post extrainfo docs to.

Referenced by router_supports_extrainfo(), routerstatus_has_changed(), and routerstatus_parse_entry_from_string().

3.102.2.25 unsigned int routerstatus_t::version_supports_v3_dir

True iff this router is a version that, if it caches directory info, we can get v3 downloads from.

Referenced by router_get_my_share_of_directory_requests(), routerstatus_has_changed(), and routerstatus_parse_entry_from_string().

3.102.2.26 unsigned int routerstatus_t::has_bandwidth

The vote/consensus had bw info

Referenced by routerstatus_format_entry(), routerstatus_parse_entry_from_string(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

3.102.2.27 unsigned int routerstatus_t::has_exitsummary

The vote/consensus had exit summaries

Referenced by routerstatus_parse_entry_from_string().

3.102.2.28 unsigned int routerstatus_t::has_measured_bw

The vote/consensus had a measured bw

Referenced by measured_bw_line_apply(), routerstatus_format_entry(), and routerstatus_parse_entry_from_string().

3.102.2.29 uint32_t routerstatus_t::measured_bw

Measured bandwidth (capacity) of the router

Referenced by measured_bw_line_apply(), routerstatus_format_entry(), and routerstatus_parse_entry_from_string().

3.102.2.30 uint32_t routerstatus_t::bandwidth

Bandwidth (capacity) of the router as reported in the vote/consensus, in kilobytes/sec.

Referenced by routerstatus_format_entry(), routerstatus_parse_entry_from_string(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

3.102.2.31 char* routerstatus_t::exitsummary

exit policy summary - XXX weasel: this probably should not stay a string.

Referenced by routerstatus_free(), routerstatus_parse_entry_from_string(), and vote_routerstatus_free().

3.102.2.32 unsigned int routerstatus_t::need_to_mirror

True if we, as a directory mirror, want to download the corresponding routerinfo from the authority who gave us this routerstatus. (That is, if we don't have the routerinfo, and if we haven't already tried to get it from this authority.) Applies in v2 networkstatus document only.

Referenced by router_add_to_routerlist().

3.102.2.33 unsigned int routerstatus_t::name_lookup_warned

Have we warned the user for referring to this (unnamed) router by nickname?

3.102.2.34 time_t routerstatus_t::last_dir_503_at

When did this router last tell us that it was too busy to serve directory info?

Referenced by connection_dir_client_reached_eof().

The documentation for this struct was generated from the following file:

- **or.h**

3.103 service_fns Struct Reference

Public Member Functions

- **BOOL** (WINAPI ∗ChangeServiceConfig2A_fn)(SC_HANDLE hService
- **BOOL** (WINAPI ∗CloseServiceHandle_fn)(SC_HANDLE hSCObject)
- **BOOL** (WINAPI ∗ControlService_fn)(SC_HANDLE hService
- **SC_HANDLE** (WINAPI ∗CreateServiceA_fn)(SC_HANDLE hSCManager
- **BOOL** (WINAPI ∗DeleteService_fn)(SC_HANDLE hService)
- **SC_HANDLE** (WINAPI ∗OpenSCManagerA_fn)(LPCTSTR lpMachineName
- **SC_HANDLE** (WINAPI ∗OpenServiceA_fn)(SC_HANDLE hSCManager
- **BOOL** (WINAPI ∗QueryServiceStatus_fn)(SC_HANDLE hService
- **SERVICE_STATUS_HANDLE** (WINAPI ∗RegisterServiceCtrlHandlerA_fn)(LPCTSTR lpServiceName
- **BOOL** (WINAPI ∗SetServiceStatus_fn)(SERVICE_STATUS_HANDLE
- **BOOL** (WINAPI ∗StartServiceCtrlDispatcherA_fn)(const SERVICE_TABLE_ENTRY ∗lpServiceTable)
- **BOOL** (WINAPI ∗StartServiceA_fn)(SC_HANDLE hService
- **BOOL** (WINAPI ∗LookupAccountNameA_fn)(LPCTSTR lpSystemName

Data Fields

- int **loaded**
- DWORD **dwInfoLevel**
- DWORD LPVOID **lpInfo**
- DWORD **dwControl**
- DWORD LPSERVICE_STATUS **lpServiceStatus**
- LPCTSTR **lpServiceName**
- LPCTSTR LPCTSTR **lpDisplayName**
- LPCTSTR LPCTSTR DWORD **dwDesiredAccess**
- LPCTSTR LPCTSTR DWORD DWORD **dwServiceType**
- LPCTSTR LPCTSTR DWORD DWORD DWORD **dwStartType**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD **dwErrorControl**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD LPCTSTR **lpBinaryPathName**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD LPCTSTR LPCTSTR **lpLoadOrderGroup**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD LPCTSTR LPCTSTR LPDWORD **lpdwTagId**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD LPCTSTR LPCTSTR LPDWORD LPCTSTR **lpDependencies**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD LPCTSTR LPCTSTR LPDWORD LPCTSTR LPCTSTR **lpServiceStartName**
- LPCTSTR LPCTSTR DWORD DWORD DWORD DWORD LPCTSTR LPCTSTR LPDWORD LPCTSTR LPCTSTR LPCTSTR **lpPassword**
- LPCTSTR **lpDatabaseName**
- LPCTSTR DWORD **dwDesiredAccess**
- LPSERVICE_STATUS **lpServiceStatus**
- LPHANDLER_FUNCTION **lpHandlerProc**
- **LPSERVICE_STATUS**

- DWORD **dwNumServiceArgs**
- DWORD LPCTSTR * **lpServiceArgVectors**
- LPCTSTR **lpAccountName**
- LPCTSTR PSID **Sid**
- LPCTSTR PSID LPDWORD **cbSid**
- LPCTSTR PSID LPDWORD LPTSTR **ReferencedDomainName**
- LPCTSTR PSID LPDWORD LPTSTR LPDWORD **cchReferencedDomainName**
- LPCTSTR PSID LPDWORD LPTSTR LPDWORD PSID_NAME_USE **peUse**

3.103.1 Detailed Description

Struct to hold dynamically loaded NT-service related function pointers.

The documentation for this struct was generated from the following file:

- ntmain.c

3.104 signed_descriptor_t Struct Reference

`#include <or.h>`

Data Fields

- char * **signed_descriptor_body**
- size_t **annotations_len**
- size_t **signed_descriptor_len**
- char **signed_descriptor_digest** [DIGEST_LEN]
- char **identity_digest** [DIGEST_LEN]
- time_t **published_on**
- char **extra_info_digest** [DIGEST_LEN]
- **download_status_t ei_dl_status**
- **saved_location_t saved_location**
- off_t **saved_offset**
- int **routerlist_index**
- time_t **last_listed_as_valid_until**
- unsigned int **do_not_cache**: 1
- unsigned int **is_extrainfo**: 1
- unsigned int **extrainfo_is_bogus**: 1
- unsigned int **send_unencrypted**: 1

3.104.1 Detailed Description

Information need to cache an onion router's descriptor.

3.104.2 Field Documentation

3.104.2.1 char* signed_descriptor_t::signed_descriptor_body

Pointer to the raw server descriptor, preceded by annotations. Not necessarily NUL-terminated. If saved_location is SAVED_IN_CACHE, this pointer is null.

Referenced by authority_cert_dup(), authority_cert_free(), authority_cert_parse_from_string(), dirserv_add_descriptor(), dirvote_add_vote(), extrainfo_free(), extrainfo_parse_entry_from_string(), format_networkstatus_vote(), init_keys(), router_parse_entry_from_string(), router_rebuild_descriptor(), router_upload_dir_desc_to_dirservers(), routerinfo_free(), routerlist_reparse_old(), signed_descriptor_free(), signed_descriptor_from_routerinfo(), and signed_descriptor_get_body_impl().

3.104.2.2 size_t signed_descriptor_t::annotations_len

Length of the annotations preceding the server descriptor.

Referenced by router_parse_entry_from_string(), routerlist_reparse_old(), signed_desc_append_to_journal(), and signed_descriptor_get_body_impl().

3.104.2.3 size_t signed_descriptor_t::signed_descriptor_len

Length of the server descriptor.

Referenced by authority_cert_dup(), authority_cert_parse_from_string(), connection_dirserv_add_servers_to_outbuf(), dirserv_add_descriptor(), dirserv_add_extrainfo(), dirserv_estimate_data_size(), extrainfo_insert(), extrainfo_parse_entry_from_string(), format_networkstatus_vote(), getinfo_helper_dir(), munge_extrainfo_into_routerinfo(), router_get_my_descriptor(), router_parse_entry_from_string(), router_rebuild_descriptor(), router_upload_dir_desc_to_dirservers(), routerlist_remove(), routerlist_remove_old(), routerlist_reparse_old(), routerlist_replace(), signed_desc_append_to_journal(), and signed_descriptor_get_body_impl().

3.104.2.4 char signed_descriptor_t::signed_descriptor_digest[DIGEST_LEN]

Digest of the server descriptor, computed as specified in dir-spec.txt.

Referenced by already_have_cert(), authority_cert_parse_from_string(), extrainfo_insert(), extrainfo_parse_entry_from_string(), router_add_to_routerlist(), router_extrainfo_digest_is_me(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_incompatible_with_extrainfo(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), routerlist_replace(), routerstatus_format_entry(), set_routerstatus_from_routerinfo(), and signed_desc_digest_is_recognized().

3.104.2.5 char signed_descriptor_t::identity_digest[DIGEST_LEN]

Identity digest of the router.

Referenced by _compare_old_routers_by_identity(), _compare_routerinfo_by_id_digest(), _compare_routerinfo_by_ip_and_bw(), add_an_entry_guard(), already_have_cert(), authority_cert_parse_from_string(), circuit_rep_hist_note_result(), connection_ap_can_use_exit(), consider_testing_reachability(), directory_get_from_dirserver(), dirserv_add_descriptor(), dirserv_add_extrainfo(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_set_router_is_running(), dirserv_single_reachability_test(), dirserv_thinks_router_is_unreliable(), dirvote_act(), dirvote_add_vote(), dirvote_get_vote(), extend_info_from_router(), extrainfo_dump_to_string(), extrainfo_insert(), extrainfo_parse_entry_from_string(), format_networkstatus_vote(), getinfo_helper_misc(), hid_serv_acting_as_directory(), hid_serv_responsible_for_desc_id(), list_single_server_status(), networkstatus_check_document_signature(), networkstatus_parse_vote_from_string(), router_add_to_routerlist(), router_get_my_share_of_directory_requests(), router_get_verbose_nickname(), router_hex_digest_matches(), router_is_me(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_get_configured_bridge(), routerinfo_incompatible_with_extrainfo(), routerlist_insert(), routerlist_remove(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), routerlist_replace(), routerset_contains_router(), set_routerstatus_from_routerinfo(), signed_desc_digest_is_recognized(), smartlist_choose_by_bandwidth(), smartlist_choose_by_bandwidth_weights(), trusted_dirs_load_certs_from_string(), and update_extrainfo_downloads().

3.104.2.6 time_t signed_descriptor_t::published_on

Declared publication time of the descriptor.

Referenced by _compare_old_routers_by_identity(), _compare_signed_descriptors_by_age(), authdir_wants_to_reject_router(), authority_cert_get_newest_by_id(), authority_cert_parse_from_string(), dirserv_add_descriptor(), dirserv_get_routerdescs(), extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), get_signed_descriptor_by_fp(), real_uptime(), rep_hist_get_router_stability_doc(), router_add_to_routerlist(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_is_active(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_incompatible_with_extrainfo(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), set_routerstatus_from_routerinfo(), trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

3.104.2.7 char signed_descriptor_t::extra_info_digest[DIGEST_LEN]

For routerdescs only: digest of the corresponding extrainfo.

Referenced by get_signed_descriptor_by_fp(), router_dump_router_to_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerinfo_incompatible_with_extrainfo(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_replace(), and update_extrainfo_downloads().

3.104.2.8 download_status_t signed_descriptor_t::ei_dl_status

For routerdescs only: Status of downloading the corresponding extrainfo.

Referenced by dir_routerdesc_download_failed(), and update_extrainfo_downloads().

3.104.2.9 saved_location_t signed_descriptor_t::saved_location

Where is the descriptor saved?

Referenced by authority_cert_dup(), authority_cert_parse_from_string(), router_get_my_descriptor(), router_parse_list_from_string(), signed_desc_append_to_journal(), and signed_descriptor_get_body_impl().

3.104.2.10 off_t signed_descriptor_t::saved_offset

If saved_location is SAVED_IN_CACHE or SAVED_IN_JOURNAL, the offset of this descriptor in the corresponding file.

Referenced by router_parse_list_from_string(), signed_desc_append_to_journal(), and signed_descriptor_get_body_impl().

3.104.2.11 int signed_descriptor_t::routerlist_index

What position is this descriptor within routerlist->routers or routerlist->old_routers? -1 for none.

Referenced by router_parse_entry_from_string(), router_rebuild_descriptor(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_routers(), routerlist_reparse_old(), routerlist_replace(), and signed_descriptor_from_routerinfo().

3.104.2.12 time_t signed_descriptor_t::last_listed_as_valid_until

The valid-until time of the most recent consensus that listed this descriptor, or a bit after the publication time of the most recent v2 networkstatus that listed it. 0 for "never listed in a consensus or status, so far as we know."

Referenced by router_set_networkstatus_v2(), routerlist_remove_old_cached_routers_with_id(), and routerlist_remove_old_routers().

The documentation for this struct was generated from the following file:

- **or.h**

3.105 sized_chunk_t Struct Reference

`#include <util.h>`

Data Fields

- const char * **bytes**
- size_t **len**

3.105.1 Detailed Description

An ad-hoc type to hold a string of characters and a count; used by write_chunks_to_file.

The documentation for this struct was generated from the following file:

- **util.h**

3.106 smartlist_t Struct Reference

`#include <container.h>`

Data Fields

- void ** **list**
- int **num_used**
- int **capacity**

3.106.1 Detailed Description

A resizeable list of pointers, with associated helpful functionality.

The members of this struct are exposed only so that macros and inlines can use them; all access to smartlist internals should go through the functions and macros defined here.

3.106.2 Field Documentation

3.106.2.1 void** smartlist_t::list

list has enough capacity to store exactly **capacity** elements before it needs to be resized. Only the first **num_used** (<= capacity) elements point to valid data.

Referenced by smartlist_add(), smartlist_add_all(), smartlist_bsearch_idx(), smartlist_create(), smartlist_del(), smartlist_del_keeporder(), smartlist_digest_isin(), smartlist_ensure_capacity(), smartlist_free(), smartlist_get_most_frequent(), smartlist_heapify(), smartlist_insert(), smartlist_intersect(), smartlist_isin(), smartlist_join_strings2(), smartlist_overlap(), smartlist_pop_last(), smartlist_pqueue_add(), smartlist_pqueue_assert_ok(), smartlist_pqueue_pop(), smartlist_pqueue_remove(), smartlist_remove(), smartlist_reverse(), smartlist_sort(), smartlist_string_isin(), smartlist_string_isin_case(), smartlist_string_pos(), smartlist_string_remove(), smartlist_subtract(), and smartlist_uniq().

The documentation for this struct was generated from the following file:

- container.h

3.107 socks_request_t Struct Reference

`#include <or.h>`

Data Fields

- char **socks_version**
- int **command**
- size_t **replylen**
- char **reply** [MAX_SOCKS_REPLY_LEN]
- char **address** [MAX_SOCKS_ADDR_LEN]
- uint16_t **port**
- unsigned int **has_finished**: 1

3.107.1 Detailed Description

State of a SOCKS request from a user to an OP. Also used to encode other information for non-socks user request (such as those on TransPort and DNSPort)

3.107.2 Field Documentation

3.107.2.1 char socks_request_t::socks_version

Which version of SOCKS did the client use? One of "0, 4, 5" – where 0 means that no socks handshake ever took place, and this is just a stub connection (e.g. see **connection_ap_make_link()** (p. 441)).

Referenced by connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), connection_ap_make_link(), and fetch_from_buf_socks().

3.107.2.2 int socks_request_t::command

What is this stream's goal? One from the above list.

Referenced by _connection_mark_unattached_ap(), circuit_get_open_circ_or_launch(), connection_ap_can_use_exit(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_process_relay_cell_not_open(), control_event_stream_status(), dnsserv_launch_request(), dnsserv_resolved(), evdns_server_callback(), fetch_from_buf_socks(), and getinfo_helper_events().

3.107.2.3 size_t socks_request_t::replylen

Length of **reply**.

Referenced by connection_ap_handshake_process_socks(), and fetch_from_buf_socks().

3.107.2.4 char socks_request_t::reply[MAX_SOCKS_REPLY_LEN]

Write an entry into this string if we want to specify our own socks reply, rather than using the default socks4 or socks5 socks reply. We use this for the two-stage socks5 handshake.

Referenced by connection_ap_handshake_process_socks(), and fetch_from_buf_socks().

3.107.2.5 char socks_request_t::address[MAX_SOCKS_ADDR_LEN]

What address did the client ask to connect to/resolve?

Referenced by circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), connection_ap_can_use_exit(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_ap_get_original_destination(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_resolved(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_edge_process_relay_cell_not_open(), consider_recording_trackhost(), directory_all_unreachable(), dnsserv_launch_request(), dnsserv_resolved(), evdns_server_callback(), fetch_from_buf_socks(), handle_control_redirectstream(), remap_event_helper(), tell_controller_about_resolved_result(), and write_stream_target_to_buf().

3.107.2.6 uint16_t socks_request_t::port

What port did the client ask to connect to?

Referenced by circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_stream_is_being_handled(), connection_ap_can_use_exit(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_ap_get_original_destination(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_begin(), connection_ap_make_link(), connection_ap_process_natd(), directory_all_unreachable(), fetch_from_buf_socks(), handle_control_redirectstream(), and write_stream_target_to_buf().

3.107.2.7 unsigned int socks_request_t::has_finished

Has the SOCKS handshake finished? Used to make sure we send back a socks reply for every connection.

Referenced by _connection_mark_unattached_ap(), assert_connection_ok(), connection_about_to_close_connection(), connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), connection_ap_make_link(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_reached_eof(), and connection_handle_read_impl().

The documentation for this struct was generated from the following file:

- **or.h**

3.108 token_rule_t Struct Reference

Data Fields

- const char * **t**
- **directory_keyword v**
- int **min_args**
- int **max_args**
- int **concat_args**
- **obj_syntax os**
- int **min_cnt**
- int **max_cnt**
- int **pos**
- int **is_annotation**

3.108.1 Detailed Description

Determines the parsing rules for a single token type.

3.108.2 Field Documentation

3.108.2.1 const char* token_rule_t::t

The string value of the keyword identifying the type of item.

Referenced by get_next_token(), and tokenize_string().

3.108.2.2 directory_keyword token_rule_t::v

The corresponding directory_keyword enum.

Referenced by get_next_token().

3.108.2.3 int token_rule_t::min_args

Minimum number of arguments for this item

Referenced by get_next_token().

3.108.2.4 int token_rule_t::max_args

Maximum number of arguments for this item

Referenced by get_next_token().

3.108.2.5 int token_rule_t::concat_args

If true, we concatenate all arguments for this item into a single string.

3.108.2.6 obj_syntax token_rule_t::os

Requirements on object syntax for this item.

Referenced by get_next_token().

3.108.2.7 int token_rule_t::min_cnt

Lowest number of times this item may appear in a document.

3.108.2.8 int token_rule_t::max_cnt

Highest number of times this item may appear in a document.

3.108.2.9 int token_rule_t::pos

One or more of AT_START/AT_END to limit where the item may appear in a document.

3.108.2.10 int token_rule_t::is_annotation

True iff this token is an annotation.

The documentation for this struct was generated from the following file:

- **routerparse.c**

3.109 tor_addr_t Struct Reference

`#include <address.h>`

Data Fields

- sa_family_t **family**
- union {
  struct in_addr **in_addr**
  struct in6_addr **in6_addr**
  } **addr**

3.109.1 Detailed Description

Holds an IPv4 or IPv6 address. (Uses less memory than struct sockaddr_storage.)

The documentation for this struct was generated from the following file:

- **address.h**

3.110 tor_lockfile_t Struct Reference

Data Fields

- char * **filename**
- int **fd**

3.110.1 Detailed Description

Represents a lockfile on which we hold the lock.

The documentation for this struct was generated from the following file:

- **compat.c**

3.111 tor_mmap_t Struct Reference

`#include <compat.h>`

Data Fields

- const char * **data**
- size_t **size**
- size_t **mapping_size**

3.111.1 Detailed Description

Represents an mmaped file. Allocated via tor_mmap_file; freed with tor_munmap_file.

3.111.2 Field Documentation

3.111.2.1 const char* tor_mmap_t::data

Mapping of the file's contents.

Referenced by microdesc_cache_reload(), router_reload_router_list_impl(), signed_descriptor_get_body_impl(), tor_mmap_file(), and tor_munmap_file().

3.111.2.2 size_t tor_mmap_t::size

Size of the file.

Referenced by microdesc_cache_reload(), router_reload_router_list_impl(), and tor_mmap_file().

3.111.2.3 size_t tor_mmap_t::mapping_size

Size of the actual mapping. (This is this file size, rounded up to the nearest page.)

Referenced by tor_mmap_file(), and tor_munmap_file().

The documentation for this struct was generated from the following file:

- compat.h

3.112 tor_mutex_t Struct Reference

`#include <compat.h>`

Data Fields

- pthread_mutex_t **mutex**

3.112.1 Detailed Description

A generic lock structure for multithreaded builds.

The documentation for this struct was generated from the following file:

- compat.h

3.113 tor_pthread_data_t Struct Reference

Data Fields

- void(∗ **func**)(void ∗)
- void ∗ **data**

3.113.1 Detailed Description

Wraps a void (∗)(void∗) function and its argument so we can invoke them in a way pthreads would expect.

The documentation for this struct was generated from the following file:

- **compat.c**

3.114 tor_tls_context_t Struct Reference

Data Fields

- int **refcnt**
- SSL_CTX ∗ **ctx**
- X509 ∗ **my_cert**
- X509 ∗ **my_id_cert**
- **crypto_pk_env_t** ∗ **key**

3.114.1 Detailed Description

Structure holding the TLS state for a single connection.

The documentation for this struct was generated from the following file:

- **tortls.c**

3.115 tor_tls_t Struct Reference

Public Member Functions

- **HT_ENTRY** (**tor_tls_t**)

Data Fields

- state **__pad0__**: 3
- unsigned int **isServer**:1
- unsigned int **wasV2Handshake**:1
- unsigned int **got_renegotiate**:1
- size_t **wantwrite_n**
- unsigned long **last_write_count**
- unsigned long **last_read_count**
- void(∗ **negotiated_callback**)(**tor_tls_t** ∗tls, void ∗arg)
- void ∗ **callback_arg**

3.115.1 Detailed Description

Holds a SSL object and its associated data. Members are only accessed from within **tortls.c** (p. 1219).

3.115.2 Field Documentation

3.115.2.1 state tor_tls_t::__pad0__

The current SSL state, depending on which operations have completed successfully.

3.115.2.2 unsigned int tor_tls_t::isServer

True iff this is a server-side connection

Referenced by tor_tls_handshake(), tor_tls_is_server(), tor_tls_new(), tor_tls_renegotiate(), and tor_tls_used_v1_handshake().

3.115.2.3 unsigned int tor_tls_t::wasV2Handshake

True iff the original handshake for this connection used the updated version of the connection protocol (client sends different cipher list, server sends only one certificate).

Referenced by tor_tls_handshake(), tor_tls_server_info_callback(), and tor_tls_used_v1_handshake().

3.115.2.4 unsigned int tor_tls_t::got_renegotiate

True iff we should call negotiated_callback when we're done reading.

Referenced by tor_tls_read(), tor_tls_server_info_callback(), and tor_tls_set_renegotiate_callback().

3.115.2.5 size_t tor_tls_t::wantwrite_n

0 normally, >0 if we returned wantwrite last time.

Referenced by tor_tls_get_forced_write_size(), tor_tls_new(), and tor_tls_write().

3.115.2.6 unsigned long tor_tls_t::last_write_count

Last values retrieved from BIO_number_read()/write(); see **tor_tls_get_n_raw_bytes()** (p. 1225) for usage.

Referenced by tor_tls_get_n_raw_bytes(), and tor_tls_new().

3.115.2.7 void(* tor_tls_t::negotiated_callback)(tor_tls_t *tls, void *arg)

If set, a callback to invoke whenever the client tries to renegotiate the handshake.

Referenced by tor_tls_free(), tor_tls_read(), tor_tls_server_info_callback(), and tor_tls_set_renegotiate_callback().

3.115.2.8 void* tor_tls_t::callback_arg

Argument to pass to negotiated_callback.

Referenced by tor_tls_read(), and tor_tls_set_renegotiate_callback().

The documentation for this struct was generated from the following file:

- **tortls.c**

3.116 tor_version_t Struct Reference

`#include <or.h>`

Public Types

- enum { **VER_PRE** = 0, **VER_RC** = 1, **VER_RELEASE** = 2 }

Data Fields

- int **major**
- int **minor**
- int **micro**
- enum tor_version_t:: { ... } **status**
- int **patchlevel**
- char **status_tag** [MAX_STATUS_TAG_LEN]
- int **svn_revision**
- int **git_tag_len**
- char **git_tag** [DIGEST_LEN]

3.116.1 Detailed Description

Structure to hold parsed Tor versions. This is a little messier than we would like it to be, because we changed version schemes with 0.1.0.

See version-spec.txt for the whole business.

3.116.2 Member Enumeration Documentation

3.116.2.1 anonymous enum

Release status. For version in the post-0.1 format, this is always VER_RELEASE.

3.116.3 Field Documentation

3.116.3.1 enum { ... } tor_version_t::status

Release status. For version in the post-0.1 format, this is always VER_RELEASE.

Referenced by tor_version_compare(), and tor_version_parse().

The documentation for this struct was generated from the following file:

- **or.h**

3.117 tor_zlib_state_t Struct Reference

Data Fields

- struct z_stream_s **stream**
- int **compress**

3.117.1 Detailed Description

Internal state for an incremental zlib compression/decompression. The body of this struct is not exposed.

The documentation for this struct was generated from the following file:

- **torgzip.c**

3.118 trusted_dir_server_t Struct Reference

`#include <or.h>`

Data Fields

- char * **description**
- char * **nickname**
- char * **address**
- uint32_t **addr**
- uint16_t **dir_port**
- uint16_t **or_port**
- char **digest** [DIGEST_LEN]
- char **v3_identity_digest** [DIGEST_LEN]
- unsigned int **is_running**:1
- unsigned int **has_accepted_serverdesc**:1
- **authority_type_t type**
- **download_status_t v2_ns_dl_status**
- time_t **addr_current_at**
- **routerstatus_t fake_status**

3.118.1 Detailed Description

Represents information about a single trusted directory server.

3.118.2 Field Documentation

3.118.2.1 char* trusted_dir_server_t::address

Hostname.

Referenced by add_trusted_dir_server(), trusted_dir_server_free(), and trusted_dirs_load_certs_from_string().

3.118.2.2 uint32_t trusted_dir_server_t::addr

IPv4 address.

Referenced by add_trusted_dir_server(), and trusted_dirs_load_certs_from_string().

3.118.2.3 uint16_t trusted_dir_server_t::dir_port

Directory port.

Referenced by add_trusted_dir_server(), and trusted_dirs_load_certs_from_string().

3.118.2.4 uint16_t trusted_dir_server_t::or_port

OR port: Used for tunneling connections.

Referenced by add_trusted_dir_server().

3.118.2.5 char trusted_dir_server_t::digest[DIGEST_LEN]

Digest of identity key.

Referenced by add_trusted_dir_server().

3.118.2.6 char trusted_dir_server_t::v3_identity_digest[DIGEST_LEN]

Digest of v3 (authority only, high-security) identity key.

Referenced by add_trusted_dir_server(), and init_keys().

3.118.2.7 unsigned int trusted_dir_server_t::is_running

True iff we think this server is running.

Referenced by add_trusted_dir_server(), and update_router_descriptor_cache_downloads_v2().

3.118.2.8 unsigned int trusted_dir_server_t::has_accepted_serverdesc

True iff this server has accepted the most recent server descriptor we tried to upload to it.

Referenced by connection_dir_client_reached_eof().

3.118.2.9 authority_type_t trusted_dir_server_t::type

What kind of authority is this? (Bitfield.)

Referenced by add_trusted_dir_server(), init_keys(), and router_set_networkstatus_v2().

3.118.2.10 download_status_t trusted_dir_server_t::v2_ns_dl_status

Status of downloading this server's v2 network status.

Referenced by dir_networkstatus_download_failed(), and router_set_networkstatus_v2().

3.118.2.11 time_t trusted_dir_server_t::addr_current_at

When was the document that we derived the address information from published?

Referenced by trusted_dirs_load_certs_from_string().

3.118.2.12 routerstatus_t trusted_dir_server_t::fake_status

Used when we need to pass this trusted dir_server_t to directory_initiate_command_∗ as a **routerstatus_t** (p. 218). Not updated by the router-status management code!

Referenced by add_trusted_dir_server(), connection_dir_client_reached_eof(), and update_consensus_router_descriptor_downloads().

The documentation for this struct was generated from the following file:

- **or.h**

3.119 unit_table_t Struct Reference

Data Fields

- const char * **unit**
- uint64_t **multiplier**

3.119.1 Detailed Description

Mapping from a unit name to a multiplier for converting that unit into a base unit.

The documentation for this struct was generated from the following file:

- **config.c**

3.120 var_cell_t Struct Reference

`#include <or.h>`

Data Fields

- uint8_t **command**
- **circid_t circ_id**
- uint16_t **payload_len**
- char **payload** [1]

3.120.1 Detailed Description

Parsed variable-length onion routing cell.

The documentation for this struct was generated from the following file:

- **or.h**

3.121 virtaddress_entry_t Struct Reference

Data Fields

- char * **ipv4_address**
- char * **hostname_address**

3.121.1 Detailed Description

Entry for mapping addresses to which virtual address we mapped them to.

The documentation for this struct was generated from the following file:

- **connection_edge.c**

3.122 vote_routerstatus_t Struct Reference

`#include <or.h>`

Data Fields

- **routerstatus_t status**
- uint64_t **flags**
- char * **version**
- vote_microdesc_hash_t * **microdesc**

3.122.1 Detailed Description

The claim about a single router, made in a vote.

3.122.2 Field Documentation

3.122.2.1 routerstatus_t vote_routerstatus_t::status

Underlying 'status' object for this router. Flags are redundant.

Referenced by compare_vote_rs(), compute_routerstatus_consensus(), networkstatus_parse_vote_from_string(), routerstatus_parse_entry_from_string(), and vote_routerstatus_free().

3.122.2.2 uint64_t vote_routerstatus_t::flags

Bit-field for all recognized flags; index into **networkstatus_t.known_flags** (p. 124).

Referenced by routerstatus_parse_entry_from_string().

3.122.2.3 char* vote_routerstatus_t::version

The version that the authority says this router is running.

Referenced by networkstatus_parse_vote_from_string(), routerstatus_parse_entry_from_string(), and vote_routerstatus_free().

The documentation for this struct was generated from the following file:

- **or.h**

3.123 vote_timing_t Struct Reference

`#include <or.h>`

Data Fields

- int **vote_interval**
- int **n_intervals_valid**
- int **vote_delay**
- int **dist_delay**

3.123.1 Detailed Description

Describes the schedule by which votes should be generated.

The documentation for this struct was generated from the following file:

- **or.h**

Chapter 4

File Documentation

4.1 address.c File Reference

Functions to use and manipulate the **tor_addr_t** (p. 237) structure.

`#include "orconfig.h"`

`#include "compat.h"`

`#include "util.h"`

`#include "address.h"`

`#include "log.h"`

`#include <sys/time.h>`

`#include <unistd.h>`

`#include <errno.h>`

`#include <netinet/in.h>`

`#include <arpa/inet.h>`

`#include <sys/socket.h>`

`#include <netdb.h>`

`#include <sys/param.h>`

`#include <stdarg.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <string.h>`

`#include <assert.h>`

Defines

- #define **MAX_ADDRESS_LENGTH** (TOR_ADDR_BUF_LEN+2+(1+INET_NTOA_BUF_LEN)+12+1)
- #define **TRISTATE**(a, b) (((a)<(b))?-1: (((a)==(b))?0:1))

Functions

- socklen_t **tor_addr_to_sockaddr** (const **tor_addr_t** *a, uint16_t port, struct sockaddr *sa_out, socklen_t len)
- int **tor_addr_from_sockaddr** (**tor_addr_t** *a, const struct sockaddr *sa, uint16_t *port_out)
- void **tor_addr_make_unspec** (**tor_addr_t** *a)
- int **tor_addr_lookup** (const char *name, uint16_t family, **tor_addr_t** *addr)
- int **tor_addr_is_internal** (const **tor_addr_t** *addr, int for_listening)
- const char * **tor_addr_to_str** (char *dest, const **tor_addr_t** *addr, int len, int decorate)
- int **tor_addr_parse_reverse_lookup_name** (**tor_addr_t** *result, const char *address, int family, int accept_regular)
- int **tor_addr_to_reverse_lookup_name** (char *out, size_t outlen, const **tor_addr_t** *addr)
- int **tor_addr_parse_mask_ports** (const char *s, **tor_addr_t** *addr_out, **maskbits_t** *maskbits_out, uint16_t *port_min_out, uint16_t *port_max_out)
- int **tor_addr_is_v4** (const **tor_addr_t** *addr)
- int **tor_addr_is_null** (const **tor_addr_t** *addr)
- int **tor_addr_is_loopback** (const **tor_addr_t** *addr)
- void **tor_addr_from_ipv4n** (**tor_addr_t** *dest, uint32_t v4addr)
- void **tor_addr_from_ipv6_bytes** (**tor_addr_t** *dest, const char *ipv6_bytes)
- void **tor_addr_from_in6** (**tor_addr_t** *dest, const struct in6_addr *in6)
- void **tor_addr_copy** (**tor_addr_t** *dest, const **tor_addr_t** *src)
- int **tor_addr_compare** (const **tor_addr_t** *addr1, const **tor_addr_t** *addr2, **tor_addr_comparison_t** how)
- int **tor_addr_compare_masked** (const **tor_addr_t** *addr1, const **tor_addr_t** *addr2, **maskbits_t** mbits, **tor_addr_comparison_t** how)
- unsigned int **tor_addr_hash** (const **tor_addr_t** *addr)
- char * **tor_dup_addr** (const **tor_addr_t** *addr)
- const char * **fmt_addr** (const **tor_addr_t** *addr)
- int **tor_addr_from_str** (**tor_addr_t** *addr, const char *src)
- int **tor_addr_port_parse** (const char *s, **tor_addr_t** *addr_out, uint16_t *port_out)
- int **get_interface_address6** (int severity, sa_family_t family, **tor_addr_t** *addr)
- int **is_internal_IP** (uint32_t ip, int for_listening)
- int **parse_addr_port** (int severity, const char *addrport, char **address, uint32_t *addr, uint16_t *port_out)
- int **addr_mask_get_bits** (uint32_t mask)
- int **addr_mask_cmp_bits** (uint32_t a1, uint32_t a2, **maskbits_t** bits)
- int **parse_port_range** (const char *port, uint16_t *port_min_out, uint16_t *port_max_out)
- int **parse_addr_and_port_range** (const char *s, uint32_t *addr_out, **maskbits_t** *maskbits_out, uint16_t *port_min_out, uint16_t *port_max_out)
- int **tor_inet_ntoa** (const struct in_addr *in, char *buf, size_t buf_len)
- char * **tor_dup_ip** (uint32_t addr)
- int **get_interface_address** (int severity, uint32_t *addr)

4.1.1 Detailed Description

Functions to use and manipulate the **tor_addr_t** (p. 237) structure.

4.1.2 Function Documentation

4.1.2.1 int addr_mask_cmp_bits (uint32_t *a1*, uint32_t *a2*, maskbits_t *bits*)

Compare two addresses **a1** and **a2** for equality under a netmask of **mbits** bits. Return -1, 0, or 1.

XXXX_IP6 Temporary function to allow masks as bitcounts everywhere. This will be replaced with an IPv6-aware version as soon as 32-bit addresses are no longer passed around.

Referenced by address_is_in_virtual_range(), addressmap_get_virtual_address(), and parse_virtual_addr_network().

4.1.2.2 int addr_mask_get_bits (uint32_t *mask*)

If **mask** is an address mask for a bit-prefix, return the number of bits. Otherwise, return -1.

Referenced by parse_addr_and_port_range(), and tor_addr_parse_mask_ports().

4.1.2.3 const char∗ fmt_addr (const tor_addr_t ∗ *addr*)

Return a string representing the address **addr**. This string is statically allocated, and must not be freed. Each call to **fmt_addr** invalidates the last result of the function. This function is not thread-safe.

References tor_addr_to_str().

Referenced by circuit_extend(), circuit_handle_first_hop(), connection_edge_finished_connecting(), connection_handle_listener_read(), connection_proxy_connect(), fetch_bridge_descriptors(), parse_bridge_line(), and rend_add_service().

4.1.2.4 int get_interface_address (int *severity*, uint32_t ∗ *addr*)

Set ∗**addr** to the host-order IPv4 address (if any) of whatever interface connects to the Internet. This address should only be used in checking whether our address has changed. Return 0 on success, -1 on failure.

References get_interface_address6(), and tor_addr_to_ipv4h().

Referenced by client_check_address_changed(), and resolve_my_address().

4.1.2.5 int get_interface_address6 (int *severity*, sa_family_t *family*, tor_addr_t ∗ *addr*)

Set ∗**addr** to the IP address (if any) of whatever interface connects to the Internet. This address should only be used in checking whether our address has changed. Return 0 on success, -1 on failure.

References LD_NET, tor_addr_from_sockaddr(), tor_assert, tor_close_socket(), and tor_open_socket().

Referenced by get_interface_address().

4.1.2.6 int is_internal_IP (uint32_t *ip*, int *for_listening*)

Return true iff **ip** (in host order) is an IP reserved to localhost, or reserved for local networks by RFC 1918.

References tor_addr_t::addr, tor_addr_t::family, tor_addr_t::in_addr, and tor_addr_is_internal().

Referenced by connection_ap_process_end_not_open(), connection_edge_process_relay_cell_not_open(), dirserv_router_has_valid_address(), http_set_address_origin(), options_validate(), resolve_my_address(), and router_new_address_suggestion().

4.1.2.7 int parse_addr_and_port_range (const char ∗ *s*, uint32_t ∗ *addr_out*, maskbits_t ∗ *maskbits_out*, uint16_t ∗ *port_min_out*, uint16_t ∗ *port_max_out*)

Parse a string **s** in the format of (IP(/mask|/mask-bits)?|∗)(:(∗|port(-maxport))?)?, setting the various ∗out pointers as appropriate. Return 0 on success, -1 on failure.

References addr_mask_get_bits(), escaped(), LD_GENERAL, parse_port_range(), tor_assert, tor_free, and tor_inet_aton().

Referenced by parse_virtual_addr_network().

4.1.2.8 int parse_addr_port (int *severity*, const char ∗ *addrport*, char ∗∗ *address*, uint32_t ∗ *addr*, uint16_t ∗ *port_out*)

Parse a string of the form "host[:port]" from **addrport**. If **address** is provided, set ∗**address** to a copy of the host portion of the string. If **addr** is provided, try to resolve the host portion of the string and store it into ∗**addr** (in host byte order). If **port_out** is provided, store the port number into ∗**port_out**, or 0 if no port is given. If **port_out** is NULL, then there must be no port number in **addrport**. Return 0 on success, -1 on failure.

References esc_for_log(), escaped(), LD_GENERAL, LD_NET, tor_assert, tor_free, tor_lookup_hostname(), and tor_parse_long().

Referenced by authority_cert_parse_from_string(), connection_exit_begin_conn(), create_inet_sockaddr(), options_validate(), parse_dir_server_line(), and retry_listeners().

4.1.2.9 int parse_port_range (const char ∗ *port*, uint16_t ∗ *port_min_out*, uint16_t ∗ *port_max_out*)

Parse a string **s** in the format of (∗|port(-maxport)?)?, setting the various ∗out pointers as appropriate. Return 0 on success, -1 on failure.

References escaped(), LD_GENERAL, tor_assert, and tor_parse_long().

Referenced by parse_addr_and_port_range(), router_parse_addr_policy_private(), and tor_addr_parse_mask_ports().

4.1.2.10 int tor_addr_compare (const tor_addr_t ∗ *addr1*, const tor_addr_t ∗ *addr2*, tor_addr_comparison_t *how*)

Given two addresses **addr1** and **addr2**, return 0 if the two addresses are equivalent under the mask mbits, less than 0 if addr1 precedes addr2, and greater than 0 otherwise.

Different address families (IPv4 vs IPv6) are always considered unequal if **how** is CMP_EXACT; otherwise, IPv6-mapped IPv4 addresses are considered equivalent to their IPv4 equivalents.

References tor_addr_compare_masked().

Referenced by cmp_single_addr_policy(), connection_or_get_for_extend(), connection_or_group_set_badness(), and policy_summary_add_item().

4.1.2.11 int tor_addr_compare_masked (const tor_addr_t ∗ *addr1*, const tor_addr_t ∗ *addr2*, maskbits_t *mbits*, tor_addr_comparison_t *how*)

As **tor_addr_compare()** (p. 258), but only looks at the first **mask** bits of the address.

Reduce over-specific masks (>128 for ipv6, >32 for ipv4) to 128 or 32.

The mask is interpreted relative to **addr1**, so that if a is ffff:1.2.3.4, and b is 3.4.5.6, tor_addr_compare_masked(a,b,100,CMP_SEMANTIC) is the same as -tor_addr_compare_masked(b,a,4,CMP_SEMANTIC).

We guarantee that the ordering from tor_addr_compare_masked is a total order on addresses, but not that it is any particular order, or that it will be the same from one version to the next.

References tor_addr_family(), tor_addr_is_v4(), tor_addr_to_ipv4h(), tor_assert, and tor_fragile_assert.

Referenced by addr_policy_covers(), addr_policy_intersects(), compare_known_tor_addr_to_addr_policy(), compare_known_tor_addr_to_addr_policy_noport(), and tor_addr_compare().

4.1.2.12 void tor_addr_copy (tor_addr_t ∗ *dest*, const tor_addr_t ∗ *src*)

Copy a **tor_addr_t** (p. 237) from **src** to **dest**.

References tor_assert.

Referenced by bridge_add_from_config(), connection_exit_begin_conn(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_or_connect(), connection_or_init_conn_from_address(), directory_initiate_command_rend(), dns_resolve_impl(), evdns_server_callback(), extend_info_alloc(), parse_port_config(), and tor_addr_port_parse().

4.1.2.13 void tor_addr_from_in6 (tor_addr_t ∗ *dest*, const struct in6_addr ∗ *in6*)

Set **dest** equal to the IPv6 address in the in6_addr **in6**.

References tor_addr_from_ipv6_bytes().

Referenced by connection_ap_get_original_destination(), tor_addr_from_sockaddr(), tor_addr_from_str(), tor_addr_lookup(), tor_addr_parse_mask_ports(), and tor_addr_parse_reverse_lookup_name().

4.1.2.14 void tor_addr_from_ipv4n (tor_addr_t ∗ *dest*, uint32_t *v4addr*)

Set **dest** to equal the IPv4 address in **v4addr** (given in network order).

References tor_addr_t::addr, tor_addr_t::family, tor_addr_t::in_addr, and tor_assert.

Referenced by connection_ap_get_original_destination(), decode_address_from_payload(), fetch_from_buf_socks(), rend_service_introduce(), and tor_addr_from_sockaddr().

4.1.2.15 void tor_addr_from_ipv6_bytes (tor_addr_t ∗ *dest*, const char ∗ *ipv6_bytes*)

Set **dest** to equal the IPv6 address in the 16 bytes at **ipv6_bytes**.

References tor_addr_t::addr, tor_addr_t::family, tor_addr_t::in6_addr, and tor_assert.

Referenced by decode_address_from_payload(), fetch_from_buf_socks(), and tor_addr_from_in6().

4.1.2.16 int tor_addr_from_sockaddr (tor_addr_t ∗ *a*, const struct sockaddr ∗ *sa*, uint16_t ∗ *port_out*)

Set the **tor_addr_t** (p. 237) in **a** to contain the socket address contained in **sa**.

References tor_addr_from_in6(), tor_addr_from_ipv4n(), tor_addr_make_unspec(), and tor_assert.

Referenced by connection_ap_get_original_destination(), connection_handle_listener_read(), evdns_server_callback(), and get_interface_address6().

4.1.2.17 int tor_addr_from_str (tor_addr_t ∗ *addr*, const char ∗ *src*)

Convert the string in **src** to a **tor_addr_t** (p. 237) **addr**. The string may be an IPv4 address, an IPv6 address, or an IPv6 address surrounded by square brackets.

Return an address family on success, or -1 if an invalid address string is provided.

References tor_addr_from_in6(), tor_assert, tor_free, and tor_inet_pton().

Referenced by circuit_get_open_circ_or_launch(), circuit_is_acceptable(), configure_nameservers(), connection_ap_fail_onehop(), dns_resolve_impl(), rend_parse_introduction_points(), and tor_addr_parse_reverse_lookup_name().

4.1.2.18 unsigned int tor_addr_hash (const tor_addr_t ∗ *addr*)

Return a hash code based on the address addr

References tor_addr_family(), tor_addr_to_ipv4h(), and tor_fragile_assert.

Referenced by policy_hash().

4.1.2.19 int tor_addr_is_internal (const tor_addr_t ∗ *addr*, int *for_listening*)

Return true iff **ip** is an IP reserved to localhost or local networks in RFC1918 or RFC4193 or RFC4291. (fec0::/10, deprecated by RFC3879, is also treated as internal for now.)

References LD_BUG, tor_addr_family(), tor_addr_is_v4(), and tor_addr_to_ipv4h().

Referenced by connection_is_rate_limited(), dns_resolve_impl(), exit_policy_is_general_exit_helper(), is_internal_IP(), and is_local_addr().

4.1.2.20 int tor_addr_is_loopback (const tor_addr_t ∗ *addr*)

Return true iff **addr** is a loopback address

References tor_addr_family(), tor_addr_to_ipv4h(), tor_assert, and tor_fragile_assert.

Referenced by connection_connect().

4.1.2.21 int tor_addr_is_null (const tor_addr_t ∗ *addr*)

Determine whether an address **addr** is null, either all zeroes or belonging to family AF_UNSPEC.

References LD_BUG, tor_addr_family(), tor_addr_to_ipv4n(), and tor_assert.

Referenced by compare_tor_addr_to_addr_policy(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_exit_begin_conn(), and router_compare_to_my_exit_policy().

4.1.2.22 int tor_addr_is_v4 (const tor_addr_t ∗ *addr*)

Determine whether an address is IPv4, either native or IPv4-mapped IPv6. Note that this is about representation only, as any decent stack will reject IPv4-mapped addresses received on the wire (and won't use them on the wire either).

References tor_addr_family(), and tor_assert.

Referenced by tor_addr_compare_masked(), tor_addr_is_internal(), and tor_addr_parse_mask_ports().

4.1.2.23 int tor_addr_lookup (const char ∗ *name*, uint16_t *family*, tor_addr_t ∗ *addr*)

Similar behavior to Unix gethostbyname: resolve **name**, and set ∗**addr** to the proper IP address and family. The **family** argument (which must be AF_INET, AF_INET6, or AF_UNSPEC) declares a *preferred* family, though another one may be returned if only one family is implemented for this address.

Return 0 on success, -1 on failure; 1 on transient failure.

References tor_addr_from_in6(), tor_assert, and tor_inet_pton().

Referenced by tor_addr_port_parse(), and tor_lookup_hostname().

4.1.2.24 void tor_addr_make_unspec (tor_addr_t ∗ *a*)

Set address **a** to the unspecified address. This address belongs to no family.

References tor_addr_t::family.

Referenced by connection_ap_make_link(), connection_handle_listener_read(), decode_address_from_payload(), and tor_addr_from_sockaddr().

4.1.2.25 int tor_addr_parse_mask_ports (const char * *s*, tor_addr_t * *addr_out*, maskbits_t * *maskbits_out*, uint16_t * *port_min_out*, uint16_t * *port_max_out*)

Parse a string **s** containing an IPv4/IPv6 address, and possibly a mask and port or port range. Store the parsed address in **addr_out**, a mask (if any) in **mask_out**, and port(s) (if any) in **port_min_out** and **port_max_out**.

The syntax is: Address OptMask OptPortRange Address ::= IPv4Address / "[" IPv6Address "]" / "*" OptMask ::= "/" Integer / OptPortRange ::= ":*" / ":" Integer / ":" Integer "-" Integer /

- If mask, minport, or maxport are NULL, we do not want these options to be set; treat them as an error if present.
- If the string has no mask, the mask is set to /32 (IPv4) or /128 (IPv6).
- If the string has one port, it is placed in both min and max port variables.
- If the string has no port(s), port_(min|max)_out are set to 1 and 65535.

Return an address family on success, or -1 if an invalid address string is provided.

References addr_mask_get_bits(), escaped(), LD_GENERAL, parse_port_range(), tor_addr_family(), tor_addr_from_in6(), tor_addr_from_ipv4h, tor_addr_is_v4(), tor_assert, tor_free, and tor_inet_pton().

Referenced by policy_expand_private(), policy_summary_add_item(), and router_parse_addr_policy().

4.1.2.26 int tor_addr_parse_reverse_lookup_name (tor_addr_t * *result*, const char * *address*, int *family*, int *accept_regular*)

Parse an .in-addr.arpa or .ip6.arpa address from **address**. Return 0 if this is not an .in-addr.arpa address or an .ip6.arpa address. Return -1 if this is an ill-formed .in-addr.arpa address or an .ip6.arpa address. Also return -1 if **family** is not AF_UNSPEC, and the parsed address family does not match **family**. On success, return 1, and store the result, if any, into **result**, if provided.

If **accept_regular** is set and the address is in neither recognized reverse lookup hostname format, try parsing the address as a regular IPv4 or IPv6 address too.

References hex_decode_digit(), INET_NTOA_BUF_LEN, strcasecmpend(), tor_addr_from_in6(), tor_addr_from_str(), and tor_inet_aton().

Referenced by connection_ap_handshake_send_resolve(), dns_resolve_impl(), and launch_resolve().

4.1.2.27 int tor_addr_port_parse (const char * *s*, tor_addr_t * *addr_out*, uint16_t * *port_out*)

Parse an address or address-port combination from **s**, resolve the address as needed, and put the result in **addr_out** and (optionally) **port_out**. Return 0 on success, negative on failure.

References eat_whitespace(), tor_addr_copy(), tor_addr_lookup(), tor_assert, tor_free, and tor_parse_long().

Referenced by parse_bridge_line(), and parse_port_config().

4.1.2.28 int tor_addr_to_reverse_lookup_name (char * *out*, size_t *outlen*, const tor_addr_t * *addr*)

Convert **addr** to an in-addr.arpa name or a .ip6.arpa name, and store the result in the **outlen**-byte buffer at **out**. Return 0 on success, -1 on failure.

References tor_addr_t::family, REVERSE_LOOKUP_NAME_BUF_LEN, tor_addr_to_ipv4h(), and tor_snprintf().

Referenced by connection_ap_handshake_send_resolve().

4.1.2.29 socklen_t tor_addr_to_sockaddr (const tor_addr_t * *a*, uint16_t *port*, struct sockaddr * *sa_out*, socklen_t *len*)

Convert the **tor_addr_t** (p. 237) in **a**, with port in **port**, into a socklen object in ***sa_out** of object size **len**. If not enough room is free, or on error, return -1. Else return the length of the sockaddr.

References tor_addr_family(), tor_addr_to_in6(), and tor_addr_to_ipv4n().

Referenced by configure_nameservers(), and connection_connect().

4.1.2.30 const char* tor_addr_to_str (char * *dest*, const tor_addr_t * *addr*, int *len*, int *decorate*)

Convert a **tor_addr_t** (p. 237) **addr** into a string, and store it in **dest** of size **len**. Returns a pointer to dest on success, or NULL on failure. If **decorate**, surround IPv6 addresses with brackets.

References tor_addr_t::addr, tor_addr_t::in6_addr, tor_addr_t::in_addr, tor_addr_family(), tor_assert, and tor_inet_ntop().

Referenced by connection_ap_get_original_destination(), fetch_from_buf_socks(), fmt_addr(), policy_write_item(), and tor_dup_addr().

4.1.2.31 char* tor_dup_addr (const tor_addr_t * *addr*)

Return a newly allocated string with a representation of **addr**.

References tor_addr_to_str().

Referenced by connection_exit_begin_conn(), connection_handle_listener_read(), connection_or_init_conn_from_address(), evdns_server_callback(), launch_direct_bridge_descriptor_fetch(), and rend_encode_v2_intro_points().

4.1.2.32 char* tor_dup_ip (uint32_t *addr*)

Given a host-order **addr**, call **tor_inet_ntop()** (p. 362) on it and return a strdup of the resulting address.

References tor_inet_ntop().

Referenced by directory_post_to_hs_dir(), dirserv_generate_networkstatus_vote_obj(), getinfo_helper_misc(), rep_hist_get_router_stability_doc(), router_rebuild_descriptor(), and trusted_dirs_load_certs_from_string().

4.1.2.33 int tor_inet_ntoa (const struct in_addr * *in*, char * *buf*, size_t *buf_len*)

Given an IPv4 in_addr struct ***in** (in network order, as usual), write it as a string into the **buf_len**-byte buffer in **buf**.

References tor_snprintf().

Referenced by addressmap_get_virtual_address(), client_dns_set_addressmap(), connection_edge_process_relay_cell_not_open(), directory_initiate_command_routerstatus_rend(), evdns_callback(), evdns_wildcard_check_callback(), fetch_from_buf_socks(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), log_addr_has_changed(), networkstatus_compute_consensus(), remap_event_helper(), resolve_my_address(), routerstatus_format_entry(), tell_controller_about_resolved_result(), and tor_inet_ntop().

4.2 address.h File Reference

Headers for **address.h** (p. 265).

`#include "orconfig.h"`

`#include "torint.h"`

`#include "compat.h"`

Data Structures

- struct **tor_addr_t**

Defines

- #define **tor_addr_to_in6_addr8**(x) tor_addr_to_in6(x) → s6_addr
- #define **tor_addr_to_in6_addr16**(x) S6_ADDR16(∗tor_addr_to_in6(x))
- #define **tor_addr_to_in6_addr32**(x) S6_ADDR32(∗tor_addr_to_in6(x))
- #define **TOR_ADDR_BUF_LEN** 48
- #define **tor_addr_eq**(a, b) (0==tor_addr_compare((a),(b),CMP_EXACT))
- #define **REVERSE_LOOKUP_NAME_BUF_LEN** 73
- #define **tor_addr_from_ipv4h**(dest, v4addr) tor_addr_from_ipv4n((dest), htonl(v4addr))
- #define **tor_addr_from_in**(dest, in) tor_addr_from_ipv4n((dest), (in) → s_addr);
- #define **INET_NTOA_BUF_LEN** 16

Typedefs

- typedef uint8_t **maskbits_t**

Enumerations

- enum **tor_addr_comparison_t** { **CMP_EXACT**, **CMP_SEMANTIC** }

Functions

- static INLINE struct in6_addr ∗ **tor_addr_to_in6** (const **tor_addr_t** ∗a)
- static INLINE uint32_t **tor_addr_to_ipv4n** (const **tor_addr_t** ∗a)
- static INLINE uint32_t **tor_addr_to_ipv4h** (const **tor_addr_t** ∗a)
- static INLINE uint32_t **tor_addr_to_mapped_ipv4h** (const **tor_addr_t** ∗a)
- static INLINE sa_family_t **tor_addr_family** (const **tor_addr_t** ∗a)
- static INLINE struct in_addr ∗ **tor_addr_to_in** (const **tor_addr_t** ∗a)
- static INLINE int **tor_addr_eq_ipv4h** (const **tor_addr_t** ∗a, uint32_t u)
- socklen_t **tor_addr_to_sockaddr** (const **tor_addr_t** ∗a, uint16_t port, struct sockaddr ∗sa_out, socklen_t len)
- int **tor_addr_from_sockaddr** (**tor_addr_t** ∗a, const struct sockaddr ∗sa, uint16_t ∗port_out)
- void **tor_addr_make_unspec** (**tor_addr_t** ∗a)
- int **tor_addr_lookup** (const char ∗name, uint16_t family, **tor_addr_t** ∗addr_out)

- char * **tor_dup_addr** (const **tor_addr_t** *addr) ATTR_MALLOC
- const char * **fmt_addr** (const **tor_addr_t** *addr)
- int **get_interface_address6** (int severity, sa_family_t family, **tor_addr_t** *addr)
- int **tor_addr_compare** (const **tor_addr_t** *addr1, const **tor_addr_t** *addr2, **tor_addr_comparison_t** how)
- int **tor_addr_compare_masked** (const **tor_addr_t** *addr1, const **tor_addr_t** *addr2, **maskbits_t** mask, **tor_addr_comparison_t** how)
- unsigned int **tor_addr_hash** (const **tor_addr_t** *addr)
- int **tor_addr_is_v4** (const **tor_addr_t** *addr)
- int **tor_addr_is_internal** (const **tor_addr_t** *ip, int for_listening) ATTR_PURE
- int **tor_addr_to_reverse_lookup_name** (char *out, size_t outlen, const **tor_addr_t** *addr)
- int **tor_addr_parse_reverse_lookup_name** (**tor_addr_t** *result, const char *address, int family, int accept_regular)
- int **tor_addr_port_parse** (const char *s, **tor_addr_t** *addr_out, uint16_t *port_out)
- int **tor_addr_parse_mask_ports** (const char *s, **tor_addr_t** *addr_out, **maskbits_t** *mask_out, uint16_t *port_min_out, uint16_t *port_max_out)
- const char * **tor_addr_to_str** (char *dest, const **tor_addr_t** *addr, int len, int decorate)
- int **tor_addr_from_str** (**tor_addr_t** *addr, const char *src)
- void **tor_addr_copy** (**tor_addr_t** *dest, const **tor_addr_t** *src)
- void **tor_addr_from_ipv4n** (**tor_addr_t** *dest, uint32_t v4addr)
- void **tor_addr_from_ipv6_bytes** (**tor_addr_t** *dest, const char *bytes)
- void **tor_addr_from_in6** (**tor_addr_t** *dest, const struct in6_addr *in6)
- int **tor_addr_is_null** (const **tor_addr_t** *addr)
- int **tor_addr_is_loopback** (const **tor_addr_t** *addr)
- int **is_internal_IP** (uint32_t ip, int for_listening) ATTR_PURE
- int **parse_addr_port** (int severity, const char *addrport, char **address, uint32_t *addr, uint16_t *port_out)
- int **parse_port_range** (const char *port, uint16_t *port_min_out, uint16_t *port_max_out)
- int **parse_addr_and_port_range** (const char *s, uint32_t *addr_out, **maskbits_t** *maskbits_out, uint16_t *port_min_out, uint16_t *port_max_out)
- int **addr_mask_get_bits** (uint32_t mask)
- int **addr_mask_cmp_bits** (uint32_t a1, uint32_t a2, **maskbits_t** bits)
- int **tor_inet_ntoa** (const struct in_addr *in, char *buf, size_t buf_len)
- char * **tor_dup_ip** (uint32_t addr) ATTR_MALLOC
- int **get_interface_address** (int severity, uint32_t *addr)

4.2.1 Detailed Description

Headers for **address.h** (p. 265).

4.2.2 Define Documentation

4.2.2.1 #define INET_NTOA_BUF_LEN 16

Length of a buffer to allocate to hold the results of tor_inet_ntoa.

Referenced by client_dns_set_addressmap(), client_dns_set_addressmap_impl(), connection_edge_process_relay_cell_not_open(), directory_initiate_command_routerstatus_rend(), evdns_callback(), evdns_wildcard_check_callback(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), log_addr_has_changed(), networkstatus_compute_consensus(), resolve_my_address(), routerstatus_format_entry(), tell_controller_about_resolved_result(), and tor_addr_parse_reverse_lookup_name().

4.2.2.2 #define REVERSE_LOOKUP_NAME_BUF_LEN 73

Longest length that can be required for a reverse lookup name.

Referenced by connection_ap_handshake_send_resolve(), and tor_addr_to_reverse_lookup_name().

4.2.2.3 #define tor_addr_eq(a, b) (0==tor_addr_compare((a),(b),CMP_EXACT))

Return true iff a and b are the same address. The comparison is done "exactly".

Referenced by circuit_dump_by_conn(), circuit_get_all_pending_on_or_conn(), circuit_is_acceptable(), circuit_n_conn_done(), command_process_netinfo_cell(), connection_ap_fail_onehop(), and connection_get_by_type_addr_port_purpose().

4.2.2.4 #define tor_addr_from_ipv4h(dest, v4addr) tor_addr_from_ipv4n((dest), htonl(v4addr))

Set **dest** to the IPv4 address encoded in **v4addr** in host order.

Referenced by addr_policy_permits_address(), circuit_extend(), compare_addr_to_addr_policy(), connection_or_init_conn_from_address(), connection_or_send_netinfo(), consider_testing_reachability(), directory_get_from_dirserver(), directory_initiate_command_routerstatus_rend(), directory_post_to_dirservers(), dirserv_single_reachability_test(), dns_found_answer(), dns_resolve_impl(), exit_policy_is_general_exit_helper(), extend_info_from_router(), fascist_firewall_allows_or(), parse_port_config(), router_find_exact_exit_enclave(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), routerset_contains_router(), routerset_contains_routerstatus(), tor_addr_parse_mask_ports(), and update_v2_networkstatus_cache_downloads().

4.2.3 Typedef Documentation

4.2.3.1 typedef uint8_t maskbits_t

The number of bits from an address to consider while doing a masked comparison.

4.2.4 Enumeration Type Documentation

4.2.4.1 enum tor_addr_comparison_t

Flag to specify how to do a comparison between addresses. In an "exact" comparison, addresses are equivalent only if they are in the same family with the same value. In a "semantic" comparison, IPv4 addresses match all IPv6 encodings of those addresses.

4.2.5 Function Documentation

4.2.5.1 int addr_mask_cmp_bits (uint32_t *a1*, uint32_t *a2*, maskbits_t *bits*)

Compare two addresses **a1** and **a2** for equality under a netmask of **mbits** bits. Return -1, 0, or 1.

XXXX_IP6 Temporary function to allow masks as bitcounts everywhere. This will be replaced with an IPv6-aware version as soon as 32-bit addresses are no longer passed around.

Referenced by address_is_in_virtual_range(), addressmap_get_virtual_address(), and parse_virtual_addr_network().

4.2.5.2 int addr_mask_get_bits (uint32_t *mask*)

If **mask** is an address mask for a bit-prefix, return the number of bits. Otherwise, return -1.

Referenced by parse_addr_and_port_range(), and tor_addr_parse_mask_ports().

4.2.5.3 const char∗ fmt_addr (const tor_addr_t ∗ *addr*)

Return a string representing the address **addr**. This string is statically allocated, and must not be freed. Each call to **fmt_addr** invalidates the last result of the function. This function is not thread-safe.

References tor_addr_to_str().

Referenced by circuit_extend(), circuit_handle_first_hop(), connection_edge_finished_connecting(), connection_handle_listener_read(), connection_proxy_connect(), fetch_bridge_descriptors(), parse_bridge_line(), and rend_add_service().

4.2.5.4 int get_interface_address (int *severity*, uint32_t ∗ *addr*)

Set ∗**addr** to the host-order IPv4 address (if any) of whatever interface connects to the Internet. This address should only be used in checking whether our address has changed. Return 0 on success, -1 on failure.

References get_interface_address6(), and tor_addr_to_ipv4h().

Referenced by client_check_address_changed(), and resolve_my_address().

4.2.5.5 int get_interface_address6 (int *severity*, sa_family_t *family*, tor_addr_t ∗ *addr*)

Set ∗**addr** to the IP address (if any) of whatever interface connects to the Internet. This address should only be used in checking whether our address has changed. Return 0 on success, -1 on failure.

References LD_NET, tor_addr_from_sockaddr(), tor_assert, tor_close_socket(), and tor_open_socket().

Referenced by get_interface_address().

4.2.5.6 int is_internal_IP (uint32_t *ip*, int *for_listening*)

Return true iff **ip** (in host order) is an IP reserved to localhost, or reserved for local networks by RFC 1918.

References tor_addr_t::addr, tor_addr_t::family, tor_addr_t::in_addr, and tor_addr_is_internal().

Referenced by connection_ap_process_end_not_open(), connection_edge_process_relay_cell_not_open(), dirserv_router_has_valid_address(), http_set_address_origin(), options_validate(), resolve_my_address(), and router_new_address_suggestion().

4.2.5.7 int parse_addr_and_port_range (const char ∗ *s*, uint32_t ∗ *addr_out*, maskbits_t ∗ *maskbits_out*, uint16_t ∗ *port_min_out*, uint16_t ∗ *port_max_out*)

Parse a string **s** in the format of (IP(/mask|/mask-bits)?|∗)(:(∗|port(-maxport))?)?, setting the various ∗out pointers as appropriate. Return 0 on success, -1 on failure.

References addr_mask_get_bits(), escaped(), LD_GENERAL, parse_port_range(), tor_assert, tor_free, and tor_inet_aton().

Referenced by parse_virtual_addr_network().

4.2.5.8 int parse_addr_port (int *severity*, const char ∗ *addrport*, char ∗∗ *address*, uint32_t ∗ *addr*, uint16_t ∗ *port_out*)

Parse a string of the form "host[:port]" from **addrport**. If **address** is provided, set ∗**address** to a copy of the host portion of the string. If **addr** is provided, try to resolve the host portion of the string and store it into ∗**addr** (in host byte order). If **port_out** is provided, store the port number into ∗**port_out**, or 0 if no port is given. If **port_out** is NULL, then there must be no port number in **addrport**. Return 0 on success, -1 on failure.

References esc_for_log(), escaped(), LD_GENERAL, LD_NET, tor_assert, tor_free, tor_lookup_hostname(), and tor_parse_long().

Referenced by authority_cert_parse_from_string(), connection_exit_begin_conn(), create_inet_sockaddr(), options_validate(), parse_dir_server_line(), and retry_listeners().

4.2.5.9 int parse_port_range (const char ∗ *port*, uint16_t ∗ *port_min_out*, uint16_t ∗ *port_max_out*)

Parse a string **s** in the format of (∗|port(-maxport)?)?, setting the various ∗out pointers as appropriate. Return 0 on success, -1 on failure.

References escaped(), LD_GENERAL, tor_assert, and tor_parse_long().

Referenced by parse_addr_and_port_range(), router_parse_addr_policy_private(), and tor_addr_parse_mask_ports().

4.2.5.10 int tor_addr_compare (const tor_addr_t ∗ *addr1*, const tor_addr_t ∗ *addr2*, tor_addr_comparison_t *how*)

Given two addresses **addr1** and **addr2**, return 0 if the two addresses are equivalent under the mask mbits, less than 0 if addr1 precedes addr2, and greater than 0 otherwise.

Different address families (IPv4 vs IPv6) are always considered unequal if **how** is CMP_EXACT; otherwise, IPv6-mapped IPv4 addresses are considered equivalent to their IPv4 equivalents.

References tor_addr_compare_masked().

Referenced by cmp_single_addr_policy(), connection_or_get_for_extend(), connection_or_group_set_badness(), and policy_summary_add_item().

4.2.5.11 int tor_addr_compare_masked (const tor_addr_t ∗ *addr1*, const tor_addr_t ∗ *addr2*, maskbits_t *mbits*, tor_addr_comparison_t *how*)

As **tor_addr_compare()** (p. 258), but only looks at the first **mask** bits of the address.

Reduce over-specific masks (>128 for ipv6, >32 for ipv4) to 128 or 32.

The mask is interpreted relative to **addr1**, so that if a is ffff:1.2.3.4, and b is 3.4.5.6, tor_addr_compare_masked(a,b,100,CMP_SEMANTIC) is the same as -tor_addr_compare_masked(b,a,4,CMP_SEMANTIC).

We guarantee that the ordering from tor_addr_compare_masked is a total order on addresses, but not that it is any particular order, or that it will be the same from one version to the next.

References tor_addr_family(), tor_addr_is_v4(), tor_addr_to_ipv4h(), tor_assert, and tor_fragile_assert.

Referenced by addr_policy_covers(), addr_policy_intersects(), compare_known_tor_addr_to_addr_policy(), compare_known_tor_addr_to_addr_policy_noport(), and tor_addr_compare().

4.2.5.12 void tor_addr_copy (tor_addr_t ∗ *dest*, const tor_addr_t ∗ *src*)

Copy a **tor_addr_t** (p. 237) from **src** to **dest**.

References tor_assert.

Referenced by bridge_add_from_config(), connection_exit_begin_conn(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_or_connect(), connection_or_init_conn_from_address(), directory_initiate_command_rend(), dns_resolve_impl(), evdns_server_callback(), extend_info_alloc(), parse_port_config(), and tor_addr_port_parse().

4.2.5.13 static INLINE int tor_addr_eq_ipv4h (const tor_addr_t ∗ *a*, uint32_t *u*) `[static]`

Return true iff **a** is an IPv4 address equal to the host-ordered address in **u**.

References tor_addr_t::family, and tor_addr_to_ipv4h().

Referenced by connection_or_init_conn_from_address(), directory_conn_is_self_reachability_test(), directory_handle_command_get(), router_new_address_suggestion(), and routerinfo_get_configured_bridge().

4.2.5.14 static INLINE sa_family_t tor_addr_family (const tor_addr_t ∗ *a*) `[static]`

Return the address family of **a**. Possible values are: AF_INET6, AF_INET, AF_UNSPEC.

References tor_addr_t::family.

Referenced by append_address_to_payload(), circuit_send_next_onion_skin(), connection_connect(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_connect(), connection_is_rate_limited(), connection_or_set_state_open(), connection_proxy_connect(), connection_send_socks5_connect(), dns_resolve_impl(), is_local_addr(), launch_resolve(), policy_write_item(), rend_parse_introduction_points(), router_add_exit_policy(), router_compare_to_my_exit_policy(), tor_addr_compare_masked(), tor_addr_hash(), tor_addr_is_internal(), tor_addr_is_loopback(), tor_addr_is_null(), tor_addr_is_v4(), tor_addr_parse_mask_ports(), tor_addr_to_sockaddr(), tor_addr_to_str(), and tor_lookup_hostname().

4.2.5.15 void tor_addr_from_in6 (tor_addr_t ∗ *dest*, const struct in6_addr ∗ *in6*)

Set **dest** equal to the IPv6 address in the in6_addr **in6**.

References tor_addr_from_ipv6_bytes().

Referenced by connection_ap_get_original_destination(), tor_addr_from_sockaddr(), tor_addr_from_str(), tor_addr_lookup(), tor_addr_parse_mask_ports(), and tor_addr_parse_reverse_lookup_name().

4.2.5.16 void tor_addr_from_ipv4n (tor_addr_t ∗ *dest*, uint32_t *v4addr*)

Set **dest** to equal the IPv4 address in **v4addr** (given in network order).

References tor_addr_t::addr, tor_addr_t::family, tor_addr_t::in_addr, and tor_assert.

Referenced by connection_ap_get_original_destination(), decode_address_from_payload(), fetch_from_buf_socks(), rend_service_introduce(), and tor_addr_from_sockaddr().

4.2.5.17 void tor_addr_from_ipv6_bytes (tor_addr_t ∗ *dest*, const char ∗ *ipv6_bytes*)

Set **dest** to equal the IPv6 address in the 16 bytes at **ipv6_bytes**.

References tor_addr_t::addr, tor_addr_t::family, tor_addr_t::in6_addr, and tor_assert.

Referenced by decode_address_from_payload(), fetch_from_buf_socks(), and tor_addr_from_in6().

4.2.5.18 int tor_addr_from_sockaddr (tor_addr_t ∗ *a*, const struct sockaddr ∗ *sa*, uint16_t ∗ *port_out*)

Set the **tor_addr_t** (p. 237) in **a** to contain the socket address contained in **sa**.

References tor_addr_from_in6(), tor_addr_from_ipv4n(), tor_addr_make_unspec(), and tor_assert.

Referenced by connection_ap_get_original_destination(), connection_handle_listener_read(), evdns_server_callback(), and get_interface_address6().

4.2.5.19 int tor_addr_from_str (tor_addr_t ∗ *addr*, const char ∗ *src*)

Convert the string in **src** to a **tor_addr_t** (p. 237) **addr**. The string may be an IPv4 address, an IPv6 address, or an IPv6 address surrounded by square brackets.

Return an address family on success, or -1 if an invalid address string is provided.

References tor_addr_from_in6(), tor_assert, tor_free, and tor_inet_pton().

Referenced by circuit_get_open_circ_or_launch(), circuit_is_acceptable(), configure_nameservers(), connection_ap_fail_onehop(), dns_resolve_impl(), rend_parse_introduction_points(), and tor_addr_parse_reverse_lookup_name().

4.2.5.20 unsigned int tor_addr_hash (const tor_addr_t * *addr*)

Return a hash code based on the address addr

References tor_addr_family(), tor_addr_to_ipv4h(), and tor_fragile_assert.

Referenced by policy_hash().

4.2.5.21 int tor_addr_is_internal (const tor_addr_t * *addr*, int *for_listening*)

Return true iff **ip** is an IP reserved to localhost or local networks in RFC1918 or RFC4193 or RFC4291. (fec0::/10, deprecated by RFC3879, is also treated as internal for now.)

References LD_BUG, tor_addr_family(), tor_addr_is_v4(), and tor_addr_to_ipv4h().

Referenced by connection_is_rate_limited(), dns_resolve_impl(), exit_policy_is_general_exit_helper(), is_internal_IP(), and is_local_addr().

4.2.5.22 int tor_addr_is_loopback (const tor_addr_t * *addr*)

Return true iff **addr** is a loopback address

References tor_addr_family(), tor_addr_to_ipv4h(), tor_assert, and tor_fragile_assert.

Referenced by connection_connect().

4.2.5.23 int tor_addr_is_null (const tor_addr_t * *addr*)

Determine whether an address **addr** is null, either all zeroes or belonging to family AF_UNSPEC.

References LD_BUG, tor_addr_family(), tor_addr_to_ipv4n(), and tor_assert.

Referenced by compare_tor_addr_to_addr_policy(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_exit_begin_conn(), and router_compare_to_my_exit_policy().

4.2.5.24 int tor_addr_is_v4 (const tor_addr_t * *addr*)

Determine whether an address is IPv4, either native or IPv4-mapped IPv6. Note that this is about representation only, as any decent stack will reject IPv4-mapped addresses received on the wire (and won't use them on the wire either).

References tor_addr_family(), and tor_assert.

Referenced by tor_addr_compare_masked(), tor_addr_is_internal(), and tor_addr_parse_mask_ports().

4.2.5.25 int tor_addr_lookup (const char ∗ *name*, uint16_t *family*, tor_addr_t ∗ *addr*)

Similar behavior to Unix gethostbyname: resolve **name**, and set ∗**addr** to the proper IP address and family. The **family** argument (which must be AF_INET, AF_INET6, or AF_UNSPEC) declares a *preferred* family, though another one may be returned if only one family is implemented for this address.

Return 0 on success, -1 on failure; 1 on transient failure.

References tor_addr_from_in6(), tor_assert, and tor_inet_pton().

Referenced by tor_addr_port_parse(), and tor_lookup_hostname().

4.2.5.26 void tor_addr_make_unspec (tor_addr_t ∗ *a*)

Set address **a** to the unspecified address. This address belongs to no family.

References tor_addr_t::family.

Referenced by connection_ap_make_link(), connection_handle_listener_read(), decode_address_from_payload(), and tor_addr_from_sockaddr().

4.2.5.27 int tor_addr_parse_mask_ports (const char ∗ *s*, tor_addr_t ∗ *addr_out*, maskbits_t ∗ *maskbits_out*, uint16_t ∗ *port_min_out*, uint16_t ∗ *port_max_out*)

Parse a string **s** containing an IPv4/IPv6 address, and possibly a mask and port or port range. Store the parsed address in **addr_out**, a mask (if any) in **mask_out**, and port(s) (if any) in **port_min_out** and **port_max_out**.

The syntax is: Address OptMask OptPortRange Address ::= IPv4Address / "[" IPv6Address "]" / "*" OptMask ::= "/" Integer / OptPortRange ::= ":*" / ":" Integer / ":" Integer "-" Integer /

- If mask, minport, or maxport are NULL, we do not want these options to be set; treat them as an error if present.
- If the string has no mask, the mask is set to /32 (IPv4) or /128 (IPv6).
- If the string has one port, it is placed in both min and max port variables.
- If the string has no port(s), port_(min|max)_out are set to 1 and 65535.

Return an address family on success, or -1 if an invalid address string is provided.

References addr_mask_get_bits(), escaped(), LD_GENERAL, parse_port_range(), tor_addr_family(), tor_addr_from_in6(), tor_addr_from_ipv4h, tor_addr_is_v4(), tor_assert, tor_free, and tor_inet_pton().

Referenced by policy_expand_private(), policy_summary_add_item(), and router_parse_addr_policy().

4.2.5.28 int tor_addr_parse_reverse_lookup_name (tor_addr_t ∗ *result*, const char ∗ *address*, int *family*, int *accept_regular*)

Parse an .in-addr.arpa or .ip6.arpa address from **address**. Return 0 if this is not an .in-addr.arpa address or an .ip6.arpa address. Return -1 if this is an ill-formed .in-addr.arpa address or an

.ip6.arpa address. Also return -1 if **family** is not AF_UNSPEC, and the parsed address family does not match **family**. On success, return 1, and store the result, if any, into **result**, if provided.

If **accept_regular** is set and the address is in neither recognized reverse lookup hostname format, try parsing the address as a regular IPv4 or IPv6 address too.

References hex_decode_digit(), INET_NTOA_BUF_LEN, strcasecmpend(), tor_addr_from_in6(), tor_addr_from_str(), and tor_inet_aton().

Referenced by connection_ap_handshake_send_resolve(), dns_resolve_impl(), and launch_resolve().

4.2.5.29 int tor_addr_port_parse (const char ∗ *s*, tor_addr_t ∗ *addr_out*, uint16_t ∗ *port_out*)

Parse an address or address-port combination from **s**, resolve the address as needed, and put the result in **addr_out** and (optionally) **port_out**. Return 0 on success, negative on failure.

References eat_whitespace(), tor_addr_copy(), tor_addr_lookup(), tor_assert, tor_free, and tor_parse_long().

Referenced by parse_bridge_line(), and parse_port_config().

4.2.5.30 static INLINE struct in_addr ∗ tor_addr_to_in (const tor_addr_t ∗ *a*) `[static, read]`

Return an in_addr∗ equivalent to **a**, or NULL if **a** is not an IPv4 address.

References tor_addr_t::addr, tor_addr_t::family, and tor_addr_t::in_addr.

Referenced by launch_resolve().

4.2.5.31 static INLINE struct in6_addr ∗ tor_addr_to_in6 (const tor_addr_t ∗ *a*) `[static, read]`

Return an in6_addr∗ equivalent to **a**, or NULL if **a** is not an IPv6 address.

References tor_addr_t::addr, tor_addr_t::family, and tor_addr_t::in6_addr.

Referenced by connection_ap_get_original_destination(), connection_send_socks5_connect(), launch_resolve(), and tor_addr_to_sockaddr().

4.2.5.32 static INLINE uint32_t tor_addr_to_ipv4h (const tor_addr_t ∗ *a*) `[static]`

Return an IPv4 address in host order for **a**, or 0 if **a** is not an IPv4 address.

References tor_addr_to_ipv4n().

Referenced by connection_or_set_state_open(), exit_policy_is_general_exit_helper(), get_interface_address(), is_local_addr(), routerset_contains(), tor_addr_compare_masked(), tor_addr_eq_ipv4h(), tor_addr_hash(), tor_addr_is_internal(), tor_addr_is_loopback(), tor_addr_to_reverse_lookup_name(), and tor_lookup_hostname().

4.2.5.33 static INLINE uint32_t tor_addr_to_ipv4n (const tor_addr_t * *a*) [static]

Return an IPv4 address in network order for **a**, or 0 if **a** is not an IPv4 address.

References tor_addr_t::addr, tor_addr_t::family, and tor_addr_t::in_addr.

Referenced by append_address_to_payload(), circuit_send_next_onion_skin(), connection_ap_get_original_destination(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_connect(), connection_proxy_connect(), connection_send_socks5_connect(), send_resolved_cell(), tor_addr_is_null(), tor_addr_to_ipv4h(), and tor_addr_to_sockaddr().

4.2.5.34 int tor_addr_to_reverse_lookup_name (char * *out*, size_t *outlen*, const tor_addr_t * *addr*)

Convert **addr** to an in-addr.arpa name or a .ip6.arpa name, and store the result in the **outlen**-byte buffer at **out**. Return 0 on success, -1 on failure.

References tor_addr_t::family, REVERSE_LOOKUP_NAME_BUF_LEN, tor_addr_to_ipv4h(), and tor_snprintf().

Referenced by connection_ap_handshake_send_resolve().

4.2.5.35 socklen_t tor_addr_to_sockaddr (const tor_addr_t * *a*, uint16_t *port*, struct sockaddr * *sa_out*, socklen_t *len*)

Convert the **tor_addr_t** (p. 237) in **a**, with port in **port**, into a socklen object in ***sa_out** of object size **len**. If not enough room is free, or on error, return -1. Else return the length of the sockaddr.

References tor_addr_family(), tor_addr_to_in6(), and tor_addr_to_ipv4n().

Referenced by configure_nameservers(), and connection_connect().

4.2.5.36 const char* tor_addr_to_str (char * *dest*, const tor_addr_t * *addr*, int *len*, int *decorate*)

Convert a **tor_addr_t** (p. 237) **addr** into a string, and store it in **dest** of size **len**. Returns a pointer to dest on success, or NULL on failure. If **decorate**, surround IPv6 addresses with brackets.

References tor_addr_t::addr, tor_addr_t::in6_addr, tor_addr_t::in_addr, tor_addr_family(), tor_assert, and tor_inet_ntop().

Referenced by connection_ap_get_original_destination(), fetch_from_buf_socks(), fmt_addr(), policy_write_item(), and tor_dup_addr().

4.2.5.37 char* tor_dup_addr (const tor_addr_t * *addr*)

Return a newly allocated string with a representation of **addr**.

References tor_addr_to_str().

Referenced by connection_exit_begin_conn(), connection_handle_listener_read(), connection_or_init_conn_from_address(), evdns_server_callback(), launch_direct_bridge_descriptor_fetch(), and rend_encode_v2_intro_points().

4.2.5.38 char* tor_dup_ip (uint32_t *addr*)

Given a host-order **addr**, call **tor_inet_ntop()** (p. 362) on it and return a strdup of the resulting address.

References tor_inet_ntop().

Referenced by directory_post_to_hs_dir(), dirserv_generate_networkstatus_vote_obj(), getinfo_helper_misc(), rep_hist_get_router_stability_doc(), router_rebuild_descriptor(), and trusted_dirs_load_certs_from_string().

4.2.5.39 int tor_inet_ntoa (const struct in_addr * *in*, char * *buf*, size_t *buf_len*)

Given an IPv4 in_addr struct ***in** (in network order, as usual), write it as a string into the **buf_len**-byte buffer in **buf**.

References tor_snprintf().

Referenced by addressmap_get_virtual_address(), client_dns_set_addressmap(), connection_edge_process_relay_cell_not_open(), directory_initiate_command_routerstatus_rend(), evdns_callback(), evdns_wildcard_check_callback(), fetch_from_buf_socks(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), log_addr_has_changed(), networkstatus_compute_consensus(), remap_event_helper(), resolve_my_address(), routerstatus_format_entry(), tell_controller_about_resolved_result(), and tor_inet_ntop().

4.3 aes.c File Reference

Implements the AES cipher (with 128-bit keys and blocks), and a counter-mode stream cipher on top of AES. This code is taken from the main Rijndael distribution. (We include this because many people are running older versions of OpenSSL without AES support.).

`#include "orconfig.h"`

`#include <openssl/opensslv.h>`

`#include <assert.h>`

`#include <stdlib.h>`

`#include <string.h>`

`#include "compat.h"`

`#include "aes.h"`

`#include "util.h"`

`#include "log.h"`

Data Structures

- struct **aes_cnt_cipher**

Defines

- #define **USE_BUILTIN_AES**
- #define **USE_RIJNDAEL_COUNTER_OPTIMIZATION**
- #define **MAXNR** 14
- #define **USING_COUNTER_VARS**
- #define **COUNTER**(c, n) ((c) → counter ## n)
- #define **UPDATE_CTR_BUF**(c, n)
- #define **SWAP**(x) (_lrotl(x, 8) & 0x00ff00ff | _lrotr(x, 8) & 0xff00ff00)
- #define **GETU32**(pt) (((u32)(pt)[0] << 24) ^ ((u32)(pt)[1] << 16) ^ ((u32)(pt)[2] << 8) ^ ((u32)(pt)[3]))
- #define **PUTU32**(ct, st) { (ct)[0] = (u8)((st) >> 24); (ct)[1] = (u8)((st) >> 16); (ct)[2] = (u8)((st) >> 8); (ct)[3] = (u8)(st); }

Typedefs

- typedef uint64_t **u64**
- typedef uint32_t **u32**
- typedef uint8_t **u8**

Functions

- static int **rijndaelKeySetupEnc** (u32 rk[ ], const u8 cipherKey[ ], int keyBits)
- static void **rijndaelEncrypt** (const u32 rk[ ], int Nr, u32 ctr3, u32 ctr2, u32 ctr1, u32 ctr0, u8 ct[16])
- static INLINE void **_aes_fill_buf** (**aes_cnt_cipher_t** ∗cipher)

- **aes_cnt_cipher_t** ∗ **aes_new_cipher** (void)
- void **aes_set_key** (**aes_cnt_cipher_t** ∗cipher, const char ∗key, int key_bits)
- void **aes_free_cipher** (**aes_cnt_cipher_t** ∗cipher)
- void **aes_crypt** (**aes_cnt_cipher_t** ∗cipher, const char ∗input, size_t len, char ∗output)
- void **aes_crypt_inplace** (**aes_cnt_cipher_t** ∗cipher, char ∗data, size_t len)
- void **aes_set_iv** (**aes_cnt_cipher_t** ∗cipher, const char ∗iv)

Variables

- static const u32 **Te0** [256]
- static const u32 **Te1** [256]
- static const u32 **Te2** [256]
- static const u32 **Te3** [256]
- static const u32 **Te4** [256]
- static const u32 **rcon** [ ]

4.3.1 Detailed Description

Implements the AES cipher (with 128-bit keys and blocks), and a counter-mode stream cipher on top of AES. This code is taken from the main Rijndael distribution. (We include this because many people are running older versions of OpenSSL without AES support.).

4.3.2 Define Documentation

4.3.2.1 #define USE_BUILTIN_AES

Defined iff we're using OpenSSL's AES functions for AES. Defined iff we're using OpenSSL's EVP code for AES. Defined iff we're using Tor's internal AES implementation, defined below.

4.3.3 Function Documentation

4.3.3.1 static INLINE void _aes_fill_buf (aes_cnt_cipher_t ∗ *cipher*) `[static]`

Helper function: set **cipher**'s internal buffer to the encrypted value of the current counter.

References aes_cnt_cipher::buf, aes_cnt_cipher::counter0, aes_cnt_cipher::counter1, aes_cnt_cipher::counter2, aes_cnt_cipher::counter3, aes_cnt_cipher::nr, and aes_cnt_cipher::rk.

Referenced by aes_crypt(), aes_crypt_inplace(), aes_set_iv(), and aes_set_key().

4.3.3.2 void aes_crypt (aes_cnt_cipher_t ∗ *cipher*, const char ∗ *input*, size_t *len*, char ∗ *output*)

Encrypt **len** bytes from **input**, storing the result in **output**. Uses the key in **cipher**, and advances the counter by **len** bytes as it encrypts.

References _aes_fill_buf(), aes_cnt_cipher::buf, and aes_cnt_cipher::pos.

Referenced by crypto_cipher_decrypt(), and crypto_cipher_encrypt().

4.3.3.3 void aes_crypt_inplace (aes_cnt_cipher_t * *cipher*, char * *data*, size_t *len*)

Encrypt **len** bytes from **input**, storing the results in place. Uses the key in **cipher**, and advances the counter by **len** bytes as it encrypts.

References _aes_fill_buf(), aes_cnt_cipher::buf, and aes_cnt_cipher::pos.

Referenced by crypto_cipher_crypt_inplace().

4.3.3.4 void aes_free_cipher (aes_cnt_cipher_t * *cipher*)

Release storage held by **cipher**

References tor_free.

Referenced by crypto_free_cipher_env().

4.3.3.5 aes_cnt_cipher_t* aes_new_cipher (void)

Return a newly allocated counter-mode AES128 cipher implementation.

Referenced by crypto_new_cipher_env().

4.3.3.6 void aes_set_iv (aes_cnt_cipher_t * *cipher*, const char * *iv*)

Reset the 128-bit counter of **cipher** to the 16-bit big-endian value in **iv**.

References _aes_fill_buf(), aes_cnt_cipher::buf, aes_cnt_cipher::counter0, aes_cnt_cipher::counter1, aes_cnt_cipher::counter2, aes_cnt_cipher::counter3, get_uint32(), and aes_cnt_cipher::pos.

Referenced by crypto_cipher_set_iv().

4.3.3.7 void aes_set_key (aes_cnt_cipher_t * *cipher*, const char * *key*, int *key_bits*)

Set the key of **cipher** to **key**, which is **key_bits** bits long (must be 128, 192, or 256). Also resets the counter to 0.

References _aes_fill_buf(), aes_cnt_cipher::buf, aes_cnt_cipher::counter0, aes_cnt_cipher::counter1, aes_cnt_cipher::counter2, aes_cnt_cipher::counter3, aes_cnt_cipher::nr, aes_cnt_cipher::pos, rijndaelKeySetupEnc(), aes_cnt_cipher::rk, and tor_assert.

Referenced by crypto_cipher_decrypt_init_cipher(), and crypto_cipher_encrypt_init_cipher().

4.3.3.8 static int rijndaelKeySetupEnc (u32 *rk*[ ], const u8 *cipherKey*[ ], int *keyBits*) `[static]`

Expand the cipher key into the encryption key schedule.

Returns:

the number of rounds for the given cipher key size.

Referenced by aes_set_key().

4.3.4 Variable Documentation

4.3.4.1 const u32 rcon[] `[static]`

Initial value:

```
 {
        0x01000000, 0x02000000, 0x04000000, 0x08000000,
        0x10000000, 0x20000000, 0x40000000, 0x80000000,
        0x1B000000, 0x36000000,
}
```

4.3.4.2 const u32 Te0[256] `[static]`

rijndael-alg-fst.c

Version:

3.0 (December 2000)

Optimized ANSI C code for the Rijndael cipher (now AES)

Author:

Vincent Rijmen <`vincent.rijmen@esat.kuleuven.ac.be`>
Antoon Bosselaers <`antoon.bosselaers@esat.kuleuven.ac.be`>
Paulo Barreto <`paulo.barreto@terra.com.br`>

This code is hereby placed in the public domain.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.4 aes.h File Reference

Headers for **aes.c** (p. 277).

`#include "torint.h"`

Typedefs

- typedef struct **aes_cnt_cipher aes_cnt_cipher_t**

Functions

- **aes_cnt_cipher_t** ∗ **aes_new_cipher** (void)
- void **aes_free_cipher** (**aes_cnt_cipher_t** ∗cipher)
- void **aes_set_key** (**aes_cnt_cipher_t** ∗cipher, const char ∗key, int key_bits)
- void **aes_crypt** (**aes_cnt_cipher_t** ∗cipher, const char ∗input, size_t len, char ∗output)
- void **aes_crypt_inplace** (**aes_cnt_cipher_t** ∗cipher, char ∗data, size_t len)
- void **aes_set_iv** (**aes_cnt_cipher_t** ∗cipher, const char ∗iv)

4.4.1 Detailed Description

Headers for **aes.c** (p. 277).

4.4.2 Function Documentation

4.4.2.1 void aes_crypt (aes_cnt_cipher_t ∗ *cipher*, const char ∗ *input*, size_t *len*, char ∗ *output*)

Encrypt **len** bytes from **input**, storing the result in **output**. Uses the key in **cipher**, and advances the counter by **len** bytes as it encrypts.

References _aes_fill_buf(), aes_cnt_cipher::buf, and aes_cnt_cipher::pos.

Referenced by crypto_cipher_decrypt(), and crypto_cipher_encrypt().

4.4.2.2 void aes_crypt_inplace (aes_cnt_cipher_t ∗ *cipher*, char ∗ *data*, size_t *len*)

Encrypt **len** bytes from **input**, storing the results in place. Uses the key in **cipher**, and advances the counter by **len** bytes as it encrypts.

References _aes_fill_buf(), aes_cnt_cipher::buf, and aes_cnt_cipher::pos.

Referenced by crypto_cipher_crypt_inplace().

4.4.2.3 void aes_free_cipher (aes_cnt_cipher_t ∗ *cipher*)

Release storage held by **cipher**

References tor_free.

Referenced by crypto_free_cipher_env().

4.4.2.4 aes_cnt_cipher_t* aes_new_cipher (void)

Return a newly allocated counter-mode AES128 cipher implementation.

Referenced by crypto_new_cipher_env().

4.4.2.5 void aes_set_iv (aes_cnt_cipher_t * *cipher*, const char * *iv*)

Reset the 128-bit counter of **cipher** to the 16-bit big-endian value in **iv**.

References _aes_fill_buf(), aes_cnt_cipher::buf, aes_cnt_cipher::counter0, aes_cnt_cipher::counter1, aes_cnt_cipher::counter2, aes_cnt_cipher::counter3, get_uint32(), and aes_cnt_cipher::pos.

Referenced by crypto_cipher_set_iv().

4.4.2.6 void aes_set_key (aes_cnt_cipher_t * *cipher*, const char * *key*, int *key_bits*)

Set the key of **cipher** to **key**, which is **key_bits** bits long (must be 128, 192, or 256). Also resets the counter to 0.

References _aes_fill_buf(), aes_cnt_cipher::buf, aes_cnt_cipher::counter0, aes_cnt_cipher::counter1, aes_cnt_cipher::counter2, aes_cnt_cipher::counter3, aes_cnt_cipher::nr, aes_cnt_cipher::pos, rijndaelKeySetupEnc(), aes_cnt_cipher::rk, and tor_assert.

Referenced by crypto_cipher_decrypt_init_cipher(), and crypto_cipher_encrypt_init_cipher().

4.5 buffers.c File Reference

Implements a generic interface buffer. Buffers are fairly opaque string holders that can read to or flush from: memory, file descriptors, or TLS connections.

`#include "or.h"`

`#include "../common/util.h"`

`#include "../common/log.h"`

`#include <unistd.h>`

`#include <sys/uio.h>`

Data Structures

- struct **chunk_t**
- struct **chunk_freelist_t**
- struct **buf_t**
- struct **buf_pos_t**

Defines

- #define **BUFFERS_PRIVATE**
- #define **check**() STMT_NIL
- #define **CHUNK_HEADER_LEN** STRUCT_OFFSET(**chunk_t**, mem[0])
- #define **CHUNK_ALLOC_SIZE**(memlen) (CHUNK_HEADER_LEN + (memlen))
- #define **CHUNK_SIZE_WITH_ALLOC**(memlen) ((memlen) - CHUNK_HEADER_LEN)
- #define **FL**(a, m, s) { a, m, s, 0, 0, 0, 0, 0, NULL }
- #define **MIN_READ_LEN** 8
- #define **MIN_CHUNK_ALLOC** 256
- #define **MAX_CHUNK_ALLOC** 65536
- #define **BUFFER_MAGIC** 0xB0FFF312u
- #define **N_IOV** 3
- #define **CONTENT_LENGTH** "\r\nContent-Length: "

Functions

- static INLINE char ∗ **CHUNK_WRITE_PTR** (**chunk_t** ∗chunk)
- static INLINE size_t **CHUNK_REMAINING_CAPACITY** (const **chunk_t** ∗chunk)
- static INLINE void **chunk_repack** (**chunk_t** ∗chunk)
- static void **assert_freelist_ok** (**chunk_freelist_t** ∗fl)
- static INLINE **chunk_freelist_t** ∗ **get_freelist** (size_t alloc)
- static void **chunk_free_unchecked** (**chunk_t** ∗chunk)
- static INLINE **chunk_t** ∗ **chunk_new_with_alloc_size** (size_t alloc)
- static INLINE **chunk_t** ∗ **chunk_grow** (**chunk_t** ∗chunk, size_t sz)
- static INLINE size_t **preferred_chunk_size** (size_t target)
- void **buf_shrink_freelists** (int free_all)
- void **buf_dump_freelist_sizes** (int severity)
- static void **buf_pullup** (**buf_t** ∗buf, size_t bytes, int nulterminate)

- void **buf_shrink** (**buf_t** ∗buf)
- static INLINE void **buf_remove_from_front** (**buf_t** ∗buf, size_t n)
- **buf_t** ∗ **buf_new_with_capacity** (size_t size)
- **buf_t** ∗ **buf_new** (void)
- void **buf_clear** (**buf_t** ∗buf)
- size_t **buf_datalen** (const **buf_t** ∗buf)
- size_t **buf_allocation** (const **buf_t** ∗buf)
- size_t **buf_slack** (const **buf_t** ∗buf)
- void **buf_free** (**buf_t** ∗buf)
- static **chunk_t** ∗ **buf_add_chunk_with_capacity** (**buf_t** ∗buf, size_t capacity, int capped)
- static INLINE int **read_to_chunk** (**buf_t** ∗buf, **chunk_t** ∗chunk, int fd, size_t at_most, int ∗reached_eof, int ∗socket_error)
- static INLINE int **read_to_chunk_tls** (**buf_t** ∗buf, **chunk_t** ∗chunk, **tor_tls_t** ∗tls, size_t at_most)
- int **read_to_buf** (int s, size_t at_most, **buf_t** ∗buf, int ∗reached_eof, int ∗socket_error)
- int **read_to_buf_tls** (**tor_tls_t** ∗tls, size_t at_most, **buf_t** ∗buf)
- static INLINE int **flush_chunk** (int s, **buf_t** ∗buf, **chunk_t** ∗chunk, size_t sz, size_t ∗buf_flushlen)
- static INLINE int **flush_chunk_tls** (**tor_tls_t** ∗tls, **buf_t** ∗buf, **chunk_t** ∗chunk, size_t sz, size_t ∗buf_flushlen)
- int **flush_buf** (int s, **buf_t** ∗buf, size_t sz, size_t ∗buf_flushlen)
- int **flush_buf_tls** (**tor_tls_t** ∗tls, **buf_t** ∗buf, size_t flushlen, size_t ∗buf_flushlen)
- int **write_to_buf** (const char ∗string, size_t string_len, **buf_t** ∗buf)
- static INLINE void **peek_from_buf** (char ∗string, size_t string_len, const **buf_t** ∗buf)
- int **fetch_from_buf** (char ∗string, size_t string_len, **buf_t** ∗buf)
- int **fetch_var_cell_from_buf** (**buf_t** ∗buf, **var_cell_t** ∗∗out, int linkproto)
- int **move_buf_to_buf** (**buf_t** ∗buf_out, **buf_t** ∗buf_in, size_t ∗buf_flushlen)
- static void **buf_pos_init** (const **buf_t** ∗buf, **buf_pos_t** ∗out)
- static off_t **buf_find_pos_of_char** (char ch, **buf_pos_t** ∗out)
- static INLINE int **buf_pos_inc** (**buf_pos_t** ∗pos)
- static int **buf_matches_at_pos** (const **buf_pos_t** ∗pos, const char ∗s, size_t n)
- int **buf_find_string_offset** (const **buf_t** ∗buf, const char ∗s, size_t n)
- int **fetch_from_buf_http** (**buf_t** ∗buf, char ∗∗headers_out, size_t max_headerlen, char ∗∗body_out, size_t ∗body_used, size_t max_bodylen, int force_complete)
- int **fetch_from_buf_socks** (**buf_t** ∗buf, **socks_request_t** ∗req, int log_sockstype, int safe_socks)
- int **fetch_from_buf_socks_client** (**buf_t** ∗buf, int state, char ∗∗reason)
- int **peek_buf_has_control0_command** (**buf_t** ∗buf)
- static off_t **buf_find_offset_of_char** (**buf_t** ∗buf, char ch)
- int **fetch_from_buf_line** (**buf_t** ∗buf, char ∗data_out, size_t ∗data_len)
- int **write_to_buf_zlib** (**buf_t** ∗buf, **tor_zlib_state_t** ∗state, const char ∗data, size_t data_len, int done)
- void **assert_buf_ok** (**buf_t** ∗buf)

Variables

- static **chunk_freelist_t** **freelists** []
- static uint64_t **n_freelist_miss** = 0

4.5.1 Detailed Description

Implements a generic interface buffer. Buffers are fairly opaque string holders that can read to or flush from: memory, file descriptors, or TLS connections.

4.5.2 Define Documentation

4.5.2.1 #define BUFFER_MAGIC 0xB0FFF312u

Magic value for **buf_t.magic** (p. 18), to catch pointer errors.

Referenced by assert_buf_ok(), and buf_new().

4.5.2.2 #define CHUNK_ALLOC_SIZE(memlen) (CHUNK_HEADER_LEN + (memlen))

Return the number of bytes needed to allocate a chunk to hold **memlen** bytes.

Referenced by assert_freelist_ok(), buf_add_chunk_with_capacity(), chunk_free_unchecked(), and chunk_grow().

4.5.2.3 #define CHUNK_SIZE_WITH_ALLOC(memlen) ((memlen) - CHUNK_HEADER_LEN)

Return the number of usable bytes in a chunk allocated with malloc(**memlen**).

Referenced by buf_pullup(), chunk_new_with_alloc_size(), and preferred_chunk_size().

4.5.2.4 #define FL(a, m, s) { a, m, s, 0, 0, 0, 0, 0, NULL }

Macro to help define freelists.

4.5.2.5 #define MAX_CHUNK_ALLOC 65536

No chunk should take up more than this many bytes.

Referenced by buf_add_chunk_with_capacity().

4.5.2.6 #define MIN_CHUNK_ALLOC 256

Every chunk should take up at least this many bytes.

Referenced by preferred_chunk_size().

4.5.2.7 #define MIN_READ_LEN 8

If a read onto the end of a chunk would be smaller than this number, then just start a new chunk.

Referenced by read_to_buf(), and read_to_buf_tls().

4.5.2.8 #define N_IOV 3

If we're using readv and writev, how many chunks are we willing to read/write at a time?

4.5.3 Function Documentation

4.5.3.1 void assert_buf_ok (buf_t * *buf*)

Log an error and exit if **buf** is corrupted.

References BUFFER_MAGIC, chunk_t::data, chunk_t::datalen, buf_t::datalen, buf_t::head, buf_t::magic, chunk_t::mem, chunk_t::memlen, chunk_t::next, buf_t::tail, and tor_assert.

Referenced by assert_connection_ok().

4.5.3.2 static void assert_freelist_ok (chunk_freelist_t * *fl*) `[static]`

Log an error and exit if **fl** is corrupted.

References chunk_freelist_t::alloc_size, CHUNK_ALLOC_SIZE, chunk_freelist_t::cur_length, chunk_freelist_t::head, chunk_freelist_t::lowest_length, chunk_t::memlen, chunk_t::next, and tor_assert.

Referenced by buf_shrink_freelists().

4.5.3.3 static chunk_t* buf_add_chunk_with_capacity (buf_t * *buf*, size_t *capacity*, int *capped*) `[static]`

Append a new chunk with enough capacity to hold **capacity** bytes to the tail of **buf**. If **capped**, don't allocate a chunk bigger than MAX_CHUNK_ALLOC.

References CHUNK_ALLOC_SIZE, chunk_new_with_alloc_size(), buf_t::default_chunk_size, buf_t::head, MAX_CHUNK_ALLOC, chunk_t::next, preferred_chunk_size(), buf_t::tail, and tor_assert.

Referenced by read_to_buf(), read_to_buf_tls(), write_to_buf(), and write_to_buf_zlib().

4.5.3.4 size_t buf_allocation (const buf_t * *buf*)

Return the total length of all chunks used in **buf**.

References buf_t::head, chunk_t::memlen, and chunk_t::next.

Referenced by connection_dump_buffer_mem_stats(), and dumpstats().

4.5.3.5 void buf_clear (buf_t * *buf*)

Remove all data from **buf**.

References chunk_free_unchecked(), buf_t::datalen, buf_t::head, chunk_t::next, and buf_t::tail.

Referenced by buf_free(), connection_close_immediate(), fetch_from_buf_socks(), and fetch_from_buf_socks_client().

4.5.3.6 size_t buf_datalen (const buf_t * *buf*)

Return the number of bytes stored in **buf**

References buf_t::datalen.

Referenced by _connection_free(), _connection_write_to_buf_impl(), append_cell_to_circuit_queue(), conn_close_if_marked(), connection_cpu_process_inbuf(), connection_dir_process_inbuf(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dirserv_flushed_some(), connection_dump_buffer_mem_stats(), connection_edge_package_raw_inbuf(), connection_edge_reached_eof(), connection_handle_read_impl(), connection_handle_write_impl(), connection_or_flushed_some(), connection_or_process_cells_from_inbuf(), connection_read_to_buf(), connection_should_read_from_linked_conn(), dumpstats(), and run_connection_housekeeping().

4.5.3.7 void buf_dump_freelist_sizes (int *severity*)

Describe the current status of the freelists at log level **severity**.

References chunk_freelist_t::alloc_size, LD_MM, and n_freelist_miss.

Referenced by dumpmemusage().

4.5.3.8 static off_t buf_find_offset_of_char (buf_t * *buf*, char *ch*) `[static]`

Return the index within **buf** at which **ch** first appears, or -1 if **ch** does not appear on buf.

References chunk_t::data, chunk_t::datalen, buf_t::head, and chunk_t::next.

Referenced by fetch_from_buf_line().

4.5.3.9 static off_t buf_find_pos_of_char (char *ch*, buf_pos_t * *out*) `[static]`

Advance **out** to the first appearance of **ch** at the current position of **out**, or later. Return -1 if no instances are found; otherwise returns the absolute position of the character.

References buf_pos_t::chunk, buf_pos_t::chunk_pos, chunk_t::data, chunk_t::datalen, chunk_t::next, buf_pos_t::pos, and tor_assert.

Referenced by buf_find_string_offset().

4.5.3.10 int buf_find_string_offset (const buf_t * *buf*, const char * *s*, size_t *n*)

Return the first position in **buf** at which the **n**-character string **s** occurs, or -1 if it does not occur.

References buf_find_pos_of_char(), buf_matches_at_pos(), buf_pos_inc(), buf_pos_init(), buf_pos_t::chunk_pos, buf_pos_t::pos, and tor_assert.

Referenced by fetch_from_buf_http().

4.5.3.11 void buf_free (buf_t * *buf*)

Release storage held by **buf**.

References buf_clear(), buf_t::magic, and tor_free.

Referenced by _connection_free().

4.5.3.12 static int buf_matches_at_pos (const buf_pos_t * *pos*, const char * *s*, size_t *n*) [static]

Return true iff the **n**-character string in **s** appears (verbatim) at **pos**.

References buf_pos_inc(), buf_pos_t::chunk, chunk_t::data, and buf_pos_t::pos.

Referenced by buf_find_string_offset().

4.5.3.13 buf_t* buf_new (void)

Allocate and return a new buffer with default capacity.

References BUFFER_MAGIC, buf_t::default_chunk_size, and buf_t::magic.

Referenced by buf_new_with_capacity(), and connection_init().

4.5.3.14 buf_t* buf_new_with_capacity (size_t *size*)

Create and return a new buf with default chunk capacity **size**.

References buf_new(), buf_t::default_chunk_size, and preferred_chunk_size().

4.5.3.15 static INLINE int buf_pos_inc (buf_pos_t * *pos*) [static]

Advance **pos** by a single character, if there are any more characters in the buffer. Returns 0 on success, -1 on failure.

References buf_pos_t::chunk, buf_pos_t::chunk_pos, chunk_t::datalen, chunk_t::next, and buf_pos_t::pos.

Referenced by buf_find_string_offset(), and buf_matches_at_pos().

4.5.3.16 static void buf_pos_init (const buf_t * *buf*, buf_pos_t * *out*) [static]

Initialize **out** to point to the first character of **buf**.

References buf_pos_t::chunk, buf_pos_t::chunk_pos, buf_t::head, and buf_pos_t::pos.

Referenced by buf_find_string_offset().

4.5.3.17 static void buf_pullup (buf_t * *buf*, size_t *bytes*, int *nulterminate*) [static]

Collapse data from the first N chunks from **buf** into buf->head, growing it as necessary, until buf->head has the first **bytes** bytes of data from the buffer, or until buf->head has all the data in **buf**.

If **nulterminate** is true, ensure that there is a 0 byte in buf->head->mem right after all the data.

References chunk_free_unchecked(), chunk_grow(), CHUNK_REMAINING_CAPACITY(), chunk_repack(), CHUNK_SIZE_WITH_ALLOC, CHUNK_WRITE_PTR(), chunk_t::data, chunk_t::datalen, buf_t::datalen, buf_t::head, chunk_t::memlen, chunk_t::next, preferred_chunk_size(), buf_t::tail, and tor_assert.

Referenced by fetch_from_buf_http(), fetch_from_buf_socks(), and fetch_from_buf_socks_client().

4.5.3.18 static INLINE void buf_remove_from_front (buf_t * *buf*, size_t *n*) `[static]`

Remove the first **n** bytes from buf.

References chunk_free_unchecked(), chunk_t::data, chunk_t::datalen, buf_t::datalen, buf_t::head, chunk_t::next, buf_t::tail, and tor_assert.

Referenced by fetch_from_buf(), fetch_from_buf_socks(), fetch_from_buf_socks_client(), fetch_var_cell_from_buf(), flush_chunk(), and flush_chunk_tls().

4.5.3.19 void buf_shrink (buf_t * *buf*)

Resize buf so it won't hold extra memory that we haven't been using lately.

Referenced by run_scheduled_events().

4.5.3.20 void buf_shrink_freelists (int *free_all*)

Remove from the freelists most chunks that have not been used since the last call to **buf_shrink_freelists()** (p. 289).

References chunk_freelist_t::alloc_size, assert_freelist_ok(), chunk_freelist_t::cur_length, chunk_freelist_t::head, LD_BUG, LD_MM, chunk_freelist_t::lowest_length, chunk_freelist_t::n_free, chunk_t::next, chunk_freelist_t::slack, tor_assert, and tor_free.

Referenced by run_scheduled_events(), and tor_free_all().

4.5.3.21 size_t buf_slack (const buf_t * *buf*)

Return the number of bytes that can be added to **buf** without performing any additional allocation.

References CHUNK_REMAINING_CAPACITY(), and buf_t::tail.

Referenced by connection_read_to_buf().

4.5.3.22 static void chunk_free_unchecked (chunk_t * *chunk*) `[static]`

Deallocate a chunk or put it on a freelist

References CHUNK_ALLOC_SIZE, chunk_freelist_t::cur_length, freelist, get_freelist(), chunk_freelist_t::head, chunk_freelist_t::max_length, chunk_t::memlen, chunk_freelist_t::n_free, chunk_t::next, and tor_free.

4.5.3.23 static INLINE chunk_t* chunk_grow (chunk_t * *chunk*, size_t *sz*) `[static]`

Expand **chunk** until it can hold **sz** bytes, and return a new pointer to **chunk**. Old pointers are no longer valid.

References CHUNK_ALLOC_SIZE, chunk_t::data, chunk_t::mem, chunk_t::memlen, and tor_assert.

Referenced by buf_pullup().

4.5.3.24 static INLINE chunk_t∗ chunk_new_with_alloc_size (size_t *alloc*) [static]

Allocate a new chunk with a given allocation size, or get one from the freelist. Note that a chunk with allocation size A can actually hold only **CHUNK_SIZE_WITH_ALLOC(A)** (p. 285) bytes in its mem field.

References CHUNK_SIZE_WITH_ALLOC, chunk_freelist_t::cur_length, chunk_t::data, chunk_t::datalen, freelist, get_freelist(), chunk_freelist_t::head, chunk_freelist_t::lowest_length, chunk_t::mem, chunk_t::memlen, chunk_freelist_t::n_alloc, n_freelist_miss, chunk_freelist_t::n_hit, chunk_t::next, and tor_assert.

Referenced by buf_add_chunk_with_capacity().

4.5.3.25 static INLINE size_t CHUNK_REMAINING_CAPACITY (const chunk_t ∗ *chunk*) [static]

Return the number of bytes that can be written onto **chunk** without running out of space.

References chunk_t::data, chunk_t::datalen, chunk_t::mem, and chunk_t::memlen.

Referenced by buf_pullup(), buf_slack(), read_to_buf(), read_to_buf_tls(), read_to_chunk(), read_to_chunk_tls(), write_to_buf(), and write_to_buf_zlib().

4.5.3.26 static INLINE void chunk_repack (chunk_t ∗ *chunk*) [static]

Move all bytes stored in **chunk** to the front of **chunk**->mem, to free up space at the end.

References chunk_t::data, chunk_t::datalen, and chunk_t::mem.

Referenced by buf_pullup().

4.5.3.27 static INLINE char∗ CHUNK_WRITE_PTR (chunk_t ∗ *chunk*) [static]

Return the next character in **chunk** onto which data can be appended. If the chunk is full, this might be off the end of chunk->mem.

References chunk_t::data, and chunk_t::datalen.

Referenced by buf_pullup(), fetch_from_buf_socks(), read_to_chunk(), read_to_chunk_tls(), write_to_buf(), and write_to_buf_zlib().

4.5.3.28 int fetch_from_buf (char ∗ *string*, size_t *string_len*, buf_t ∗ *buf*)

Remove **string_len** bytes from the front of **buf**, and store them into **string**. Return the new buffer size. **string_len** must be <= the number of bytes on the buffer.

References buf_remove_from_front(), buf_t::datalen, peek_from_buf(), and tor_assert.

Referenced by connection_fetch_from_buf(), fetch_from_buf_http(), fetch_from_buf_line(), and move_buf_to_buf().

4.5.3.29 int fetch_from_buf_http (buf_t ∗ *buf*, char ∗∗ *headers_out*, size_t *max_headerlen*, char ∗∗ *body_out*, size_t ∗ *body_used*, size_t *max_bodylen*, int *force_complete*)

There is a (possibly incomplete) http statement on **buf**, of the form "\%s\\r\\n\\r\\n\%s", headers, body. (body may contain NULs.) If a) the headers include a Content-Length field and all bytes in the body are present, or b) there's no Content-Length field and all headers are present, then:

- strdup headers into ∗**headers_out**, and NUL-terminate it.
- memdup body into ∗**body_out**, and NUL-terminate it.
- Then remove them from **buf**, and return 1.

- If headers or body is NULL, discard that part of the buf.
- If a headers or body doesn't fit in the arg, return -1. (We ensure that the headers or body don't exceed max len, _even if_ we're planning to discard them.)
- If force_complete is true, then succeed even if not all of the content has arrived.

Else, change nothing and return 0.

References buf_find_string_offset(), buf_pullup(), chunk_t::data, buf_t::datalen, chunk_t::datalen, fetch_from_buf(), buf_t::head, LD_HTTP, LD_PROTOCOL, and tor_assert.

Referenced by connection_dir_client_reached_eof(), connection_read_https_proxy_response(), and directory_handle_command().

4.5.3.30 int fetch_from_buf_line (buf_t ∗ *buf*, char ∗ *data_out*, size_t ∗ *data_len*)

Try to read a single LF-terminated line from **buf**, and write it, NUL-terminated, into the ∗**data_len** byte buffer at **data_out**. Set ∗**data_len** to the number of bytes in the line, not counting the terminating NUL. Return 1 if we read a whole line, return 0 if we don't have a whole line yet, and return -1 if the line length exceeds ∗**data_len**.

References buf_find_offset_of_char(), fetch_from_buf(), and buf_t::head.

Referenced by connection_ap_process_natd(), and connection_control_process_inbuf().

4.5.3.31 int fetch_from_buf_socks (buf_t ∗ *buf*, socks_request_t ∗ *req*, int *log_sockstype*, int *safe_socks*)

There is a (possibly incomplete) socks handshake on **buf**, of one of the forms

- socks4: "socksheader username\\0"
- socks4a: "socksheader username\\0 destaddr\\0"
- socks5 phase one: "version #methods methods"

- socks5 phase two: "version command 0 addresstype..." If it's a complete and valid handshake, and destaddr fits in MAX_SOCKS_ADDR_LEN bytes, then pull the handshake off the buf, assign to **req**, and return 1.

If it's invalid or too big, return -1.

Else it's not all there yet, leave buf alone and return 0.

If you want to specify the socks reply, write it into **req->reply** and set **req->replylen**, else leave **req->replylen** alone.

If **log_sockstype** is non-zero, then do a notice-level log of whether the connection is possibly leaking DNS requests locally or not.

If **safe_socks** is true, then reject unsafe socks protocols.

If returning 0 or -1, **req->address** and **req->port** are undefined.

References socks_request_t::address, addressmap_have_mapping(), buf_clear(), buf_pullup(), buf_remove_from_front(), CHUNK_WRITE_PTR(), socks_request_t::command, control_event_client_status(), chunk_t::data, chunk_t::datalen, buf_t::datalen, escaped(), get_uint16(), get_uint32(), buf_t::head, LD_APP, LD_PROTOCOL, socks_request_t::port, socks_request_t::reply, socks_request_t::replylen, safe_str_client(), SOCKS4_NETWORK_LEN, SOCKS_COMMAND_CONNECT, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, socks_request_t::socks_version, tor_addr_from_ipv4n(), tor_addr_from_ipv6_bytes(), tor_addr_to_str(), tor_assert, tor_free, tor_inet_ntoa(), and tor_strisprint().

Referenced by connection_ap_handshake_process_socks().

4.5.3.32 int fetch_from_buf_socks_client (buf_t * *buf*, int *state*, char ** *reason*)

Inspect a reply from SOCKS server stored in **buf** according to **state**, removing the protocol data upon success. Return 0 on incomplete response, 1 on success and -1 on error, in which case **reason** is set to a descriptive message (free() when finished with it).

As a special case, 2 is returned when user/pass is required during SOCKS5 handshake and user/pass is configured.

References buf_clear(), buf_pullup(), buf_remove_from_front(), chunk_t::data, chunk_t::datalen, buf_t::datalen, buf_t::head, LD_NET, socks4_response_code_to_string(), socks5_response_code_to_string(), and tor_assert.

Referenced by connection_read_proxy_handshake().

4.5.3.33 int fetch_var_cell_from_buf (buf_t * *buf*, var_cell_t ** *out*, int *linkproto*)

Check **buf** for a variable-length cell according to the rules of link protocol version **linkproto**. If one is found, pull it off the buffer and assign a newly allocated **var_cell_t** (p. 250) to ***out**, and return 1. Return 0 if whatever is on the start of **buf_t** (p. 18) is not a variable-length cell. Return 1 and set ***out** to NULL if there seems to be the start of a variable-length cell on **buf**, but the whole thing isn't there yet.

References buf_remove_from_front(), CELL_COMMAND_IS_VAR_LENGTH, var_cell_t::circ_id, var_cell_t::command, buf_t::datalen, get_uint16(), var_cell_t::payload, peek_from_buf(), VAR_CELL_HEADER_SIZE, and var_cell_new().

Referenced by connection_fetch_var_cell_from_buf().

4.5.3.34 int flush_buf (int *s*, buf_t ∗ *buf*, size_t *sz*, size_t ∗ *buf_flushlen*)

Write data from **buf** to the socket **s**. Write at most **sz** bytes, decrement ∗**buf_flushlen** by the number of bytes actually written, and remove the written bytes from the buffer. Return the number of bytes written on success, -1 on failure. Return 0 if write() would block.

References chunk_t::datalen, flush_chunk(), buf_t::head, and tor_assert.

Referenced by conn_close_if_marked(), and connection_handle_write_impl().

4.5.3.35 int flush_buf_tls (tor_tls_t ∗ *tls*, buf_t ∗ *buf*, size_t *flushlen*, size_t ∗ *buf_flushlen*)

As **flush_buf()** (p. 293), but writes data to a TLS connection. Can write more than **flushlen** bytes.

References chunk_t::datalen, flush_chunk_tls(), buf_t::head, and tor_assert.

Referenced by conn_close_if_marked(), and connection_handle_write_impl().

4.5.3.36 static INLINE int flush_chunk (int *s*, buf_t ∗ *buf*, chunk_t ∗ *chunk*, size_t *sz*, size_t ∗ *buf_flushlen*) `[static]`

Helper for **flush_buf()** (p. 293): try to write **sz** bytes from chunk **chunk** of buffer **buf** onto socket **s**. On success, deduct the bytes written from ∗**buf_flushlen**. Return the number of bytes written on success, 0 on blocking, -1 on failure.

References buf_remove_from_front(), chunk_t::data, chunk_t::datalen, LD_NET, chunk_t::next, and tor_assert.

Referenced by flush_buf().

4.5.3.37 static INLINE int flush_chunk_tls (tor_tls_t ∗ *tls*, buf_t ∗ *buf*, chunk_t ∗ *chunk*, size_t *sz*, size_t ∗ *buf_flushlen*) `[static]`

Helper for **flush_buf_tls()** (p. 293): try to write **sz** bytes from chunk **chunk** of buffer **buf** onto socket **s**. (Tries to write more if there is a forced pending write size.) On success, deduct the bytes written from ∗**buf_flushlen**. Return the number of bytes written on success, and a TOR_TLS error code on failure or blocking.

References buf_remove_from_front(), chunk_t::data, buf_t::datalen, LD_NET, tor_assert, tor_tls_get_forced_write_size(), and tor_tls_write().

Referenced by flush_buf_tls().

4.5.3.38 static INLINE chunk_freelist_t∗ get_freelist (size_t *alloc*) `[static]`

Return the freelist to hold chunks of size **alloc**, or NULL if no freelist exists for that size.

References chunk_freelist_t::alloc_size.

Referenced by chunk_free_unchecked(), and chunk_new_with_alloc_size().

4.5.3.39 int move_buf_to_buf (buf_t * *buf_out*, buf_t * *buf_in*, size_t * *buf_flushlen*)

Move up to ***buf_flushlen** bytes from **buf_in** to **buf_out**, and modify ***buf_flushlen** appropriately. Return the number of bytes actually copied.

References buf_t::datalen, fetch_from_buf(), tor_assert, and write_to_buf().

Referenced by conn_close_if_marked(), and connection_read_to_buf().

4.5.3.40 int peek_buf_has_control0_command (buf_t * *buf*)

Return 1 iff buf looks more like it has an (obsolete) v0 controller command on it than any valid v1 controller command.

References buf_t::datalen, get_uint16(), and peek_from_buf().

Referenced by connection_control_process_inbuf().

4.5.3.41 static INLINE void peek_from_buf (char * *string*, size_t *string_len*, const buf_t * *buf*) `[static]`

Helper: copy the first **string_len** bytes from **buf** onto **string**.

References chunk_t::data, chunk_t::datalen, buf_t::head, chunk_t::next, and tor_assert.

Referenced by fetch_from_buf(), fetch_var_cell_from_buf(), and peek_buf_has_control0_command().

4.5.3.42 static INLINE size_t preferred_chunk_size (size_t *target*) `[static]`

Return the allocation size we'd like to use to hold **target** bytes.

References CHUNK_SIZE_WITH_ALLOC, and MIN_CHUNK_ALLOC.

Referenced by buf_add_chunk_with_capacity(), buf_new_with_capacity(), and buf_pullup().

4.5.3.43 int read_to_buf (int *s*, size_t *at_most*, buf_t * *buf*, int * *reached_eof*, int * *socket_error*)

Read from socket **s**, writing onto end of **buf**. Read at most **at_most** bytes, growing the buffer as necessary. If recv() returns 0 (because of EOF), set ***reached_eof** to 1 and return 0. Return -1 on error; else return the number of bytes read.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), chunk_t::memlen, MIN_READ_LEN, read_to_chunk(), buf_t::tail, and tor_assert.

Referenced by connection_read_to_buf().

4.5.3.44 int read_to_buf_tls (tor_tls_t * *tls*, size_t *at_most*, buf_t * *buf*)

As read_to_buf, but reads from a TLS connection, and returns a TLS status value rather than the number of bytes read.

Using TLS on OR connections complicates matters in two ways.

First, a TLS stream has its own read buffer independent of the connection's read buffer. (TLS needs to read an entire frame from the network before it can decrypt any data. Thus, trying to read 1 byte from TLS can require that several KB be read from the network and decrypted. The extra data is stored in TLS's decrypt buffer.) Because the data hasn't been read by Tor (it's still inside the TLS), this means that sometimes a connection "has stuff to read" even when poll() didn't return POLLIN. The tor_tls_get_pending_bytes function is used in **connection.c** (p. 406) to detect TLS objects with non-empty internal buffers and read from them again.

Second, the TLS stream's events do not correspond directly to network events: sometimes, before a TLS stream can read, the network must be ready to write – or vice versa.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), chunk_-t::memlen, MIN_READ_LEN, read_to_chunk_tls(), buf_t::tail, and tor_assert.

Referenced by connection_read_to_buf().

4.5.3.45 static INLINE int read_to_chunk (buf_t * *buf*, chunk_t * *chunk*, int *fd*, size_t *at_most*, int * *reached_eof*, int * *socket_error*) `[static]`

Read up to **at_most** bytes from the socket **fd** into **chunk** (which must be on **buf**). If we get an EOF, set *__reached_eof__* to 1. Return -1 on error, 0 on eof or blocking, and the number of bytes read otherwise.

References CHUNK_REMAINING_CAPACITY(), CHUNK_WRITE_PTR(), chunk_-t::datalen, buf_t::datalen, LD_NET, chunk_t::next, and tor_assert.

Referenced by read_to_buf().

4.5.3.46 static INLINE int read_to_chunk_tls (buf_t * *buf*, chunk_t * *chunk*, tor_tls_t * *tls*, size_t *at_most*) `[static]`

As **read_to_chunk()** (p. 295), but return (negative) error code on error, blocking, or TLS, and the number of bytes read otherwise.

References CHUNK_REMAINING_CAPACITY(), CHUNK_WRITE_PTR(), chunk_-t::datalen, buf_t::datalen, tor_assert, and tor_tls_read().

Referenced by read_to_buf_tls().

4.5.3.47 int write_to_buf (const char * *string*, size_t *string_len*, buf_t * *buf*)

Append **string_len** bytes from **string** to the end of **buf**.

Return the new length of the buffer on success, -1 on failure.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), CHUNK_WRITE_PTR(), chunk_t::datalen, buf_t::datalen, buf_t::tail, and tor_assert.

Referenced by _connection_write_to_buf_impl(), and move_buf_to_buf().

4.5.3.48 int write_to_buf_zlib (buf_t * *buf*, tor_zlib_state_t * *state*, const char * *data*, size_t *data_len*, int *done*)

Compress on uncompress the **data_len** bytes in **data** using the zlib state **state**, appending the result to **buf**. If **done** is true, flush the data in the state and finish the compression/uncompression. Return -1 on failure, 0 on success.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), CHUNK_WRITE_PTR(), chunk_t::datalen, buf_t::datalen, buf_t::tail, and tor_zlib_process().

Referenced by _connection_write_to_buf_impl().

4.5.4 Variable Documentation

4.5.4.1 chunk_freelist_t freelists[ ] `[static]`

Initial value:

```
 {
  FL(4096, 256, 8), FL(8192, 128, 4), FL(16384, 64, 4), FL(32768, 32, 2),

}
```

Static array of freelists, sorted by alloc_len, terminated by an entry with alloc_size of 0.

4.5.4.2 uint64_t n_freelist_miss = 0 `[static]`

How many times have we looked for a chunk of a size that no freelist could help with?

Referenced by buf_dump_freelist_sizes(), and chunk_new_with_alloc_size().

4.6 circuitbuild.c File Reference

The actual details of building circuits.

`#include "or.h"`

`#include "crypto.h"`

`#include <math.h>`

Data Structures

- struct **entry_guard_t**
- struct **bridge_info_t**

Defines

- #define **CIRCUIT_PRIVATE**
- #define **ENTRY_GUARD_REMOVE_AFTER** (30*24*60*60)

Functions

- static int **circuit_deliver_create_cell** (**circuit_t** *circ, uint8_t cell_type, const char *payload)
- static int **onion_pick_cpath_exit** (**origin_circuit_t** *circ, **extend_info_t** *exit)
- static **crypt_path_t** * **onion_next_hop_in_cpath** (**crypt_path_t** *cpath)
- static int **onion_extend_cpath** (**origin_circuit_t** *circ)
- static int **count_acceptable_routers** (**smartlist_t** *routers)
- static int **onion_append_hop** (**crypt_path_t** **head_ptr, **extend_info_t** *choice)
- static void **entry_guards_changed** (void)
- static int32_t **circuit_build_times_max_timeouts** (void)
- static int32_t **circuit_build_times_min_circs_to_observe** (void)
- double **circuit_build_times_quantile_cutoff** (void)
- static int32_t **circuit_build_times_test_frequency** (void)
- static int32_t **circuit_build_times_min_timeout** (void)
- int32_t **circuit_build_times_initial_timeout** (void)
- static int32_t **circuit_build_times_recent_circuit_count** (void)
- void **circuit_build_times_new_consensus_params** (**circuit_build_times_t** *cbt, **networkstatus_t** *ns)
- void **circuitbuild_running_unit_tests** (void)
- static double **circuit_build_times_get_initial_timeout** (void)
- void **circuit_build_times_reset** (**circuit_build_times_t** *cbt)
- void **circuit_build_times_init** (**circuit_build_times_t** *cbt)
- static void **circuit_build_times_rewind_history** (**circuit_build_times_t** *cbt, int n)
- int **circuit_build_times_add_time** (**circuit_build_times_t** *cbt, **build_time_t** time)
- static **build_time_t** **circuit_build_times_max** (**circuit_build_times_t** *cbt)
- static uint32_t * **circuit_build_times_create_histogram** (**circuit_build_times_t** *cbt, **build_time_t** *nbins)
- static **build_time_t** **circuit_build_times_mode** (**circuit_build_times_t** *cbt)

- void **circuit_build_times_update_state** (**circuit_build_times_t** *cbt, **or_state_t** *state)
- static void **circuit_build_times_shuffle_and_store_array** (**circuit_build_times_t** *cbt, **build_time_t** *raw_times, int num_times)
- int **circuit_build_times_parse_state** (**circuit_build_times_t** *cbt, **or_state_t** *state, char **msg)
- void **circuit_build_times_update_alpha** (**circuit_build_times_t** *cbt)
- double **circuit_build_times_calculate_timeout** (**circuit_build_times_t** *cbt, double quantile)
- double **circuit_build_times_cdf** (**circuit_build_times_t** *cbt, double x)
- **build_time_t circuit_build_times_generate_sample** (**circuit_build_times_t** *cbt, double q_lo, double q_hi)
- void **circuit_build_times_add_timeout_worker** (**circuit_build_times_t** *cbt, double quantile_cutoff)
- void **circuit_build_times_initial_alpha** (**circuit_build_times_t** *cbt, double quantile, double timeout_ms)
- static void **circuit_build_times_count_pretimeouts** (**circuit_build_times_t** *cbt)
- int **circuit_build_times_needs_circuits** (**circuit_build_times_t** *cbt)
- int **circuit_build_times_needs_circuits_now** (**circuit_build_times_t** *cbt)
- void **circuit_build_times_network_is_live** (**circuit_build_times_t** *cbt)
- void **circuit_build_times_network_circ_success** (**circuit_build_times_t** *cbt)
- static void **circuit_build_times_network_timeout** (**circuit_build_times_t** *cbt, int did_onehop, time_t start_time)
- int **circuit_build_times_network_check_live** (**circuit_build_times_t** *cbt)
- int **circuit_build_times_network_check_changed** (**circuit_build_times_t** *cbt)
- int **circuit_build_times_add_timeout** (**circuit_build_times_t** *cbt, int did_onehop, time_t start_time)
- void **circuit_build_times_set_timeout** (**circuit_build_times_t** *cbt)
- static **circid_t get_unique_circ_id_by_conn** (**or_connection_t** *conn)
- static char * **circuit_list_path_impl** (**origin_circuit_t** *circ, int verbose, int verbose_names)
- char * **circuit_list_path** (**origin_circuit_t** *circ, int verbose)
- char * **circuit_list_path_for_controller** (**origin_circuit_t** *circ)
- void **circuit_log_path** (int severity, unsigned int domain, **origin_circuit_t** *circ)
- void **circuit_rep_hist_note_result** (**origin_circuit_t** *circ)
- static int **onion_populate_cpath** (**origin_circuit_t** *circ)
- **origin_circuit_t** * **origin_circuit_init** (uint8_t purpose, int flags)
- **origin_circuit_t** * **circuit_establish_circuit** (uint8_t purpose, **extend_info_t** *exit, int flags)
- int **circuit_handle_first_hop** (**origin_circuit_t** *circ)
- void **circuit_n_conn_done** (**or_connection_t** *or_conn, int status)
- int **inform_testing_reachability** (void)
- static INLINE int **should_use_create_fast_for_circuit** (**origin_circuit_t** *circ)
- int **circuit_send_next_onion_skin** (**origin_circuit_t** *circ)
- void **circuit_note_clock_jumped** (int seconds_elapsed)
- int **circuit_extend** (**cell_t** *cell, **circuit_t** *circ)
- int **circuit_init_cpath_crypto** (**crypt_path_t** *cpath, const char *key_data, int reverse)
- int **circuit_finish_handshake** (**origin_circuit_t** *circ, uint8_t reply_type, const char *reply)

- int **circuit_truncated** (**origin_circuit_t** ∗circ, **crypt_path_t** ∗layer)
- int **onionskin_answer** (**or_circuit_t** ∗circ, uint8_t cell_type, const char ∗payload, const char ∗keys)
- static int **new_route_len** (uint8_t purpose, **extend_info_t** ∗exit, **smartlist_t** ∗routers)
- static **smartlist_t** ∗ **circuit_get_unhandled_ports** (time_t now)
- int **circuit_all_predicted_ports_handled** (time_t now, int ∗need_uptime, int ∗need_capacity)
- static int **router_handles_some_port** (**routerinfo_t** ∗router, **smartlist_t** ∗needed_ports)
- static int **ap_stream_wants_exit_attention** (**connection_t** ∗conn)
- static **routerinfo_t** ∗ **choose_good_exit_server_general** (**routerlist_t** ∗dir, int need_uptime, int need_capacity)
- static **routerinfo_t** ∗ **choose_good_exit_server** (uint8_t purpose, **routerlist_t** ∗dir, int need_uptime, int need_capacity, int is_internal)
- static void **warn_if_last_router_excluded** (**origin_circuit_t** ∗circ, const **extend_info_t** ∗exit)
- int **circuit_append_new_exit** (**origin_circuit_t** ∗circ, **extend_info_t** ∗exit)
- int **circuit_extend_to_new_exit** (**origin_circuit_t** ∗circ, **extend_info_t** ∗exit)
- void **onion_append_to_cpath** (**crypt_path_t** ∗∗head_ptr, **crypt_path_t** ∗new_hop)
- static **routerinfo_t** ∗ **choose_good_middle_server** (uint8_t purpose, **cpath_build_state_t** ∗state, **crypt_path_t** ∗head, int cur_len)
- static **routerinfo_t** ∗ **choose_good_entry_server** (uint8_t purpose, **cpath_build_state_t** ∗state)
- **extend_info_t** ∗ **extend_info_alloc** (const char ∗nickname, const char ∗digest, **crypto_pk_env_t** ∗onion_key, const **tor_addr_t** ∗addr, uint16_t port)
- **extend_info_t** ∗ **extend_info_from_router** (**routerinfo_t** ∗r)
- void **extend_info_free** (**extend_info_t** ∗info)
- **extend_info_t** ∗ **extend_info_dup** (**extend_info_t** ∗info)
- **routerinfo_t** ∗ **build_state_get_exit_router** (**cpath_build_state_t** ∗state)
- const char ∗ **build_state_get_exit_nickname** (**cpath_build_state_t** ∗state)
- static int **entry_guard_set_status** (**entry_guard_t** ∗e, **routerinfo_t** ∗ri, time_t now, **or_options_t** ∗options, const char ∗∗reason)
- static int **entry_is_time_to_retry** (**entry_guard_t** ∗e, time_t now)
- static INLINE **routerinfo_t** ∗ **entry_is_live** (**entry_guard_t** ∗e, int need_uptime, int need_capacity, int assume_reachable, const char ∗∗msg)
- static int **num_live_entry_guards** (void)
- static INLINE **entry_guard_t** ∗ **is_an_entry_guard** (const char ∗digest)
- static void **log_entry_guards** (int severity)
- static void **control_event_guard_deferred** (void)
- static **routerinfo_t** ∗ **add_an_entry_guard** (**routerinfo_t** ∗chosen, int reset_status)
- static void **pick_entry_guards** (void)
- static void **entry_guard_free** (**entry_guard_t** ∗e)
- static int **remove_obsolete_entry_guards** (void)
- static int **remove_dead_entry_guards** (void)
- void **entry_guards_compute_status** (void)
- int **entry_guard_register_connect_status** (const char ∗digest, int succeeded, int mark_relay_status, time_t now)
- void **entry_nodes_should_be_added** (void)
- static void **entry_guards_prepend_from_config** (void)

- int **entry_list_is_constrained** (**or_options_t** ∗options)
- static int **entry_list_is_totally_static** (**or_options_t** ∗options)
- **routerinfo_t** ∗ **choose_random_entry** (**cpath_build_state_t** ∗state)
- int **entry_guards_parse_state** (**or_state_t** ∗state, int set, char ∗∗msg)
- void **entry_guards_update_state** (**or_state_t** ∗state)
- int **getinfo_helper_entry_guards** (**control_connection_t** ∗conn, const char ∗question, char ∗∗answer)
- void **clear_bridge_list** (void)
- static **bridge_info_t** ∗ **routerinfo_get_configured_bridge** (**routerinfo_t** ∗ri)
- int **routerinfo_is_a_configured_bridge** (**routerinfo_t** ∗ri)
- void **bridge_add_from_config** (const **tor_addr_t** ∗addr, uint16_t port, char ∗digest)
- static **bridge_info_t** ∗ **find_bridge_by_digest** (const char ∗digest)
- static void **launch_direct_bridge_descriptor_fetch** (**bridge_info_t** ∗bridge)
- void **retry_bridge_descriptor_fetch_directly** (const char ∗digest)
- void **fetch_bridge_descriptors** (time_t now)
- void **learned_bridge_descriptor** (**routerinfo_t** ∗ri, int from_cache)
- int **any_bridge_descriptors_known** (void)
- int **any_pending_bridge_descriptor_fetches** (void)
- static int **bridges_retry_helper** (int act)
- int **bridges_known_but_down** (void)
- void **bridges_retry_all** (void)
- void **entry_guards_free_all** (void)

Variables

- **circuit_build_times_t circ_times**
- **circuit_t** ∗ **global_circuitlist**
- static **smartlist_t** ∗ **entry_guards** = NULL
- static int **entry_guards_dirty** = 0
- static int **unit_tests** = 0
- static int **should_add_entry_nodes** = 0
- static **smartlist_t** ∗ **bridge_list** = NULL

4.6.1 Detailed Description

The actual details of building circuits.

4.6.2 Define Documentation

4.6.2.1 #define ENTRY_GUARD_REMOVE_AFTER (30∗24∗60∗60)

How long (in seconds) do we allow an entry guard to be nonfunctional, unlisted, excluded, or otherwise nonusable before we give up on it?

Referenced by remove_dead_entry_guards().

4.6.3 Function Documentation

4.6.3.1 static routerinfo_t∗ add_an_entry_guard (routerinfo_t ∗ *chosen*, int *reset_status*) [static]

Add a new (preferably stable and fast) router to our entry_guards list. Return a pointer to the router if we succeed, or NULL if we can't find any more suitable entries.

If **chosen** is defined, use that one, and if it's not already in our entry_guards list, put it at the ∗beginning∗. Else, put the one we pick at the end of the list.

References entry_guard_t::bad_since, routerinfo_t::cache_info, entry_guard_t::can_retry, choose_good_entry_server(), entry_guard_t::chosen_by_version, entry_guard_t::chosen_on_date, CIRCUIT_PURPOSE_C_GENERAL, control_event_guard(), control_event_guard_deferred(), crypto_rand_int(), DIGEST_LEN, entry_guard_t::identity, signed_descriptor_t::identity_digest, is_an_entry_guard(), LD_CIRC, log_entry_guards(), entry_guard_t::nickname, routerinfo_t::nickname, smartlist_add(), and smartlist_insert().

Referenced by choose_random_entry(), learned_bridge_descriptor(), and pick_entry_guards().

4.6.3.2 int any_bridge_descriptors_known (void)

Return 1 if any of our entry guards have descriptors that are marked with purpose 'bridge' and are running. Else return 0.

We use this function to decide if we're ready to start building circuits through our bridges, or if we need to wait until the directory "server/authority" requests finish.

References choose_random_entry(), get_options(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), control_event_bootstrap_problem(), learned_bridge_descriptor(), and should_delay_dir_fetches().

4.6.3.3 int any_pending_bridge_descriptor_fetches (void)

Return 1 if there are any directory conns fetching bridge descriptors that aren't marked for close. We use this to guess if we should tell the controller that we have a problem.

References CONN_TYPE_DIR, DIR_PURPOSE_FETCH_SERVERDESC, get_connection_array(), LD_DIR, ROUTER_PURPOSE_BRIDGE, and TO_DIR_CONN().

Referenced by control_event_bootstrap_problem().

4.6.3.4 static int ap_stream_wants_exit_attention (connection_t ∗ *conn*) [static]

Return true iff **conn** needs another general circuit to be built.

References AP_CONN_STATE_CIRCUIT_WAIT, circuit_stream_is_being_handled(), CONN_TYPE_AP, connection_edge_is_rendezvous_stream(), connection_t::marked_for_close, MIN_CIRCUITS_HANDLING_STREAM, connection_t::state, TO_EDGE_CONN(), and connection_t::type.

Referenced by choose_good_exit_server_general().

4.6.3.5 void bridge_add_from_config (const tor_addr_t * *addr*, uint16_t *port*, char * *digest*)

Remember a new bridge at **addr**:**port**. If **digest** is set, it tells us the identity key too.

References bridge_info_t::addr, DIGEST_LEN, bridge_info_t::fetch_status, bridge_info_t::identity, bridge_info_t::port, download_status_t::schedule, smartlist_add(), smartlist_create(), and tor_addr_copy().

Referenced by parse_bridge_line().

4.6.3.6 int bridges_known_but_down (void)

Do we know any descriptors for our bridges, and are they all down?

References bridges_retry_helper().

Referenced by circuit_get_open_circ_or_launch().

4.6.3.7 void bridges_retry_all (void)

Mark all down known bridges up.

References bridges_retry_helper().

Referenced by circuit_get_open_circ_or_launch().

4.6.3.8 static int bridges_retry_helper (int *act*) `[static]`

Return 1 if we have at least one descriptor for a bridge and all descriptors we know are down. Else return 0. If **act** is 1, then mark the down bridges up; else just observe and report.

References routerinfo_t::is_running, LD_DIR, routerinfo_t::purpose, router_get_by_digest(), ROUTER_PURPOSE_BRIDGE, and smartlist_create().

Referenced by bridges_known_but_down(), and bridges_retry_all().

4.6.3.9 const char* build_state_get_exit_nickname (cpath_build_state_t * *state*)

Return the nickname for the chosen exit router in **state**. If there is no chosen exit, or if we don't know the **routerinfo_t** (p. 206) for the chosen exit, return NULL.

References cpath_build_state_t::chosen_exit, and extend_info_t::nickname.

Referenced by _circuit_mark_for_close(), circuit_build_failed(), circuit_launch_by_extend_info(), circuit_list_path_impl(), and onion_extend_cpath().

4.6.3.10 routerinfo_t* build_state_get_exit_router (cpath_build_state_t * *state*)

Return the **routerinfo_t** (p. 206) for the chosen exit router in **state**. If there is no chosen exit, or if we don't know the **routerinfo_t** (p. 206) for the chosen exit, return NULL.

References cpath_build_state_t::chosen_exit, extend_info_t::identity_digest, and router_get_by_digest().

Referenced by choose_good_entry_server(), choose_good_middle_server(), choose_random_entry(), circuit_is_acceptable(), and circuit_stream_is_being_handled().

4.6.3.11 static routerinfo_t* choose_good_entry_server (uint8_t *purpose*, cpath_build_state_t * *state*) `[static]`

Pick a good entry server for the circuit to be built according to **state**. Don't reuse a chosen exit (if any), don't use this router (if we're an OR), and respect firewall settings; if we're configured to use entry guards, return one.

If **state** is NULL, we're choosing a router to serve as an entry guard, not for any particular circuit.

References or_options_t::BridgeRelay, build_state_get_exit_router(), choose_random_entry(), CIRCUIT_PURPOSE_TESTING, fascist_firewall_allows_or(), firewall_is_fascist_or(), get_options(), router_get_by_digest(), router_get_routerlist(), routerlist_add_family(), routerlist_t::routers, smartlist_add(), smartlist_create(), and or_options_t::UseEntryGuards.

Referenced by add_an_entry_guard(), and onion_extend_cpath().

4.6.3.12 static routerinfo_t* choose_good_exit_server (uint8_t *purpose*, routerlist_t * *dir*, int *need_uptime*, int *need_capacity*, int *is_internal*) `[static]`

Return a pointer to a suitable router to be the exit node for the circuit of purpose **purpose** that we're about to build (or NULL if no router is suitable).

For general-purpose circuits, pass it off to **choose_good_exit_server_general()** (p. 303)

For client-side rendezvous circuits, choose a random node, weighted toward the preferences in 'options'.

References or_options_t::_AllowInvalid, choose_good_exit_server_general(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, or_options_t::ExcludeNodes, get_options(), LD_BUG, router_choose_random_node(), and tor_fragile_assert.

Referenced by onion_pick_cpath_exit().

4.6.3.13 static routerinfo_t* choose_good_exit_server_general (routerlist_t * *dir*, int *need_uptime*, int *need_capacity*) `[static]`

Return a pointer to a suitable router to be the exit node for the general-purpose circuit we're about to build.

Look through the connection array, and choose a router that maximizes the number of pending streams that can exit from this router.

Return NULL if we can't find any suitable routers.

References or_options_t::_AllowInvalid, or_options_t::_ExcludeExitNodesUnion, routerinfo_t::allow_single_hop_exits, ap_stream_wants_exit_attention(), circuit_get_unhandled_ports(), connection_ap_can_use_exit(), or_options_t::ExcludeSingleHopRelays, or_options_t::ExitNodes, get_connection_array(), get_options(), routerinfo_t::is_bad_exit, routerinfo_t::is_running, routerinfo_t::is_valid, LD_CIRC, routerinfo_t::nickname, router_exit_policy_rejects_all(), router_handles_some_port(), router_is_me(), router_is_unreliable(), routerlist_sl_choose_by_bandwidth(), routerlist_t::routers, routerset_contains_router(), routersets_get_disjunction(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), or_options_t::StrictNodes, TO_EDGE_CONN(), and tor_free.

Referenced by choose_good_exit_server().

4.6.3.14 static routerinfo_t∗ choose_good_middle_server (uint8_t *purpose*, cpath_build_state_t ∗ *state*, crypt_path_t ∗ *head*, int *cur_len*) `[static]`

A helper function used by **onion_extend_cpath()** (p. 323). Use **purpose** and **state** and the cpath **head** (currently populated only to length **cur_len** to decide a suitable middle hop for a circuit. In particular, make sure we don't pick the exit node or its family, and make sure we don't duplicate any previous nodes or their families.

References or_options_t::_AllowInvalid, build_state_get_exit_router(), or_options_t::ExcludeNodes, get_options(), LD_CIRC, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, crypt_path_t::next, router_choose_random_node(), router_get_by_digest(), routerlist_add_family(), smartlist_add(), smartlist_create(), smartlist_free(), and tor_assert.

Referenced by onion_extend_cpath().

4.6.3.15 routerinfo_t∗ choose_random_entry (cpath_build_state_t ∗ *state*)

Pick a live (up and listed) entry guard from entry_guards. If **state** is non-NULL, this is for a specific circuit – make sure not to pick this circuit's exit or any node in the exit's family. If **state** is NULL, we're looking for a random guard (likely a bridge).

References add_an_entry_guard(), build_state_get_exit_router(), entry_guards_changed(), entry_guards_prepend_from_config(), entry_is_live(), entry_list_is_constrained(), or_options_t::EntryNodes, get_options(), LD_CIRC, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, or_options_t::NumEntryGuards, pick_entry_guards(), routerlist_add_family(), routerlist_sl_choose_by_bandwidth(), routerset_contains_router(), smartlist_add(), smartlist_choose(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_isin(), or_options_t::StrictNodes, and tor_fragile_assert.

Referenced by any_bridge_descriptors_known(), choose_good_entry_server(), and directory_get_from_dirserver().

4.6.3.16 int circuit_all_predicted_ports_handled (time_t *now*, int ∗ *need_uptime*, int ∗ *need_capacity*)

Return 1 if we already have circuits present or on the way for all anticipated ports. Return 0 if we should make more.

If we're returning 0, set need_uptime and need_capacity to indicate any requirements that the unhandled ports have.

References circuit_get_unhandled_ports(), get_options(), or_options_t::LongLivedPorts, smartlist_free(), smartlist_string_num_isin(), tor_assert, and tor_free.

Referenced by circuit_predict_and_launch_new().

4.6.3.17 int circuit_append_new_exit (origin_circuit_t * *circ*, extend_info_t * *exit*)

Give **circ** a new exit destination to **exit**, and add a hop to the cpath reflecting this. Don't send the next extend cell – the caller will do this if it wants to.

References origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, origin_circuit_t::cpath, cpath_build_state_t::desired_path_len, extend_info_dup(), extend_info_free(), onion_append_hop(), and tor_assert.

Referenced by circuit_extend_to_new_exit(), and handle_control_extendcircuit().

4.6.3.18 int circuit_build_times_add_time (circuit_build_times_t * *cbt*, build_time_t *time*)

Add a new timeout value **time** to the set of build times. Time units are milliseconds.

circuit_build_times **cbt is a circular array, so loop around when array is full.**

References or_options_t::AvoidDiskWrites, circuit_build_times_t::build_times_idx, CBT_NCIRCUITS_TO_OBSERVE, CBT_SAVE_STATE_EVERY, circuit_build_times_t::circuit_build_times, get_options(), get_or_state(), LD_CIRC, or_state_mark_dirty(), tor_assert, circuit_build_times_t::total_build_times, and unit_tests.

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_shuffle_and_store_array(), and circuit_send_next_onion_skin().

4.6.3.19 int circuit_build_times_add_timeout (circuit_build_times_t * *cbt*, int *did_onehop*, time_t *start_time*)

Store a timeout as a synthetic value.

Returns true if the store was successful and we should possibly update our timeout estimate.

References circuit_build_times_add_timeout_worker(), circuit_build_times_count_pretimeouts(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), circuit_build_times_network_timeout(), circuit_build_times_t::have_computed_timeout, LD_CIRC, circuit_build_times_t::pre_timeouts, and circuit_build_times_t::total_build_times.

Referenced by circuit_expire_building().

4.6.3.20 void circuit_build_times_add_timeout_worker (circuit_build_times_t * *cbt*, double *quantile_cutoff*)

Generate points in [cutoff, 1.0) on the CDF.

References circuit_build_times_t::alpha, CBT_MAX_SYNTHETIC_QUANTILE, circuit_build_times_add_time(), circuit_build_times_generate_sample(), LD_CIRC, circuit_build_times_t::timeout_ms, tor_lround(), and circuit_build_times_t::Xm.

Referenced by circuit_build_times_add_timeout(), and circuit_build_times_count_pretimeouts().

4.6.3.21 double circuit_build_times_calculate_timeout (circuit_build_times_t * *cbt*, double *quantile*)

This is the Pareto Quantile Function. It calculates the point x in the distribution such that F(x) = quantile (ie quantile*100% of the mass of the density function is below x on the curve).

We use it to calculate the timeout and also to generate synthetic values of time for circuits that timeout before completion.

See `http://en.wikipedia.org/wiki/Quantile_function`, `http://en.wikipedia.org/wiki/Inverse_transform_sampling` and `http://en.wikipedia.org/wiki/Pareto_distribution#Generating_a_` random_sample_from_Pareto_distribution That's right. I'll cite wikipedia all day long.

Return value is in milliseconds.

References circuit_build_times_t::alpha, tor_assert, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_generate_sample(), and circuit_build_times_set_timeout().

4.6.3.22 double circuit_build_times_cdf (circuit_build_times_t * *cbt*, double *x*)

Pareto CDF

References circuit_build_times_t::alpha, tor_assert, and circuit_build_times_t::Xm.

4.6.3.23 static void circuit_build_times_count_pretimeouts (circuit_build_times_t * *cbt*) `[static]`

Generate synthetic timeout values for the timeouts that have happened before we estimated our parameters.

References CBT_MAX_SYNTHETIC_QUANTILE, circuit_build_times_add_timeout_worker(), circuit_build_times_initial_alpha(), circuit_build_times_mode(), circuit_build_times_t::pre_timeouts, circuit_build_times_t::timeout_ms, tor_assert, circuit_build_times_t::total_build_times, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_add_timeout(), and circuit_build_times_set_timeout().

4.6.3.24 static uint32_t* circuit_build_times_create_histogram (circuit_build_times_t * *cbt*, build_time_t * *nbins*) `[static]`

Calculate and return a histogram for the set of build times.

Returns an allocated array of histrogram bins representing the frequency of index*CBT_BIN_WIDTH millisecond build times. Also outputs the number of bins in nbins.

The return value must be freed by the caller.

References CBT_BIN_WIDTH, circuit_build_times_t::circuit_build_times, and circuit_build_times_max().

Referenced by circuit_build_times_mode(), and circuit_build_times_update_state().

4.6.3.25 build_time_t circuit_build_times_generate_sample (circuit_build_times_t * *cbt*, double *q_lo*, double *q_hi*)

Generate a synthetic time using our distribution parameters.

The return value will be within the [q_lo, q_hi) quantile points on the CDF.

References circuit_build_times_calculate_timeout(), crypto_rand_uint64(), tor_assert, and tor_lround().

Referenced by circuit_build_times_add_timeout_worker().

4.6.3.26 static double circuit_build_times_get_initial_timeout (void) `[static]`

Return the initial default or configured timeout in milliseconds

References or_options_t::CircuitBuildTimeout, get_options(), LD_CIRC, and unit_tests.

Referenced by circuit_build_times_init(), circuit_build_times_network_check_changed(), and circuit_build_times_network_check_live().

4.6.3.27 void circuit_build_times_init (circuit_build_times_t * *cbt*)

Initialize the buildtimes structure for first use.

Sets the initial timeout value based to either the config setting or BUILD_TIMEOUT_INITIAL_VALUE.

References circuit_build_times_get_initial_timeout(), control_event_buildtimeout_set(), circuit_build_times_t::liveness, network_liveness_t::num_recent_circs, circuit_build_times_t::timeout_ms, and network_liveness_t::timeouts_after_firsthop.

Referenced by circuit_build_times_parse_state().

4.6.3.28 void circuit_build_times_initial_alpha (circuit_build_times_t * *cbt*, double *quantile*, double *timeout_ms*)

Estimate an initial alpha parameter by solving the quantile function with a quantile point and a specific timeout value.

References circuit_build_times_t::alpha, tor_assert, tor_mathlog(), and circuit_build_times_t::Xm.

Referenced by circuit_build_times_count_pretimeouts().

4.6.3.29 static build_time_t circuit_build_times_max (circuit_build_times_t * *cbt*) `[static]`

Return maximum circuit build time

References CBT_NCIRCUITS_TO_OBSERVE, and circuit_build_times_t::circuit_build_times.

Referenced by circuit_build_times_create_histogram().

4.6.3.30 static build_time_t circuit_build_times_mode (circuit_build_times_t * *cbt*) `[static]`

Return the most frequent build time (rounded to CBT_BIN_WIDTH ms).

Ties go in favor of the slower time.

References CBT_BIN_WIDTH, circuit_build_times_create_histogram(), and tor_free.

Referenced by circuit_build_times_count_pretimeouts(), and circuit_build_times_update_alpha().

4.6.3.31 int circuit_build_times_needs_circuits (circuit_build_times_t * *cbt*)

Returns true if we need circuits to be built

References circuit_build_times_t::total_build_times.

Referenced by circuit_build_times_needs_circuits_now(), and circuit_expire_old_circuits_clientside().

4.6.3.32 int circuit_build_times_needs_circuits_now (circuit_build_times_t * *cbt*)

Returns true if we should build a timeout test circuit right now.

References approx_time(), circuit_build_times_needs_circuits(), and circuit_build_times_t::last_circ_at.

Referenced by circuit_predict_and_launch_new().

4.6.3.33 int circuit_build_times_network_check_changed (circuit_build_times_t * *cbt*)

Returns true if we have seen more than MAX_RECENT_TIMEOUT_COUNT of the past RECENT_CIRCUITS time out after the first hop. Used to detect if the network connection has changed significantly.

Also resets the entire timeout history in this case and causes us to restart the process of building test circuits and estimating a new timeout.

References network_liveness_t::after_firsthop_idx, circuit_build_times_get_initial_timeout(), circuit_build_times_reset(), control_event_buildtimeout_set(), LD_CIRC, circuit_build_times_t::liveness, network_liveness_t::num_recent_circs, circuit_build_times_t::timeout_ms, network_liveness_t::timeouts_after_firsthop, tor_lround(), and circuit_build_times_t::total_build_times.

Referenced by circuit_build_times_add_timeout().

4.6.3.34 int circuit_build_times_network_check_live (circuit_build_times_t * *cbt*)

Returns false if the network has not received a cell or tls handshake in the past NETWORK_NOTLIVE_TIMEOUT_COUNT circuits.

Also has the side effect of rewinding the circuit time history in the case of recent liveness changes.

References approx_time(), CBT_NETWORK_NONLIVE_DISCARD_COUNT, CBT_NETWORK_NONLIVE_TIMEOUT_COUNT, circuit_build_times_get_initial_timeout(), circuit_build_times_rewind_history(), control_event_buildtimeout_set(), LD_CIRC, circuit_build_times_t::liveness, network_liveness_t::net_suspended, network_liveness_t::network_last_live, network_liveness_t::nonlive_discarded, network_liveness_t::nonlive_timeouts, circuit_build_times_t::timeout_ms, and tor_lround().

Referenced by circuit_build_times_add_timeout().

4.6.3.35 void circuit_build_times_network_circ_success (circuit_build_times_t ∗ *cbt*)

Called to indicate that we completed a circuit. Because this circuit succeeded, it doesn't count as a timeout-after-the-first-hop.

References network_liveness_t::after_firsthop_idx, circuit_build_times_t::liveness, network_liveness_t::num_recent_circs, and network_liveness_t::timeouts_after_firsthop.

Referenced by circuit_send_next_onion_skin().

4.6.3.36 void circuit_build_times_network_is_live (circuit_build_times_t ∗ *cbt*)

Called to indicate that the network showed some signs of liveness.

This function is called every time we receive a cell. Avoid syscalls, events, and other high-intensity work.

References approx_time(), circuit_build_times_t::liveness, network_liveness_t::network_last_live, network_liveness_t::nonlive_discarded, and network_liveness_t::nonlive_timeouts.

Referenced by connection_or_process_cells_from_inbuf(), connection_or_set_state_open(), and connection_tls_finish_handshake().

4.6.3.37 static void circuit_build_times_network_timeout (circuit_build_times_t ∗ *cbt*, int *did_onehop*, time_t *start_time*) `[static]`

A circuit just timed out. If there has been no recent network activity at all, but this circuit was launched back when we thought the network was live, increment the number of "nonlive" circuit timeouts.

Also distinguish between whether it failed before the first hop and record that in our history for later deciding if the network has changed.

References network_liveness_t::after_firsthop_idx, circuit_build_times_t::liveness, network_liveness_t::network_last_live, network_liveness_t::nonlive_timeouts, network_liveness_t::num_recent_circs, circuit_build_times_t::timeout_ms, and network_liveness_t::timeouts_after_firsthop.

Referenced by circuit_build_times_add_timeout().

4.6.3.38 void circuit_build_times_new_consensus_params (circuit_build_times_t ∗ *cbt*, networkstatus_t ∗ *ns*)

This function is called when we get a consensus update.

It checks to see if we have changed any consensus parameters that require reallocation or discard of previous stats.

References network_liveness_t::after_firsthop_idx, CBT_DEFAULT_RECENT_CIRCUITS, LD_CIRC, circuit_build_times_t::liveness, networkstatus_get_param(), network_liveness_t::num_recent_circs, network_liveness_t::timeouts_after_firsthop, tor_assert, and tor_free.

Referenced by networkstatus_set_current_consensus().

4.6.3.39 int circuit_build_times_parse_state (circuit_build_times_t ∗ *cbt*, or_state_t ∗ *state*, char ∗∗ *msg*)

Load histogram from **state**, shuffling the resulting array after we do so. Use this result to estimate parameters and calculate the timeout.

Returns -1 and sets msg on error. Msg must be freed by the caller.

References or_state_t::BuildtimeHistogram, circuit_build_times_t::circuit_build_times, circuit_build_times_init(), circuit_build_times_set_timeout(), circuit_build_times_shuffle_and_store_array(), LD_CIRC, config_line_t::next, smartlist_create(), smartlist_free(), smartlist_split_string(), tor_assert, tor_free, tor_parse_ulong(), circuit_build_times_t::total_build_times, or_state_t::TotalBuildTimes, and config_line_t::value.

Referenced by or_state_set().

4.6.3.40 void circuit_build_times_reset (circuit_build_times_t ∗ *cbt*)

Reset the build time state.

Leave estimated parameters, timeout and network liveness intact for future use.

References circuit_build_times_t::build_times_idx, circuit_build_times_t::circuit_build_times, circuit_build_times_t::have_computed_timeout, circuit_build_times_t::pre_timeouts, and circuit_build_times_t::total_build_times.

Referenced by circuit_build_times_network_check_changed().

4.6.3.41 static void circuit_build_times_rewind_history (circuit_build_times_t ∗ *cbt*, int *n*) `[static]`

Rewind our timeout history by n timeout positions.

References circuit_build_times_t::build_times_idx, CBT_NCIRCUITS_TO_OBSERVE, circuit_build_times_t::circuit_build_times, LD_CIRC, circuit_build_times_t::pre_timeouts, and circuit_build_times_t::total_build_times.

Referenced by circuit_build_times_network_check_live().

4.6.3.42 void circuit_build_times_set_timeout (circuit_build_times_t ∗ *cbt*)

Estimate a new timeout based on history and set our timeout variable accordingly.

References circuit_build_times_t::alpha, circuit_build_times_calculate_timeout(), circuit_build_times_count_pretimeouts(), circuit_build_times_update_alpha(), control_event_buildtimeout_set(), circuit_build_times_t::have_computed_timeout, LD_CIRC, circuit_build_times_t::timeout_ms, tor_lround(), circuit_build_times_t::total_build_times, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_parse_state(), circuit_expire_building(), and circuit_send_next_onion_skin().

4.6.3.43 static void circuit_build_times_shuffle_and_store_array (circuit_build_times_t * *cbt*, build_time_t * *raw_times*, int *num_times*) [static]

Shuffle the build times array.

Stolen from `http://en.wikipedia.org/wiki/Fisher`

References circuit_build_times_add_time(), crypto_rand_int(), and LD_CIRC.

Referenced by circuit_build_times_parse_state().

4.6.3.44 void circuit_build_times_update_alpha (circuit_build_times_t * *cbt*)

Estimates the Xm and Alpha parameters using `http://en.wikipedia.org/wiki/Pareto_distribution#Parameter_estimation`

The notable difference is that we use mode instead of min to estimate Xm. This is because our distribution is frechet-like. We claim this is an acceptable approximation because we are only concerned with the accuracy of the CDF of the tail.

References circuit_build_times_t::alpha, circuit_build_times_t::circuit_build_times, circuit_build_times_mode(), LD_CIRC, tor_assert, tor_mathlog(), circuit_build_times_t::total_build_times, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_set_timeout().

4.6.3.45 void circuit_build_times_update_state (circuit_build_times_t * *cbt*, or_state_t * *state*)

Output a histogram of current circuit build times to the **or_state_t** (p. 178) state structure.

References or_options_t::AvoidDiskWrites, or_state_t::BuildtimeHistogram, CBT_BIN_WIDTH, circuit_build_times_create_histogram(), config_free_lines(), get_options(), get_or_state(), config_line_t::key, config_line_t::next, or_state_mark_dirty(), tor_free, tor_snprintf(), circuit_build_times_t::total_build_times, or_state_t::TotalBuildTimes, unit_tests, and config_line_t::value.

Referenced by or_state_save().

4.6.3.46 static int circuit_deliver_create_cell (circuit_t * *circ*, uint8_t *cell_type*, const char * *payload*) [static]

Find a new circid that isn't currently in use on the circ->n_conn for the outgoing circuit **circ**, and deliver a cell of type **cell_type** (either CELL_CREATE or CELL_CREATE_FAST) with payload **payload** to this circuit. Return -1 if we failed to find a suitable circid, else return 0.

References append_cell_to_circuit_queue(), CELL_DIRECTION_OUT, cell_t::circ_id, CIRCUIT_IS_ORIGIN, circuit_set_n_circid_orconn(), or_connection_t::client_used, cell_t::command, get_unique_circ_id_by_conn(), LD_CIRC, circuit_t::n_circ_id, circuit_t::n_conn, cell_t::payload, and tor_assert.

Referenced by circuit_extend(), circuit_n_conn_done(), and circuit_send_next_onion_skin().

4.6.3.47 origin_circuit_t* circuit_establish_circuit (uint8_t *purpose*, extend_info_t * *exit*, int *flags*)

Build a new circuit for **purpose**. If **exit** is defined, then use that as your exit router, else choose a suitable exit node.

Also launch a connection to the first OR in the chosen path, if it's not open already.

References circuit_handle_first_hop(), control_event_circuit_status(), END_CIRC_REASON_NOPATH, onion_pick_cpath_exit(), onion_populate_cpath(), origin_circuit_init(), and TO_CIRCUIT.

Referenced by circuit_launch_by_extend_info().

4.6.3.48 int circuit_extend (cell_t * *cell*, circuit_t * *circ*)

Take the 'extend' **cell**, pull out addr/port plus the onion skin and identity digest for the next hop. If we're already connected, pass the onion skin to the next hop using a create cell; otherwise launch a new OR connection, and **circ** will notice when the connection succeeds or fails.

Return -1 if we want to warn and tear down the circuit, else return 0.

References or_connection_t::_base, connection_t::address, circuit_deliver_create_cell(), circuit_set_state(), CIRCUIT_STATE_OR_WAIT, connection_or_connect(), connection_or_get_for_extend(), DIGEST_LEN, extend_info_alloc(), fmt_addr(), get_options(), get_uint16(), get_uint32(), LD_CIRC, LD_OR, LD_PROTOCOL, relay_header_t::length, circuit_t::n_conn, circuit_t::n_conn_onionskin, circuit_t::n_hop, cell_t::payload, connection_t::port, RELAY_HEADER_SIZE, relay_header_unpack(), server_mode(), TO_OR_CIRCUIT(), tor_addr_from_ipv4h, tor_assert, and tor_digest_is_zero().

Referenced by connection_edge_process_relay_cell().

4.6.3.49 int circuit_extend_to_new_exit (origin_circuit_t * *circ*, extend_info_t * *exit*)

Take an open **circ**, and add a new hop at the end, based on **info**. Set its state back to CIRCUIT_STATE_BUILDING, and then send the next extend cell to begin connecting to that hop.

References circuit_append_new_exit(), circuit_send_next_onion_skin(), circuit_set_state(), CIRCUIT_STATE_BUILDING, LD_CIRC, extend_info_t::nickname, TO_CIRCUIT, and warn_if_last_router_excluded().

Referenced by circuit_launch_by_extend_info(), and rend_client_introduction_acked().

4.6.3.50 int circuit_finish_handshake (origin_circuit_t * *circ*, uint8_t *reply_type*, const char * *reply*)

A created or extended cell came back to us on the circuit, and it included **reply** as its body. (If **reply_type** is CELL_CREATED, the body contains (the second DH key, plus KH). If **reply_type** is CELL_CREATED_FAST, the body contains a secret y and a hash H(x|y).)

Calculate the appropriate keys and digests, make sure KH is correct, and initialize this hop of the cpath.

Return - reason if we want to mark circ for close, else return 0.

References CIPHER_KEY_LEN, circuit_init_cpath_crypto(), circuit_log_path(), control_event_circuit_status(), origin_circuit_t::cpath, crypto_dh_free(), crypt_path_t::dh_handshake_state, DIGEST_LEN, fast_client_handshake(), crypt_path_t::fast_handshake_state, crypt_path_t::handshake_digest, LD_CIRC, LD_PROTOCOL, onion_next_hop_in_cpath(), onion_skin_client_handshake(), crypt_path_t::state, and tor_assert.

Referenced by command_process_created_cell(), and connection_edge_process_relay_cell().

4.6.3.51 static smartlist_t∗ circuit_get_unhandled_ports (time_t *now*) `[static]`

Fetch the list of predicted ports, dup it into a smartlist of uint16_t's, remove the ones that are already handled by an existing circuit, and return it.

References circuit_remove_handled_ports(), rep_hist_get_predicted_ports(), smartlist_add(), and smartlist_create().

Referenced by choose_good_exit_server_general(), and circuit_all_predicted_ports_handled().

4.6.3.52 int circuit_handle_first_hop (origin_circuit_t ∗ *circ*)

Start establishing the first hop of our circuit. Figure out what OR we should connect to, and if necessary start the connection to it. If we're already connected, then send the 'create' cell. Return 0 for ok, -reason if circ should be marked-for-close.

References origin_circuit_t::_base, extend_info_t::addr, origin_circuit_t::build_state, circuit_send_next_onion_skin(), connection_or_connect(), connection_or_get_for_extend(), control_event_bootstrap(), origin_circuit_t::cpath, crypt_path_t::extend_info, extend_info_dup(), fmt_addr(), extend_info_t::identity_digest, LD_CIRC, circuit_t::n_conn, circuit_t::n_hop, extend_info_t::nickname, cpath_build_state_t::onehop_tunnel, onion_next_hop_in_cpath(), extend_info_t::port, safe_str_client(), and tor_assert.

Referenced by circuit_establish_circuit(), and handle_control_extendcircuit().

4.6.3.53 int circuit_init_cpath_crypto (crypt_path_t ∗ *cpath*, const char ∗ *key_data*, int *reverse*)

Initialize cpath->{f|b}_{crypto|digest} from the key material in key_data. key_data must contain CPATH_KEY_MATERIAL bytes, which are used as follows:

- 20 to initialize f_digest
- 20 to initialize b_digest
- 16 to key f_crypto
- 16 to key b_crypto

(If 'reverse' is true, then f_XX and b_XX are swapped.)

References crypt_path_t::b_crypto, crypt_path_t::b_digest, CIPHER_KEY_LEN, crypto_create_init_cipher(), crypto_digest_add_bytes(), crypto_new_digest_env(), DIGEST_LEN, crypt_path_t::f_crypto, crypt_path_t::f_digest, LD_BUG, and tor_assert.

Referenced by circuit_finish_handshake(), onionskin_answer(), rend_client_receive_rendezvous(), and rend_service_introduce().

4.6.3.54 char∗ circuit_list_path (origin_circuit_t ∗ *circ*, int *verbose*)

If **verbose** is false, allocate and return a comma-separated list of the currently built elements of **circuit_t** (p. 40). If **verbose** is true, also list information about link status in a more verbose format using spaces.

References circuit_list_path_impl().

Referenced by circuit_log_path().

4.6.3.55 char∗ circuit_list_path_for_controller (origin_circuit_t ∗ *circ*)

Allocate and return a comma-separated list of the currently built elements of **circuit_t** (p. 40), giving each as a verbose nickname.

References circuit_list_path_impl().

Referenced by control_event_circuit_status(), and getinfo_helper_events().

4.6.3.56 static char∗ circuit_list_path_impl (origin_circuit_t ∗ *circ*, int *verbose*, int *verbose_names*) `[static]`

If **verbose** is false, allocate and return a comma-separated list of the currently built elements of **circuit_t** (p. 40). If **verbose** is true, also list information about link status in a more verbose format using spaces. If **verbose_names** is false, give nicknames for Named routers and hex digests for others; if **verbose_names** is true, use $DIGEST=Name style names.

References origin_circuit_t::_base, base16_encode(), origin_circuit_t::build_state, build_state_get_exit_nickname(), CIRCUIT_STATE_OPEN, origin_circuit_t::cpath, cpath_build_state_t::desired_path_len, DIGEST_LEN, crypt_path_t::extend_info, HEX_DIGEST_LEN, extend_info_t::identity_digest, cpath_build_state_t::is_internal, is_legal_nickname(), routerinfo_t::is_named, MAX_NICKNAME_LEN, MAX_VERBOSE_NICKNAME_LEN, cpath_build_state_t::need_uptime, crypt_path_t::next, extend_info_t::nickname, router_get_by_digest(), router_get_consensus_status_by_id(), router_get_verbose_nickname(), routerstatus_get_verbose_nickname(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), crypt_path_t::state, circuit_t::state, tor_asprintf(), tor_assert, tor_free, and tor_snprintf().

Referenced by circuit_list_path(), and circuit_list_path_for_controller().

4.6.3.57 void circuit_log_path (int *severity*, unsigned int *domain*, origin_circuit_t ∗ *circ*)

Log, at severity **severity**, the nicknames of each router in circ's cpath. Also log the length of the cpath, and the intended exit point.

References circuit_list_path(), tor_free, and tor_log().

Referenced by circuit_dump_details(), circuit_expire_building(), circuit_finish_handshake(), connection_ap_handshake_attach_circuit(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell_not_open(), and warn_if_last_router_excluded().

4.6.3.58 void circuit_n_conn_done (or_connection_t * *or_conn*, int *status*)

Find any circuits that are waiting on **or_conn** to become open and get them to send their create cells forward.

Status is 1 if connect succeeded, or 0 if connect failed.

References or_connection_t::_base, connection_t::addr, connection_t::address, circuit_deliver_create_cell(), circuit_get_all_pending_on_or_conn(), CIRCUIT_IS_ORIGIN, circuit_send_next_onion_skin(), circuit_set_state(), CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, DIGEST_LEN, extend_info_free(), or_connection_t::identity_digest, LD_CIRC, or_connection_t::nickname, connection_t::port, smartlist_create(), smartlist_free(), TO_ORIGIN_CIRCUIT(), tor_addr_eq, tor_assert, tor_digest_is_zero(), and tor_free.

Referenced by connection_about_to_close_connection(), and connection_or_set_state_open().

4.6.3.59 void circuit_note_clock_jumped (int *seconds_elapsed*)

Our clock just jumped by **seconds_elapsed**. Assume something has also gone wrong with our network: notify the user, and abandon all not-yet-used circuits.

References circuit_expire_all_dirty_circs(), circuit_mark_all_unused_circs(), control_event_client_status(), control_event_general_status(), get_options(), has_completed_circuit, LD_GENERAL, server_mode(), and tor_log().

Referenced by second_elapsed_callback().

4.6.3.60 void circuit_rep_hist_note_result (origin_circuit_t * *circ*)

Tell the rep(utation)hist(ory) module about the status of the links in circ. Hops that have become OPEN are marked as successfully extended; the _first_ hop that isn't open (if any) is marked as unable to extend.

References routerinfo_t::cache_info, origin_circuit_t::cpath, crypt_path_t::extend_info, get_options(), extend_info_t::identity_digest, signed_descriptor_t::identity_digest, crypt_path_t::next, rep_hist_note_extend_failed(), rep_hist_note_extend_succeeded(), router_get_by_digest(), router_get_my_routerinfo(), server_mode(), and crypt_path_t::state.

Referenced by _circuit_mark_for_close(), and circuit_send_next_onion_skin().

4.6.3.61 int circuit_send_next_onion_skin (origin_circuit_t * *circ*)

This is the backbone function for building circuits.

If circ's first hop is closed, then we need to build a create cell and send it forward.

Otherwise, we need to build a relay extend cell and send it forward.

Return -reason if we want to tear down circ, else return 0.

References origin_circuit_t::_base, extend_info_t::addr, origin_circuit_t::build_state, check_whether_orport_reachable(), circuit_build_times_add_time(), circuit_build_times_network_circ_success(), circuit_build_times_set_timeout(), circuit_deliver_create_cell(), circuit_has_opened(), circuit_rep_hist_note_result(), circuit_reset_failure_count(), circuit_set_state(), CIRCUIT_STATE_BUILDING, CIRCUIT_STATE_OPEN, consider_testing_reachability(), control_event_bootstrap(), control_event_client_status(), origin_circuit_-

t::cpath, crypto_rand(), crypt_path_t::dh_handshake_state, DIGEST_LEN, crypt_path_t::extend_info, crypt_path_t::fast_handshake_state, get_options(), has_completed_circuit, circuit_t::highres_created, extend_info_t::identity_digest, or_connection_t::identity_digest, inform_testing_reachability(), LD_BUG, LD_CIRC, LD_GENERAL, circuit_t::n_conn, routerinfo_t::nickname, cpath_build_state_t::onehop_tunnel, extend_info_t::onion_key, onion_next_hop_in_cpath(), onion_skin_create(), extend_info_t::port, crypt_path_t::prev, relay_send_command_from_edge(), router_get_by_digest(), server_mode(), set_uint16(), set_uint32(), should_use_create_fast_for_circuit(), circuit_t::state, crypt_path_t::state, TO_CIRCUIT, tor_addr_family(), tor_addr_to_ipv4n(), tor_assert, tor_gettimeofday(), and tv_mdiff().

Referenced by circuit_extend_to_new_exit(), circuit_handle_first_hop(), circuit_n_conn_done(), command_process_created_cell(), connection_edge_process_relay_cell(), and handle_control_extendcircuit().

4.6.3.62 int circuit_truncated (origin_circuit_t * *circ*, crypt_path_t * *layer*)

We received a relay truncated cell on circ.

Since we don't ask for truncates currently, getting a truncated means that a connection broke or an extend failed. For now, just give up: for circ to close, and return 0.

References circuit_free_cpath_node(), origin_circuit_t::cpath, END_CIRC_REASON_FLAG_REMOTE, LD_APP, LD_CIRC, crypt_path_t::next, edge_connection_t::next_stream, origin_circuit_t::p_streams, TO_CIRCUIT, and tor_assert.

Referenced by connection_edge_process_relay_cell().

4.6.3.63 void circuitbuild_running_unit_tests (void)

Make a note that we're running unit tests (rather than running Tor itself), so we avoid clobbering our state file.

References unit_tests.

4.6.3.64 void clear_bridge_list (void)

Initialize the bridge list to empty, creating it if needed.

References smartlist_clear(), smartlist_create(), and tor_free.

Referenced by entry_guards_free_all(), and options_act().

4.6.3.65 static void control_event_guard_deferred (void) `[static]`

Called when one or more guards that we would previously have used for some purpose are no longer in use because a higher-priority guard has become usable again.

References control_event_guard(), entry_is_live(), get_options(), and or_options_t::NumEntryGuards.

Referenced by add_an_entry_guard().

4.6.3.66 static int count_acceptable_routers (smartlist_t * *routers*) `[static]`

Return the number of routers in **routers** that are currently up and available for building circuits through.

References routerinfo_t::is_running, and routerinfo_t::is_valid.

Referenced by new_route_len().

4.6.3.67 static void entry_guard_free (entry_guard_t * *e*) `[static]`

Release all storage held by **e**.

References entry_guard_t::chosen_by_version, and tor_free.

Referenced by entry_guards_free_all(), entry_guards_parse_state(), remove_dead_entry_guards(), and remove_obsolete_entry_guards().

4.6.3.68 int entry_guard_register_connect_status (const char * *digest*, int *succeeded*, int *mark_relay_status*, time_t *now*)

Called when a connection to an OR with the identity digest **digest** is established (**succeeded**==1) or has failed (**succeeded**==0). If the OR is an entry, change that entry's up/down status. Return 0 normally, or -1 if we want to tear down the new connection.

If **mark_relay_status**, also call **router_set_status()** (p. 1004) on this relay.

XXX022 change succeeded and mark_relay_status into 'int flags'.

References DIGEST_LEN, and HEX_DIGEST_LEN.

Referenced by circuit_build_failed(), connection_about_to_close_connection(), connection_or_check_valid_tls_handshake(), connection_or_connect(), and connection_or_set_state_open().

4.6.3.69 static int entry_guard_set_status (entry_guard_t * *e*, routerinfo_t * *ri*, time_t *now*, or_options_t * *options*, const char ** *reason*) `[static]`

Check whether the entry guard **e** is usable, given the directory authorities' opinion about the router (stored in **ri**) and the user's configuration (in **options**). Set **e**->bad_since accordingly. Return true iff the entry guard's status changes.

If it's not usable, set ***reason** to a static string explaining why.

References entry_guard_t::bad_since, base16_encode(), control_event_guard(), DIGEST_LEN, or_options_t::EntryNodes, or_options_t::ExcludeNodes, HEX_DIGEST_LEN, entry_guard_t::identity, routerinfo_t::is_possible_guard, routerinfo_t::is_running, LD_CIRC, entry_guard_t::nickname, routerinfo_t::purpose, ROUTER_PURPOSE_BRIDGE, routerset_contains_router(), tor_assert, and or_options_t::UseBridges.

Referenced by entry_guards_compute_status().

4.6.3.70 static void entry_guards_changed (void) `[static]`

Our list of entry guards has changed, or some element of one of our entry guards has changed. Write the changes to disk within the next few minutes.

References or_options_t::AvoidDiskWrites, entry_guards_dirty, get_options(), get_or_state(), and or_state_mark_dirty().

Referenced by choose_random_entry(), entry_guards_compute_status(), and pick_entry_guards().

4.6.3.71 void entry_guards_compute_status (void)

A new directory or router-status has arrived; update the down/listed status of the entry guards.

An entry is 'down' if the directory lists it as nonrunning. An entry is 'unlisted' if the directory doesn't include it.

Don't call this on startup; only on a fresh download. Otherwise we'll think that things are unlisted.

References digestmap_free(), digestmap_get(), digestmap_new(), digestmap_set(), entry_guard_set_status(), entry_guards_changed(), entry_is_live(), entry_nodes_should_be_added(), or_options_t::EntryNodes, get_options(), LD_CIRC, log_entry_guards(), num_live_entry_guards(), remove_dead_entry_guards(), router_get_by_digest(), and tor_assert.

Referenced by directory_info_has_arrived().

4.6.3.72 void entry_guards_free_all (void)

Release all storage held by the list of entry guards and related memory structs.

References bridge_list, clear_bridge_list(), entry_guard_free(), and smartlist_free().

Referenced by tor_free_all().

4.6.3.73 int entry_guards_parse_state (or_state_t * *state*, int *set*, char ** *msg*)

Parse **state** and learn about the entry guards it describes. If **set** is true, and there are no errors, replace the global entry_list with what we find. On success, return 0. On failure, alloc into ***msg** a string describing the error, and return -1.

References _tor_free(), entry_guard_t::bad_since, base16_decode(), crypto_rand_int(), DIGEST_LEN, digestmap_free(), digestmap_get(), digestmap_new(), digestmap_set(), entry_guard_free(), entry_guards_dirty, or_state_t::EntryGuards, escaped(), HEX_DIGEST_LEN, entry_guard_t::identity, is_legal_nickname(), config_line_t::key, entry_guard_t::last_attempted, LD_BUG, entry_guard_t::made_contact, MAX_NICKNAME_LEN, config_line_t::next, entry_guard_t::nickname, parse_iso_time(), remove_obsolete_entry_guards(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, or_state_t::TorVersion, entry_guard_t::unreachable_since, and config_line_t::value.

Referenced by or_state_set(), and or_state_validate().

4.6.3.74 static void entry_guards_prepend_from_config (void) `[static]`

Add all nodes in EntryNodes that aren't currently guard nodes to the list of guard nodes, at the front.

References or_options_t::EntryNodes, get_options(), is_an_entry_guard(), LD_CIRC, routerset_get_all_routers(), routerset_to_string(), smartlist_add(), smartlist_create(), smartlist_digest_isin(), tor_assert, and tor_free.

Referenced by choose_random_entry().

4.6.3.75 void entry_guards_update_state (or_state_t ∗ *state*)

If the entry guard info has not changed, do nothing and return. Otherwise, free the EntryGuards piece of **state** and create a new one out of the global entry_guards list, and then mark **state** dirty so it will get saved to disk.

References or_options_t::AvoidDiskWrites, base16_encode(), config_free_lines(), DIGEST_LEN, entry_guards_dirty, or_state_t::EntryGuards, format_iso_time(), get_options(), get_or_state(), HEX_DIGEST_LEN, config_line_t::key, MAX_NICKNAME_LEN, config_line_t::next, or_state_mark_dirty(), smartlist_create(), tor_snprintf(), and config_line_t::value.

Referenced by or_state_save().

4.6.3.76 static INLINE routerinfo_t∗ entry_is_live (entry_guard_t ∗ *e*, int *need_uptime*, int *need_capacity*, int *assume_reachable*, const char ∗∗ *msg*) `[static]`

Return the router corresponding to **e**, if **e** is working well enough that we are willing to use it as an entry right now. (Else return NULL.) In particular, it must be

- Listed as either up or never yet contacted;
- Present in the routerlist;
- Listed as 'stable' or 'fast' by the current dirserver consensus, if demanded by **need_uptime** or **need_capacity** (unless it's a configured EntryNode);
- Allowed by our current ReachableORAddresses config option; and
- Currently thought to be reachable by us (unless **assume_reachable** is true).

If the answer is no, set ∗**msg** to an explanation of why.

References entry_guard_t::bad_since, entry_guard_t::can_retry, entry_is_time_to_retry(), or_options_t::EntryNodes, fascist_firewall_allows_or(), get_options(), entry_guard_t::identity, routerinfo_t::purpose, router_get_by_digest(), router_is_unreliable(), ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_GENERAL, routerset_contains_router(), tor_assert, and entry_guard_t::unreachable_since.

Referenced by choose_random_entry(), control_event_guard_deferred(), entry_guards_compute_status(), log_entry_guards(), and num_live_entry_guards().

4.6.3.77 static int entry_is_time_to_retry (entry_guard_t ∗ *e*, time_t *now*) `[static]`

Return true iff enough time has passed since we last tried to connect to the unreachable guard **e** that we're willing to try again.

References entry_guard_t::last_attempted, and entry_guard_t::unreachable_since.

Referenced by entry_is_live().

4.6.3.78 int entry_list_is_constrained (or_options_t ∗ *options*)

Return 0 if we're fine adding arbitrary routers out of the directory to our entry guard list, or return 1 if we have a list already and we'd prefer to stick to it.

References or_options_t::EntryNodes, and or_options_t::UseBridges.

Referenced by choose_random_entry(), and connection_dir_request_failed().

4.6.3.79 void entry_nodes_should_be_added (void)

Called when the value of EntryNodes changes in our configuration.

References LD_CIRC.

Referenced by entry_guards_compute_status(), and options_act().

4.6.3.80 extend_info_t∗ extend_info_alloc (const char ∗ *nickname*, const char ∗ *digest*, crypto_pk_env_t ∗ *onion_key*, const tor_addr_t ∗ *addr*, uint16_t *port*)

Allocate a new extend_info object based on the various arguments.

References extend_info_t::addr, crypto_pk_dup_key(), DIGEST_LEN, extend_info_t::identity_digest, extend_info_t::nickname, extend_info_t::onion_key, extend_info_t::port, and tor_addr_copy().

Referenced by circuit_extend(), circuit_get_open_circ_or_launch(), and extend_info_from_router().

4.6.3.81 extend_info_t∗ extend_info_dup (extend_info_t ∗ *info*)

Allocate and return a new **extend_info_t** (p. 97) with the same contents as **info**.

References crypto_pk_dup_key(), extend_info_t::onion_key, and tor_assert.

Referenced by circuit_append_new_exit(), circuit_handle_first_hop(), onion_append_hop(), onion_extend_cpath(), onion_pick_cpath_exit(), rend_client_get_random_intro(), rend_service_launch_establish_intro(), and rend_service_update_descriptor().

4.6.3.82 void extend_info_free (extend_info_t ∗ *info*)

Release storage held by an **extend_info_t** (p. 97) struct.

References crypto_free_pk_env(), extend_info_t::onion_key, and tor_free.

Referenced by circuit_append_new_exit(), circuit_free(), circuit_free_cpath_node(), circuit_get_open_circ_or_launch(), circuit_launch_by_router(), circuit_n_conn_done(), handle_control_extendcircuit(), onion_extend_cpath(), rend_client_get_random_intro(), rend_client_introduction_acked(), rend_intro_point_free(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.6.3.83 extend_info_t∗ extend_info_from_router (routerinfo_t ∗ *r*)

Allocate and return a new **extend_info_t** (p. 97) that can be used to build a circuit to or through the router **r**.

References routerinfo_t::addr, routerinfo_t::cache_info, extend_info_alloc(), signed_descriptor_t::identity_digest, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::or_port, tor_addr_from_ipv4h, and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), circuit_launch_by_router(), handle_control_extendcircuit(), onion_extend_cpath(), onion_pick_cpath_exit(), rend_client_get_random_intro(), and rend_service_introduce().

4.6.3.84 void fetch_bridge_descriptors (time_t *now*)

For each bridge in our list for which we don't currently have a descriptor, fetch a new copy of its descriptor – either directly from the bridge or via a bridge authority.

References base16_encode(), BRIDGE_AUTHORITY, DIGEST_LEN, DIR_PURPOSE_FETCH_SERVERDESC, directory_get_from_dirserver(), download_status_failed, download_status_is_ready(), fascist_firewall_allows_address_or(), fmt_addr(), get_n_authorities(), get_options(), HEX_DIGEST_LEN, IMPOSSIBLE_TO_DOWNLOAD, launch_direct_bridge_descriptor_fetch(), LD_DIR, ROUTER_PURPOSE_BRIDGE, tor_digest_is_zero(), and or_options_t::UpdateBridgesFromAuthority.

Referenced by run_scheduled_events().

4.6.3.85 static bridge_info_t∗ find_bridge_by_digest (const char ∗ *digest*) `[static]`

If **digest** is one of our known bridges, return it.

References DIGEST_LEN.

Referenced by retry_bridge_descriptor_fetch_directly().

4.6.3.86 static circid_t get_unique_circ_id_by_conn (or_connection_t ∗ *conn*) `[static]`

Iterate over values of circ_id, starting from conn->next_circ_id, and with the high bit specified by conn->circ_id_type, until we get a circ_id that is not in use by any other circuit on that conn.

Return it, or 0 if can't get a unique circ_id.

References or_connection_t::circ_id_type, CIRC_ID_TYPE_HIGHER, CIRC_ID_TYPE_NEITHER, circuit_id_in_use_on_orconn(), LD_BUG, LD_CIRC, or_connection_t::next_circ_id, and tor_assert.

Referenced by circuit_deliver_create_cell().

4.6.3.87 int getinfo_helper_entry_guards (control_connection_t ∗ *conn*, const char ∗ *question*, char ∗∗ *answer*)

If **question** is the string "entry-guards", then dump to ∗**answer** a newly allocated string describing all of the nodes in the global entry_guards list. See control-spec.txt for details. For backward compatibility, we also handle the string "helper-nodes".

References base16_encode(), DIGEST_LEN, format_iso_time(), MAX_VERBOSE_NICKNAME_LEN, router_get_by_digest(), router_get_verbose_nickname(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

4.6.3.88 int inform_testing_reachability (void)

We've decided to start our reachability testing. If all is set, log this to the user. Return 1 if we did, or 0 if we chose not to log anything.

References routerinfo_t::address, control_event_server_status(), routerinfo_t::dir_port, LD_OR, routerinfo_t::or_port, router_get_my_routerinfo(), TIMEOUT_UNTIL_UNREACHABILITY_COMPLAINT, and tor_snprintf().

Referenced by circuit_send_next_onion_skin(), and options_act().

4.6.3.89 static INLINE entry_guard_t∗ is_an_entry_guard (const char ∗ *digest*) `[static]`

If **digest** matches the identity of any node in the entry_guards list, return that node. Else return NULL.

References DIGEST_LEN.

Referenced by add_an_entry_guard(), and entry_guards_prepend_from_config().

4.6.3.90 static void launch_direct_bridge_descriptor_fetch (bridge_info_t ∗ *bridge*) `[static]`

We need to ask **bridge** for its server descriptor. **address** is a helpful string describing this bridge.

References bridge_info_t::addr, CONN_TYPE_DIR, connection_get_by_type_addr_port_purpose(), DIR_PURPOSE_FETCH_SERVERDESC, directory_initiate_command(), bridge_info_t::identity, bridge_info_t::port, ROUTER_PURPOSE_BRIDGE, tor_dup_addr(), and tor_free.

Referenced by fetch_bridge_descriptors(), and retry_bridge_descriptor_fetch_directly().

4.6.3.91 void learned_bridge_descriptor (routerinfo_t ∗ *ri*, int *from_cache*)

We just learned a descriptor for a bridge. See if that digest is in our entry guard list, and add it if not.

References add_an_entry_guard(), any_bridge_descriptors_known(), download_status_reset(), bridge_info_t::fetch_status, get_options(), routerinfo_t::is_running, LD_DIR, routerinfo_t::nickname, routerinfo_t::purpose, ROUTER_PURPOSE_BRIDGE, routerinfo_get_configured_bridge(), routerlist_retry_directory_downloads(), and tor_assert.

Referenced by routerlist_descriptors_added().

4.6.3.92 static void log_entry_guards (int *severity*) `[static]`

Dump a description of our list of entry guards to the log at level **severity**.

References entry_is_live(), LD_CIRC, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_asprintf(), and tor_free.

Referenced by add_an_entry_guard(), entry_guards_compute_status(), remove_dead_entry_guards(), and remove_obsolete_entry_guards().

4.6.3.93 static int new_route_len (uint8_t *purpose*, extend_info_t * *exit*, smartlist_t * *routers*) [static]

Choose a length for a circuit of purpose **purpose**. Default length is 3 + the number of endpoints that would give something away. If the routerlist **routers** doesn't have enough routers to handle the desired path length, return as large a path length as is feasible, except if it's less than 2, in which case return -1.

References CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, count_acceptable_routers(), LD_CIRC, and tor_assert.

Referenced by onion_pick_cpath_exit().

4.6.3.94 static int num_live_entry_guards (void) [static]

Return the number of entry guards that we think are usable.

References entry_is_live().

Referenced by entry_guards_compute_status(), and pick_entry_guards().

4.6.3.95 static int onion_append_hop (crypt_path_t ** *head_ptr*, extend_info_t * *choice*) [static]

Create a new hop, annotate it with information about its corresponding router **choice**, and append it to the end of the cpath **head_ptr**.

References circuit_initial_package_window(), CIRCWINDOW_START, crypt_path_t::deliver_window, crypt_path_t::extend_info, extend_info_dup(), crypt_path_t::magic, onion_append_to_cpath(), crypt_path_t::package_window, and crypt_path_t::state.

Referenced by circuit_append_new_exit(), and onion_extend_cpath().

4.6.3.96 void onion_append_to_cpath (crypt_path_t ** *head_ptr*, crypt_path_t * *new_hop*)

Add **new_hop** to the end of the doubly-linked-list **head_ptr**. This function is used to extend cpath by another hop.

References crypt_path_t::next, and crypt_path_t::prev.

Referenced by onion_append_hop(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

4.6.3.97 static int onion_extend_cpath (origin_circuit_t * *circ*) [static]

Choose a suitable next hop in the cpath **head_ptr**, based on **state**. Append the hop info to head_ptr.

References origin_circuit_t::_base, origin_circuit_t::build_state, build_state_get_exit_nickname(), choose_good_entry_server(), choose_good_middle_server(), cpath_build_state_t::chosen_exit, circuit_get_cpath_len(), origin_circuit_t::cpath, cpath_build_state_t::desired_path_len, extend_info_dup(), extend_info_free(), extend_info_from_router(), LD_CIRC, extend_info_t::nickname, onion_append_hop(), and circuit_t::purpose.

Referenced by onion_populate_cpath().

4.6.3.98 static crypt_path_t * onion_next_hop_in_cpath (crypt_path_t * *cpath*) [static]

Return the first non-open hop in cpath, or return NULL if all hops are open.

References crypt_path_t::next, and crypt_path_t::state.

Referenced by circuit_finish_handshake(), circuit_handle_first_hop(), and circuit_send_next_onion_skin().

4.6.3.99 static int onion_pick_cpath_exit (origin_circuit_t * *circ*, extend_info_t * *exit*) [static]

Decide a suitable length for circ's cpath, and pick an exit router (or use **exit** if provided). Store these in the cpath. Return 0 if ok, -1 if circuit should be closed.

References origin_circuit_t::_base, origin_circuit_t::build_state, choose_good_exit_server(), cpath_build_state_t::chosen_exit, cpath_build_state_t::desired_path_len, extend_info_dup(), extend_info_from_router(), cpath_build_state_t::is_internal, LD_CIRC, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, new_route_len(), extend_info_t::nickname, cpath_build_state_t::onehop_tunnel, circuit_t::purpose, router_get_routerlist(), routerlist_t::routers, and warn_if_last_router_excluded().

Referenced by circuit_establish_circuit().

4.6.3.100 static int onion_populate_cpath (origin_circuit_t * *circ*) [static]

Pick all the entries in our cpath. Stop and return 0 when we're happy, or return -1 if an error occurs.

References LD_CIRC, and onion_extend_cpath().

Referenced by circuit_establish_circuit().

4.6.3.101 int onionskin_answer (or_circuit_t * *circ*, uint8_t *cell_type*, const char * *payload*, const char * *keys*)

Given a response payload and keys, initialize, then send a created cell back.

References or_connection_t::_base, connection_t::addr, append_cell_to_circuit_queue(), crypt_path_t::b_crypto, crypt_path_t::b_digest, CELL_DIRECTION_IN, cell_t::circ_id, circuit_init_cpath_crypto(), circuit_set_state(), CIRCUIT_STATE_OPEN, cell_t::command, connection_or_nonopen_was_started_here(), DIGEST_LEN, crypt_path_t::f_crypto, crypt_path_t::f_digest, or_circuit_t::handshake_digest, or_circuit_t::is_first_hop, is_local_addr(), LD_BUG, LD_CIRC, crypt_path_t::magic, or_circuit_t::n_crypto, or_circuit_t::n_digest, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::p_crypto, or_circuit_t::p_digest, cell_t::payload, router_orport_found_reachable(), TO_CIRCUIT, and tor_free.

Referenced by command_process_create_cell(), and connection_cpu_process_inbuf().

4.6.3.102 origin_circuit_t* origin_circuit_init (uint8_t *purpose*, int *flags*)

Create and return a new origin circuit. Initialize its purpose and build-state based on our arguments. The **flags** argument is a bitfield of CIRCLAUNCH_* flags.

References origin_circuit_t::_base, origin_circuit_t::build_state, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, CIRCLAUNCH_ONEHOP_TUNNEL, circuit_set_state(), CIRCUIT_STATE_OR_WAIT, cpath_build_state_t::is_internal, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, cpath_build_state_t::onehop_tunnel, origin_circuit_new(), circuit_t::purpose, and TO_CIRCUIT.

Referenced by circuit_establish_circuit(), and handle_control_extendcircuit().

4.6.3.103 static void pick_entry_guards (void) `[static]`

If the use of entry guards is configured, choose more entry guards until we have enough in the list.

References add_an_entry_guard(), entry_guards_changed(), get_options(), num_live_entry_guards(), or_options_t::NumEntryGuards, and tor_assert.

Referenced by choose_random_entry().

4.6.3.104 static int remove_dead_entry_guards (void) `[static]`

Remove all entry guards that have been down or unlisted for so long that we don't think they'll come up again. Return 1 if we removed any, or 0 if we did nothing.

References entry_guard_t::bad_since, base16_encode(), control_event_guard(), DIGEST_LEN, entry_guard_free(), ENTRY_GUARD_REMOVE_AFTER, format_local_iso_time(), HEX_DIGEST_LEN, entry_guard_t::identity, LD_CIRC, log_entry_guards(), entry_guard_t::nickname, and smartlist_del_keeporder().

Referenced by entry_guards_compute_status().

4.6.3.105 static int remove_obsolete_entry_guards (void) `[static]`

Remove any entry guard which was selected by an unknown version of Tor, or which was selected by a version of Tor that's known to select entry guards badly.

References base16_encode(), entry_guard_t::chosen_by_version, entry_guard_t::chosen_on_date, control_event_guard(), DIGEST_LEN, entry_guard_free(), escaped(), HEX_DIGEST_LEN, entry_guard_t::identity, LD_CIRC, log_entry_guards(), entry_guard_t::nickname, smartlist_del_keeporder(), tor_assert, tor_free, tor_snprintf(), tor_version_as_new_as(), and tor_version_parse().

Referenced by entry_guards_parse_state().

4.6.3.106 void retry_bridge_descriptor_fetch_directly (const char * *digest*)

Fetching the bridge descriptor from the bridge authority returned a "not found". Fall back to trying a direct fetch.

References find_bridge_by_digest(), and launch_direct_bridge_descriptor_fetch().

Referenced by dir_routerdesc_download_failed().

4.6.3.107 static int router_handles_some_port (routerinfo_t * *router*, smartlist_t * *needed_ports*) [static]

Return 1 if **router** can handle one or more of the ports in **needed_ports**, else return 0.

References ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, compare_addr_to_addr_policy(), routerinfo_t::exit_policy, and tor_assert.

Referenced by choose_good_exit_server_general().

4.6.3.108 static bridge_info_t* routerinfo_get_configured_bridge (routerinfo_t * *ri*) [static]

Return a bridge pointer if **ri** is one of our known bridges (either by comparing keys if possible, else by comparing addr/port). Else return NULL.

References routerinfo_t::addr, routerinfo_t::cache_info, DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::or_port, tor_addr_eq_ipv4h(), and tor_digest_is_zero().

Referenced by learned_bridge_descriptor(), and routerinfo_is_a_configured_bridge().

4.6.3.109 int routerinfo_is_a_configured_bridge (routerinfo_t * *ri*)

Return 1 if **ri** is one of our known bridges, else 0.

References routerinfo_get_configured_bridge().

Referenced by router_add_to_routerlist().

4.6.3.110 static INLINE int should_use_create_fast_for_circuit (origin_circuit_t * *circ*) [static]

Return true iff we should send a create_fast cell to start building a given circuit

References origin_circuit_t::cpath, crypt_path_t::extend_info, or_options_t::FastFirstHopPK, get_options(), extend_info_t::onion_key, server_mode(), and tor_assert.

Referenced by circuit_send_next_onion_skin().

4.6.3.111 static void warn_if_last_router_excluded (origin_circuit_t * *circ*, const extend_info_t * *exit*) [static]

Log a warning if the user specified an exit for the circuit that has been excluded from use by ExcludeNodes or ExcludeExitNodes.

References origin_circuit_t::_base, or_options_t::_ExcludeExitNodesUnion, origin_circuit_t::build_state, circuit_log_path(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCE_ACKED, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_PURPOSE_CONTROLLER, CIRCUIT_PURPOSE_INTRO_POINT, CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_ESTABLISHED, CIRCUIT_PURPOSE_REND_POINT_WAITING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_REND_JOINED, CIRCUIT_PURPOSE_TESTING, or_options_t::ExcludeNodes, get_options(), cpath_build_state_t::is_internal, LD_BUG, LD_CIRC, extend_info_t::nickname, cpath_build_state_t::onehop_tunnel, circuit_t::purpose, and routerset_contains_extendinfo().

Referenced by circuit_extend_to_new_exit(), and onion_pick_cpath_exit().

4.6.4 Variable Documentation

4.6.4.1 smartlist_t∗ bridge_list = NULL `[static]`

A list of configured bridges. Whenever we actually get a descriptor for one, we add it as an entry guard.

Referenced by entry_guards_free_all().

4.6.4.2 circuit_build_times_t circ_times

Global list of circuit build times

Referenced by circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_predict_and_launch_new(), connection_or_process_cells_from_inbuf(), connection_or_set_state_open(), connection_tls_finish_handshake(), networkstatus_set_current_consensus(), or_state_save(), or_state_set(), and origin_circuit_new().

4.6.4.3 smartlist_t∗ entry_guards = NULL `[static]`

A list of our chosen entry guards.

4.6.4.4 int entry_guards_dirty = 0 `[static]`

A value of 1 means that the entry_guards list has changed and those changes need to be flushed to disk.

Referenced by entry_guards_changed(), entry_guards_parse_state(), and entry_guards_update_state().

4.6.4.5 circuit_t∗ global_circuitlist

A global list of all circuits at this hop.

4.6.4.6 int should_add_entry_nodes = 0 `[static]`

When we try to choose an entry guard, should we parse and add config's EntryNodes first?

4.6.4.7 int unit_tests = 0 `[static]`

If set, we're running the unit tests: we should avoid clobbering our state file or accessing **get_options()** (p. 389) or **get_or_state()** (p. 390)

Referenced by circuit_build_times_add_time(), circuit_build_times_get_initial_timeout(), circuit_build_times_update_state(), and circuitbuild_running_unit_tests().

4.7 circuitlist.c File Reference

Manage the global circuit list.

`#include "or.h"`

`#include "ht.h"`

Data Structures

- struct **orconn_circid_circuit_map_t**

Functions

- static void **circuit_free** (**circuit_t** *circ)
- static void **circuit_free_cpath** (**crypt_path_t** *cpath)
- static void **circuit_free_cpath_node** (**crypt_path_t** *victim)
- static INLINE int **_orconn_circid_entries_eq** (**orconn_circid_circuit_map_t** *a, **orconn_circid_circuit_map_t** *b)
- static INLINE unsigned int **_orconn_circid_entry_hash** (**orconn_circid_circuit_map_t** *a)
- static **HT_HEAD** (HT_PROTOTYPE(orconn_circid_map, HT_PROTOTYPE(**orconn_circid_circuit_map_t**)
- void **circuit_set_p_circid_orconn** (**or_circuit_t** *circ, **circid_t** id, **or_connection_t** *conn)
- void **circuit_set_n_circid_orconn** (**circuit_t** *circ, **circid_t** id, **or_connection_t** *conn)
- void **circuit_set_state** (**circuit_t** *circ, uint8_t state)
- static void **circuit_add** (**circuit_t** *circ)
- void **circuit_get_all_pending_on_or_conn** (**smartlist_t** *out, **or_connection_t** *or_conn)
- int **circuit_count_pending_on_or_conn** (**or_connection_t** *or_conn)
- void **circuit_close_all_marked** (void)
- **circuit_t** * **_circuit_get_global_list** (void)
- const char * **circuit_state_to_string** (int state)
- const char * **circuit_purpose_to_controller_string** (uint8_t purpose)
- int32_t **circuit_initial_package_window** (void)
- static void **init_circuit_base** (**circuit_t** *circ)
- **origin_circuit_t** * **origin_circuit_new** (void)
- **or_circuit_t** * **or_circuit_new** (**circid_t** p_circ_id, **or_connection_t** *p_conn)
- void **circuit_free_all** (void)
- static void **circuit_dump_details** (int severity, **circuit_t** *circ, int conn_array_index, const char *type, int this_circid, int other_circid)
- void **circuit_dump_by_conn** (**connection_t** *conn, int severity)
- **origin_circuit_t** * **circuit_get_by_global_id** (uint32_t id)
- static INLINE **circuit_t** * **circuit_get_by_circid_orconn_impl** (**circid_t** circ_id, **or_connection_t** *conn)
- **circuit_t** * **circuit_get_by_circid_orconn** (**circid_t** circ_id, **or_connection_t** *conn)
- int **circuit_id_in_use_on_orconn** (**circid_t** circ_id, **or_connection_t** *conn)
- **circuit_t** * **circuit_get_by_edge_conn** (**edge_connection_t** *conn)

- void **circuit_unlink_all_from_or_conn** (**or_connection_t** ∗conn, int reason)
- **origin_circuit_t** ∗ **circuit_get_by_rend_query_and_purpose** (const char ∗rend_query, uint8_t purpose)
- **origin_circuit_t** ∗ **circuit_get_next_by_pk_and_purpose** (**origin_circuit_t** ∗start, const char ∗digest, uint8_t purpose)
- static **or_circuit_t** ∗ **circuit_get_by_rend_token_and_purpose** (uint8_t purpose, const char ∗token, size_t len)
- **or_circuit_t** ∗ **circuit_get_rendezvous** (const char ∗cookie)
- **or_circuit_t** ∗ **circuit_get_intro_point** (const char ∗digest)
- **origin_circuit_t** ∗ **circuit_find_to_cannibalize** (uint8_t purpose, **extend_info_t** ∗info, int flags)
- int **circuit_get_cpath_len** (**origin_circuit_t** ∗circ)
- **crypt_path_t** ∗ **circuit_get_cpath_hop** (**origin_circuit_t** ∗circ, int hopnum)
- void **circuit_mark_all_unused_circs** (void)
- void **circuit_expire_all_dirty_circs** (void)
- void **_circuit_mark_for_close** (**circuit_t** ∗circ, int reason, int line, const char ∗file)
- void **assert_cpath_layer_ok** (const **crypt_path_t** ∗cp)
- static void **assert_cpath_ok** (const **crypt_path_t** ∗cp)
- void **assert_circuit_ok** (const **circuit_t** ∗c)

Variables

- **circuit_t** ∗ **global_circuitlist** = NULL
- static **smartlist_t** ∗ **circuits_pending_or_conns** = NULL

4.7.1 Detailed Description

Manage the global circuit list.

4.7.2 Function Documentation

4.7.2.1 circuit_t∗ _circuit_get_global_list (void)

Return the head of the global linked list of circuits.

Referenced by count_established_intro_points(), dump_cell_pool_usage(), getinfo_helper_-events(), rend_config_services(), and rep_hist_buffer_stats_write().

4.7.2.2 void _circuit_mark_for_close (circuit_t ∗ *circ*, int *reason*, int *line*, const char ∗ *file*)

Mark **circ** to be closed next time we call **circuit_close_all_marked()** (p. 331). Do any cleanup needed:

- If state is onionskin_pending, remove circ from the onion_pending list.
- If circ isn't open yet: call **circuit_build_failed()** (p. 342) if we're the origin, and in either case call **circuit_rep_hist_note_result()** (p. 315) to note stats.
- If purpose is C_INTRODUCE_ACK_WAIT, remove the intro point we just tried from our list of intro points for that service descriptor.

- Send appropriate destroys and edge_destroys for conns and streams attached to circ.
- If circ->rend_splice is set (we are the midpoint of a joined rendezvous stream), then mark the other circuit to close as well.

References or_circuit_t::_base, edge_connection_t::_base, assert_circuit_ok(), origin_circuit_t::build_state, build_state_get_exit_nickname(), cpath_build_state_t::chosen_exit, circuit_build_failed(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, circuit_rep_hist_note_result(), CIRCUIT_STATE_ONIONSKIN_PENDING, CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, connection_edge_destroy(), connection_or_send_destroy(), control_event_circuit_status(), edge_connection_t::edge_has_sent_end, END_CIRC_AT_ORIGIN, END_CIRC_REASON_FLAG_REMOTE, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED, LD_BUG, LD_REND, connection_t::marked_for_close, circuit_t::marked_for_close, circuit_t::marked_for_close_file, circuit_t::n_circ_id, circuit_t::n_conn, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, rend_data_t::onion_address, onion_pending_remove(), or_circuit_t::p_circ_id, or_circuit_t::p_conn, origin_circuit_t::p_streams, circuit_t::purpose, rend_client_remove_intro_point(), origin_circuit_t::rend_data, or_circuit_t::rend_splice, or_circuit_t::resolving_streams, safe_str_client(), smartlist_remove(), circuit_t::state, TO_CIRCUIT, TO_CONN, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_assert.

4.7.2.3 static INLINE int _orconn_circid_entries_eq (orconn_circid_circuit_map_t * *a*, orconn_circid_circuit_map_t * *b*) [static]

Helper for hash tables: compare the OR connection and circuit ID for a and b, and return less than, equal to, or greater than zero appropriately.

4.7.2.4 static INLINE unsigned int _orconn_circid_entry_hash (orconn_circid_circuit_map_t * *a*) [static]

Helper: return a hash based on circuit ID and the pointer value of or_conn in **a**.

4.7.2.5 void assert_circuit_ok (const circuit_t * *c*)

Verify that circuit **c** has all of its invariants correct. Trigger an assert if anything is invalid.

References edge_connection_t::_base, assert_cpath_ok(), circuit_get_by_circid_orconn_impl(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_REND_ESTABLISHED, CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, CONN_TYPE_EXIT, origin_circuit_t::cpath, circuit_t::deliver_window, circuit_t::magic, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, circuit_t::n_conn_onionskin, or_circuit_t::n_crypto, or_circuit_t::n_digest, circuit_t::n_hop, or_circuit_t::n_streams, edge_connection_t::next_stream, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::p_crypto, or_circuit_t::p_digest, circuit_t::package_window, circuit_t::purpose, or_circuit_t::rend_splice, smartlist_isin(), circuit_t::state, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and connection_t::type.

Referenced by _circuit_mark_for_close(), circuit_package_relay_cell(), connection_ap_handshake_attach_circuit(), connection_exit_begin_conn(), and connection_exit_begin_resolve().

4.7.2.6 void assert_cpath_layer_ok (const crypt_path_t * *cp*)

Verify that cpath layer **cp** has all of its invariants correct. Trigger an assert if anything is invalid.

References crypt_path_t::b_crypto, crypt_path_t::deliver_window, crypt_path_t::dh_handshake_state, crypt_path_t::f_crypto, LD_BUG, crypt_path_t::magic, crypt_path_t::package_window, crypt_path_t::state, and tor_assert.

Referenced by assert_connection_ok(), and assert_cpath_ok().

4.7.2.7 static void assert_cpath_ok (const crypt_path_t * *cp*) `[static]`

Verify that cpath **cp** has all of its invariants correct. Trigger an assert if anything is invalid.

References assert_cpath_layer_ok(), crypt_path_t::next, crypt_path_t::prev, crypt_path_t::state, and tor_assert.

Referenced by assert_circuit_ok().

4.7.2.8 static void circuit_add (circuit_t * *circ*) `[static]`

Add **circ** to the global list of circuits. This is called only from within circuit_new.

References circuit_t::next.

Referenced by init_circuit_base().

4.7.2.9 void circuit_close_all_marked (void)

Detach from the global circuit list, and deallocate, all circuits that have been marked for close.

References circuit_free(), circuit_t::marked_for_close, and circuit_t::next.

Referenced by run_scheduled_events().

4.7.2.10 int circuit_count_pending_on_or_conn (or_connection_t * *or_conn*)

Return the number of circuits in state OR_WAIT, waiting for the given connection.

References circuit_get_all_pending_on_or_conn(), LD_CIRC, or_connection_t::nickname, smartlist_create(), and smartlist_free().

Referenced by control_event_or_conn_status().

4.7.2.11 void circuit_dump_by_conn (connection_t * *conn*, int *severity*)

Log, at severity **severity**, information about each circuit that is connected to **conn**.

References connection_t::addr, extend_info_t::addr, circuit_dump_details(), CIRCUIT_IS_ORIGIN, CIRCUIT_STATE_OPEN, connection_t::conn_array_index, CONN_TYPE_OR, DIGEST_LEN, extend_info_t::identity_digest, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, circuit_t::n_hop, circuit_t::next, edge_connection_t::next_stream, or_circuit_t::p_circ_id, connection_t::port, extend_info_t::port, circuit_t::state, TO_CONN, TO_OR_CIRCUIT(), TO_OR_CONN(), TO_ORIGIN_CIRCUIT(), tor_addr_eq, and connection_t::type.

Referenced by dumpstats().

4.7.2.12 static void circuit_dump_details (int *severity*, circuit_t ∗ *circ*, int *conn_array_index*, const char ∗ *type*, int *this_circid*, int *other_circid*) `[static]`

A helper function for **circuit_dump_by_conn()** (p. 331) below. Log a bunch of information about circuit **circ**.

References CIRCUIT_IS_ORIGIN, circuit_log_path(), circuit_state_to_string(), LD_CIRC, circuit_t::state, circuit_t::timestamp_created, and TO_ORIGIN_CIRCUIT().

Referenced by circuit_dump_by_conn().

4.7.2.13 void circuit_expire_all_dirty_circs (void)

Go through the circuitlist; for each circuit that starts at us and is dirty, frob its timestamp_dirty so we won't use it for any new streams.

This is useful for letting the user change pseudonyms, so new streams will not be linkable to old streams.

References CIRCUIT_IS_ORIGIN, get_options(), circuit_t::marked_for_close, or_options_t::MaxCircuitDirtiness, circuit_t::next, and circuit_t::timestamp_dirty.

Referenced by circuit_note_clock_jumped(), do_hup(), options_act(), and signewnym_impl().

4.7.2.14 origin_circuit_t∗ circuit_find_to_cannibalize (uint8_t *purpose*, extend_info_t ∗ *info*, int *flags*)

Return a circuit that is open, is CIRCUIT_PURPOSE_C_GENERAL, has a timestamp_dirty value of 0, has flags matching the CIRCLAUNCH_∗ flags in **flags**, and if info is defined, does not already use info as any of its hops; or NULL if no circuit fits this description.

The **purpose** argument (currently ignored) refers to the purpose of the circuit we want to create, not the purpose of the circuit we want to cannibalize.

If !CIRCLAUNCH_NEED_UPTIME, prefer returning non-uptime circuits.

References origin_circuit_t::build_state, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_STATE_OPEN, origin_circuit_t::cpath, DIGEST_LEN, crypt_path_t::extend_info, extend_info_t::identity_digest, cpath_build_state_t::is_internal, LD_CIRC, circuit_t::marked_for_close, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, crypt_path_t::next, circuit_t::next, circuit_t::purpose, origin_circuit_t::remaining_relay_early_cells, router_get_by_digest(), routers_in_same_family(), circuit_t::state, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by circuit_launch_by_extend_info().

4.7.2.15 static void circuit_free (circuit_t ∗ *circ*) `[static]`

Deallocate space associated with circ.

References or_circuit_t::_base, origin_circuit_t::build_state, cell_queue_clear(), cpath_build_state_t::chosen_exit, circuit_free_cpath(), circuit_free_cpath_node(), CIRCUIT_IS_ORIGIN, circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), origin_circuit_t::cpath, crypto_free_cipher_env(), crypto_free_digest_env(), crypto_free_pk_env(), extend_info_free(), get_options(), origin_circuit_t::intro_key, circuit_t::magic, circuit_t::n_conn_cells, circuit_t::n_conn_onionskin, or_circuit_t::n_crypto, or_circuit_t::n_digest, circuit_t::n_hop, or_circuit_t::p_conn_cells, or_circuit_t::p_crypto, or_circuit_t::p_digest, cpath_build_state_t::pending_final_cpath, origin_circuit_t::rend_data, rend_data_free(), or_circuit_t::rend_splice, rep_hist_buffer_stats_add_circ(), TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and tor_free.

Referenced by circuit_close_all_marked(), and circuit_free_all().

4.7.2.16 void circuit_free_all (void)

Release all storage held by circuits.

References circuit_free(), CIRCUIT_IS_ORIGIN, connection_free(), circuit_t::next, edge_connection_t::next_stream, or_circuit_t::resolving_streams, smartlist_free(), TO_CONN, and TO_OR_CIRCUIT().

Referenced by tor_free_all().

4.7.2.17 static void circuit_free_cpath (crypt_path_t * *cpath*) `[static]`

Deallocate space associated with the linked list **cpath**.

References circuit_free_cpath_node(), and crypt_path_t::next.

Referenced by circuit_free().

4.7.2.18 static void circuit_free_cpath_node (crypt_path_t * *victim*) `[static]`

Deallocate space associated with the cpath node **victim**.

References crypt_path_t::b_crypto, crypt_path_t::b_digest, crypto_dh_free(), crypto_free_cipher_env(), crypto_free_digest_env(), crypt_path_t::dh_handshake_state, crypt_path_t::extend_info, extend_info_free(), crypt_path_t::f_crypto, crypt_path_t::f_digest, and tor_free.

Referenced by circuit_free(), circuit_free_cpath(), and circuit_truncated().

4.7.2.19 void circuit_get_all_pending_on_or_conn (smartlist_t * *out*, or_connection_t * *or_conn*)

Append to **out** all circuits in state OR_WAIT waiting for the given connection.

References or_connection_t::_base, connection_t::addr, CIRCUIT_STATE_OR_WAIT, DIGEST_LEN, or_connection_t::identity_digest, connection_t::port, smartlist_add(), tor_addr_eq, tor_assert, and tor_digest_is_zero().

Referenced by circuit_count_pending_on_or_conn(), and circuit_n_conn_done().

4.7.2.20 circuit_t* circuit_get_by_circid_orconn (circid_t *circ_id*, or_connection_t * *conn*)

Return a circ such that:

- circ->n_circ_id or circ->p_circ_id is equal to **circ_id**, and

- circ is attached to **conn**, either as p_conn or n_conn.
- circ is not marked for close. Return NULL if no such circuit exists.

References circuit_get_by_circid_orconn_impl(), and circuit_t::marked_for_close.

Referenced by command_process_created_cell(), command_process_destroy_cell(), command_process_relay_cell(), and connection_cpu_process_inbuf().

4.7.2.21 static INLINE circuit_t* circuit_get_by_circid_orconn_impl (circid_t *circ_id*, or_connection_t * *conn*) [static]

Return a circ such that:

- circ->n_circ_id or circ->p_circ_id is equal to **circ_id**, and
- circ is attached to **conn**, either as p_conn or n_conn. Return NULL if no such circuit exists.

References CIRCUIT_IS_ORIGIN, LD_BUG, circuit_t::n_circ_id, circuit_t::n_conn, circuit_t::next, or_circuit_t::p_circ_id, or_circuit_t::p_conn, and TO_OR_CIRCUIT().

Referenced by assert_circuit_ok(), circuit_get_by_circid_orconn(), and circuit_id_in_use_on_orconn().

4.7.2.22 circuit_t* circuit_get_by_edge_conn (edge_connection_t * *conn*)

Return the circuit that a given edge connection is using.

References CIRCUIT_IS_ORIGIN, circuit_t::magic, edge_connection_t::on_circuit, and tor_assert.

Referenced by _connection_write_to_buf_impl(), connection_about_to_close_connection(), connection_ap_expire_beginning(), connection_edge_consider_sending_sendme(), connection_edge_end(), connection_edge_package_raw_inbuf(), connection_exit_connect(), control_event_stream_status(), dns_cancel_pending_resolve(), dns_found_answer(), getinfo_helper_events(), handle_control_attachstream(), and purge_expired_resolves().

4.7.2.23 origin_circuit_t* circuit_get_by_global_id (uint32_t *id*)

Return the circuit whose global ID is **id**, or NULL if no such circuit exists.

References CIRCUIT_IS_ORIGIN, circuit_t::marked_for_close, circuit_t::next, and TO_ORIGIN_CIRCUIT().

Referenced by get_circ().

4.7.2.24 origin_circuit_t* circuit_get_by_rend_query_and_purpose (const char * *rend_query*, uint8_t *purpose*)

Return a circ such that:

- circ->rend_data->query is equal to **rend_query**, and

- circ->purpose is equal to **purpose**.

Return NULL if no such circuit exists.

References CIRCUIT_PURPOSE_IS_ORIGIN, circuit_t::marked_for_close, circuit_t::next, rend_data_t::onion_address, circuit_t::purpose, rend_cmp_service_ids(), origin_circuit_t::rend_data, TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by rend_client_introduction_acked().

4.7.2.25 static or_circuit_t* circuit_get_by_rend_token_and_purpose (uint8_t *purpose*, const char * *token*, size_t *len*) [static]

Return the first OR circuit in the global list whose purpose is **purpose**, and whose rend_token is the **len**-byte **token**.

References circuit_t::marked_for_close, circuit_t::next, circuit_t::purpose, and TO_OR_CIRCUIT().

Referenced by circuit_get_intro_point(), and circuit_get_rendezvous().

4.7.2.26 crypt_path_t* circuit_get_cpath_hop (origin_circuit_t * *circ*, int *hopnum*)

Return the **hopnum**th hop in **circ**->cpath, or NULL if there aren't that many hops in the list.

References origin_circuit_t::cpath, and crypt_path_t::next.

Referenced by handle_control_attachstream().

4.7.2.27 int circuit_get_cpath_len (origin_circuit_t * *circ*)

Return the number of hops in circuit's path.

References origin_circuit_t::cpath, and crypt_path_t::next.

Referenced by handle_control_attachstream(), and onion_extend_cpath().

4.7.2.28 or_circuit_t* circuit_get_intro_point (const char * *digest*)

Return the circuit waiting for intro cells of the given digest. Return NULL if no such circuit is found.

References circuit_get_by_rend_token_and_purpose(), CIRCUIT_PURPOSE_INTRO_POINT, and DIGEST_LEN.

Referenced by rend_mid_establish_intro(), and rend_mid_introduce().

4.7.2.29 origin_circuit_t* circuit_get_next_by_pk_and_purpose (origin_circuit_t * *start*, const char * *digest*, uint8_t *purpose*)

Return the first circuit originating here in global_circuitlist after **start** whose purpose is **purpose**, and where **digest** (if set) matches the rend_pk_digest field. Return NULL if no circuit is found. If **start** is NULL, begin at the start of the list.

References CIRCUIT_PURPOSE_IS_ORIGIN, DIGEST_LEN, circuit_t::marked_for_close, circuit_t::next, circuit_t::purpose, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by find_intro_circuit(), and router_perform_bandwidth_test().

4.7.2.30 or_circuit_t* circuit_get_rendezvous (const char * *cookie*)

Return the circuit waiting for a rendezvous with the provided cookie. Return NULL if no such circuit is found.

References circuit_get_by_rend_token_and_purpose(), CIRCUIT_PURPOSE_REND_POINT_WAITING, and REND_COOKIE_LEN.

Referenced by rend_mid_establish_rendezvous(), and rend_mid_rendezvous().

4.7.2.31 int circuit_id_in_use_on_orconn (circid_t *circ_id*, or_connection_t * *conn*)

Return true iff the circuit ID **circ_id** is currently used by a circuit, marked or not, on **conn**.

References circuit_get_by_circid_orconn_impl().

Referenced by command_process_create_cell(), and get_unique_circ_id_by_conn().

4.7.2.32 int32_t circuit_initial_package_window (void)

Pick a reasonable package_window to start out for our circuits. Originally this was hard-coded at 1000, but now the consensus votes on the answer. See proposal 168.

References CIRCWINDOW_START, and networkstatus_get_param().

Referenced by init_circuit_base(), onion_append_hop(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

4.7.2.33 void circuit_mark_all_unused_circs (void)

Go through the circuitlist; mark-for-close each circuit that starts at us but has not yet been used.

References CIRCUIT_IS_ORIGIN, circuit_t::marked_for_close, circuit_t::next, and circuit_t::timestamp_dirty.

Referenced by circuit_note_clock_jumped(), and options_act().

4.7.2.34 const char* circuit_purpose_to_controller_string (uint8_t *purpose*)

Map a circuit purpose to a string suitable to be displayed to a controller.

References CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCE_ACKED, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_PURPOSE_CONTROLLER, CIRCUIT_PURPOSE_INTRO_POINT, CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_ESTABLISHED, CIRCUIT_PURPOSE_REND_POINT_WAITING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_-

ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, CIRCUIT_PURPOSE_S_REND_-JOINED, CIRCUIT_PURPOSE_TESTING, and tor_snprintf().

Referenced by control_event_circuit_status(), and getinfo_helper_events().

4.7.2.35 void circuit_set_n_circid_orconn (circuit_t ∗ *circ*, circid_t *id*, or_connection_t ∗ *conn*)

Set the n_conn field of a circuit **circ**, along with the corresponding circuit ID, and add the circuit as appropriate to the (orconn,id)->circuit map.

References bool_eq, CELL_DIRECTION_OUT, cell_queue_t::n, circuit_t::n_conn_cells, circuit_t::next_active_on_n_conn, and tor_assert.

Referenced by circuit_deliver_create_cell(), circuit_free(), circuit_unlink_all_from_or_-conn(), command_process_destroy_cell(), and connection_edge_process_relay_cell().

4.7.2.36 void circuit_set_p_circid_orconn (or_circuit_t ∗ *circ*, circid_t *id*, or_connection_t ∗ *conn*)

Set the p_conn field of a circuit **circ**, along with the corresponding circuit ID, and add the circuit as appropriate to the (orconn,id)->circuit map.

References bool_eq, CELL_DIRECTION_IN, cell_queue_t::n, or_circuit_t::next_active_-on_p_conn, or_circuit_t::p_conn_cells, TO_CIRCUIT, and tor_assert.

Referenced by circuit_free(), circuit_unlink_all_from_or_conn(), command_process_-destroy_cell(), and or_circuit_new().

4.7.2.37 void circuit_set_state (circuit_t ∗ *circ*, uint8_t *state*)

Change the state of **circ** to **state**, adding it to or removing it from lists as appropriate.

References CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, circuit_t::n_conn_-onionskin, smartlist_add(), smartlist_create(), smartlist_remove(), circuit_t::state, and tor_-assert.

Referenced by circuit_extend(), circuit_extend_to_new_exit(), circuit_n_conn_done(), circuit_send_next_onion_skin(), command_process_create_cell(), handle_control_-extendcircuit(), onionskin_answer(), and origin_circuit_init().

4.7.2.38 const char∗ circuit_state_to_string (int *state*)

Function to make circ->state human-readable

References CIRCUIT_STATE_BUILDING, CIRCUIT_STATE_ONIONSKIN_PENDING, CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, LD_BUG, and tor_snprintf().

Referenced by circuit_dump_details(), circuit_expire_building(), and rend_service_dump_-stats().

4.7.2.39 void circuit_unlink_all_from_or_conn (or_connection_t ∗ *conn*, int *reason*)

For each circuit that has **conn** as n_conn or p_conn, unlink the circuit from the orconn,circid map, and mark it for close if it hasn't been marked already.

References CIRCUIT_IS_ORIGIN, circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), connection_or_unlink_all_active_circs(), circuit_t::marked_for_close, circuit_t::n_conn, circuit_t::next, or_circuit_t::p_conn, and TO_OR_CIRCUIT().

Referenced by connection_about_to_close_connection().

4.7.2.40 static HT_HEAD (HT_PROTOTYPE(*orconn_circid_map*, HT_PROTOTYPE(*orconn_circid_circuit_map_t*) `[static]`

Map from [orconn,circid] to circuit.

References or_connection_t::_base, CELL_DIRECTION_OUT, connection_t::magic, make_circuit_active_on_conn(), make_circuit_inactive_on_conn(), cell_queue_t::n, or_connection_t::n_circuits, or_circuit_t::next_active_on_p_conn, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::p_conn_cells, TO_OR_CIRCUIT(), tor_assert, and tor_free.

4.7.2.41 static void init_circuit_base (circuit_t ∗ *circ*) `[static]`

Initialize the common elements in a **circuit_t** (p. 40), and add it to the global list.

References cell_ewma_t::cell_count, cell_ewma_get_tick(), circuit_add(), circuit_initial_package_window(), CIRCWINDOW_START, circuit_t::deliver_window, cell_ewma_t::heap_index, circuit_t::highres_created, cell_ewma_t::is_for_p_conn, cell_ewma_t::last_adjusted_tick, circuit_t::n_cell_ewma, circuit_t::package_window, circuit_t::timestamp_created, and tor_gettimeofday().

Referenced by or_circuit_new(), and origin_circuit_new().

4.7.2.42 or_circuit_t∗ or_circuit_new (circid_t *p_circ_id*, or_connection_t ∗ *p_conn*)

Allocate a new **or_circuit_t** (p. 135), connected to **p_conn** as **p_circ_id**. If **p_conn** is NULL, the circuit is unattached.

References or_circuit_t::_base, cell_ewma_t::cell_count, cell_ewma_get_tick(), circuit_set_p_circid_orconn(), cell_ewma_t::heap_index, init_circuit_base(), cell_ewma_t::is_for_p_conn, cell_ewma_t::last_adjusted_tick, circuit_t::magic, MAX_RELAY_EARLY_CELLS_PER_CIRCUIT, or_circuit_t::p_cell_ewma, or_circuit_t::remaining_relay_early_cells, and TO_CIRCUIT.

Referenced by command_process_create_cell().

4.7.2.43 origin_circuit_t∗ origin_circuit_new (void)

Allocate space for a new circuit, initializing with **p_circ_id** and **p_conn**. Add it to the global circuit list.

References origin_circuit_t::_base, approx_time(), circ_times, crypto_rand_int(), origin_circuit_t::global_identifier, init_circuit_base(), circuit_build_times_t::last_circ_at, circuit_t::magic, MAX_RELAY_EARLY_CELLS_PER_CIRCUIT, origin_circuit_t::next_stream_id, origin_circuit_t::remaining_relay_early_cells, and TO_CIRCUIT.

Referenced by origin_circuit_init().

4.7.3 Variable Documentation

4.7.3.1 smartlist_t* circuits_pending_or_conns = NULL `[static]`

A list of all the circuits in CIRCUIT_STATE_OR_WAIT.

4.7.3.2 circuit_t* global_circuitlist = NULL

A global list of all circuits at this hop.

4.8 circuituse.c File Reference

Launch the right sort of circuits and attach streams to them.

`#include "or.h"`

Defines

- #define **REND_PARALLEL_INTRO_DELAY** 15
- #define **MAX_UNUSED_OPEN_CIRCUITS** 14
- #define **TESTING_CIRCUIT_INTERVAL** 300
- #define **IDLE_TIMEOUT_WHILE_LEARNING** (10*60)
- #define **IDLE_ONE_HOP_CIRC_TIMEOUT** 60
- #define **NUM_PARALLEL_TESTING_CIRCS** 4
- #define **MAX_CIRCUIT_FAILURES** 5

Functions

- static void **circuit_expire_old_circuits_clientside** (time_t now)
- static void **circuit_increment_failure_count** (void)
- long int **lround** (double x)
- static int **circuit_is_acceptable** (**circuit_t** *circ, **edge_connection_t** *conn, int must_be_open, uint8_t purpose, int need_uptime, int need_internal, time_t now)
- static int **circuit_is_better** (**circuit_t** *a, **circuit_t** *b, uint8_t purpose)
- static **origin_circuit_t** * **circuit_get_best** (**edge_connection_t** *conn, int must_be_open, uint8_t purpose, int need_uptime, int need_internal)
- int **circuit_conforms_to_options** (const **origin_circuit_t** *circ, const **or_options_t** *options)
- void **circuit_expire_building** (time_t now)
- void **circuit_remove_handled_ports** (**smartlist_t** *needed_ports)
- int **circuit_stream_is_being_handled** (**edge_connection_t** *conn, uint16_t port, int min)
- static void **circuit_predict_and_launch_new** (void)
- void **circuit_build_needed_circs** (time_t now)
- void **circuit_detach_stream** (**circuit_t** *circ, **edge_connection_t** *conn)
- void **circuit_expire_old_circuits_serverside** (time_t now)
- void **reset_bandwidth_test** (void)
- int **circuit_enough_testing_circs** (void)
- static void **circuit_testing_opened** (**origin_circuit_t** *circ)
- static void **circuit_testing_failed** (**origin_circuit_t** *circ, int at_last_hop)
- void **circuit_has_opened** (**origin_circuit_t** *circ)
- void **circuit_build_failed** (**origin_circuit_t** *circ)
- **origin_circuit_t** * **circuit_launch_by_router** (uint8_t purpose, **routerinfo_t** *exit, int flags)
- **origin_circuit_t** * **circuit_launch_by_extend_info** (uint8_t purpose, **extend_info_t** *extend_info, int flags)
- void **circuit_reset_failure_count** (int timeout)
- static int **circuit_get_open_circ_or_launch** (**edge_connection_t** *conn, uint8_t desired_circuit_purpose, **origin_circuit_t** **circp)
- static int **cpath_is_on_circuit** (**origin_circuit_t** *circ, **crypt_path_t** *crypt_path)

- static void **link_apconn_to_circ** (**edge_connection_t** ∗apconn, **origin_circuit_t** ∗circ, **crypt_path_t** ∗cpath)
- static void **consider_recording_trackhost** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ)
- int **connection_ap_handshake_attach_chosen_circuit** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ, **crypt_path_t** ∗cpath)
- int **connection_ap_handshake_attach_circuit** (**edge_connection_t** ∗conn)

Variables

- **circuit_t** ∗ **global_circuitlist**
- static int **have_performed_bandwidth_test** = 0
- static int **n_circuit_failures** = 0
- static int **did_circs_fail_last_period** = 0

4.8.1 Detailed Description

Launch the right sort of circuits and attach streams to them.

4.8.2 Define Documentation

4.8.2.1 #define IDLE_ONE_HOP_CIRC_TIMEOUT 60

How long do we wait before killing circuits with the properties described below?

Probably we could choose a number here as low as 5 to 10 seconds, since these circs are used for begindir, and a) generally you either ask another begindir question right after or you don't for a long time, b) clients at least through 0.2.1.x choose from the whole set of directory mirrors at each choice, and c) re-establishing a one-hop circuit via create-fast is a light operation assuming the TLS conn is still there.

I expect "b" to go away one day when we move to using directory guards, but I think "a" and "c" are good enough reasons that a low number is safe even then.

Referenced by circuit_expire_old_circuits_serverside().

4.8.2.2 #define IDLE_TIMEOUT_WHILE_LEARNING (10∗60)

If we haven't yet decided on a good timeout value for circuit building, we close idles circuits aggressively so we can get more data points.

Referenced by circuit_expire_old_circuits_clientside().

4.8.2.3 #define MAX_CIRCUIT_FAILURES 5

Don't retry launching a new circuit if we try this many times with no success.

Referenced by circuit_launch_by_extend_info(), and circuit_reset_failure_count().

4.8.2.4 #define MAX_UNUSED_OPEN_CIRCUITS 14

Don't keep more than this many unused open circuits around.

Referenced by circuit_predict_and_launch_new().

4.8.2.5 #define NUM_PARALLEL_TESTING_CIRCS 4

Number of testing circuits we want open before testing our bandwidth.

Referenced by circuit_enough_testing_circs(), and circuit_testing_opened().

4.8.2.6 #define TESTING_CIRCUIT_INTERVAL 300

Build a new test circuit every 5 minutes

Referenced by circuit_build_needed_circs().

4.8.3 Function Documentation

4.8.3.1 void circuit_build_failed (origin_circuit_t * *circ*)

Called whenever a circuit could not be successfully built.

References or_connection_t::_base, origin_circuit_t::_base, connection_t::address, origin_circuit_t::build_state, build_state_get_exit_nickname(), circuit_discard_optional_exit_enclaves(), circuit_increment_failure_count(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, CIRCUIT_STATE_OPEN, circuit_testing_failed(), connection_ap_fail_onehop(), origin_circuit_t::cpath, entry_guard_register_connect_status(), escaped(), crypt_path_t::extend_info, extend_info_t::identity_digest, or_connection_t::is_bad_for_new_circs, LD_OR, LD_REND, circuit_t::n_conn, connection_t::port, crypt_path_t::prev, circuit_t::purpose, rend_service_relaunch_rendezvous(), circuit_t::state, and crypt_path_t::state.

Referenced by _circuit_mark_for_close().

4.8.3.2 void circuit_build_needed_circs (time_t *now*)

This function is called once a second, if router_have_min_dir_info() is true. Its job is to make sure all services we offer have enough circuits available. Some services just want enough circuits for current tasks, whereas others want a minimum set of idle circuits hanging around.

References addressmap_clean(), circuit_expire_old_circuits_clientside(), circuit_launch_by_router(), circuit_predict_and_launch_new(), CIRCUIT_PURPOSE_C_GENERAL, circuit_reset_failure_count(), connection_ap_attach_pending(), or_options_t::DisablePredictedCircuits, get_options(), or_options_t::NewCircuitPeriod, proxy_mode(), rend_services_introduce(), and TESTING_CIRCUIT_INTERVAL.

Referenced by run_scheduled_events().

4.8.3.3 int circuit_conforms_to_options (const origin_circuit_t * *circ*, const or_options_t * *options*)

Check whether, according to the policies in **options**, the circuit **circ** makes sense.

References origin_circuit_t::cpath, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, crypt_path_t::extend_info, crypt_path_t::next, and routerset_contains_extendinfo().

4.8.3.4 void circuit_detach_stream (circuit_t * *circ*, edge_connection_t * *conn*)

If the stream **conn** is a member of any of the linked lists of **circ**, then remove it from the list.

References CIRCUIT_IS_ORIGIN, edge_connection_t::cpath_layer, LD_BUG, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, origin_circuit_t::p_streams, or_circuit_t::resolving_streams, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and tor_fragile_assert.

Referenced by connection_about_to_close_connection(), connection_ap_detach_retriable(), connection_exit_connect(), dns_cancel_pending_resolve(), dns_found_answer(), handle_control_attachstream(), and purge_expired_resolves().

4.8.3.5 int circuit_enough_testing_circs (void)

Return 1 if we've already exercised our bandwidth, or if we have fewer than NUM_PARALLEL_TESTING_CIRCS testing circuits established or on the way. Else return 0.

References CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_TESTING, CIRCUIT_STATE_OPEN, have_performed_bandwidth_test, circuit_t::marked_for_close, circuit_t::next, NUM_PARALLEL_TESTING_CIRCS, circuit_t::purpose, and circuit_t::state.

Referenced by circuit_testing_opened(), consider_testing_reachability(), and rep_hist_circbuilding_dormant().

4.8.3.6 void circuit_expire_building (time_t *now*)

Close all circuits that start at us, aren't open, and were born at least CircuitBuildTimeout seconds ago.

References or_connection_t::_base, connection_t::address, origin_circuit_t::build_state, circ_times, circuit_build_times_add_timeout(), circuit_build_times_set_timeout(), CIRCUIT_IS_ORIGIN, circuit_log_path(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_STATE_OPEN, circuit_state_to_string(), origin_circuit_t::cpath, LD_CIRC, LD_REND, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, circuit_t::next, cpath_build_state_t::onehop_tunnel, connection_t::port, circuit_t::purpose, crypt_path_t::state, circuit_t::state, circuit_build_times_t::timeout_ms, circuit_t::timestamp_created, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by run_scheduled_events().

4.8.3.7 static void circuit_expire_old_circuits_clientside (time_t *now*) [static]

Find each circuit that has been unused for too long, or dirty for too long and has no streams on it: mark it for close.

References circ_times, circuit_build_times_needs_circuits(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_STATE_OPEN, or_options_t::CircuitIdleTimeout, get_options(), IDLE_TIMEOUT_WHILE_LEARNING, LD_CIRC, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::next, circuit_t::purpose, circuit_t::state, circuit_t::timestamp_created, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by circuit_build_needed_circs().

4.8.3.8 void circuit_expire_old_circuits_serverside (time_t *now*)

Find each non-origin circuit that has been unused for too long, has no streams on it, used a create_fast, and ends here: mark it for close.

References CIRCUIT_IS_ORIGIN, IDLE_ONE_HOP_CIRC_TIMEOUT, or_circuit_t::is_first_hop, LD_CIRC, circuit_t::marked_for_close, circuit_t::n_conn, or_circuit_t::n_streams, circuit_t::next, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::resolving_streams, or_connection_t::timestamp_last_added_nonpadding, and TO_OR_CIRCUIT().

Referenced by run_scheduled_events().

4.8.3.9 static origin_circuit_t∗ circuit_get_best (edge_connection_t ∗ *conn*, int *must_be_open*, uint8_t *purpose*, int *need_uptime*, int *need_internal*) [static]

Find the best circ that conn can use, preferably one which is dirty. Circ must not be too old.

Conn must be defined.

If must_be_open, ignore circs not in CIRCUIT_STATE_OPEN.

circ_purpose specifies what sort of circuit we must have. It can be C_GENERAL, C_INTRODUCE_ACK_WAIT, or C_REND_JOINED.

If it's REND_JOINED and must_be_open==0, then return the closest rendezvous-purposed circuit that you can find.

If it's INTRODUCE_ACK_WAIT and must_be_open==0, then return the closest introduce-purposed circuit that you can find.

References circuit_is_acceptable(), circuit_is_better(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_STATE_OPEN, LD_CIRC, LD_REND, circuit_t::next, circuit_t::state, circuit_t::timestamp_created, TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch().

4.8.3.10 static int circuit_get_open_circ_or_launch (edge_connection_t ∗ *conn*, uint8_t *desired_circuit_purpose*, origin_circuit_t ∗∗ *circp*) [static]

Find an open circ that we're happy to use for **conn** and return 1. If there isn't one, and there isn't one on the way, launch one and return 0. If it will never work, return -1.

Write the found or in-progress or launched circ into *circp.

References origin_circuit_t::_base, edge_connection_t::_base, socks_request_t::address, any_bridge_descriptors_known(), AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RENDDESC_WAIT, base16_decode(), bridges_known_but_down(), bridges_retry_all(), edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, CIRCLAUNCH_ONEHOP_TUNNEL, circuit_get_best(), circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_STATE_OPEN, socks_request_t::command, CONN_TYPE_DIR, connection_ap_can_use_exit(), connection_edge_is_rendezvous_stream(), connection_get_by_type(), DIGEST_LEN, escaped_safe_str_client(), extend_info_alloc(), extend_info_free(), extend_info_from_router(), get_options(), HEX_DIGEST_LEN, LD_APP, LD_CIRC, LD_DIR, LD_REND, or_options_t::LongLivedPorts, extend_info_t::nickname, rend_data_t::onion_address, socks_request_t::port, circuit_t::purpose, rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_client_rendcirc_has_opened(), origin_circuit_t::rend_data, edge_connection_t::rend_data, rend_data_dup(), rep_hist_note_used_internal(), router_exit_policy_all_routers_reject(), router_get_by_nickname(), router_have_minimum_dir_info(), routerlist_retry_directory_downloads(), safe_str_client(), smartlist_string_num_isin(), SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, circuit_t::state, connection_t::state, tor_addr_from_str(), tor_assert, tor_free, tor_inet_aton(), edge_connection_t::use_begindir, or_options_t::UseBridges, and edge_connection_t::want_onehop.

Referenced by connection_ap_handshake_attach_circuit().

4.8.3.11 void circuit_has_opened (origin_circuit_t * *circ*)

The circuit **circ** has just become open. Take the next step: for rendezvous circuits, we pass circ to the appropriate function in rendclient or rendservice. For general circuits, we call connection_ap_attach_pending, which looks for pending streams that could use circ.

References CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, circuit_testing_opened(), connection_ap_attach_pending(), control_event_circuit_status(), rend_client_introcirc_has_opened(), rend_client_rendcirc_has_opened(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), and TO_CIRCUIT.

Referenced by circuit_send_next_onion_skin(), and rend_service_intro_has_opened().

4.8.3.12 static void circuit_increment_failure_count (void) `[static]`

Record another failure at opening a general circuit. When we have too many, we'll stop trying for the remainder of this minute.

References LD_CIRC, and n_circuit_failures.

Referenced by circuit_build_failed().

4.8.3.13 static int circuit_is_acceptable (circuit_t * *circ*, edge_connection_t * *conn*, int *must_be_open*, uint8_t *purpose*, int *need_uptime*, int *need_internal*, time_t *now*) `[static]`

Return 1 if **circ** could be returned by **circuit_get_best()** (p. 344). Else return 0.

References extend_info_t::addr, socks_request_t::address, origin_circuit_t::build_state, build_state_get_exit_router(), cpath_build_state_t::chosen_exit, edge_connection_t::chosen_exit_name, CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_STATE_OPEN, connection_ap_can_use_exit(), DIGEST_LEN, get_options(), hexdigest_to_digest(), extend_info_t::identity_digest, cpath_build_state_t::is_internal, LD_CIRC, circuit_t::marked_for_close, circuit_t::n_conn, cpath_build_state_t::need_uptime, cpath_build_state_t::onehop_tunnel, rend_data_t::onion_address, socks_request_t::port, extend_info_t::port, circuit_t::purpose, rend_cmp_service_ids(), origin_circuit_t::rend_data, edge_connection_t::rend_data, edge_connection_t::socks_request, circuit_t::state, circuit_t::timestamp_dirty, TO_ORIGIN_CIRCUIT(), tor_addr_eq, tor_addr_from_str(), tor_assert, tor_digest_is_zero(), and edge_connection_t::want_onehop.

Referenced by circuit_get_best().

4.8.3.14 static int circuit_is_better (circuit_t * *a*, circuit_t * *b*, uint8_t *purpose*) `[static]`

Return 1 if circuit **a** is better than circuit **b** for **purpose**, and return 0 otherwise. Used by circuit_get_best.

References CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_REND_JOINED, circuit_t::purpose, circuit_t::timestamp_created, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by circuit_get_best().

4.8.3.15 origin_circuit_t* circuit_launch_by_extend_info (uint8_t *purpose*, extend_info_t * *extend_info*, int *flags*)

Launch a new circuit with purpose **purpose** and exit node **extend_info** (or NULL to select a random exit node). If flags contains CIRCLAUNCH_NEED_UPTIME, choose among routers with high uptime. If CIRCLAUNCH_NEED_CAPACITY is set, choose among routers with high bandwidth. If CIRCLAUNCH_IS_INTERNAL is true, the last hop need not be an exit node. If CIRCLAUNCH_ONEHOP_TUNNEL is set, the circuit will have only one hop. Return the newly allocated circuit on success, or NULL on failure.

References origin_circuit_t::_base, origin_circuit_t::build_state, build_state_get_exit_nickname(), CIRCLAUNCH_ONEHOP_TUNNEL, circuit_establish_circuit(), circuit_extend_to_new_exit(), circuit_find_to_cannibalize(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, did_circs_fail_last_period, LD_BUG, LD_CIRC, MAX_CIRCUIT_FAILURES, n_circuit_failures, circuit_t::purpose, router_have_minimum_dir_info(), circuit_t::timestamp_created, tor_assert, and tor_-

fragile_assert.

Referenced by circuit_get_open_circ_or_launch(), circuit_launch_by_router(), rend_client_introduction_acked(), rend_service_introduce(), rend_service_launch_establish_intro(), and rend_service_relaunch_rendezvous().

4.8.3.16 origin_circuit_t* circuit_launch_by_router (uint8_t *purpose*, routerinfo_t ∗ *exit*, int *flags*)

Launch a new circuit; see **circuit_launch_by_extend_info()** (p. 346) for details on arguments.

References circuit_launch_by_extend_info(), extend_info_free(), and extend_info_from_router().

Referenced by circuit_build_needed_circs(), circuit_predict_and_launch_new(), consider_testing_reachability(), and handle_control_extendcircuit().

4.8.3.17 static void circuit_predict_and_launch_new (void) `[static]`

Figure out how many circuits we have open that are clean. Make sure it's enough for all the upcoming behaviors we predict we'll have. But put an upper bound on the total number of circuits.

References origin_circuit_t::build_state, circ_times, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, circuit_all_predicted_ports_handled(), circuit_build_times_needs_circuits_now(), CIRCUIT_IS_ORIGIN, circuit_launch_by_router(), CIRCUIT_PURPOSE_C_GENERAL, cpath_build_state_t::is_internal, LD_CIRC, circuit_t::marked_for_close, MAX_UNUSED_OPEN_CIRCUITS, cpath_build_state_t::need_uptime, circuit_t::next, num_rend_services(), cpath_build_state_t::onehop_tunnel, circuit_t::purpose, rep_hist_get_predicted_internal(), circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by circuit_build_needed_circs().

4.8.3.18 void circuit_remove_handled_ports (smartlist_t ∗ *needed_ports*)

Remove any elements in **needed_ports** that are handled by an open or in-progress circuit.

References circuit_stream_is_being_handled(), LD_CIRC, MIN_CIRCUITS_HANDLING_STREAM, smartlist_del(), tor_assert, and tor_free.

Referenced by circuit_get_unhandled_ports().

4.8.3.19 void circuit_reset_failure_count (int *timeout*)

Reset the failure count for opening general circuits. This means we will try MAX_CIRCUIT_FAILURES times more (if necessary) before stopping again.

References did_circs_fail_last_period, MAX_CIRCUIT_FAILURES, and n_circuit_failures.

Referenced by circuit_build_needed_circs(), and circuit_send_next_onion_skin().

4.8.3.20 int circuit_stream_is_being_handled (edge_connection_t ∗ *conn*, uint16_t *port*, int *min*)

Return 1 if at least **min** general-purpose non-internal circuits will have an acceptable exit node for exit stream **conn** if it is defined, else for "∗:port". Else return 0.

References ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, origin_circuit_t::build_state, build_state_get_exit_router(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_GENERAL, compare_addr_to_addr_policy(), connection_ap_can_use_exit(), routerinfo_t::exit_policy, get_options(), cpath_build_state_t::is_internal, circuit_t::marked_for_close, cpath_build_state_t::need_uptime, circuit_t::next, cpath_build_state_t::onehop_tunnel, socks_request_t::port, circuit_t::purpose, smartlist_string_num_isin(), edge_connection_t::socks_request, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by ap_stream_wants_exit_attention(), and circuit_remove_handled_ports().

4.8.3.21 static void circuit_testing_failed (origin_circuit_t ∗ *circ*, int *at_last_hop*) `[static]`

A testing circuit has failed to build. Take whatever stats we want.

References check_whether_orport_reachable(), get_options(), LD_GENERAL, and server_mode().

Referenced by circuit_build_failed().

4.8.3.22 static void circuit_testing_opened (origin_circuit_t ∗ *circ*) `[static]`

A testing circuit has completed. Take whatever stats we want. Noticing reachability is taken care of in **onionskin_answer()** (p. 324), so there's no need to record anything here. But if we still want to do the bandwidth test, and we now have enough testing circuits open, do it.

References check_whether_orport_reachable(), circuit_enough_testing_circs(), consider_testing_reachability(), END_CIRC_AT_ORIGIN, have_performed_bandwidth_test, NUM_PARALLEL_TESTING_CIRCS, router_perform_bandwidth_test(), and TO_CIRCUIT.

Referenced by circuit_has_opened().

4.8.3.23 int connection_ap_handshake_attach_chosen_circuit (edge_connection_t ∗ *conn*, origin_circuit_t ∗ *circ*, crypt_path_t ∗ *cpath*)

Attempt to attach the connection **conn** to **circ**, and send a begin or resolve cell as appropriate. Return values are as for connection_ap_handshake_attach_circuit. The stream will exit from the hop indicated by **cpath**, or from the last hop in circ's cpath if **cpath** is NULL.

References origin_circuit_t::_base, edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, CIRCUIT_STATE_OPEN, socks_request_t::command, connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), consider_recording_trackhost(), link_apconn_to_circ(), SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, circuit_t::state, connection_t::state, circuit_t::timestamp_dirty, tor_assert, and edge_connection_t::use_begindir.

Referenced by connection_ap_handshake_attach_circuit().

4.8.3.24 int connection_ap_handshake_attach_circuit (edge_connection_t * *conn*)

Try to find a safe live circuit for CONN_TYPE_AP connection conn. If we don't find one: if conn cannot be handled by any known nodes, warn and return -1 (conn needs to die, and is maybe already marked); else launch new circuit (if necessary) and return 0. Otherwise, associate conn with a safe live circuit, do the right next step, and return 1.

References origin_circuit_t::_base, edge_connection_t::_base, connection_t::addr, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, assert_circuit_ok(), edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, circuit_get_open_circ_or_launch(), CIRCUIT_IS_ORIGIN, circuit_log_path(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_STATE_OPEN, connection_ap_can_use_exit(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_send_begin(), connection_edge_is_rendezvous_stream(), edge_connection_t::cpath_layer, get_options(), LD_APP, LD_CIRC, LD_REND, link_apconn_to_circ(), circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::next, rend_data_t::onion_address, socks_request_t::port, connection_t::port, circuit_t::purpose, rend_client_send_introduction(), rend_cmp_service_ids(), edge_connection_t::rend_data, origin_circuit_t::rend_data, router_get_by_nickname(), safe_str_client(), edge_connection_t::socks_request, or_options_t::SocksTimeout, connection_t::state, connection_t::timestamp_created, circuit_t::timestamp_dirty, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), tor_addr_is_null(), tor_assert, tor_free, and edge_connection_t::want_onehop.

Referenced by connection_ap_attach_pending(), connection_ap_detach_retriable(), connection_ap_make_link(), and rend_client_desc_trynow().

4.8.3.25 static void consider_recording_trackhost (edge_connection_t * *conn*, origin_circuit_t * *circ*) `[static]`

If an exit wasn't specifically chosen, save the history for future use.

References socks_request_t::address, addressmap_have_mapping(), get_options(), HEX_DIGEST_LEN, edge_connection_t::socks_request, strcasecmpend(), or_options_t::TrackHostExits, and or_options_t::TrackHostExitsExpire.

Referenced by connection_ap_handshake_attach_chosen_circuit().

4.8.3.26 static int cpath_is_on_circuit (origin_circuit_t * *circ*, crypt_path_t * *crypt_path*) `[static]`

Return true iff **crypt_path** is one of the crypt_paths for **circ**.

References origin_circuit_t::cpath, and crypt_path_t::next.

Referenced by link_apconn_to_circ().

4.8.3.27 static void link_apconn_to_circ (edge_connection_t * *apconn*, origin_circuit_t * *circ*, crypt_path_t * *cpath*) `[static]`

Attach the AP stream **apconn** to circ's linked list of p_streams. Also set apconn's cpath_layer to **cpath**, or to the last hop in circ's cpath if **cpath** is NULL.

References edge_connection_t::_base, origin_circuit_t::_base, origin_circuit_t::cpath, cpath_is_on_circuit(), edge_connection_t::cpath_layer, LD_APP, LD_CIRC, circuit_t::n_circ_id, edge_connection_t::next_stream, edge_connection_t::on_circuit, origin_circuit_t::p_streams, crypt_path_t::prev, crypt_path_t::state, connection_t::timestamp_lastread, TO_CIRCUIT, and tor_assert.

Referenced by connection_ap_handshake_attach_chosen_circuit(), and connection_ap_handshake_attach_circuit().

4.8.3.28 void reset_bandwidth_test (void)

Reset have_performed_bandwidth_test, so we'll start building testing circuits again so we can exercise our bandwidth.

References have_performed_bandwidth_test.

Referenced by ip_address_changed(), and run_scheduled_events().

4.8.4 Variable Documentation

4.8.4.1 int did_circs_fail_last_period = 0 `[static]`

Before the last time we called **circuit_reset_failure_count()** (p. 347), were there a lot of failures?

Referenced by circuit_launch_by_extend_info(), and circuit_reset_failure_count().

4.8.4.2 circuit_t∗ global_circuitlist

A global list of all circuits at this hop.

4.8.4.3 int have_performed_bandwidth_test = 0 `[static]`

True iff we've ever had enough testing circuits open to test our bandwidth.

Referenced by circuit_enough_testing_circs(), circuit_testing_opened(), and reset_bandwidth_test().

4.8.4.4 int n_circuit_failures = 0 `[static]`

Number of consecutive failures so far; should only be touched by circuit_launch_new and circuit_∗_failure_count.

Referenced by circuit_increment_failure_count(), circuit_launch_by_extend_info(), and circuit_reset_failure_count().

4.9 command.c File Reference

Functions for processing incoming cells.

`#include "or.h"`

Defines

- #define **PROCESS_CELL**(tp, cl, cn) command_process_ ## tp ## _cell(cl, cn)
- #define **NETINFO_NOTICE_SKEW** 3600

Functions

- static void **command_process_create_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- static void **command_process_created_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- static void **command_process_relay_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- static void **command_process_destroy_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- static void **command_process_versions_cell** (**var_cell_t** ∗cell, **or_connection_t** ∗conn)
- static void **command_process_netinfo_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- void **command_process_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- void **command_process_var_cell** (**var_cell_t** ∗cell, **or_connection_t** ∗conn)

Variables

- uint64_t **stats_n_padding_cells_processed** = 0
- uint64_t **stats_n_create_cells_processed** = 0
- uint64_t **stats_n_created_cells_processed** = 0
- uint64_t **stats_n_relay_cells_processed** = 0
- uint64_t **stats_n_destroy_cells_processed** = 0
- uint64_t **stats_n_versions_cells_processed** = 0
- uint64_t **stats_n_netinfo_cells_processed** = 0

4.9.1 Detailed Description

Functions for processing incoming cells.

4.9.2 Function Documentation

4.9.2.1 void command_process_cell (cell_t ∗ *cell*, or_connection_t ∗ *conn*)

Process a **cell** that was just received on **conn**. Keep internal statistics about how many of each cell we've processed so far this second, and the total number of microseconds it took to process each type of cell.

References or_connection_t::_base, cell_t::command, LD_OR, LD_PROTOCOL, OR_CONN_STATE_OR_HANDSHAKING, connection_t::state, stats_n_create_cells_processed, stats_n_created_cells_processed, stats_n_destroy_cells_processed, stats_n_netinfo_cells_processed, stats_n_padding_cells_processed, stats_n_relay_cells_processed, and tor_fragile_assert.

Referenced by connection_or_process_cells_from_inbuf().

4.9.2.2 static void command_process_create_cell (cell_t ∗ *cell*, or_connection_t ∗ *conn*) `[static]`

Process a 'create' **cell** that just arrived from **conn**. Make a new circuit with the p_circ_id specified in cell. Put the circuit in state onionskin_pending, and pass the onionskin to the cpuworker. Circ will get picked up again when the cpuworker finishes decrypting it.

References or_circuit_t::_base, or_connection_t::_base, connection_t::address, assign_onionskin_to_cpuworker(), cell_t::circ_id, or_connection_t::circ_id_type, CIRC_ID_TYPE_HIGHER, CIRC_ID_TYPE_LOWER, circuit_id_in_use_on_orconn(), CIRCUIT_PURPOSE_OR, circuit_set_state(), CIRCUIT_STATE_ONIONSKIN_PENDING, cell_t::command, connection_or_send_destroy(), DIGEST_LEN, escaped(), fast_server_handshake(), get_options(), or_connection_t::identity_digest, LD_GENERAL, LD_OR, LD_PROTOCOL, routerinfo_t::nickname, onionskin_answer(), or_circuit_new(), cell_t::payload, routerinfo_t::platform, connection_t::port, circuit_t::purpose, router_get_by_digest(), server_mode(), connection_t::timestamp_created, TO_CIRCUIT, tor_assert, and we_are_hibernating().

4.9.2.3 static void command_process_created_cell (cell_t ∗ *cell*, or_connection_t ∗ *conn*) `[static]`

Process a 'created' **cell** that just arrived from **conn**. Find the circuit that it's intended for. If we're not the origin of the circuit, package the 'created' cell in an 'extended' relay cell and pass it back. If we are the origin of the circuit, send it to **circuit_finish_handshake()** (p. 312) to finish processing keys, and then call **circuit_send_next_onion_skin()** (p. 315) to extend to the next hop in the circuit if necessary.

References cell_t::circ_id, circuit_finish_handshake(), circuit_get_by_circid_orconn(), CIRCUIT_IS_ORIGIN, circuit_send_next_onion_skin(), cell_t::command, LD_OR, LD_PROTOCOL, circuit_t::n_circ_id, cell_t::payload, relay_send_command_from_edge(), and TO_ORIGIN_CIRCUIT().

4.9.2.4 static void command_process_destroy_cell (cell_t ∗ *cell*, or_connection_t ∗ *conn*) `[static]`

Process a 'destroy' **cell** that just arrived from **conn**. Find the circ that it refers to (if any).

If the circ is in state onionskin_pending, then call **onion_pending_remove()** (p. 737) to remove it from the pending onion list (note that if it's already being processed by the cpuworker, it won't be in the list anymore; but when the cpuworker returns it, the circuit will be gone, and the cpuworker response will be dropped).

Then mark the circuit for close (which marks all edges for close, and passes the destroy cell onward if necessary).

References or_connection_t::_base, connection_t::address, cell_t::circ_id, circuit_get_by_circid_orconn(), CIRCUIT_IS_ORIGIN, circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), END_CIRC_REASON_FLAG_REMOTE, LD_OR, cell_t::payload, connection_t::port, relay_send_command_from_edge(), and TO_OR_CIRCUIT().

4.9.2.5 static void command_process_netinfo_cell (cell_t ∗ *cell*, or_connection_t ∗ *conn*) `[static]`

Process a 'netinfo' cell: read and act on its contents, and set the connection state to "open".

Warn when we get a netinfo skew with at least this value.

References or_connection_t::_base, connection_t::address, assert_connection_ok(), CELL_PAYLOAD_SIZE, connection_or_set_state_open(), control_event_general_status(), decode_address_from_payload(), format_time_interval(), get_uint32(), or_connection_t::handshake_state, or_connection_t::identity_digest, or_connection_t::is_canonical, LD_GENERAL, LD_OR, or_connection_t::link_proto, OR_CONN_STATE_OR_HANDSHAKING, cell_t::payload, connection_t::port, or_connection_t::real_addr, or_handshake_state_t::received_versions, router_get_by_digest(), safe_str_client(), or_handshake_state_t::sent_versions_at, connection_t::state, TO_CONN, tor_addr_eq, and tor_assert.

4.9.2.6 static void command_process_relay_cell (cell_t ∗ *cell*, or_connection_t ∗ *conn*) `[static]`

Process a 'relay' or 'relay_early' **cell** that just arrived from **conn**. Make sure it came in with a recognized circ_id. Pass it on to **circuit_receive_relay_cell()** (p. 850) for actual processing.

References or_connection_t::_base, connection_t::address, CELL_DIRECTION_IN, CELL_DIRECTION_OUT, cell_t::circ_id, circuit_get_by_circid_orconn(), CIRCUIT_IS_ORIGIN, circuit_receive_relay_cell(), CIRCUIT_STATE_ONIONSKIN_PENDING, or_connection_t::client_used, cell_t::command, LD_OR, LD_PROTOCOL, connection_t::port, or_circuit_t::remaining_relay_early_cells, safe_str(), circuit_t::state, and TO_OR_CIRCUIT().

4.9.2.7 void command_process_var_cell (var_cell_t ∗ *cell*, or_connection_t ∗ *conn*)

Process a **cell** that was just received on **conn**. Keep internal statistics about how many of each cell we've processed so far this second, and the total number of microseconds it took to process each type of cell.

References or_connection_t::_base, var_cell_t::command, LD_BUG, LD_OR, OR_CONN_STATE_OR_HANDSHAKING, connection_t::state, stats_n_versions_cells_processed, and tor_fragile_assert.

Referenced by connection_or_process_cells_from_inbuf().

4.9.2.8 static void command_process_versions_cell (var_cell_t ∗ *cell*, or_connection_t ∗ *conn*) `[static]`

Process a 'versions' cell. The current link protocol version must be 0 to indicate that no version has yet been negotiated. We compare the versions in the cell to the list of versions we support, pick the highest version we have in common, and continue the negotiation from there.

References or_connection_t::_base, connection_t::address, connection_or_send_netinfo(), get_uint16(), or_connection_t::handshake_state, is_or_protocol_version_known(), LD_OR, or_connection_t::link_proto, OR_CONN_STATE_OR_HANDSHAKING, var_cell_t::payload, var_cell_t::payload_len, connection_t::port, or_handshake_state_t::received_versions, safe_str_client(), connection_t::state, TO_CONN, and tor_assert.

4.9.3 Variable Documentation

4.9.3.1 uint64_t stats_n_create_cells_processed = 0

How many CELL_CREATE cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.9.3.2 uint64_t stats_n_created_cells_processed = 0

How many CELL_CREATED cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.9.3.3 uint64_t stats_n_destroy_cells_processed = 0

How many CELL_DESTROY cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.9.3.4 uint64_t stats_n_netinfo_cells_processed = 0

How many CELL_NETINFO cells have we received, ever?

Referenced by command_process_cell().

4.9.3.5 uint64_t stats_n_padding_cells_processed = 0

How many CELL_PADDING cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.9.3.6 uint64_t stats_n_relay_cells_processed = 0

How many CELL_RELAY cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.9.3.7 uint64_t stats_n_versions_cells_processed = 0

How many CELL_VERSIONS cells have we received, ever?

Referenced by command_process_var_cell().

4.10 compat.c File Reference

Wrappers to make calls more portable. This code defines functions such as tor_malloc, tor_snprintf, get/set various data types, renaming, setting socket options, switching user IDs. It is basically where the non-portable items are conditionally included depending on the platform.

`#include "compat.h"`

`#include <sys/utsname.h>`

`#include <unistd.h>`

`#include <sys/fcntl.h>`

`#include <pwd.h>`

`#include <grp.h>`

`#include <fcntl.h>`

`#include <errno.h>`

`#include <arpa/inet.h>`

`#include <netdb.h>`

`#include <sys/param.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <assert.h>`

`#include <signal.h>`

`#include <utime.h>`

`#include <sys/mman.h>`

`#include <sys/file.h>`

`#include "log.h"`

`#include "util.h"`

`#include "container.h"`

`#include "address.h"`

`#include "strlcpy.c"`

`#include "strlcat.c"`

Data Structures

- struct **tor_lockfile_t**
- struct **tor_pthread_data_t**

Defines

- #define **_GNU_SOURCE**
- #define **SEEK_CUR** 1
- #define **SEEK_END** 2

- #define **mark_socket_open**(s) STMT_NIL
- #define **ULIMIT_BUFFER** 32
- #define **CREDENTIAL_LOG_LEVEL** LOG_INFO
- #define **TIME_FNS_NEED_LOCKS**

Functions

- **tor_mmap_t** ∗ **tor_mmap_file** (const char ∗filename)
- void **tor_munmap_file** (**tor_mmap_t** ∗handle)
- int **tor_snprintf** (char ∗str, size_t size, const char ∗format,...)
- int **tor_vsnprintf** (char ∗str, size_t size, const char ∗format, va_list args)
- int **tor_asprintf** (char ∗∗strp, const char ∗fmt,...)
- int **tor_vasprintf** (char ∗∗strp, const char ∗fmt, va_list args)
- const void ∗ **tor_memmem** (const void ∗_haystack, size_t hlen, const void ∗_needle, size_t nlen)
- char ∗ **tor_strtok_r_impl** (char ∗str, const char ∗sep, char ∗∗lasts)
- uint16_t **get_uint16** (const char ∗cp)
- uint32_t **get_uint32** (const char ∗cp)
- uint64_t **get_uint64** (const char ∗cp)
- void **set_uint16** (char ∗cp, uint16_t v)
- void **set_uint32** (char ∗cp, uint32_t v)
- void **set_uint64** (char ∗cp, uint64_t v)
- int **replace_file** (const char ∗from, const char ∗to)
- int **touch_file** (const char ∗fname)
- **tor_lockfile_t** ∗ **tor_lockfile_lock** (const char ∗filename, int blocking, int ∗locked_out)
- void **tor_lockfile_unlock** (**tor_lockfile_t** ∗**lockfile**)
- off_t **tor_fd_getpos** (int fd)
- int **tor_fd_seekend** (int fd)
- static INLINE void **socket_accounting_lock** (void)
- static INLINE void **socket_accounting_unlock** (void)
- int **tor_close_socket** (int s)
- int **tor_open_socket** (int domain, int type, int protocol)
- int **tor_accept_socket** (int sockfd, struct sockaddr ∗addr, socklen_t ∗len)
- int **get_n_open_sockets** (void)
- void **set_socket_nonblocking** (int socket)
- int **tor_socketpair** (int family, int type, int protocol, int fd[2])
- int **set_max_file_descriptors** (rlim_t limit, int ∗max_out)
- static int **log_credential_status** (void)
- int **switch_id** (const char ∗user)
- char ∗ **get_user_homedir** (const char ∗username)
- int **tor_inet_aton** (const char ∗str, struct in_addr ∗addr)
- const char ∗ **tor_inet_ntop** (int af, const void ∗src, char ∗dst, size_t len)
- int **tor_inet_pton** (int af, const char ∗src, void ∗dst)
- int **tor_lookup_hostname** (const char ∗name, uint32_t ∗addr)
- const char ∗ **get_uname** (void)
- static void ∗ **tor_pthread_helper_fn** (void ∗_data)
- int **spawn_func** (void(∗func)(void ∗), void ∗data)
- void **spawn_exit** (void)
- void **tor_gettimeofday** (struct timeval ∗timeval)
- void **tor_mutex_init** (**tor_mutex_t** ∗mutex)

- void **tor_mutex_acquire** (**tor_mutex_t** ∗m)
- void **tor_mutex_release** (**tor_mutex_t** ∗m)
- void **tor_mutex_uninit** (**tor_mutex_t** ∗m)
- unsigned long **tor_get_thread_id** (void)
- **tor_mutex_t** ∗ **tor_mutex_new** (void)
- void **tor_mutex_free** (**tor_mutex_t** ∗m)
- void **tor_threads_init** (void)
- int **tor_mlockall** (void)
- void **set_main_thread** (void)
- int **in_main_thread** (void)
- int **network_init** (void)

Variables

- const uint32_t **TOR_ISALPHA_TABLE** [8]
- const uint32_t **TOR_ISALNUM_TABLE** [8]
- const uint32_t **TOR_ISSPACE_TABLE** [8] = { 0x3e00, 0x1, 0, 0, 0, 0, 0, 0 }
- const uint32_t **TOR_ISXDIGIT_TABLE** [8]
- const uint32_t **TOR_ISDIGIT_TABLE** [8] = { 0, 0x3ff0000, 0, 0, 0, 0, 0, 0 }
- const uint32_t **TOR_ISPRINT_TABLE** [8]
- const uint32_t **TOR_ISUPPER_TABLE** [8] = { 0, 0, 0x7fffffe, 0, 0, 0, 0, 0 }
- const uint32_t **TOR_ISLOWER_TABLE** [8] = { 0, 0, 0, 0x7fffffe, 0, 0, 0, 0 }
- const char **TOR_TOUPPER_TABLE** [256]
- const char **TOR_TOLOWER_TABLE** [256]
- static int **n_sockets_open** = 0
- static **tor_mutex_t** ∗ **socket_accounting_mutex** = NULL
- static char **uname_result** [256]
- static int **uname_result_is_set** = 0
- static pthread_mutexattr_t **attr_reentrant**
- static int **threads_initialized** = 0
- static unsigned long **main_thread_id** = -1

4.10.1 Detailed Description

Wrappers to make calls more portable. This code defines functions such as tor_malloc, tor_snprintf, get/set various data types, renaming, setting socket options, switching user IDs. It is basically where the non-portable items are conditionally included depending on the platform.

4.10.2 Define Documentation

4.10.2.1 #define TIME_FNS_NEED_LOCKS

Defined iff we need to add locks when defining fake versions of reentrant versions of time-related functions.

4.10.3 Function Documentation

4.10.3.1 int get_n_open_sockets (void)

Return the number of sockets we currently have opened.

Referenced by connection_connect(), connection_create_listener(), and warn_too_many_conns().

4.10.3.2 uint16_t get_uint16 (const char * *cp*)

Read a 16-bit value beginning at **cp**. Equivalent to *(uint16_t*)(cp), but will not cause segfaults on platforms that forbid unaligned memory access.

Referenced by circuit_extend(), command_process_versions_cell(), detect_compression_method(), fetch_from_buf_socks(), fetch_var_cell_from_buf(), peek_buf_has_control0_command(), relay_header_unpack(), rend_mid_establish_intro(), rend_parse_service_descriptor(), rend_service_introduce(), and tag_unpack().

4.10.3.3 uint32_t get_uint32 (const char * *cp*)

Read a 32-bit value beginning at **cp**. Equivalent to *(uint32_t*)(cp), but will not cause segfaults on platforms that forbid unaligned memory access.

Referenced by accounting_set_wakeup_time(), aes_set_iv(), circuit_extend(), command_process_netinfo_cell(), connection_ap_handshake_socks_resolved(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell_not_open(), decode_address_from_payload(), fetch_from_buf_socks(), rend_parse_service_descriptor(), rend_service_introduce(), and tell_controller_about_resolved_result().

4.10.3.4 uint64_t get_uint64 (const char * *cp*)

Read a 64-bit value beginning at **cp**. Equivalent to *(uint64_t*)(cp), but will not cause segfaults on platforms that forbid unaligned memory access.

Referenced by tag_unpack().

4.10.3.5 const char* get_uname (void)

Return a pointer to a description of our platform.

References tor_snprintf().

Referenced by get_platform_str(), options_validate(), and tor_init().

4.10.3.6 char* get_user_homedir (const char * *username*)

Allocate and return a string containing the home directory for the user **username**. Only works on posix-like systems.

References LD_CONFIG, and tor_assert.

Referenced by expand_filename().

4.10.3.7 int in_main_thread (void)

Return true iff called from the main thread.

References tor_get_thread_id().

Referenced by control_event_logmsg().

4.10.3.8 static int log_credential_status (void) `[static]`

Log details of current user and group credentials. Return 0 on success. Logs and return -1 on failure.

References LD_GENERAL, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

Referenced by switch_id().

4.10.3.9 int network_init (void)

On Windows, WSAEWOULDBLOCK is not always correct: when you see it, you need to ask the socket for its actual errno. Also, you need to get your errors from WSAGetLastError, not errno. (If you supply a socket of -1, we check WSAGetLastError, but don't correct WSAEWOULDBLOCKs.)

The upshot of all of this is that when a socket call fails, you should call tor_socket_errno *at most once* on the failing socket to get the error. Called before we make any calls to network-related functions. (Some operating systems require their network libraries to be initialized.)

References LD_NET.

Referenced by tor_init().

4.10.3.10 int replace_file (const char ∗ *from*, const char ∗ *to*)

Rename the file **from** to the file **to**. On Unix, this is the same as rename(2). On windows, this removes **to** first if it already exists. Returns 0 on success. Returns -1 and sets errno on failure.

References file_status().

Referenced by finish_writing_to_file_impl(), rotate_onion_key(), and router_rebuild_store().

4.10.3.11 void set_main_thread (void)

Start considering the current thread to be the 'main thread'. This has no effect on anything besides **in_main_thread()** (p. 359).

References tor_get_thread_id().

Referenced by start_daemon(), and tor_threads_init().

4.10.3.12 int set_max_file_descriptors (rlim_t *limit*, int ∗ *max_out*)

Learn the maximum allowed number of file descriptors. (Some systems have a low soft limit.

We compute this by finding the largest number that we can use. If we can't find a number greater than or equal to **limit**, then we fail: return -1.

Otherwise, return 0 and store the maximum we found inside **max_out**.

References LD_CONFIG, LD_NET, and tor_assert.

Referenced by options_act_reversible().

4.10.3.13 void set_socket_nonblocking (int *socket*)

Turn **socket** into a nonblocking socket.

Referenced by connection_connect(), connection_create_listener(), connection_handle_listener_read(), and spawn_cpuworker().

4.10.3.14 void set_uint16 (char ∗ *cp*, uint16_t *v*)

Set a 16-bit value beginning at **cp** to **v**. Equivalent to ∗(uint16_t∗)(cp) = v, but will not cause segfaults on platforms that forbid unaligned memory access.

Referenced by circuit_send_next_onion_skin(), connection_ap_handshake_socks_resolved(), connection_control_process_inbuf(), connection_or_send_versions(), relay_header_pack(), rend_service_intro_has_opened(), tag_pack(), and var_cell_pack_header().

4.10.3.15 void set_uint32 (char ∗ *cp*, uint32_t *v*)

Set a 32-bit value beginning at **cp** to **v**. Equivalent to ∗(uint32_t∗)(cp) = v, but will not cause segfaults on platforms that forbid unaligned memory access.

Referenced by circuit_send_next_onion_skin(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_connect(), connection_or_send_netinfo(), send_resolved_cell(), and send_resolved_hostname_cell().

4.10.3.16 void set_uint64 (char ∗ *cp*, uint64_t *v*)

Set a 64-bit value beginning at **cp** to **v**. Equivalent to ∗(uint64_t∗)(cp) = v, but will not cause segfaults on platforms that forbid unaligned memory access.

Referenced by tag_pack().

4.10.3.17 void spawn_exit (void)

End the current thread/process.

References tor_assert.

Referenced by cpuworker_main().

4.10.3.18 int spawn_func (void(∗)(void ∗) *func*, void ∗ *data*)

Minimalist interface to run a void function in the background. On Unix calls fork, on win32 calls beginthread. Returns -1 on failure. func should not return, but rather should call spawn_exit.

NOTE: if **data** is used, it should not be allocated on the stack, since in a multithreaded environment, there is no way to be sure that the caller's stack will still be around when the called function is running.

References tor_pthread_data_t::data, tor_pthread_data_t::func, tor_assert, and tor_pthread_helper_fn().

Referenced by spawn_cpuworker().

4.10.3.19 int switch_id (const char ∗ *user*)

Call setuid and setgid to run as **user** and switch to their primary group. Return 0 on success. On failure, log and return -1.

References LD_CONFIG, LD_GENERAL, log_credential_status(), and tor_assert.

Referenced by options_act_reversible().

4.10.3.20 int tor_accept_socket (int *sockfd*, struct sockaddr ∗ *addr*, socklen_t ∗ *len*)

As socket(), but counts the number of open sockets.

Referenced by connection_handle_listener_read(), and tor_socketpair().

4.10.3.21 int tor_asprintf (char ∗∗ *strp*, const char ∗ *fmt*, *...*)

Portable asprintf implementation. Does a printf() into a newly malloc'd string. Sets ∗**strp** to this string, and returns its length (not including the terminating NUL character).

You can treat this function as if its implementation were something like

```
     char buf[_INFINITY_];
     tor_snprintf(buf, sizeof(buf), fmt, args);
strp = tor_strdup(buf);
     return strlen(*strp):
```

Where _INFINITY_ is an imaginary constant so big that any string can fit into it.

References LD_BUG, tor_assert, and tor_vasprintf().

Referenced by check_nickname_list(), circuit_list_path_impl(), config_assign_line(), config_assign_value(), config_dump(), ensure_bandwidth_cap(), geoip_get_client_history(), geoip_get_request_history(), get_assigned_option(), getinfo_helper_config(), list_getinfo_options(), log_entry_guards(), networkstatus_compute_consensus(), options_act_reversible(), options_init_from_string(), options_transition_allowed(), options_validate(), or_state_save(), parse_bridge_stats_controller(), rep_hist_buffer_stats_write(), rep_hist_format_router_status(), validate_ports_csv(), and write_configuration_file().

4.10.3.22 int tor_close_socket (int *s*)

As close(), but guaranteed to work for sockets across platforms (including Windows, where close()ing a socket doesn't work. Returns 0 on success, -1 on failure.

References LD_BUG, LD_NET, and tor_assert.

Referenced by _connection_free(), connection_close_immediate(), connection_connect(), connection_create_listener(), connection_handle_listener_read(), cpuworker_main(), get_interface_address6(), spawn_cpuworker(), and tor_socketpair().

4.10.3.23 off_t tor_fd_getpos (int *fd*)

Return the position of **fd** with respect to the start of the file.

Referenced by log_tor_version().

4.10.3.24 int tor_fd_seekend (int *fd*)

Move **fd** to the end of the file. Return -1 on error, 0 on success.

Referenced by add_file_log(), and start_writing_to_file().

4.10.3.25 unsigned long tor_get_thread_id (void)

Return an integer representing this thread.

Referenced by in_main_thread(), set_main_thread(), and setup_openssl_threading().

4.10.3.26 void tor_gettimeofday (struct timeval * *timeval*)

Set *timeval to the current time of day. On error, log and terminate. (Same as gettimeofday(timeval,NULL), but never returns -1.)

References LD_GENERAL.

Referenced by _log_prefix(), circuit_send_next_onion_skin(), geoip_change_dirreq_state(), geoip_get_dirreq_history(), geoip_start_dirreq(), and init_circuit_base().

4.10.3.27 int tor_inet_aton (const char * *str*, struct in_addr * *addr*)

Set *addr to the IP address (in dotted-quad notation) stored in c. Return 1 on success, 0 if c is badly formatted. (Like inet_aton(c,addr), but works on Windows and Solaris.)

References tor_sscanf().

Referenced by address_is_in_virtual_range(), authority_cert_parse_from_string(), circuit_get_open_circ_or_launch(), client_dns_set_addressmap(), connection_ap_can_use_exit(), connection_ap_handshake_rewrite_and_attach(), connection_ap_process_end_not_open(), connection_connect(), directory_handle_command_get(), dirserv_router_has_valid_address(), getinfo_helper_geoip(), http_set_address_origin(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_addr_and_port_range(), resolve_my_address(), router_find_exact_exit_enclave(), router_new_address_suggestion(), router_parse_entry_from_string(), routerstatus_parse_entry_from_string(), tor_addr_parse_reverse_lookup_name(), and tor_inet_pton().

4.10.3.28 const char* tor_inet_ntop (int *af*, const void * *src*, char * *dst*, size_t *len*)

Given **af**==AF_INET and **src** a struct in_addr, or **af**==AF_INET6 and **src** a struct in6_addr, try to format the address and store it in the **len**-byte buffer **dst**. Returns **dst** on success, NULL on failure.

(Like inet_ntop(af,src,dst,len), but works on platforms that don't have it: Tor sometimes needs to format ipv6 addresses even on platforms without ipv6 support.)

References tor_inet_ntoa(), and tor_snprintf().

Referenced by tor_addr_to_str(), and tor_dup_ip().

4.10.3.29 int tor_inet_pton (int *af*, const char * *src*, void * *dst*)

Given **af**==AF_INET or **af**==AF_INET6, and a string **src** encoding an IPv4 address or IPv6 address correspondingly, try to parse the address and store the result in **dst** (which must have space for a struct in_addr or a struct in6_addr, as appropriate). Return 1 on success, 0 on a bad parse, and -1 on a bad **af**.

(Like inet_pton(af,src,dst) but works on platforms that don't have it: Tor sometimes needs to format ipv6 addresses even on platforms without ipv6 support.)

References tor_assert, tor_inet_aton(), and tor_sscanf().

Referenced by tor_addr_from_str(), tor_addr_lookup(), and tor_addr_parse_mask_ports().

4.10.3.30 tor_lockfile_t* tor_lockfile_lock (const char * *filename*, int *blocking*, int * *locked_out*)

Try to get a lock on the lockfile **filename**, creating it as necessary. If someone else has the lock and **blocking** is true, wait until the lock is available. Otherwise return immediately whether we succeeded or not.

Set ***locked_out** to true if somebody else had the lock, and to false otherwise.

Return a **tor_lockfile_t** (p. **238**) on success, NULL on failure.

(Implementation note: because we need to fall back to fcntl on some platforms, these locks are per-process, not per-thread. If you want to do in-process locking, use **tor_mutex_t** (p. 240) like a normal person.)

References tor_lockfile_t::fd, tor_lockfile_t::filename, and LD_FS.

Referenced by try_locking().

4.10.3.31 void tor_lockfile_unlock (tor_lockfile_t * *lockfile*)

Release the lock held as **lockfile**.

References tor_lockfile_t::fd, tor_lockfile_t::filename, LD_FS, tor_assert, and tor_free.

Referenced by release_lockfile().

4.10.3.32 int tor_lookup_hostname (const char * *name*, uint32_t * *addr*)

Similar behavior to Unix gethostbyname: resolve **name**, and set ***addr** to the proper IP address, in host byte order. Returns 0 on success, -1 on failure; 1 on transient failure.

(This function exists because standard windows gethostbyname doesn't treat raw IP addresses properly.)

References tor_addr_family(), tor_addr_lookup(), and tor_addr_to_ipv4h().

Referenced by add_trusted_dir_server(), parse_addr_port(), and resolve_my_address().

4.10.3.33 const void∗ tor_memmem (const void ∗ _*haystack*, size_t *hlen*, const void ∗ _*needle*, size_t *nlen*)

Given **hlen** bytes at **haystack** and **nlen** bytes at **needle**, return a pointer to the first occurrence of the needle within the haystack, or NULL if there is no such occurrence.

Requires that nlen be greater than zero.

References tor_assert.

4.10.3.34 int tor_mlockall (void)

Attempt to lock all current and all future memory pages. This should only be called once and while we're privileged. Like mlockall() we return 0 when we're successful and -1 when we're not. Unlike mlockall() we return 1 if we've already attempted to lock memory.

References LD_GENERAL.

Referenced by options_act_reversible().

4.10.3.35 tor_mmap_t∗ tor_mmap_file (const char ∗ *filename*)

Try to create a memory mapping for **filename** and return it. On failure, return NULL. Sets errno properly, using ERANGE to mean "empty file".

References tor_mmap_t::data, LD_FS, tor_mmap_t::mapping_size, tor_mmap_t::size, and tor_assert.

Referenced by microdesc_cache_rebuild(), microdesc_cache_reload(), router_rebuild_store(), and router_reload_router_list_impl().

4.10.3.36 void tor_munmap_file (tor_mmap_t ∗ *handle*)

Release storage held for a memory mapping.

References tor_mmap_t::data, tor_mmap_t::mapping_size, and tor_free.

Referenced by microdesc_cache_clear(), microdesc_cache_rebuild(), router_rebuild_store(), router_reload_router_list_impl(), and routerlist_free().

4.10.3.37 void tor_mutex_acquire (tor_mutex_t ∗ *m*)

Wait until **m** is free, then acquire it.

References LD_GENERAL, tor_mutex_t::mutex, tor_assert, and tor_fragile_assert.

Referenced by _openssl_dynlock_lock_cb(), _openssl_locking_cb(), dup_onion_keys(), rotate_onion_key(), and set_onion_key().

4.10.3.38 void tor_mutex_free (tor_mutex_t ∗ *m*)

Release all storage and system resources held by **m**.

References tor_free, and tor_mutex_uninit().

Referenced by _openssl_dynlock_destroy_cb(), crypto_global_cleanup(), and router_free_all().

4.10.3.39 void tor_mutex_init (tor_mutex_t * *mutex*)

Initialize **mutex** so it can be locked. Every mutex must be set up with **tor_mutex_init()** (p. 365) or **tor_mutex_new()** (p. 365); not both.

References LD_GENERAL, tor_mutex_t::mutex, tor_fragile_assert, and tor_threads_init().

Referenced by init_logging(), and tor_mutex_new().

4.10.3.40 tor_mutex_t* tor_mutex_new (void)

Return a newly allocated, ready-for-use mutex.

References tor_mutex_init().

Referenced by _openssl_dynlock_create_cb(), init_keys(), and setup_openssl_threading().

4.10.3.41 void tor_mutex_release (tor_mutex_t * *m*)

Release the lock **m** so another thread can have it.

References LD_GENERAL, tor_mutex_t::mutex, tor_assert, and tor_fragile_assert.

Referenced by _openssl_dynlock_lock_cb(), _openssl_locking_cb(), dup_onion_keys(), rotate_onion_key(), and set_onion_key().

4.10.3.42 void tor_mutex_uninit (tor_mutex_t * *m*)

Clean up the mutex **m** so that it no longer uses any system resources. Does not free **m**. This function must only be called on mutexes from **tor_mutex_init()** (p. 365).

References LD_GENERAL, tor_mutex_t::mutex, tor_assert, and tor_fragile_assert.

Referenced by tor_mutex_free().

4.10.3.43 int tor_open_socket (int *domain*, int *type*, int *protocol*)

As socket(), but counts the number of open sockets.

Referenced by connection_connect(), connection_create_listener(), get_interface_address6(), and tor_socketpair().

4.10.3.44 static void* tor_pthread_helper_fn (void * _ *data*) `[static]`

Given a **tor_pthread_data_t** (p. 241) **_data**, call _data->func(d->data) and free _data. Used to make sure we can call functions the way pthread expects.

References tor_pthread_data_t::data, tor_pthread_data_t::func, and tor_free.

Referenced by spawn_func().

4.10.3.45 int tor_snprintf (char * *str*, size_t *size*, const char * *format*, *...*)

Replacement for snprintf. Differs from platform snprintf in two ways: First, always NUL-terminates its output. Second, always returns -1 if the result is truncated. (Note that this return behavior does *not* conform to C99; it just happens to be easier to emulate "return -1" with conformant implementations than it is to emulate "return number that would be written" with non-conformant implementations.)

References tor_vsnprintf().

Referenced by _log_prefix(), add_trusted_dir_server(), addressmap_get_mappings(), addressmap_rewrite_reverse(), circuit_build_times_update_state(), circuit_list_path_impl(), circuit_purpose_to_controller_string(), circuit_state_to_string(), clear_trackexithost_mappings(), client_dns_set_addressmap_impl(), client_dns_set_reverse_addressmap(), conn_state_to_string(), conn_type_to_string(), connection_ap_handshake_send_begin(), connection_proxy_connect(), control_event_bootstrap(), control_event_bootstrap_problem(), control_event_circuit_status(), control_event_descriptors_changed(), control_event_guard(), control_event_or_authdir_new_descriptor(), control_event_or_conn_status(), control_event_status(), control_event_stream_status(), control_setconf_helper(), directory_get_consensus_url(), directory_send_command(), dirserv_add_multiple_descriptors(), dirserv_dump_directory_to_string(), dirvote_compute_params(), dirvote_create_microdescriptor(), dirvote_format_microdesc_vote_line(), dump_desc(), entry_guards_update_state(), esc_for_log(), esc_router_info(), expand_filename(), extrainfo_dump_to_string(), format_msg(), format_networkstatus_vote(), format_time_interval(), generate_runningrouters(), generate_v2_networkstatus_opinion(), geoip_bridge_stats_write(), geoip_get_dirreq_history(), get_next_token(), get_platform_str(), get_uname(), get_version(), getinfo_helper_accounting(), getinfo_helper_entry_guards(), getinfo_helper_events(), handle_control_getconf(), handle_control_mapaddress(), inform_testing_reachability(), init_keys(), load_downloaded_routers(), log_credential_status(), log_tor_version(), lookup_last_hid_serv_request(), make_consensus_method_list(), networkstatus_dump_bridge_status_to_file(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), networkstatus_set_current_consensus(), options_act(), options_act_reversible(), options_get_datadir_fname2_suffix(), options_validate(), or_state_load(), orconn_target_get_name(), policies_parse_exit_policy(), policy_summarize(), policy_write_item(), remove_obsolete_entry_guards(), rend_cache_lookup_entry(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_parse_service_authorization(), rend_service_load_keys(), rep_hist_dump_stats(), rep_hist_fill_bandwidth_history(), rep_hist_get_bandwidth_lines(), rep_hist_get_router_stability_doc(), rep_hist_update_state(), router_dump_router_to_string(), router_load_single_router(), router_parse_addr_policy_item_from_string(), router_reload_consensus_networkstatus(), routerstatus_format_entry(), routerstatus_parse_entry_from_string(), smartlist_string_num_isin(), ssl_state_to_string(), start_writing_to_file(), token_check_object(), tor_addr_to_reverse_lookup_name(), tor_inet_ntoa(), tor_inet_ntop(), tor_init(), tor_listdir(), update_router_have_minimum_dir_info(), vote_routerstatus_find_microdesc_hash(), write_configuration_file(), write_http_response_header_impl(), write_http_status_line(), and write_stream_target_to_buf().

4.10.3.46 int tor_socketpair (int *family*, int *type*, int *protocol*, int *fd*[2])

Allocate a pair of connected sockets. (Like socketpair(family, type,protocol,fd), but works on systems that don't have socketpair.)

Currently, only (AF_UNIX, SOCK_STREAM, 0) sockets are supported.

Note that on systems without socketpair, this call will fail if localhost is inaccessible (for example, if the networking stack is down). And even if it succeeds, the socket pair will not be able to read while localhost is down later (the socket pair may even close, depending on OS-specific timeouts).

Returns 0 on success and -errno on failure; do not rely on the value of errno or WSAGetLastError().

References tor_accept_socket(), tor_close_socket(), and tor_open_socket().

Referenced by spawn_cpuworker().

4.10.3.47 char* tor_strtok_r_impl (char * *str*, const char * *sep*, char ** *lasts*)

Implementation of strtok_r for platforms whose coders haven't figured out how to write one. Hey guys! You can use this code here for free!

References tor_assert.

4.10.3.48 void tor_threads_init (void)

Set up common structures for use by threading.

References set_main_thread().

Referenced by tor_main(), and tor_mutex_init().

4.10.3.49 int tor_vasprintf (char ** *strp*, const char * *fmt*, va_list *args*)

Portable vasprintf implementation. Does a printf() into a newly malloc'd string. Differs from regular vasprintf in the same ways that **tor_asprintf()** (p. 361) differs from regular asprintf.

References tor_free.

Referenced by tor_asprintf().

4.10.3.50 int tor_vsnprintf (char * *str*, size_t *size*, const char * *format*, va_list *args*)

Replacement for vsnprintf; behavior differs as tor_snprintf differs from snprintf.

Referenced by connection_printf_to_buf(), format_msg(), send_control_event_impl(), and tor_snprintf().

4.10.3.51 int touch_file (const char * *fname*)

Change **fname**'s modification time to now.

Referenced by router_set_networkstatus_v2().

4.10.4 Variable Documentation

4.10.4.1 pthread_mutexattr_t attr_reentrant [static]

A mutex attribute that we're going to use to tell pthreads that we want "reentrant" mutexes (i.e., once we can re-lock if we're already holding them.)

4.10.4.2 unsigned long main_thread_id = -1 [static]

Identity of the "main" thread

4.10.4.3 int n_sockets_open = 0 [static]

Count of number of sockets currently open. (Undercounts sockets opened by eventdns and libevent.)

4.10.4.4 tor_mutex_t* socket_accounting_mutex = NULL [static]

Mutex to protect open_sockets, max_socket, and n_sockets_open.

4.10.4.5 int threads_initialized = 0 [static]

True iff we've called **tor_threads_init()** (p. 367)

4.10.4.6 const uint32_t TOR_ISALNUM_TABLE[8]

Initial value:

```
{ 0, 0x3ff0000, 0x7fffffe, 0x7fffffe, 0, 0, 0, 0 }
```

4.10.4.7 const uint32_t TOR_ISALPHA_TABLE[8]

Initial value:

```
{ 0, 0, 0x7fffffe, 0x7fffffe, 0, 0, 0, 0 }
```

4.10.4.8 const uint32_t TOR_ISPRINT_TABLE[8]

Initial value:

```
{ 0, 0xffffffff, 0xffffffff, 0x7fffffff, 0, 0, 0, 0x0 }
```

4.10.4.9 const uint32_t TOR_ISXDIGIT_TABLE[8]

Initial value:

```
{ 0, 0x3ff0000, 0x7e, 0x7e, 0, 0, 0, 0 }
```

4.10.4.10 const char TOR_TOLOWER_TABLE[256]

Initial value:

```
 {
  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
  16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
  32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,
  48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,
  64,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,
  112,113,114,115,116,117,118,119,120,121,122,91,92,93,94,95,
  96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,
  112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,
  128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,
  144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,
  160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,
  176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,
  192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,
  208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,
  224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,
  240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,
}
```

4.10.4.11 const char TOR_TOUPPER_TABLE[256]

Initial value:

```
 {
  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
  16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
  32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,
  48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,
  64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,
  80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,
  96,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,
  80,81,82,83,84,85,86,87,88,89,90,123,124,125,126,127,
  128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,
  144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,
  160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,
  176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,
  192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,
  208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,
  224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,
  240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,
}
```

4.10.4.12 char uname_result[256] `[static]`

Hold the result of our call to **uname**.

4.10.4.13 int uname_result_is_set = 0 `[static]`

True iff uname_result is set.

4.11 compat_libevent.c File Reference

Wrappers to handle porting between different versions of libevent.

`#include "orconfig.h"`

`#include "compat.h"`

`#include "compat_libevent.h"`

`#include "util.h"`

`#include "log.h"`

`#include <event.h>`

Defines

- #define **V**(major, minor, patch) (((major) << 24) | ((minor) << 16) | ((patch) << 8))
- #define **V_OLD**(major, minor, patch) V((major), (minor), (patch)-'a'+1)
- #define **LE_OLD** V(0,0,0)
- #define **LE_OTHER** V(0,0,99)

Typedefs

- typedef uint32_t **le_version_t**

Functions

- static **le_version_t** **tor_get_libevent_version** (const char **v_out)
- static void **libevent_logging_callback** (int severity, const char *msg)
- void **configure_libevent_logging** (void)
- void **suppress_libevent_log_msg** (const char *msg)
- struct event * **tor_event_new** (struct event_base *base, int sock, short what, void(*cb)(int, short, void *), void *arg)
- struct event * **tor_evtimer_new** (struct event_base *base, void(*cb)(int, short, void *), void *arg)
- struct event * **tor_evsignal_new** (struct event_base *base, int sig, void(*cb)(int, short, void *), void *arg)
- void **tor_event_free** (struct event *ev)
- void **tor_libevent_initialize** (void)
- struct event_base * **tor_libevent_get_base** (void)
- const char * **tor_libevent_get_method** (void)
- static **le_version_t** **tor_decode_libevent_version** (const char *v)
- static int **le_versions_compatibility** (**le_version_t** v)
- const char * **tor_libevent_get_version_str** (void)
- void **tor_check_libevent_version** (const char *m, int server, const char **badness_out)
- void **tor_check_libevent_header_compatibility** (void)

Variables

- static const char * **suppress_msg** = NULL
- struct event_base * **the_event_base** = NULL

4.11.1 Detailed Description

Wrappers to handle porting between different versions of libevent.

In an ideal world, we'd just use Libevent 2.0 from now on. But as of June 2009, Libevent 2.0 is still in alpha, and we will have old versions of Libevent for the forseeable future.

4.11.2 Typedef Documentation

4.11.2.1 typedef uint32_t le_version_t

A number representing a version of Libevent.

This is a 4-byte number, with the first three bytes representing the major, minor, and patchlevel respectively of the library. The fourth byte is unused.

This is equivalent to the format of LIBEVENT_VERSION_NUMBER on Libevent 2.0.1 or later. For versions of Libevent before 1.4.0, which followed the format of "1.0, 1.0a, 1.0b", we define 1.0 to be equivalent to 1.0.0, 1.0a to be equivalent to 1.0.1, and so on.

4.11.3 Function Documentation

4.11.3.1 void configure_libevent_logging (void)

Set hook to intercept log messages from libevent.

References libevent_logging_callback().

Referenced by init_libevent().

4.11.3.2 static int le_versions_compatibility (le_version_t *v*) `[static]`

Return an integer representing the binary interface of a Libevent library. Two different versions with different numbers are sure not to be binary compatible. Two different versions with the same numbers have a decent chance of binary compatibility.

References V.

Referenced by tor_check_libevent_header_compatibility().

4.11.3.3 static void libevent_logging_callback (int *severity*, const char ∗ *msg*) `[static]`

Callback function passed to event_set_log() so we can intercept log messages from libevent.

References LD_GENERAL, LD_NET, and suppress_msg.

Referenced by configure_libevent_logging().

4.11.3.4 void suppress_libevent_log_msg (const char ∗ *msg*)

Ignore any libevent log message that contains **msg**.

References suppress_msg.

Referenced by init_libevent().

4.11.3.5 void tor_check_libevent_header_compatibility (void)

See whether the headers we were built against differ from the library we linked against so much that we're likely to crash. If so, warn the user.

References LD_GENERAL, le_versions_compatibility(), and tor_decode_libevent_version().

Referenced by init_libevent().

4.11.3.6 void tor_check_libevent_version (const char ∗ *m*, int *server*, const char ∗∗ *badness_out*)

Compare the current Libevent method and version to a list of versions which are known not to work. Warn the user as appropriate.

References LD_GENERAL, tor_get_libevent_version(), and V.

Referenced by init_libevent().

4.11.3.7 static le_version_t tor_decode_libevent_version (const char ∗ *v*) `[static]`

Return the le_version_t for the current version of libevent. If the version is very new, return LE_-OTHER. If the version is so old that it doesn't support event_get_version(), return LE_OLD. DOCDOC

References V.

Referenced by tor_check_libevent_header_compatibility(), and tor_get_libevent_version().

4.11.3.8 void tor_event_free (struct event ∗ *ev*)

Work-alike replacement for event_free() on pre-Libevent-2.0 systems.

References tor_free.

4.11.3.9 struct event∗ tor_event_new (struct event_base ∗ *base*, int *sock*, short *what*, void(∗)(int, short, void ∗) *cb*, void ∗ *arg*) `[read]`

Work-alike replacement for event_new() on pre-Libevent-2.0 systems.

References tor_libevent_get_base().

Referenced by connection_add(), tor_evsignal_new(), and tor_evtimer_new().

4.11.3.10 struct event∗ tor_evsignal_new (struct event_base ∗ *base*, int *sig*, void(∗)(int, short, void ∗) *cb*, void ∗ *arg*) `[read]`

Work-alike replacement for evsignal_new() on pre-Libevent-2.0 systems.

References tor_event_new().

Referenced by handle_signals().

4.11.3.11 struct event∗ tor_evtimer_new (struct event_base ∗ *base*, void(∗)(int, short, void ∗) *cb*, void ∗ *arg*) `[read]`

Work-alike replacement for evtimer_new() on pre-Libevent-2.0 systems.

References tor_event_new().

Referenced by dns_launch_correctness_checks(), and second_elapsed_callback().

4.11.3.12 static le_version_t tor_get_libevent_version (const char ∗∗ *v_out*) `[static]`

Return the version number of the currently running version of Libevent. See le_version_t for info on the format.

References tor_decode_libevent_version().

Referenced by tor_check_libevent_version(), and tor_libevent_initialize().

4.11.3.13 struct event_base∗ tor_libevent_get_base (void) `[read]`

Return the current Libevent event base that we're set up to use.

Referenced by configure_nameservers(), connection_add(), connection_start_reading_from_linked_conn(), dns_launch_correctness_checks(), dns_reset(), do_main_loop(), handle_signals(), second_elapsed_callback(), and tor_event_new().

4.11.3.14 const char∗ tor_libevent_get_method (void)

Return the name of the Libevent backend we're using.

Referenced by do_main_loop(), init_libevent(), and tor_libevent_initialize().

4.11.3.15 const char∗ tor_libevent_get_version_str (void)

Return a string representation of the version of the currently running version of Libevent.

Referenced by init_libevent().

4.11.3.16 void tor_libevent_initialize (void)

Initialize the Libevent library and set up the event base.

References LD_GENERAL, tor_assert, tor_get_libevent_version(), and tor_libevent_get_method().

Referenced by init_libevent().

4.11.4 Variable Documentation

4.11.4.1 const char∗ suppress_msg = NULL `[static]`

A string which, if it appears in a libevent log, should be ignored.

Referenced by libevent_logging_callback(), and suppress_libevent_log_msg().

4.11.4.2 struct event_base* the_event_base = NULL

Global event base for use by the main thread.

4.12 config.c File Reference

Code to parse and interpret configuration files.

`#include "or.h"`

Data Structures

- struct **config_abbrev_t**
- struct **config_var_t**
- struct **config_var_description_t**
- struct **config_format_t**
- struct **unit_table_t**

Defines

- #define **CONFIG_PRIVATE**
- #define **PLURAL**(tok) { #tok, #tok "s", 0, 0 }
- #define **VAR**(name, conftype, member, initvalue)
- #define **V**(member, conftype, initvalue) VAR(#member, conftype, member, initvalue)
- #define **OBSOLETE**(name) { name, CONFIG_TYPE_OBSOLETE, 0, NULL }
- #define **VAR**(name, conftype, member, initvalue)
- #define **CHECK**(fmt, cfg)
- #define **OR_OPTIONS_MAGIC** 9090909
- #define **OR_STATE_MAGIC** 0x57A73f57
- #define **MIN_REND_POST_PERIOD** (10*60)
- #define **MAX_DIR_PERIOD** (MIN_ONION_KEY_LIFETIME/2)
- #define **MIN_MAX_CIRCUIT_DIRTINESS** 10
- #define **MIN_CIRCUIT_STREAM_TIMEOUT** 10
- #define **REJECT**(arg) STMT_BEGIN *msg = tor_strdup(arg); return -1; STMT_END
- #define **COMPLAIN**(arg) STMT_BEGIN log(LOG_WARN, LD_CONFIG, arg); STMT_END
- #define **GENERATED_FILE_PREFIX**
- #define **GENERATED_FILE_COMMENT**
- #define **VERY_OLD_FILE_AGE** (28*24*60*60)

Typedefs

- typedef int(* **validate_fn_t**)(void *, void *, int, char **)

Enumerations

- enum **config_type_t** {

  CONFIG_TYPE_STRING = 0, **CONFIG_TYPE_FILENAME**, **CONFIG_TYPE_UINT**, **CONFIG_TYPE_INTERVAL**,

  CONFIG_TYPE_MEMUNIT, **CONFIG_TYPE_DOUBLE**, **CONFIG_TYPE_BOOL**, **CONFIG_TYPE_ISOTIME**,

  CONFIG_TYPE_CSV, **CONFIG_TYPE_LINELIST**, **CONFIG_TYPE_LINELIST_S**, **CONFIG_TYPE_LINELIST_V**,

  CONFIG_TYPE_ROUTERSET, **CONFIG_TYPE_OBSOLETE** }

Functions

- static void **config_line_append** (**config_line_t** **lst, const char *key, const char *val)
- static void **option_clear** (**config_format_t** *fmt, **or_options_t** *options, **config_var_t** *var)
- static void **option_reset** (**config_format_t** *fmt, **or_options_t** *options, **config_var_t** *var, int use_defaults)
- static void **config_free** (**config_format_t** *fmt, void *options)
- static int **config_lines_eq** (**config_line_t** *a, **config_line_t** *b)
- static int **option_is_same** (**config_format_t** *fmt, **or_options_t** *o1, **or_options_t** *o2, const char *name)
- static **or_options_t** * **options_dup** (**config_format_t** *fmt, **or_options_t** *old)
- static int **options_validate** (**or_options_t** *old_options, **or_options_t** *options, int from_setconf, char **msg)
- static int **options_act_reversible** (**or_options_t** *old_options, char **msg)
- static int **options_act** (**or_options_t** *old_options)
- static int **options_transition_allowed** (**or_options_t** *old, **or_options_t** *new, char **msg)
- static int **options_transition_affects_workers** (**or_options_t** *old_options, **or_options_t** *new_options)
- static int **options_transition_affects_descriptor** (**or_options_t** *old_options, **or_options_t** *new_options)
- static int **check_nickname_list** (const char *lst, const char *name, char **msg)
- static void **config_register_addressmaps** (**or_options_t** *options)
- static int **parse_bridge_line** (const char *line, int validate_only)
- static int **parse_dir_server_line** (const char *line, **authority_type_t** required_type, int validate_only)
- static int **validate_data_directory** (**or_options_t** *options)
- static int **write_configuration_file** (const char *fname, **or_options_t** *options)
- static **config_line_t** * **get_assigned_option** (**config_format_t** *fmt, void *options, const char *key, int escape_val)
- static void **config_init** (**config_format_t** *fmt, void *options)
- static int **or_state_validate** (**or_state_t** *old_options, **or_state_t** *options, int from_setconf, char **msg)
- static int **or_state_load** (void)
- static int **options_init_logs** (**or_options_t** *options, int validate_only)
- static int **is_listening_on_low_port** (uint16_t port_option, const **config_line_t** *listen_options)
- static uint64_t **config_parse_memunit** (const char *s, int *ok)
- static int **config_parse_interval** (const char *s, int *ok)
- static void **init_libevent** (void)
- static int **opt_streq** (const char *s1, const char *s2)
- const char * **get_dirportfrontpage** (void)
- static void * **config_alloc** (**config_format_t** *fmt)
- **or_options_t** * **get_options** (void)
- int **set_options** (**or_options_t** *new_val, char **msg)
- const char * **get_version** (void)
- static void **or_options_free** (**or_options_t** *options)
- void **config_free_all** (void)
- const char * **safe_str_client** (const char *address)
- const char * **safe_str** (const char *address)

- const char * **escaped_safe_str_client** (const char *address)
- const char * **escaped_safe_str** (const char *address)
- static void **add_default_trusted_dir_authorities** (**authority_type_t** type)
- static int **validate_dir_authorities** (**or_options_t** *options, **or_options_t** *old_options)
- static int **consider_adding_dir_authorities** (**or_options_t** *options, **or_options_t** *old_options)
- int **options_need_geoip_info** (**or_options_t** *options, const char **reason_out)
- uint32_t **get_effective_bwrate** (**or_options_t** *options)
- uint32_t **get_effective_bwburst** (**or_options_t** *options)
- static const char * **expand_abbrev** (**config_format_t** *fmt, const char *option, int command_line, int warn_obsolete)
- static int **config_get_commandlines** (int argc, char **argv, **config_line_t** **result)
- int **config_get_lines** (const char *string, **config_line_t** **result)
- void **config_free_lines** (**config_line_t** *front)
- static **config_var_t** * **config_find_option** (**config_format_t** *fmt, const char *key)
- static int **config_assign_value** (**config_format_t** *fmt, **or_options_t** *options, **config_line_t** *c, char **msg)
- static int **config_assign_line** (**config_format_t** *fmt, **or_options_t** *options, **config_line_t** *c, int use_defaults, int clear_first, char **msg)
- static void **config_reset_line** (**config_format_t** *fmt, **or_options_t** *options, const char *key, int use_defaults)
- int **option_is_recognized** (const char *key)
- const char * **option_get_canonical_name** (const char *key)
- **config_line_t** * **option_get_assignment** (**or_options_t** *options, const char *key)
- static int **config_value_needs_escape** (const char *value)
- static **config_line_t** * **config_lines_dup** (const **config_line_t** *inp)
- static int **config_assign** (**config_format_t** *fmt, void *options, **config_line_t** *list, int use_defaults, int clear_first, char **msg)
- **setopt_err_t** **options_trial_assign** (**config_line_t** *list, int use_defaults, int clear_first, char **msg)
- static void **print_usage** (void)
- static void **list_torrc_options** (void)
- int **resolve_my_address** (int warn_severity, **or_options_t** *options, uint32_t *addr_out, char **hostname_out)
- int **is_local_addr** (const **tor_addr_t** *addr)
- static char * **get_default_nickname** (void)
- **or_options_t** * **options_new** (void)
- void **options_init** (**or_options_t** *options)
- static char * **config_dump** (**config_format_t** *fmt, void *options, int minimal, int comment_defaults)
- char * **options_dump** (**or_options_t** *options, int minimal)
- static int **validate_ports_csv** (**smartlist_t** *sl, const char *name, char **msg)
- static int **ensure_bandwidth_cap** (uint64_t *value, const char *desc, char **msg)
- static int **compute_publishserverdescriptor** (**or_options_t** *options)
- static const char * **get_default_conf_file** (void)
- static char * **find_torrc_filename** (int argc, char **argv, int *using_default_torrc, int *ignore_missing_torrc)
- static char * **load_torrc_from_disk** (int argc, char **argv)
- int **options_init_from_torrc** (int argc, char **argv)

- **setopt_err_t options_init_from_string** (const char *cf, int command, const char *command_arg, char **msg)
- const char * **get_torrc_fname** (void)
- static int **normalize_data_directory** (**or_options_t** *options)
- int **options_save_current** (void)
- static uint64_t **config_parse_units** (const char *val, struct **unit_table_t** *u, int *ok)
- **or_state_t** * **get_or_state** (void)
- char * **options_get_datadir_fname2_suffix** (**or_options_t** *options, const char *sub1, const char *sub2, const char *suffix)
- static void **or_state_set** (**or_state_t** *new_state)
- int **or_state_save** (time_t now)
- void **remove_file_if_very_old** (const char *fname, time_t now)
- int **getinfo_helper_config** (**control_connection_t** *conn, const char *question, char **answer)

Variables

- static **config_abbrev_t _option_abbrevs** []
- static **config_abbrev_t _state_abbrevs** []
- static **config_var_t _option_vars** []
- static **config_var_t testing_tor_network_defaults** []
- static **config_var_t _state_vars** []
- static **config_format_t options_format**
- static **config_var_t state_extra_var**
- static **config_format_t state_format**
- static **or_options_t** * **global_options** = NULL
- static char * **torrc_fname** = NULL
- static **or_state_t** * **global_state** = NULL
- static **config_line_t** * **global_cmdline_options** = NULL
- static char * **global_dirfrontpagecontents** = NULL
- const char **tor_git_revision** []
- static char * **_version** = NULL
- static uint32_t **last_resolved_addr** = 0
- static struct **unit_table_t memory_units** []
- static struct **unit_table_t time_units** []

4.12.1 Detailed Description

Code to parse and interpret configuration files.

4.12.2 Define Documentation

4.12.2.1 #define CHECK(fmt, cfg)

Value:

```
STMT_BEGIN                                        \
    tor_assert(fmt && cfg);                                          \
    tor_assert((fmt)->magic ==                                       \
               *(uint32_t*)STRUCT_VAR_P(cfg,fmt->magic_offset));        \
  STMT_END
```

Macro: assert that **cfg** has the right magic field for format **fmt**.

Referenced by config_alloc(), config_assign(), config_assign_line(), config_assign_value(), config_init(), config_reset_line(), get_assigned_option(), option_is_same(), and option_reset().

4.12.2.2 #define GENERATED_FILE_COMMENT

Value:

```
"# The old torrc file was renamed " \
  "to torrc.orig.1 or similar, and Tor will ignore it"
```

This string can change; it tries to give the reader an idea that editing this file by hand is not a good plan.

Referenced by write_configuration_file().

4.12.2.3 #define GENERATED_FILE_PREFIX

Value:

```
"# This file was generated by Tor; " \
  "if you edit it, comments will not be preserved"
```

This string must remain the same forevermore. It is how we recognize that the torrc file doesn't need to be backed up.

Referenced by write_configuration_file().

4.12.2.4 #define MAX_DIR_PERIOD (MIN_ONION_KEY_LIFETIME/2)

Highest allowable value for RendPostPeriod.

4.12.2.5 #define MIN_CIRCUIT_STREAM_TIMEOUT 10

Lowest allowable value for CircuitStreamTimeout; if this is too low, Tor will generate too many circuits and potentially overload the network.

4.12.2.6 #define MIN_MAX_CIRCUIT_DIRTINESS 10

Lowest allowable value for MaxCircuitDirtiness; if this is too low, Tor will generate too many circuits and potentially overload the network.

4.12.2.7 #define MIN_REND_POST_PERIOD (10*60)

Lowest allowable value for RendPostPeriod; if this is too low, hidden services can overload the directory system.

4.12.2.8 #define OBSOLETE(name) { name, CONFIG_TYPE_OBSOLETE, 0, NULL }

An entry for config_vars: "The option name is obsolete."

4.12.2.9 #define OR_OPTIONS_MAGIC 9090909

Magic value for **or_options_t** (p. 148).

Referenced by options_init_from_string().

4.12.2.10 #define OR_STATE_MAGIC 0x57A73f57

Magic value for **or_state_t** (p. 178).

Referenced by or_state_load().

4.12.2.11 #define V(member, conftype, initvalue) VAR(#member, conftype, member, initvalue)

As VAR, but the option name and member name are the same.

Referenced by le_versions_compatibility(), tor_check_libevent_version(), and tor_decode_libevent_version().

4.12.2.12 #define VAR(name, conftype, member, initvalue)

Value:

```
{ name, CONFIG_TYPE_ ## conftype, STRUCT_OFFSET(or_state_t, member),  \
      initvalue }
```

An entry for config_vars: "The option **name** has type CONFIG_TYPE_**conftype**, and corresponds to **or_options_t** (p. 148).**member**"

4.12.2.13 #define VAR(name, conftype, member, initvalue)

Value:

```
{ name, CONFIG_TYPE_ ## conftype, STRUCT_OFFSET(or_options_t, member), \
      initvalue }
```

An entry for config_vars: "The option **name** has type CONFIG_TYPE_**conftype**, and corresponds to **or_options_t** (p. 148).**member**"

4.12.3 Typedef Documentation

4.12.3.1 typedef int(* validate_fn_t)(void *, void *, int, char **)

Type of a callback to validate whether a given configuration is well-formed and consistent. See **options_trial_assign()** (p. 397) for documentation of arguments.

4.12.4 Enumeration Type Documentation

4.12.4.1 enum config_type_t

Enumeration of types which option values can take

Enumerator:

- ***CONFIG_TYPE_STRING*** An arbitrary string.
- ***CONFIG_TYPE_FILENAME*** A filename: some prefixes get expanded.
- ***CONFIG_TYPE_UINT*** A non-negative integer less than MAX_INT
- ***CONFIG_TYPE_INTERVAL*** A number of seconds, with optional units
- ***CONFIG_TYPE_MEMUNIT*** A number of bytes, with optional units
- ***CONFIG_TYPE_DOUBLE*** A floating-point value
- ***CONFIG_TYPE_BOOL*** A boolean value, expressed as 0 or 1.
- ***CONFIG_TYPE_ISOTIME*** An ISO-formatted time relative to GMT.
- ***CONFIG_TYPE_CSV*** A list of strings, separated by commas and optional whitespace.
- ***CONFIG_TYPE_LINELIST*** Uninterpreted config lines
- ***CONFIG_TYPE_LINELIST_S*** Uninterpreted, context-sensitive config lines, mixed with other keywords.
- ***CONFIG_TYPE_LINELIST_V*** Catch-all "virtual" option to summarize context-sensitive config lines when fetching.
- ***CONFIG_TYPE_ROUTERSET*** A list of router names, addrs, and fps, parsed into a **routerset_t** (p. 216).
- ***CONFIG_TYPE_OBSOLETE*** Obsolete (ignored) option.

4.12.5 Function Documentation

4.12.5.1 static void add_default_trusted_dir_authorities (authority_type_t *type*) `[static]`

Add the default directory authorities directly into the trusted dir list, but only add them insofar as they share bits with **type**.

References LD_BUG, and parse_dir_server_line().

Referenced by consider_adding_dir_authorities().

4.12.5.2 static int check_nickname_list (const char ∗ *lst*, const char ∗ *name*, char ∗∗ *msg*) `[static]`

Verify whether lst is a string containing valid-looking comma-separated nicknames, or NULL. Return 0 on success. Warn and return -1 on failure.

References is_legal_nickname_or_hexdigest(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_asprintf(), and tor_free.

4.12.5.3 static int compute_publishserverdescriptor (or_options_t ∗ *options*) [static]

Parse an authority type from **options**->PublishServerDescriptor and write it to **options**->_PublishServerDescriptor. Treat "1" as "v2,v3" unless BridgeRelay is 1, in which case treat it as "bridge". Treat "0" as "". Return 0 on success or -1 if not a recognized authority type (in which case the value of _PublishServerDescriptor is undefined).

References or_options_t::_PublishServerDescriptor, BRIDGE_AUTHORITY, or_options_t::BridgeRelay, HIDSERV_AUTHORITY, or_options_t::PublishServerDescriptor, V1_AUTHORITY, V2_AUTHORITY, and V3_AUTHORITY.

4.12.5.4 static void∗ config_alloc (config_format_t ∗ *fmt*) [static]

Allocate an empty configuration object of a given format type.

References CHECK, config_format_t::magic, config_format_t::magic_offset, config_format_t::size, and STRUCT_VAR_P.

Referenced by config_dump(), options_dup(), and options_new().

4.12.5.5 static int config_assign (config_format_t ∗ *fmt*, void ∗ *options*, config_line_t ∗ *list*, int *use_defaults*, int *clear_first*, char ∗∗ *msg*) [static]

Iterate through the linked list of requested options **list**. For each item, convert as appropriate and assign to **options**. If an item is unrecognized, set ∗msg and return -1 immediately, else return 0 for success.

If **clear_first**, interpret config options as replacing (not extending) their previous values. If **clear_first** is set, then **use_defaults** to decide if you set to defaults after clearing, or make the value 0 or NULL.

Here are the use cases: 1. A non-empty AllowInvalid line in your torrc. Appends to current if linelist, replaces current if csv. 2. An empty AllowInvalid line in your torrc. Should clear it. 3. "RESETCONF AllowInvalid" sets it to default. 4. "SETCONF AllowInvalid" makes it NULL. 5. "SETCONF AllowInvalid=foo" clears it and sets it to "foo".

Use_defaults Clear_first 0 0 "append" 1 0 undefined, don't use 0 1 "set to null first" 1 1 "set to defaults first" Return 0 on success, -1 on bad key, -2 on bad value.

As an additional special case, if a LINELIST config option has no value and clear_first is 0, then warn and ignore it.

References CHECK, config_assign_line(), config_reset_line(), expand_abbrev(), config_line_t::key, config_line_t::next, and tor_free.

Referenced by options_dup(), options_init_from_string(), options_trial_assign(), and or_state_load().

4.12.5.6 static int config_assign_line (config_format_t ∗ *fmt*, or_options_t ∗ *options*, config_line_t ∗ *c*, int *use_defaults*, int *clear_first*, char ∗∗ *msg*) [static]

If **c** is a syntactically valid configuration line, update **options** with its value and return 0. Otherwise return -1 for bad key, -2 for bad value.

If **clear_first** is set, clear the value first. Then if **use_defaults** is set, set the value to the default.

Called from **config_assign()** (p. 382).

References CHECK, config_assign_value(), config_find_option(), config_line_append(), CONFIG_TYPE_LINELIST, CONFIG_TYPE_LINELIST_S, config_format_t::extra, config_line_t::key, LD_CONFIG, config_var_t::name, option_reset(), STRUCT_VAR_P, tor_asprintf(), tor_free, config_var_t::type, config_line_t::value, and config_var_t::var_offset.

Referenced by config_assign().

4.12.5.7 static int config_assign_value (config_format_t ∗ *fmt*, or_options_t ∗ *options*, config_line_t ∗ *c*, char ∗∗ *msg*) `[static]`

c->key is known to be a real key. Update **options** with **c**->value and return 0, or return -1 if bad value.

Called from **config_assign_line()** (p. 382) and **option_reset()** (p. 393).

References CHECK, config_find_option(), config_line_append(), config_parse_interval(), config_parse_memunit(), CONFIG_TYPE_BOOL, CONFIG_TYPE_CSV, CONFIG_TYPE_DOUBLE, CONFIG_TYPE_FILENAME, CONFIG_TYPE_INTERVAL, CONFIG_TYPE_ISOTIME, CONFIG_TYPE_LINELIST, CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_LINELIST_V, CONFIG_TYPE_MEMUNIT, CONFIG_TYPE_OBSOLETE, CONFIG_TYPE_ROUTERSET, CONFIG_TYPE_STRING, CONFIG_TYPE_UINT, config_line_t::key, LD_CONFIG, parse_iso_time(), routerset_free(), routerset_new(), routerset_parse(), smartlist_clear(), smartlist_create(), smartlist_split_string(), STRUCT_VAR_P, tor_asprintf(), tor_assert, tor_free, tor_parse_long(), config_var_t::type, config_line_t::value, and config_var_t::var_offset.

Referenced by config_assign_line(), and option_reset().

4.12.5.8 static char∗ config_dump (config_format_t ∗ *fmt*, void ∗ *options*, int *minimal*, int *comment_defaults*) `[static]`

Allocate and return a new string holding the written-out values of the vars in 'options'. If 'minimal', do not write out any default-valued vars. Else, if comment_defaults, write default values as comments.

References config_alloc(), config_free(), config_free_lines(), config_init(), CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_OBSOLETE, config_format_t::extra, get_assigned_option(), config_line_t::key, LD_BUG, config_var_t::name, config_line_t::next, option_is_same(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), strcmpstart(), STRUCT_VAR_P, tor_asprintf(), tor_assert, tor_free, config_var_t::type, config_format_t::validate_fn, config_line_t::value, config_var_t::var_offset, and config_format_t::vars.

Referenced by options_dump(), and or_state_save().

4.12.5.9 static config_var_t∗ config_find_option (config_format_t ∗ *fmt*, const char ∗ *key*) `[static]`

If **key** is a configuration option, return the corresponding **config_var_t** (p. 52). Otherwise, if **key** is a non-standard abbreviation, warn, and return the corresponding **config_var_t** (p. 52). Otherwise return NULL.

References LD_CONFIG, config_var_t::name, and config_format_t::vars.

Referenced by config_assign_line(), config_assign_value(), config_reset_line(), get_assigned_option(), option_get_canonical_name(), option_is_recognized(), and options_init_from_string().

4.12.5.10 static void config_free (config_format_t * *fmt*, void * *options*) [static]

Release storage held by **options**.

References config_free_lines(), config_format_t::extra, config_var_t::name, option_clear(), STRUCT_VAR_P, tor_assert, tor_free, config_var_t::var_offset, and config_format_t::vars.

Referenced by config_dump(), config_free_all(), options_init_from_string(), options_trial_assign(), or_options_free(), or_state_load(), or_state_set(), and set_options().

4.12.5.11 void config_free_all (void)

Release all memory and resources held by global configuration structures.

References config_free(), config_free_lines(), or_options_free(), and tor_free.

Referenced by tor_free_all().

4.12.5.12 void config_free_lines (config_line_t * *front*)

Free all the configuration lines on the linked list **front**.

References config_line_t::key, config_line_t::next, tor_free, and config_line_t::value.

Referenced by circuit_build_times_update_state(), config_dump(), config_free(), config_free_all(), config_get_commandlines(), config_get_lines(), control_setconf_helper(), dirserv_load_fingerprint_file(), entry_guards_update_state(), option_clear(), option_is_same(), option_reset(), options_dup(), options_init_from_string(), and or_state_load().

4.12.5.13 static int config_get_commandlines (int *argc*, char ** *argv*, config_line_t ** *result*) [static]

Helper: Read a list of configuration options from the command line. If successful, put them in ***result** and return 0, and return -1 and leave ***result** alone.

References config_free_lines(), expand_abbrev(), and LD_CONFIG.

Referenced by options_init_from_torrc().

4.12.5.14 int config_get_lines (const char * *string*, config_line_t ** *result*)

Helper: parse the config string and strdup into key/value strings. Set *result to the list, or NULL if parsing the string failed. Return 0 on success, -1 on failure. Warn and ignore any misformatted lines.

References config_free_lines(), parse_config_line_from_str(), and tor_free.

Referenced by control_setconf_helper(), dirserv_load_fingerprint_file(), options_init_from_string(), and or_state_load().

4.12.5.15 static void config_init (config_format_t * *fmt*, void * *options*) [static]

Set all vars in the configuration object **options** to their default values.

References CHECK, config_var_t::initvalue, config_var_t::name, option_reset(), and config_format_t::vars.

Referenced by config_dump(), options_init(), and or_state_load().

4.12.5.16 static void config_line_append (config_line_t ** *lst*, const char * *key*, const char * *val*) [static]

Helper: allocate a new configuration option mapping 'key' to 'val', append it to ***lst**.

References config_line_t::key, config_line_t::next, and config_line_t::value.

Referenced by config_assign_line(), config_assign_value(), and options_validate().

4.12.5.17 static config_line_t* config_lines_dup (const config_line_t * *inp*) [static]

Return a newly allocated deep copy of the lines in **inp**.

References config_line_t::key, config_line_t::next, and config_line_t::value.

Referenced by get_assigned_option(), and options_validate().

4.12.5.18 static int config_lines_eq (config_line_t * *a*, config_line_t * *b*) [static]

Return true iff a and b contain identical keys and values in identical order.

References config_line_t::key, config_line_t::next, and config_line_t::value.

Referenced by consider_adding_dir_authorities(), option_is_same(), options_transition_affects_descriptor(), options_transition_affects_workers(), options_validate(), and validate_dir_authorities().

4.12.5.19 static int config_parse_interval (const char * *s*, int * *ok*) [static]

Parse a string in the format "number unit", where unit is a unit of time. On success, set ***ok** to true and return the number of seconds in the provided interval. Otherwise, set ***ok** to 0 and return -1.

References config_parse_units(), and LD_CONFIG.

Referenced by config_assign_value().

4.12.5.20 static uint64_t config_parse_memunit (const char * *s*, int * *ok*) [static]

Parse a string in the format "number unit", where unit is a unit of information (byte, KB, M, etc). On success, set ***ok** to true and return the number of bytes specified. Otherwise, set ***ok** to false and return 0.

References config_parse_units().

Referenced by config_assign_value().

4.12.5.21 static uint64_t config_parse_units (const char * *val*, struct unit_table_t * *u*, int * *ok*) [static]

Parse a string **val** containing a number, zero or more spaces, and an optional unit string. If the unit appears in the table **u**, then multiply the number by the unit multiplier. On success, set ∗**ok** to 1 and return this product. Otherwise, set ∗**ok** to 0.

References eat_whitespace(), LD_CONFIG, unit_table_t::multiplier, tor_assert, tor_parse_double(), tor_parse_uint64(), and unit_table_t::unit.

Referenced by config_parse_interval(), and config_parse_memunit().

4.12.5.22 static void config_register_addressmaps (or_options_t * *options*) [static]

Adjust the address map based on the MapAddress elements in the configuration **options**

References address_is_invalid_destination(), or_options_t::AddressMap, addressmap_clear_configured(), addressmap_register(), ADDRMAPSRC_TORRC, LD_CONFIG, config_line_t::next, smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, and config_line_t::value.

Referenced by options_act().

4.12.5.23 static void config_reset_line (config_format_t * *fmt*, or_options_t * *options*, const char * *key*, int *use_defaults*) [static]

Restore the option named **key** in options to its default value. Called from **config_assign()** (p. 382).

References CHECK, config_find_option(), and option_reset().

Referenced by config_assign().

4.12.5.24 static int config_value_needs_escape (const char * *value*) [static]

Return true iff value needs to be quoted and escaped to be used in a configuration file.

Referenced by get_assigned_option().

4.12.5.25 static int consider_adding_dir_authorities (or_options_t * *options*, or_options_t * *old_options*) [static]

Look at all the config options and assign new dir authorities as appropriate.

References add_default_trusted_dir_authorities(), or_options_t::AlternateBridgeAuthority, or_options_t::AlternateDirAuthority, or_options_t::AlternateHSAuthority, BRIDGE_AUTHORITY, clear_trusted_dir_servers(), config_lines_eq(), or_options_t::DirServers, HIDSERV_AUTHORITY, config_line_t::next, parse_dir_server_line(), router_get_trusted_dir_servers(), V1_AUTHORITY, V2_AUTHORITY, V3_AUTHORITY, and config_line_t::value.

Referenced by options_act().

4.12.5.26 static int ensure_bandwidth_cap (uint64_t ∗ *value*, const char ∗ *desc*, char ∗∗ *msg*) `[static]`

If **value** exceeds ROUTER_MAX_DECLARED_BANDWIDTH, write a complaint into ∗**msg** using string **desc**, and return -1. Else return 0.

References tor_asprintf().

4.12.5.27 const char∗ escaped_safe_str (const char ∗ *address*)

Equivalent to escaped(safe_str(address)). See reentrancy note on **escaped()** (p. 1245): don't use this outside the main thread, or twice in the same log statement.

References escaped(), and get_options().

Referenced by connection_dns_remove(), connection_edge_finished_connecting(), connection_tls_start_handshake(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), evdns_callback(), launch_resolve(), launch_test_addresses(), and purge_expired_resolves().

4.12.5.28 const char∗ escaped_safe_str_client (const char ∗ *address*)

Equivalent to escaped(safe_str_client(address)). See reentrancy note on **escaped()** (p. 1245): don't use this outside the main thread, or twice in the same log statement.

References escaped(), and get_options().

Referenced by addressmap_rewrite(), addressmap_rewrite_reverse(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), conn_close_if_marked(), connection_ap_handshake_rewrite_and_attach(), connection_connect(), connection_exit_connect(), directory_get_from_hs_dir(), dnsserv_launch_request(), evdns_server_callback(), rend_client_introduction_acked(), rend_client_remove_intro_point(), rend_client_send_introduction(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.12.5.29 static const char∗ expand_abbrev (config_format_t ∗ *fmt*, const char ∗ *option*, int *command_line*, int *warn_obsolete*) `[static]`

If **option** is an official abbreviation for a longer option, return the longer option. Otherwise return **option**. If **command_line** is set, apply all abbreviations. Otherwise, only apply abbreviations that work for the config file and the command line. If **warn_obsolete** is set, warn about deprecated names.

References config_abbrev_t::abbreviated, config_format_t::abbrevs, config_abbrev_t::commandline_only, config_abbrev_t::full, LD_CONFIG, and config_abbrev_t::warn.

Referenced by config_assign(), and config_get_commandlines().

4.12.5.30 static char∗ find_torrc_filename (int *argc*, char ∗∗ *argv*, int ∗ *using_default_torrc*, int ∗ *ignore_missing_torrc*) `[static]`

Learn config file name from command line arguments, or use the default

References expand_filename(), file_status(), get_default_conf_file(), LD_CONFIG, and tor_free.

Referenced by load_torrc_from_disk().

4.12.5.31 static config_line_t ∗ get_assigned_option (config_format_t ∗ *fmt*, void ∗ *options*, const char ∗ *key*, int *escape_val*) [static]

Return newly allocated line or lines corresponding to **key** in the configuration **options**. If **escape_val** is true and a value needs to be quoted before it's put in a config file, quote and escape that value. Return NULL if no such key exists.

References CHECK, config_find_option(), config_lines_dup(), CONFIG_TYPE_BOOL, CONFIG_TYPE_CSV, CONFIG_TYPE_DOUBLE, CONFIG_TYPE_FILENAME, CONFIG_TYPE_INTERVAL, CONFIG_TYPE_ISOTIME, CONFIG_TYPE_LINELIST, CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_LINELIST_V, CONFIG_TYPE_MEMUNIT, CONFIG_TYPE_OBSOLETE, CONFIG_TYPE_ROUTERSET, CONFIG_TYPE_STRING, CONFIG_TYPE_UINT, config_value_needs_escape(), esc_for_log(), format_iso_time(), config_line_t::key, LD_BUG, LD_CONFIG, config_var_t::name, config_line_t::next, routerset_to_string(), smartlist_join_strings(), STRUCT_VAR_P, tor_asprintf(), tor_assert, tor_free, config_var_t::type, config_line_t::value, and config_var_t::var_offset.

Referenced by config_dump(), option_get_assignment(), option_is_same(), and options_dup().

4.12.5.32 static const char∗ get_default_conf_file (void) [static]

Return the default location for our torrc file.

Referenced by find_torrc_filename(), and get_torrc_fname().

4.12.5.33 static char∗ get_default_nickname (void) [static]

Called when we don't have a nickname set. Try to guess a good nickname based on the hostname, and return it in a newly allocated string. If we can't, return NULL and let the caller warn if it wants to.

References LEGAL_NICKNAME_CHARACTERS, MAX_NICKNAME_LEN, tor_free, and tor_strlower().

Referenced by options_validate().

4.12.5.34 const char∗ get_dirportfrontpage (void)

Return the contents of our frontpage string, or NULL if not configured.

Referenced by directory_handle_command_get().

4.12.5.35 uint32_t get_effective_bwburst (or_options_t ∗ *options*)

Return the bandwidthburst that we are going to report to the authorities based on the config options.

References or_options_t::BandwidthBurst, and or_options_t::RelayBandwidthBurst.

Referenced by options_transition_affects_descriptor(), and router_rebuild_descriptor().

4.12.5.36 uint32_t get_effective_bwrate (or_options_t * *options*)

Return the bandwidthrate that we are going to report to the authorities based on the config options.

References or_options_t::BandwidthRate, or_options_t::MaxAdvertisedBandwidth, and or_options_t::RelayBandwidthRate.

Referenced by options_transition_affects_descriptor(), and router_rebuild_descriptor().

4.12.5.37 or_options_t* get_options (void)

Return the currently configured options.

References tor_assert.

Referenced by accounting_record_bandwidth_usage(), accounting_set_wakeup_time(), add_networkstatus_to_cache(), add_trusted_dir_server(), add_wildcarded_test_address(), address_is_invalid_destination(), any_bridge_descriptors_known(), authority_certs_fetch_missing(), cell_queue_append_packed_copy(), check_descriptor_ipaddress_changed(), check_whether_dirport_reachable(), check_whether_orport_reachable(), choose_good_entry_server(), choose_good_exit_server(), choose_good_exit_server_general(), choose_good_middle_server(), choose_random_entry(), circuit_all_predicted_ports_handled(), circuit_build_needed_circs(), circuit_build_times_add_time(), circuit_build_times_get_initial_timeout(), circuit_build_times_update_state(), circuit_expire_all_dirty_circs(), circuit_expire_old_circuits_clientside(), circuit_extend(), circuit_free(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_note_clock_jumped(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), circuit_stream_is_being_handled(), circuit_testing_failed(), command_process_create_cell(), compute_retry_timeout(), configure_nameservers(), conn_close_if_marked(), connection_about_to_close_connection(), connection_ap_can_use_exit(), connection_ap_detach_retriable(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_rewrite_and_attach(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_bucket_init(), connection_bucket_refill(), connection_connect(), connection_create_listener(), connection_dir_client_reached_eof(), connection_dir_request_failed(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_handle_listener_read(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_connect_failed(), connection_or_finished_connecting(), connection_or_flush_from_first_active_circuit(), connection_or_init_conn_from_address(), connection_proxy_connect(), connection_read_proxy_handshake(), consider_hibernation(), consider_plaintext_ports(), consider_publishable_server(), consider_recording_trackhost(), control_event_bootstrap_problem(), control_event_guard_deferred(), count_loading_descriptors_progress(), cpuworkers_rotate(), decide_if_publishable_server(), dir_routerdesc_download_failed(), directories_have_accepted_server_descriptor(), directory_get_from_dirserver(), directory_handle_command_get(), directory_handle_command_post(), directory_info_has_arrived(), directory_initiate_command_rend(), directory_send_command(), dirserv_dump_directory_to_string(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_status_impl(), dirserv_load_fingerprint_file(), dirserv_orconn_tls_done(), dirserv_pick_cached_dir_obj(), dirserv_router_has_valid_address(), dirserv_set_router_is_running(), dirserv_test_reachability(), dirserv_thinks_router_is_hs_dir(), dirvote_compute_consensuses(), dirvote_get_preferred_voting_intervals(), dns_init(), dns_launch_correctness_checks(), dns_reset(), dns_servers_relaunch_checks(), do_hash_password(), do_hup(), do_list_fingerprint(), do_main_loop(), download_status_reset(), entry_guards_-

changed(), entry_guards_compute_status(), entry_guards_prepend_from_config(), entry_guards_update_state(), entry_is_live(), escaped_safe_str(), escaped_safe_str_client(), evdns_log_cb(), extrainfo_dump_to_string(), fetch_bridge_descriptors(), generate_runningrouters(), generate_v2_networkstatus_opinion(), geoip_change_dirreq_state(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), geoip_note_client_seen(), geoip_note_ns_response(), geoip_start_dirreq(), get_cookie_file(), get_possible_sybil_list(), getinfo_helper_accounting(), getinfo_helper_events(), getinfo_helper_misc(), global_write_bucket_low(), handle_control_attachstream(), handle_control_authenticate(), handle_control_getconf(), handle_control_protocolinfo(), hibernate_begin(), hibernate_go_dormant(), hibernate_hard_limit_reached(), hibernate_soft_limit_reached(), hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), init_cookie_authentication(), init_key_from_file(), init_keys(), init_libevent(), init_v3_authority_keys(), ip_address_changed(), is_local_addr(), is_test_address(), launch_resolve(), launch_router_descriptor_downloads(), launch_test_addresses(), learned_bridge_descriptor(), list_server_status_v1(), networkstatus_dump_bridge_status_to_file(), networkstatus_getinfo_by_purpose(), networkstatus_set_current_consensus(), networkstatus_v2_list_clean(), onion_pending_add(), options_act(), options_act_reversible(), options_save_current(), options_trial_assign(), or_state_save(), parse_reachable_addresses(), pick_entry_guards(), purpose_needs_anonymity(), refresh_all_country_info(), rend_client_refetch_v2_renddesc(), rend_consider_descriptor_republication(), rend_consider_services_upload(), rend_service_requires_uptime(), rend_services_introduce(), rep_hist_circbuilding_dormant(), rep_hist_dump_stats(), rep_hist_fill_bandwidth_history(), rep_hist_note_exit_bytes_read(), rep_hist_note_exit_bytes_written(), rep_hist_note_exit_stream_opened(), rep_hist_update_state(), rep_history_clean(), retry_all_listeners(), rotate_onion_key(), router_add_to_routerlist(), router_choose_random_node(), router_digest_is_trusted_dir_type(), router_dirport_found_reachable(), router_dump_router_to_string(), router_get_by_nickname(), router_get_my_extrainfo(), router_get_my_routerinfo(), router_new_address_suggestion(), router_orport_found_reachable(), router_parse_directory(), router_parse_runningrouters(), router_perform_bandwidth_test(), router_pick_directory_server(), router_pick_trusteddirserver(), router_rebuild_descriptor(), router_reload_consensus_networkstatus(), router_reload_v2_networkstatus(), router_set_networkstatus_v2(), router_upload_dir_desc_to_dirservers(), routerlist_add_family(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), routers_in_same_family(), routers_update_all_from_networkstatus(), routers_update_status_from_consensus_networkstatus(), run_connection_housekeeping(), run_scheduled_events(), safe_str(), safe_str_client(), second_elapsed_callback(), should_generate_v2_networkstatus(), should_use_create_fast_for_circuit(), signal_callback(), signed_desc_digest_is_recognized(), spawn_enough_cpuworkers(), tor_cleanup(), tor_init(), tor_main(), trusted_dirs_load_certs_from_string(), update_consensus_networkstatus_downloads(), update_consensus_networkstatus_fetch_time(), update_consensus_router_descriptor_downloads(), update_expected_bandwidth(), update_extrainfo_downloads(), update_networkstatus_downloads(), update_router_descriptor_cache_downloads_v2(), update_router_descriptor_downloads(), update_router_have_minimum_dir_info(), update_v2_networkstatus_cache_downloads(), upload_service_descriptor(), v3_authority_check_key_expiry(), and warn_if_last_router_excluded().

4.12.5.38 or_state_t* get_or_state (void)

Return the persistent state struct for this Tor.

References tor_assert.

Referenced by accounting_run_housekeeping(), circuit_build_times_add_time(), circuit_build_times_update_state(), do_hup(), entry_guards_changed(), entry_guards_update_state(), hibernate_begin(), hibernate_go_dormant(), init_keys(), read_bandwidth_usage(), rep_hist_update_state(), rotate_onion_key(), and tor_cleanup().

4.12.5.39 const char* get_torrc_fname (void)

Return the location for our configuration file.

References get_default_conf_file().

Referenced by getinfo_helper_misc(), options_save_current(), and parse_dir_server_line().

4.12.5.40 const char* get_version (void)

Return the current Tor version.

References tor_snprintf().

Referenced by get_platform_str(), getinfo_helper_misc(), options_init_from_torrc(), or_state_save(), and tor_init().

4.12.5.41 int getinfo_helper_config (control_connection_t * *conn*, const char * *question*, char ** *answer*)

Helper to implement GETINFO functions about configuration variables (not their values). Given a "config/names" question, set ***answer** to a new string describing the supported configuration variables and their types.

References CONFIG_TYPE_BOOL, CONFIG_TYPE_CSV, CONFIG_TYPE_DOUBLE, CONFIG_TYPE_FILENAME, CONFIG_TYPE_INTERVAL, CONFIG_TYPE_ISOTIME, CONFIG_TYPE_LINELIST, CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_LINELIST_V, CONFIG_TYPE_MEMUNIT, CONFIG_TYPE_OBSOLETE, CONFIG_TYPE_ROUTERSET, CONFIG_TYPE_STRING, CONFIG_TYPE_UINT, config_var_t::name, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_asprintf(), tor_free, and config_var_t::type.

4.12.5.42 static void init_libevent (void) `[static]`

Initialize the libevent library.

References configure_libevent_logging(), control_event_general_status(), get_options(), suppress_libevent_log_msg(), tor_check_libevent_header_compatibility(), tor_check_libevent_version(), tor_libevent_get_method(), tor_libevent_get_version_str(), and tor_libevent_initialize().

Referenced by options_act_reversible().

4.12.5.43 int is_local_addr (const tor_addr_t * *addr*)

Return true iff **addr** is judged to be on the same network as us, or on a private network.

References get_options(), tor_addr_family(), tor_addr_is_internal(), and tor_addr_to_ipv4h().

Referenced by onionskin_answer(), and write_http_response_header_impl().

4.12.5.44 static void list_torrc_options (void) `[static]`

Print all non-obsolete torrc options.

References CONFIG_TYPE_LINELIST_V, CONFIG_TYPE_OBSOLETE, config_var_t::name, smartlist_create(), smartlist_free(), and config_var_t::type.

Referenced by options_init_from_torrc().

4.12.5.45 static char∗ load_torrc_from_disk (int *argc*, char ∗∗ *argv*) `[static]`

Load torrc from disk, setting torrc_fname if successful

References file_status(), find_torrc_filename(), LD_CONFIG, read_file_to_str(), tor_assert, and tor_free.

Referenced by options_init_from_torrc().

4.12.5.46 static int normalize_data_directory (or_options_t ∗ *options*) `[static]`

Adjust the value of options->DataDirectory, or fill it in if it's absent. Return 0 on success, -1 on failure.

References or_options_t::DataDirectory, expand_filename(), LD_CONFIG, and tor_free.

Referenced by validate_data_directory().

4.12.5.47 static int opt_streq (const char ∗ *s1*, const char ∗ *s2*) `[static]`

Helper: return true iff s1 and s2 are both NULL, or both non-NULL equal strings.

Referenced by options_act(), options_transition_affects_descriptor(), options_transition_affects_workers(), and options_transition_allowed().

4.12.5.48 static void option_clear (config_format_t ∗ *fmt*, or_options_t ∗ *options*, config_var_t ∗ *var*) `[static]`

Reset config option **var** to 0, 0.0, NULL, or the equivalent. Called from **option_reset()** (p. 393) and **config_free()** (p. 384).

References config_free_lines(), CONFIG_TYPE_BOOL, CONFIG_TYPE_CSV, CONFIG_TYPE_DOUBLE, CONFIG_TYPE_FILENAME, CONFIG_TYPE_INTERVAL, CONFIG_TYPE_ISOTIME, CONFIG_TYPE_LINELIST, CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_LINELIST_V, CONFIG_TYPE_MEMUNIT, CONFIG_TYPE_OBSOLETE, CONFIG_TYPE_ROUTERSET, CONFIG_TYPE_STRING, CONFIG_TYPE_UINT, routerset_free(), smartlist_free(), STRUCT_VAR_P, tor_free, config_var_t::type, and config_var_t::var_offset.

Referenced by config_free(), and option_reset().

4.12.5.49 config_line_t∗ option_get_assignment (or_options_t ∗ *options*, const char ∗ *key*)

Return a canonical list of the options assigned for key.

References get_assigned_option().

Referenced by handle_control_getconf().

4.12.5.50 const char* option_get_canonical_name (const char * *key*)

Return the canonical name of a configuration option, or NULL if no such option exists.

References config_find_option(), and config_var_t::name.

Referenced by handle_control_getconf().

4.12.5.51 int option_is_recognized (const char * *key*)

Return true iff key is a valid configuration option.

References config_find_option().

Referenced by handle_control_getconf().

4.12.5.52 static int option_is_same (config_format_t * *fmt*, or_options_t * *o1*, or_options_t * *o2*, const char * *name*) `[static]`

Return true iff the option **name** has the same value in **o1** and **o2**. Must not be called for LINELIST_S or OBSOLETE options.

References CHECK, config_free_lines(), config_lines_eq(), and get_assigned_option().

Referenced by config_dump().

4.12.5.53 static void option_reset (config_format_t * *fmt*, or_options_t * *options*, config_var_t * *var*, int *use_defaults*) `[static]`

Clear the option indexed by **var** in **options**. Then if **use_defaults**, set it to its default value. Called by **config_init()** (p. 385) and option_reset_line() and option_assign_line().

References CHECK, config_assign_value(), config_free_lines(), config_var_t::initvalue, config_line_t::key, LD_BUG, config_var_t::name, option_clear(), tor_free, and config_line_t::value.

Referenced by config_assign_line(), config_init(), and config_reset_line().

4.12.5.54 static int options_act (or_options_t * *old_options*) `[static]`

Fetch the active option list, and take actions based on it. All of the things we do should survive being done repeatedly. If present, **old_options** contains the previous value of the options.

Return 0 if all goes well, return -1 if it's time to die.

Note: We haven't moved all the "act on new configuration" logic here yet. Some is still in **do_hup()** (p. 699) and other places.

References accounting_is_enabled(), accounting_parse_options(), any_predicted_circuits(), authdir_mode_v3(), bool_eq, or_options_t::BridgeRelay, or_options_t::Bridges, cell_ewma_set_scale_factor(), circuit_expire_all_dirty_circs(), circuit_mark_all_unused_circs(), clear_bridge_list(), or_options_t::command, config_register_addressmaps(), configure_accounting(), consider_adding_dir_authorities(), or_options_t::CookieAuthentication, cpuworkers_rotate(), or_options_t::DataDirectory, directory_fetches_dir_info_early(), directory_-

fetches_dir_info_later(), or_options_t::DirPortFrontPage, or_options_t::DirReqStatistics, dirvote_recalculate_timing(), dns_reset(), entry_nodes_should_be_added(), or_options_t::EntryNodes, or_options_t::EntryStatistics, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, or_options_t::ExitNodes, finish_daemon(), geoip_is_loaded(), geoip_load_file(), geoip_remove_old_clients(), or_options_t::GeoIPFile, get_options(), has_completed_circuit, have_lockfile(), inform_testing_reachability(), init_cookie_authentication(), init_keys(), ip_address_changed(), LD_BUG, LD_CIRC, LD_CONFIG, LD_GENERAL, mark_my_descriptor_dirty(), networkstatus_get_latest_consensus(), config_line_t::next, opt_streq(), options_transition_affects_descriptor(), options_transition_affects_workers(), or_state_load(), parse_bridge_line(), parse_virtual_addr_network(), or_options_t::PidFile, policies_parse_from_options(), read_file_to_str(), rend_config_services(), rend_parse_service_authorization(), rend_service_load_keys(), rep_hist_load_mtbf_data(), router_dir_info_changed(), routerset_equal(), or_options_t::RunAsDaemon, server_mode(), should_record_bridge_info(), or_options_t::StrictNodes, tor_free, tor_snprintf(), try_locking(), update_consensus_networkstatus_fetch_time(), or_options_t::UseEntryGuards, or_options_t::V3AuthoritativeDir, config_line_t::value, or_options_t::VirtualAddrNetwork, and write_pidfile().

Referenced by set_options().

4.12.5.55 static int options_act_reversible (or_options_t * *old_options*, char ** *msg*) [static]

Fetch the active option list, and take actions based on it. All of the things we do should survive being done repeatedly. If present, **old_options** contains the previous value of the options.

Return 0 if all goes well, return -1 if things went badly.

References or_options_t::_ConnLimit, add_callback_log(), check_private_dir(), close_temp_logs(), or_options_t::command, conn_type_to_string(), connection_close_immediate(), or_options_t::ConnLimit, control_adjust_event_log_severity(), control_event_logmsg(), or_options_t::ControlSocket, or_options_t::DataDirectory, directory_caches_v2_dir_info(), or_options_t::DisableAllSwap, get_options(), init_libevent(), LD_NET, mark_logs_temp(), options_init_logs(), retry_all_listeners(), rollback_log_changes(), or_options_t::RunAsDaemon, set_max_file_descriptors(), smartlist_create(), smartlist_free(), start_daemon(), switch_id(), tor_asprintf(), tor_assert, tor_free, tor_mlockall(), tor_snprintf(), or_options_t::TransPort, and or_options_t::User.

Referenced by set_options().

4.12.5.56 char* options_dump (or_options_t * *options*, int *minimal*)

Return a string containing a possible configuration file that would give the configuration in **options**. If **minimal** is true, do not include options that are the same as Tor's defaults.

References config_dump().

Referenced by getinfo_helper_misc(), and write_configuration_file().

4.12.5.57 static or_options_t * options_dup (config_format_t * *fmt*, or_options_t * *old*) [static]

Copy storage held by **old** into a new **or_options_t** (p. 148) and return it.

References config_alloc(), config_assign(), config_free_lines(), CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_OBSOLETE, get_assigned_option(), LD_BUG, config_var_t::name, tor_assert, tor_free, config_var_t::type, and config_format_t::vars.

Referenced by options_trial_assign().

4.12.5.58 char* options_get_datadir_fname2_suffix (or_options_t * *options*, const char * *sub1*, const char * *sub2*, const char * *suffix*)

Return a newly allocated string holding a filename relative to the data directory. If **sub1** is present, it is the first path component after the data directory. If **sub2** is also present, it is the second path component after the data directory. If **suffix** is present, it is appended to the filename.

Examples: get_datadir_fname2_suffix("a", NULL, NULL) -> $DATADIR/a get_datadir_fname2_suffix("a", NULL, ".tmp") -> $DATADIR/a.tmp get_datadir_fname2_suffix("a", "b", ".tmp") -> $DATADIR/a/b/.tmp get_datadir_fname2_suffix("a", "b", NULL) -> $DATADIR/a/b

Note: Consider using the get_datadir_fname* macros in **or.h** (p. 739).

References or_options_t::DataDirectory, tor_assert, and tor_snprintf().

Referenced by try_locking().

4.12.5.59 void options_init (or_options_t * *options*)

Set **options** to hold reasonable defaults for most options. Each option defaults to zero.

References config_init().

Referenced by options_init_from_string().

4.12.5.60 setopt_err_t options_init_from_string (const char * *cf*, int *command*, const char * *command_arg*, char ** *msg*)

Load the options from the configuration in **cf**, validate them for consistency and take actions based on them.

Return 0 if success, negative on error: * -1 for general errors. * -2 for failure to parse/validate, * -3 for transition not allowed * -4 for error while setting the new options

References or_options_t::_magic, or_options_t::command, or_options_t::command_arg, config_assign(), config_find_option(), config_free(), config_free_lines(), config_get_lines(), config_var_t::initvalue, config_var_t::name, options_init(), options_transition_allowed(), options_validate(), OR_OPTIONS_MAGIC, set_options(), or_options_t::TestingTorNetwork, tor_asprintf(), tor_assert, and tor_free.

Referenced by handle_control_loadconf(), and options_init_from_torrc().

4.12.5.61 int options_init_from_torrc (int *argc*, char ** *argv*)

Read a configuration file into **options**, finding the configuration file location based on the command line. After loading the file call **options_init_from_string()** (p. 395) to load the config. Return 0 if success, -1 if failure.

References config_get_commandlines(), get_version(), LD_CONFIG, list_torrc_options(), load_torrc_from_disk(), options_init_from_string(), print_usage(), and tor_free.

Referenced by do_hup(), and tor_init().

4.12.5.62 static int options_init_logs (or_options_t * *options*, int *validate_only*) [static]

Initialize the logs based on the configuration file.

References add_file_log(), add_stream_log(), add_syslog_log(), LD_CONFIG, or_options_t::Logs, config_line_t::next, parse_log_severity_config(), or_options_t::RunAsDaemon, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, and config_line_t::value.

Referenced by options_act_reversible(), and options_validate().

4.12.5.63 int options_need_geoip_info (or_options_t * *options*, const char ** *reason_out*)

If we need to have a GEOIP ip-to-country map to run with our configured options, return 1 and set ***reason_out** to a description of why.

References or_options_t::BridgeRecordUsageByCountry, or_options_t::BridgeRelay, or_options_t::EntryNodes, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, or_options_t::ExitNodes, and routerset_needs_geoip().

Referenced by geoip_load_file().

4.12.5.64 or_options_t* options_new (void)

Return a new empty **or_options_t** (p. 148). Used for testing.

References config_alloc().

4.12.5.65 int options_save_current (void)

Save the current configuration file value to disk. Return 0 on success, -1 on failure.

References get_options(), get_torrc_fname(), and write_configuration_file().

Referenced by handle_control_saveconf().

4.12.5.66 static int options_transition_affects_descriptor (or_options_t * *old_options*, or_options_t * *new_options*) [static]

Return 1 if any change from **old_options** to **new_options** will require us to generate a new descriptor; else return 0.

References or_options_t::_PublishServerDescriptor, or_options_t::AccountingMax, or_options_t::AccountingStart, or_options_t::Address, or_options_t::ClientOnly, config_lines_eq(), or_options_t::ContactInfo, or_options_t::DataDirectory, or_options_t::DirPort, or_options_t::ExitPolicy, or_options_t::ExitPolicyRejectPrivate, get_effective_bwburst(), get_effective_bwrate(), or_options_t::MyFamily, or_options_t::Nickname, or_options_t::NoPublish, opt_streq(), and or_options_t::ORPort.

Referenced by options_act().

4.12.5.67 static int options_transition_affects_workers (or_options_t * *old_options*, or_options_t * *new_options*) `[static]`

Return 1 if any change from **old_options** to **new_options** will require us to rotate the CPU and DNS workers; else return 0.

References or_options_t::ClientOnly, config_lines_eq(), or_options_t::DataDirectory, or_options_t::Logs, or_options_t::NumCpus, opt_streq(), or_options_t::ORPort, or_options_t::SafeLogging, and or_options_t::ServerDNSSearchDomains.

Referenced by options_act().

4.12.5.68 static int options_transition_allowed (or_options_t * *old*, or_options_t * *new_val*, char ** *msg*) `[static]`

Check if any of the previous options have changed but aren't allowed to.

References or_options_t::AccelDir, or_options_t::AccelName, or_options_t::CellStatistics, or_options_t::DataDirectory, or_options_t::DirReqStatistics, or_options_t::DisableAllSwap, or_options_t::EntryStatistics, or_options_t::ExitPortStatistics, or_options_t::Group, or_options_t::HardwareAccel, opt_streq(), or_options_t::PidFile, or_options_t::RunAsDaemon, or_options_t::TestingTorNetwork, tor_asprintf(), and or_options_t::User.

Referenced by options_init_from_string(), and options_trial_assign().

4.12.5.69 setopt_err_t options_trial_assign (config_line_t * *list*, int *use_defaults*, int *clear_first*, char ** *msg*)

Try assigning **list** to the global options. You do this by duping options, assigning list to the new one, then validating it. If it's ok, then throw out the old one and stick with the new one. Else, revert to old and return failure. Return SETOPT_OK on success, or a setopt_err_t on failure.

If not success, point ***msg** to a newly allocated string describing what went wrong.

References config_assign(), config_free(), get_options(), options_dup(), options_transition_allowed(), options_validate(), and set_options().

Referenced by control_setconf_helper().

4.12.5.70 static int options_validate (or_options_t * *old_options*, or_options_t * *options*, int *from_setconf*, char ** *msg*) `[static]`

Return 0 if every setting in **options** is reasonable, and a permissible transition from **old_options**. Else return -1. Should have no side effects, except for normalizing the contents of **options**.

On error, tor_strdup an error explanation into ***msg**.

XXX If **from_setconf**, we were called by the controller, and our Log line should stay empty. If it's 0, then give us a default log if there are no logs defined.

References or_options_t::_ExcludeExitNodesUnion, or_options_t::AccountingMax, authdir_mode(), authdir_mode_any_main(), or_options_t::AuthoritativeDir, or_options_t::BridgeAuthoritativeDir, or_options_t::ClientOnly, config_line_append(), config_lines_dup(), config_lines_eq(), or_options_t::ConnLimit, or_options_t::ContactInfo, or_options_t::ControlListenAddress, or_options_t::ControlPort, or_options_t::DirListenAddress, or_options_t::DirPort, dirserv_read_measured_bandwidths(), or_options_t::DNSListenAddress, or_options_t::DNSPort, or_options_t::DownloadExtraInfo, or_options_t::EntryNodes, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, or_options_t::FascistFirewall, or_options_t::FetchDirInfoEarly, or_options_t::FetchDirInfoExtraEarly, or_options_t::FirewallPorts, get_default_nickname(), get_uname(), or_options_t::HSAuthoritativeDir, is_internal_IP(), is_legal_nickname(), config_line_t::key, LD_CONFIG, or_options_t::Logs, or_options_t::LongLivedPorts, or_options_t::NatdListenAddress, or_options_t::NatdPort, config_line_t::next, or_options_t::Nickname, or_options_t::NoPublish, options_init_logs(), or_options_t::ORListenAddress, or_options_t::ORPort, parse_addr_port(), path_is_relative(), or_options_t::PublishServerDescriptor, or_options_t::ReachableAddresses, or_options_t::ReachableDirAddresses, or_options_t::ReachableORAddresses, or_options_t::RecommendedClientVersions, or_options_t::RecommendedServerVersions, or_options_t::RecommendedVersions, or_options_t::RejectPlaintextPorts, or_options_t::RendConfigLines, resolve_my_address(), routerset_is_list(), routerset_new(), routerset_union(), or_options_t::RunAsDaemon, server_mode(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_join_strings(), or_options_t::SocksListenAddress, or_options_t::SocksPort, strcmpstart(), or_options_t::StrictNodes, or_options_t::TestingTorNetwork, tor_asprintf(), tor_assert, tor_free, tor_snprintf(), or_options_t::TransListenAddress, or_options_t::TransPort, or_options_t::UseEntryGuards, or_options_t::V1AuthoritativeDir, or_options_t::V2AuthoritativeDir, or_options_t::V3AuthoritativeDir, or_options_t::V3BandwidthsFile, validate_data_directory(), validate_ports_csv(), config_line_t::value, or_options_t::VersioningAuthoritativeDir, and or_options_t::WarnPlaintextPorts.

Referenced by options_init_from_string(), and options_trial_assign().

4.12.5.71 static void or_options_free (or_options_t * *options*) [static]

Release additional memory allocated in options

References or_options_t::_ExcludeExitNodesUnion, config_free(), and routerset_free().

Referenced by config_free_all().

4.12.5.72 static int or_state_load (void) [static]

Reload the persistent state from disk, generating a new state as needed. Return 0 on success, less than 0 on failure.

References or_state_t::_magic, config_assign(), config_free(), config_free_lines(), config_get_lines(), config_init(), file_status(), get_datadir_fname, LD_BUG, LD_FS, LD_GENERAL, or_state_t::next_write, OR_STATE_MAGIC, or_state_save(), or_state_set(), or_state_validate(), read_file_to_str(), tor_free, and tor_snprintf().

Referenced by options_act().

4.12.5.73 int or_state_save (time_t *now*)

Write the persistent state to disk. Return 0 for success, <0 on failure.

References accounting_is_enabled(), accounting_run_housekeeping(), circ_times, circuit_build_times_update_state(), config_dump(), entry_guards_update_state(), format_local_iso_time(), get_datadir_fname, get_options(), get_version(), or_state_t::LastWritten, LD_FS, LD_GENERAL, or_state_t::next_write, rep_hist_update_state(), tor_asprintf(), tor_assert, tor_free, or_state_t::TorVersion, and write_str_to_file().

Referenced by or_state_load(), run_scheduled_events(), and tor_cleanup().

4.12.5.74 static void or_state_set (or_state_t * *new_state*) [static]

Replace the current persistent state with **new_state**

References circ_times, circuit_build_times_parse_state(), config_free(), entry_guards_parse_state(), LD_GENERAL, rep_hist_load_state(), tor_assert, and tor_free.

Referenced by or_state_load().

4.12.5.75 static int or_state_validate (or_state_t * *old_state*, or_state_t * *state*, int *from_setconf*, char ** *msg*) [static]

Return 0 if every setting in **state** is reasonable, and a permissible transition from **old_state**. Else warn and return -1. Should have no side effects, except for normalizing the contents of **state**.

References entry_guards_parse_state().

Referenced by or_state_load().

4.12.5.76 static int parse_bridge_line (const char * *line*, int *validate_only*) [static]

Read the contents of a Bridge line from **line**. Return 0 if the line is well-formed, and -1 if it isn't. If **validate_only** is 0, and the line is well-formed, then add the bridge described in the line to our internal bridge list.

References base16_decode(), bridge_add_from_config(), DIGEST_LEN, fmt_addr(), HEX_DIGEST_LEN, LD_CONFIG, LD_DIR, smartlist_create(), smartlist_del_keeporder(), smartlist_free(), smartlist_join_strings(), smartlist_split_string(), tor_addr_port_parse(), and tor_free.

Referenced by options_act().

4.12.5.77 static int parse_dir_server_line (const char * *line*, authority_type_t *required_type*, int *validate_only*) [static]

Read the contents of a DirServer line from **line**. If **validate_only** is 0, and the line is well-formed, and it shares any bits with **required_type** or **required_type** is 0, then add the dirserver described in the line (minus whatever bits it's missing) as a valid authority. Return 0 on success, or -1 if the line isn't well-formed or if we can't add it.

References add_trusted_dir_server(), base16_decode(), BRIDGE_AUTHORITY, DIGEST_LEN, get_torrc_fname(), HEX_DIGEST_LEN, HIDSERV_AUTHORITY, is_legal_nickname(), LD_CONFIG, LD_DIR, parse_addr_port(), smartlist_create(), smartlist_del_keeporder(), smartlist_free(), smartlist_join_strings(), smartlist_split_string(), strcasecmpstart(), tor_free, tor_parse_long(), V1_AUTHORITY, V2_AUTHORITY, and V3_AUTHORITY.

Referenced by add_default_trusted_dir_authorities(), consider_adding_dir_authorities(), and validate_dir_authorities().

4.12.5.78 static void print_usage (void) [static]

Print a usage message for tor.

Referenced by options_init_from_torrc().

4.12.5.79 void remove_file_if_very_old (const char ∗ *fname*, time_t *now*)

Given a file name check to see whether the file exists but has not been modified for a very long time. If so, remove it.

References format_local_iso_time(), and LD_GENERAL.

Referenced by router_reload_router_list_impl(), and router_reload_v2_networkstatus().

4.12.5.80 int resolve_my_address (int *warn_severity*, or_options_t ∗ *options*, uint32_t ∗ *addr_out*, char ∗∗ *hostname_out*)

Based on **options->Address**, guess our public IP address and put it (in host order) into ∗**addr_out**. If **hostname_out** is provided, set ∗**hostname_out** to a new string holding the hostname we used to get the address. Return 0 if all is well, or -1 if we can't find a suitable public IP address.

References or_options_t::_PublishServerDescriptor, or_options_t::Address, or_options_t::AlternateDirAuthority, control_event_server_status(), or_options_t::DirServers, get_interface_address(), INET_NTOA_BUF_LEN, ip_address_changed(), is_internal_IP(), LD_CONFIG, LD_NET, tor_assert, tor_inet_aton(), tor_inet_ntoa(), and tor_lookup_hostname().

Referenced by add_trusted_dir_server(), check_descriptor_ipaddress_changed(), dirserv_generate_networkstatus_vote_obj(), generate_v2_networkstatus_opinion(), options_validate(), router_new_address_suggestion(), and router_pick_published_address().

4.12.5.81 const char∗ safe_str (const char ∗ *address*)

Make **address** – a piece of information of unspecified sensitivity – safe to log according to the settings in options->SafeLogging, and return it.

(We return "[scrubbed]" if SafeLogging is anything besides "0", and address otherwise.)

References get_options(), and tor_assert.

Referenced by command_process_relay_cell(), connection_ap_process_end_not_open(), connection_edge_finished_connecting(), connection_exit_begin_conn(), directory_handle_command(), directory_handle_command_get(), evdns_callback(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store_v2_desc_as_dir(), rend_compute_v2_desc_id(), rend_mid_establish_intro(), and rend_mid_introduce().

4.12.5.82 const char∗ safe_str_client (const char ∗ *address*)

Make **address** – a piece of information related to our operation as a client – safe to log according to the settings in options->SafeLogging, and return it.

(We return "[scrubbed]" if SafeLogging is "1", and address otherwise.)

References get_options(), and tor_assert.

Referenced by _circuit_mark_for_close(), addressmap_register(), addressmap_register_virtual_address(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_handle_first_hop(), client_dns_incr_failures(), command_process_netinfo_cell(), command_process_versions_cell(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_or_check_valid_tls_handshake(), connection_tls_finish_handshake(), directory_all_-

unreachable(), directory_post_to_hs_dir(), dumpstats(), fetch_from_buf_socks(), handle_control_mapaddress(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_compute_v2_desc_id(), rend_config_services(), and rend_service_dump_stats().

4.12.5.83 int set_options (or_options_t * *new_val*, char ** *msg*)

Change the current global options to contain **new_val** instead of their current value; take action based on the new value; free the old value as necessary. Returns 0 on success, -1 on failure.

References config_free(), LD_BUG, options_act(), options_act_reversible(), and tor_assert.

Referenced by options_init_from_string(), and options_trial_assign().

4.12.5.84 static int validate_data_directory (or_options_t * *options*) `[static]`

Check and normalize the value of options->DataDirectory; return 0 if it is sane, -1 otherwise.

References or_options_t::DataDirectory, LD_CONFIG, normalize_data_directory(), and tor_assert.

Referenced by options_validate().

4.12.5.85 static int validate_dir_authorities (or_options_t * *options*, or_options_t * *old_options*) `[static]`

Look at all the config options for using alternate directory authorities, and make sure none of them are broken. Also, warn the user if we changed any dangerous ones.

References or_options_t::AlternateBridgeAuthority, or_options_t::AlternateDirAuthority, or_options_t::AlternateHSAuthority, config_lines_eq(), or_options_t::DirServers, LD_CONFIG, config_line_t::next, parse_dir_server_line(), and config_line_t::value.

4.12.5.86 static int validate_ports_csv (smartlist_t * *sl*, const char * *name*, char ** *msg*) `[static]`

Return 0 if every element of sl is a string holding a decimal representation of a port number, or if sl is NULL. Otherwise set *msg and return -1.

References tor_asprintf(), and tor_assert.

Referenced by options_validate().

4.12.5.87 static int write_configuration_file (const char * *fname*, or_options_t * *options*) `[static]`

Save a configuration file for the configuration in **options** into the file **fname**. If the file already exists, and doesn't begin with GENERATED_FILE_PREFIX, rename it. Otherwise replace it. Return 0 on success, -1 on failure.

References file_status(), GENERATED_FILE_COMMENT, GENERATED_FILE_PREFIX, LD_BUG, LD_CONFIG, LD_FS, options_dump(), read_file_to_str(), strcmpstart(), tor_asprintf(), tor_assert, tor_free, tor_snprintf(), and write_str_to_file().

Referenced by options_save_current().

4.12.6 Variable Documentation

4.12.6.1 config_abbrev_t _option_abbrevs[ ] [static]

A list of abbreviations and aliases to map command-line options, obsolete option names, or alternative option names, to their current values.

4.12.6.2 config_var_t _option_vars[ ] [static]

Array of configuration options. Until we disallow nonstandard abbreviations, order is significant, since the first matching option will be chosen first.

4.12.6.3 config_abbrev_t _state_abbrevs[ ] [static]

Initial value:

```
 {
  { "AccountingBytesReadInterval", "AccountingBytesReadInInterval", 0, 0 },
  { "HelperNode", "EntryGuard", 0, 0 },
  { "HelperNodeDownSince", "EntryGuardDownSince", 0, 0 },
  { "HelperNodeUnlistedSince", "EntryGuardUnlistedSince", 0, 0 },
  { "EntryNode", "EntryGuard", 0, 0 },
  { "EntryNodeDownSince", "EntryGuardDownSince", 0, 0 },
  { "EntryNodeUnlistedSince", "EntryGuardUnlistedSince", 0, 0 },
  { NULL, NULL, 0, 0},
}
```

A list of state-file "abbreviations," for compatibility.

4.12.6.4 config_var_t _state_vars[ ] [static]

Array of "state" variables saved to the ~/.tor/state file.

4.12.6.5 char* _version = NULL [static]

The version of this Tor process, as parsed.

4.12.6.6 config_line_t* global_cmdline_options = NULL [static]

Configuration Options set by command line.

4.12.6.7 char* global_dirfrontpagecontents = NULL [static]

Contents of most recently read DirPortFrontPage file.

4.12.6.8 or_options_t* global_options = NULL [static]

Command-line and config-file options.

4.12.6.9 or_state_t∗ global_state = NULL [static]

Persistent serialized state.

4.12.6.10 uint32_t last_resolved_addr = 0 [static]

Last value actually set by resolve_my_address.

4.12.6.11 struct unit_table_t memory_units[ ] [static]

Initial value:

```
 {
  { "",          1 },
  { "b",         1<< 0 },
  { "byte",      1<< 0 },
  { "bytes",     1<< 0 },
  { "kb",        1<<10 },
  { "kbyte",     1<<10 },
  { "kbytes",    1<<10 },
  { "kilobyte",  1<<10 },
  { "kilobytes", 1<<10 },
  { "m",         1<<20 },
  { "mb",        1<<20 },
  { "mbyte",     1<<20 },
  { "mbytes",    1<<20 },
  { "megabyte",  1<<20 },
  { "megabytes", 1<<20 },
  { "gb",        1<<30 },
  { "gbyte",     1<<30 },
  { "gbytes",    1<<30 },
  { "gigabyte",  1<<30 },
  { "gigabytes", 1<<30 },
  { "tb",        U64_LITERAL(1)<<40 },
  { "terabyte",  U64_LITERAL(1)<<40 },
  { "terabytes", U64_LITERAL(1)<<40 },
  { NULL, 0 },
}
```

Table to map the names of memory units to the number of bytes they contain.

4.12.6.12 config_format_t options_format [static]

Initial value:

```
 {
  sizeof(or_options_t),
  OR_OPTIONS_MAGIC,
  STRUCT_OFFSET(or_options_t, _magic),
  _option_abbrevs,
  _option_vars,
  (validate_fn_t)options_validate,
  NULL
}
```

Configuration format for **or_options_t** (p. 148).

4.12.6.13 config_var_t state_extra_var [static]

Initial value:

```
{
  "__extra", CONFIG_TYPE_LINELIST, STRUCT_OFFSET(or_state_t, ExtraLines), NULL
}
```

"Extra" variable in the state that receives lines we can't parse. This lets us preserve options from versions of Tor newer than us.

4.12.6.14 config_format_t state_format [static]

Initial value:

```
{
  sizeof(or_state_t),
  OR_STATE_MAGIC,
  STRUCT_OFFSET(or_state_t, _magic),
  _state_abbrevs,
  _state_vars,
  (validate_fn_t)or_state_validate,
  &state_extra_var,
}
```

Configuration format for **or_state_t** (p. 178).

4.12.6.15 config_var_t testing_tor_network_defaults[ ] [static]

Initial value:

```
{
  V(ServerDNSAllowBrokenConfig,  BOOL,  "1"),
  V(DirAllowPrivateAddresses,    BOOL,     "1"),
  V(EnforceDistinctSubnets,      BOOL,     "0"),
  V(AssumeReachable,             BOOL,     "1"),
  V(AuthDirMaxServersPerAddr,    UINT,     "0"),
  V(AuthDirMaxServersPerAuthAddr,UINT,     "0"),
  V(ClientDNSRejectInternalAddresses, BOOL,"0"),
  V(ExitPolicyRejectPrivate,     BOOL,     "0"),
  V(V3AuthVotingInterval,        INTERVAL, "5 minutes"),
  V(V3AuthVoteDelay,             INTERVAL, "20 seconds"),
  V(V3AuthDistDelay,             INTERVAL, "20 seconds"),
  V(TestingV3AuthInitialVotingInterval, INTERVAL, "5 minutes"),
  V(TestingV3AuthInitialVoteDelay, INTERVAL, "20 seconds"),
  V(TestingV3AuthInitialDistDelay, INTERVAL, "20 seconds"),
  V(TestingAuthDirTimeToLearnReachability, INTERVAL, "0 minutes"),
  V(TestingEstimatedDescriptorPropagationTime, INTERVAL, "0 minutes"),
  { NULL, CONFIG_TYPE_OBSOLETE, 0, NULL }
}
```

Override default values with these if the user sets the TestingTorNetwork option.

4.12.6.16 struct unit_table_t time_units[ ] [static]

Initial value:

```
 {
  { "",         1 },
  { "second",   1 },
  { "seconds",  1 },
  { "minute",   60 },
  { "minutes",  60 },
  { "hour",     60*60 },
  { "hours",    60*60 },
  { "day",      24*60*60 },
  { "days",     24*60*60 },
  { "week",     7*24*60*60 },
  { "weeks",    7*24*60*60 },
  { NULL, 0 },
}
```

Table to map the names of time units to the number of seconds they contain.

4.12.6.17 const char tor_git_revision[ ]

String describing which Tor subversion repository version the source was built from. This string is generated by a bit of shell kludging int src/or/Makefile.am, and is usually right.

4.12.6.18 char∗ torrc_fname = NULL `[static]`

Name of most recently read torrc file.

4.13 connection.c File Reference

General high-level functions to handle reading and writing on connections.

`#include "or.h"`

Defines

- #define **CONN_IS_CLOSED**(c) ((c) → linked ? ((c) → linked_conn_is_closed) : ((c) → s < 0))
- #define **WARN_TOO_MANY_CONNS_INTERVAL** (6∗60∗60)
- #define **CLIENT_IDLE_TIME_FOR_PRIORITY** 30
- #define **MIN_TLS_FLUSHLEN** 15872

Functions

- static **connection_t** ∗ **connection_create_listener** (struct sockaddr ∗listensockaddr, socklen_t listensocklen, int type, char ∗address)
- static void **connection_init** (time_t now, **connection_t** ∗conn, int type, int socket_family)
- static int **connection_init_accepted_conn** (**connection_t** ∗conn, uint8_t listener_type)
- static int **connection_handle_listener_read** (**connection_t** ∗conn, int new_type)
- static int **connection_bucket_should_increase** (int bucket, **or_connection_t** ∗conn)
- static int **connection_finished_flushing** (**connection_t** ∗conn)
- static int **connection_flushed_some** (**connection_t** ∗conn)
- static int **connection_finished_connecting** (**connection_t** ∗conn)
- static int **connection_reached_eof** (**connection_t** ∗conn)
- static int **connection_read_to_buf** (**connection_t** ∗conn, int ∗max_to_read, int ∗socket_error)
- static int **connection_process_inbuf** (**connection_t** ∗conn, int package_partial)
- static void **client_check_address_changed** (int sock)
- static void **set_constrained_socket_buffers** (int sock, int size)
- static const char ∗ **connection_proxy_state_to_string** (int state)
- static int **connection_read_https_proxy_response** (**connection_t** ∗conn)
- static void **connection_send_socks5_connect** (**connection_t** ∗conn)
- const char ∗ **conn_type_to_string** (int type)
- const char ∗ **conn_state_to_string** (int type, int state)
- **dir_connection_t** ∗ **dir_connection_new** (int socket_family)
- **or_connection_t** ∗ **or_connection_new** (int socket_family)
- **edge_connection_t** ∗ **edge_connection_new** (int type, int socket_family)
- **control_connection_t** ∗ **control_connection_new** (int socket_family)
- **connection_t** ∗ **connection_new** (int type, int socket_family)
- void **connection_link_connections** (**connection_t** ∗conn_a, **connection_t** ∗conn_b)
- static void **_connection_free** (**connection_t** ∗conn)
- void **connection_free** (**connection_t** ∗conn)
- void **connection_free_all** (void)
- void **connection_about_to_close_connection** (**connection_t** ∗conn)
- void **connection_close_immediate** (**connection_t** ∗conn)

- void **_connection_mark_for_close** (**connection_t** *conn, int line, const char *file)
- void **connection_expire_held_open** (void)
- static struct sockaddr_in * **create_inet_sockaddr** (const char *listenaddress, uint16_t listenport, char **readable_address, socklen_t *socklen_out)
- static struct sockaddr_un * **create_unix_sockaddr** (const char *listenaddress, char **readable_address, socklen_t *len_out)
- static void **warn_too_many_conns** (void)
- static int **check_sockaddr** (struct sockaddr *sa, int len, int level)
- static int **check_sockaddr_family_match** (sa_family_t got, **connection_t** *listener)
- int **connection_connect** (**connection_t** *conn, const char *address, const **tor_addr_t** *addr, uint16_t port, int *socket_error)
- int **connection_proxy_connect** (**connection_t** *conn, int type)
- int **connection_read_proxy_handshake** (**connection_t** *conn)
- static int **retry_listeners** (int type, **config_line_t** *cfg, int port_option, const char *default_addr, **smartlist_t** *replaced_conns, **smartlist_t** *new_conns, int disable_all_conns, int socket_family)
- int **retry_all_listeners** (**smartlist_t** *replaced_conns, **smartlist_t** *new_conns)
- static int **connection_is_rate_limited** (**connection_t** *conn)
- static int **connection_counts_as_relayed_traffic** (**connection_t** *conn, time_t now)
- static ssize_t **connection_bucket_round_robin** (int base, int priority, ssize_t global_bucket, ssize_t conn_bucket)
- static ssize_t **connection_bucket_read_limit** (**connection_t** *conn, time_t now)
- ssize_t **connection_bucket_write_limit** (**connection_t** *conn, time_t now)
- int **global_write_bucket_low** (**connection_t** *conn, size_t attempt, int priority)
- static void **connection_buckets_decrement** (**connection_t** *conn, time_t now, size_t num_read, size_t num_written)
- static void **connection_consider_empty_read_buckets** (**connection_t** *conn)
- static void **connection_consider_empty_write_buckets** (**connection_t** *conn)
- void **connection_bucket_init** (void)
- static void **connection_bucket_refill_helper** (int *bucket, int rate, int burst, int seconds_elapsed, const char *name)
- void **connection_bucket_refill** (int seconds_elapsed, time_t now)
- static int **connection_handle_read_impl** (**connection_t** *conn)
- int **connection_handle_read** (**connection_t** *conn)
- int **connection_fetch_from_buf** (char *string, size_t len, **connection_t** *conn)
- int **connection_wants_to_flush** (**connection_t** *conn)
- int **connection_outbuf_too_full** (**connection_t** *conn)
- static int **connection_handle_write_impl** (**connection_t** *conn, int force)
- int **connection_handle_write** (**connection_t** *conn, int force)
- void **_connection_write_to_buf_impl** (const char *string, size_t len, **connection_t** *conn, int zlib)
- **connection_t** * **connection_get_by_type_addr_port_purpose** (int type, const **tor_addr_t** *addr, uint16_t port, int purpose)
- **connection_t** * **connection_get_by_global_id** (uint64_t id)
- **connection_t** * **connection_get_by_type** (int type)
- **connection_t** * **connection_get_by_type_state** (int type, int state)
- **connection_t** * **connection_get_by_type_state_rendquery** (int type, int state, const char *rendquery)
- **connection_t** * **connection_get_by_type_purpose** (int type, int purpose)
- int **connection_is_listener** (**connection_t** *conn)
- int **connection_state_is_open** (**connection_t** *conn)

- int **connection_state_is_connecting** (**connection_t** *conn)
- char * **alloc_http_authenticator** (const char *authenticator)
- void **connection_dump_buffer_mem_stats** (int severity)
- void **assert_connection_ok** (**connection_t** *conn, time_t now)

Variables

- static uint32_t **last_interface_ip** = 0
- static **smartlist_t** * **outgoing_addrs** = NULL
- int **global_read_bucket**
- int **global_write_bucket**
- int **global_relayed_read_bucket**
- int **global_relayed_write_bucket**
- static int **write_buckets_empty_last_second** = 0

4.13.1 Detailed Description

General high-level functions to handle reading and writing on connections.

4.13.2 Define Documentation

4.13.2.1 #define CLIENT_IDLE_TIME_FOR_PRIORITY 30

How many seconds of no active local circuits will make the connection revert to the "relayed" bandwidth class?

Referenced by connection_counts_as_relayed_traffic().

4.13.2.2 #define CONN_IS_CLOSED(c) ((c) → linked ? ((c) → linked_conn_is_closed) : ((c) → s < 0))

Return true iff **connection_close_immediate()** (p. 414) has been called on this connection.

Referenced by connection_close_immediate(), and connection_finished_flushing().

4.13.2.3 #define MIN_TLS_FLUSHLEN 15872

OpenSSL TLS record size is 16383; this is close. The goal here is to push data out as soon as we know there's enough for a TLS record, so during periods of high load we won't read entire megabytes from input before pushing any data out. It also has the feature of not growing huge outbufs unless something is slow.

Referenced by _connection_write_to_buf_impl().

4.13.3 Function Documentation

4.13.3.1 static void _connection_free (connection_t * *conn*) `[static]`

Deallocate memory used by **conn**. Deallocate its buffers if necessary, close its socket if necessary, and mark the directory as dirty if **conn** is an OR or OP connection.

References or_connection_t::active_circuit_pqueue, connection_t::address, buf_datalen(), buf_free(), dir_connection_t::cached_dir, cached_dir_decref(), edge_connection_t::chosen_exit_name, CONN_IS_EDGE, conn_state_to_string(), CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, conn_type_to_string(), connection_is_listener(), connection_or_remove_from_identity_map(), dir_connection_t::fingerprint_stack, or_connection_t::handshake_state, connection_t::inbuf, control_connection_t::incoming_cmd, LD_BUG, LD_GENERAL, LD_NET, connection_t::linked, connection_t::magic, or_connection_t::nickname, or_handshake_state_free(), connection_t::outbuf, connection_t::read_event, dir_connection_t::rend_data, edge_connection_t::rend_data, rend_data_free(), dir_connection_t::requested_resource, connection_t::s, smartlist_free(), connection_t::socket_family, edge_connection_t::socks_request, connection_t::state, or_connection_t::tls, TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, tor_close_socket(), tor_digest_is_zero(), tor_free, tor_tls_free(), tor_zlib_free(), connection_t::type, connection_t::write_event, and dir_connection_t::zlib_state.

Referenced by connection_free(), and connection_free_all().

4.13.3.2 void _connection_mark_for_close (connection_t * *conn*, int *line*, const char * *file*)

Mark **conn** to be closed next time we loop through **conn_close_if_marked()** (p. 693) in **main.c** (p. 690).

References add_connection_to_closeable_list(), assert_connection_ok(), LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, connection_t::timestamp_lastwritten, tor_assert, and tor_fragile_assert.

Referenced by _connection_mark_unattached_ap().

4.13.3.3 void _connection_write_to_buf_impl (const char * *string*, size_t *len*, connection_t * *conn*, int *zlib*)

Append **len** bytes of **string** onto **conn**'s outbuf, and ask it to start writing.

If **zlib** is nonzero, this is a directory connection that should get its contents compressed or decompressed as they're written. If zlib is negative, this is the last data to be compressed, and the connection's zlib state should be flushed.

If it's an OR conn and an entire TLS record is ready, then try to flush the record now. Similarly, if it's a local control connection and a 64k chunk is ready, try to flush it all, so we don't end up with many megabytes of controller info queued at once.

References buf_datalen(), circuit_get_by_edge_conn(), CONN_IS_EDGE, CONN_LOG_PROTECT, CONN_TYPE_CONTROL, CONN_TYPE_OR, connection_close_immediate(), connection_is_rate_limited(), connection_start_writing(), connection_t::hold_open_until_flushed, connection_t::in_flushed_some, LD_BUG, LD_NET, connection_t::marked_for_close, MIN_TLS_FLUSHLEN, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::s, TO_DIR_CONN(), TO_EDGE_CONN(), tor_fragile_assert, connection_t::type, write_to_buf(), write_to_buf_zlib(), and dir_connection_t::zlib_state.

4.13.3.4 char* alloc_http_authenticator (const char * *authenticator*)

Allocates a base64'ed authenticator for use in http or https auth, based on the input string **authenticator**. Returns it if success, else returns NULL.

References base64_encode(), and tor_free.

Referenced by connection_proxy_connect(), directory_handle_command_get(), and directory_send_command().

4.13.3.5 void assert_connection_ok (connection_t * *conn*, time_t *now*)

Verify that connection **conn** has all of its invariants correct. Trigger an assert if anything is invalid.

References connection_t::address, AP_CONN_STATE_OPEN, assert_buf_ok(), assert_cpath_layer_ok(), edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, edge_connection_t::chosen_exit_retries, CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_EXIT, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, connection_is_listener(), connection_is_writing(), edge_connection_t::cpath_layer, EXIT_PURPOSE_CONNECT, EXIT_PURPOSE_RESOLVE, socks_request_t::has_finished, connection_t::hold_open_until_flushed, connection_t::inbuf, connection_t::linked, connection_t::linked_conn, LISTENER_STATE_READY, connection_t::magic, connection_t::marked_for_close, OR_CONN_STATE_OPEN, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::purpose, connection_t::s, edge_connection_t::socks_request, connection_t::state, or_connection_t::tls, TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, connection_t::type, and connection_t::write_blocked_on_bw.

Referenced by _connection_mark_for_close(), add_connection_to_closeable_list(), assert_all_pending_dns_resolves_ok(), command_process_netinfo_cell(), conn_close_if_marked(), conn_read_callback(), conn_write_callback(), connection_close_immediate(), connection_dns_remove(), connection_or_finished_flushing(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), and rend_client_desc_trynow().

4.13.3.6 static int check_sockaddr (struct sockaddr * *sa*, int *len*, int *level*) `[static]`

Do basic sanity checking on a newly received socket. Return 0 if it looks ok, else return -1.

References LD_NET, and tor_mem_is_zero().

Referenced by connection_handle_listener_read().

4.13.3.7 static int check_sockaddr_family_match (sa_family_t *got*, connection_t * *listener*) `[static]`

Check whether the socket family from an accepted socket **got** is the same as the one that **listener** is waiting for. If it isn't, log a useful message and return -1. Else return 0.

This is annoying, but can apparently happen on some Darwins.

References conn_type_to_string(), LD_BUG, connection_t::socket_family, and connection_t::type.

Referenced by connection_handle_listener_read().

4.13.3.8 static void client_check_address_changed (int *sock*) [static]

Given a socket handle, check whether the local address (sockname) of the socket is one that we've connected from before. If so, double-check whether our address has changed and we need to generate keys. If we do, call **init_keys()** (p. 935).

References get_interface_address(), ip_address_changed(), last_interface_ip, LD_NET, smartlist_add(), smartlist_clear(), smartlist_create(), and tor_free.

Referenced by connection_connect().

4.13.3.9 const char∗ conn_state_to_string (int *type*, int *state*)

Return the human-readable name for the connection state **state** for the connection type **type**

References AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_RESOLVE_WAIT, AP_CONN_STATE_SOCKS_WAIT, CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_EXIT, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, conn_type_to_string(), CONTROL_CONN_STATE_NEEDAUTH, CONTROL_CONN_STATE_OPEN, CPUWORKER_STATE_BUSY_ONION, CPUWORKER_STATE_IDLE, DIR_CONN_STATE_CLIENT_FINISHED, DIR_CONN_STATE_CLIENT_READING, DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_CONNECTING, DIR_CONN_STATE_SERVER_COMMAND_WAIT, DIR_CONN_STATE_SERVER_WRITING, EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, EXIT_CONN_STATE_RESOLVEFAILED, EXIT_CONN_STATE_RESOLVING, LD_BUG, LISTENER_STATE_READY, OR_CONN_STATE_CONNECTING, OR_CONN_STATE_OPEN, OR_CONN_STATE_OR_HANDSHAKING, OR_CONN_STATE_PROXY_HANDSHAKING, OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING, OR_CONN_STATE_TLS_HANDSHAKING, OR_CONN_STATE_TLS_SERVER_RENEGOTIATING, and tor_snprintf().

Referenced by _connection_free(), connection_ap_expire_beginning(), connection_buckets_decrement(), connection_close_immediate(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_expire_held_open(), and dumpstats().

4.13.3.10 const char∗ conn_type_to_string (int *type*)

Return the human-readable name for the connection type **type**

References CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_EXIT, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, LD_BUG, and tor_snprintf().

Referenced by _connection_free(), check_sockaddr_family_match(), conn_close_if_marked(), conn_read_callback(), conn_state_to_string(), conn_write_callback(), connection_add(), connection_buckets_decrement(), connection_close_immediate(), connection_create_listener(), connection_expire_held_open(), connection_remove(), dumpstats(), and options_act_-

reversible().

4.13.3.11 void connection_about_to_close_connection (connection_t * *conn*)

Do any cleanup needed:

- Directory conns that failed to fetch a rendezvous descriptor need to inform pending rendezvous streams.
- OR conns need to call rep_hist_note_*() to record status.
- AP conns need to send a socks reject if necessary.
- Exit conns need to call **connection_dns_remove()** (p. 632) if necessary.
- AP and Exit conns need to send an end cell if they can.
- DNS conns need to fail any resolves that are pending on them.
- OR and edge connections need to be unlinked from circuits.

References authdir_mode_tests_reachability(), circuit_detach_stream(), circuit_get_by_edge_conn(), circuit_n_conn_done(), circuit_unlink_all_from_or_conn(), CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_dir_request_failed(), connection_dns_remove(), connection_or_nonopen_was_started_here(), control_event_bootstrap_problem(), control_event_or_conn_status(), control_event_stream_bandwidth(), control_event_stream_status(), DIR_CONN_STATE_CLIENT_FINISHED, DIR_PURPOSE_FETCH_RENDDESC_V2, edge_connection_t::dns_server_request, dnsserv_reject_request(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, entry_guard_register_connect_status(), EXIT_CONN_STATE_RESOLVING, get_options(), socks_request_t::has_finished, connection_t::hold_open_until_flushed, or_options_t::HttpsProxy, or_connection_t::identity_digest, LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, rend_data_t::onion_address, OR_CONN_STATE_OPEN, OR_CONN_STATE_TLS_HANDSHAKING, orconn_end_reason_to_control_string(), connection_t::purpose, rend_client_refetch_v2_renddesc(), dir_connection_t::rend_data, REND_SERVICE_ID_LEN_BASE32, rep_hist_note_connect_failed(), rep_hist_note_connection_died(), rep_hist_note_disconnect(), edge_connection_t::socks_request, connection_t::state, or_connection_t::tls_error, tls_error_to_orconn_end_reason(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, tor_digest_is_zero(), tor_fragile_assert, and connection_t::type.

Referenced by connection_unlink().

4.13.3.12 void connection_bucket_init (void)

Initialize the global read bucket to options->BandwidthBurst.

References or_options_t::BandwidthBurst, get_options(), or_options_t::RelayBandwidthBurst, and or_options_t::RelayBandwidthRate.

Referenced by do_main_loop().

4.13.3.13 static ssize_t connection_bucket_read_limit (connection_t * *conn*, time_t *now*) `[static]`

How many bytes at most can we read onto this connection?

References CELL_NETWORK_SIZE, CONN_TYPE_DIR, connection_bucket_round_robin(), connection_counts_as_relayed_traffic(), connection_is_rate_limited(), OR_CONN_STATE_OPEN, or_connection_t::read_bucket, RELAY_PAYLOAD_SIZE, connection_t::state, TO_OR_CONN(), and connection_t::type.

Referenced by connection_read_to_buf().

4.13.3.14 void connection_bucket_refill (int *seconds_elapsed*, time_t *now*)

A second has rolled over; increment buckets appropriately.

References or_connection_t::bandwidthburst, or_options_t::BandwidthBurst, or_connection_t::bandwidthrate, or_options_t::BandwidthRate, connection_bucket_refill_helper(), connection_bucket_should_increase(), connection_counts_as_relayed_traffic(), connection_start_reading(), connection_start_writing(), get_connection_array(), get_options(), LD_NET, LOG_FN_CONN, OR_CONN_STATE_OPEN, or_connection_t::read_bucket, or_options_t::RelayBandwidthBurst, or_options_t::RelayBandwidthRate, TO_OR_CONN(), tor_assert, and or_connection_t::write_bucket.

Referenced by second_elapsed_callback().

4.13.3.15 static void connection_bucket_refill_helper (int * *bucket*, int *rate*, int *burst*, int *seconds_elapsed*, const char * *name*) `[static]`

Refill a single **bucket** called **name** with bandwidth rate **rate** and bandwidth burst **burst**, assuming that **seconds_elapsed** seconds have passed since the last call.

References LD_NET.

Referenced by connection_bucket_refill().

4.13.3.16 static ssize_t connection_bucket_round_robin (int *base*, int *priority*, ssize_t *global_bucket*, ssize_t *conn_bucket*) `[static]`

Helper function to decide how many bytes out of **global_bucket** we're willing to use for this transaction. **base** is the size of a cell on the network; **priority** says whether we should write many of them or just a few; and **conn_bucket** (if non-negative) provides an upper limit for our answer.

Referenced by connection_bucket_read_limit(), and connection_bucket_write_limit().

4.13.3.17 static int connection_bucket_should_increase (int *bucket*, or_connection_t * *conn*) `[static]`

Is the **bucket** for connection **conn** low enough that we should add another pile of tokens to it?

References or_connection_t::_base, or_connection_t::bandwidthburst, OR_CONN_STATE_OPEN, connection_t::state, and tor_assert.

Referenced by connection_bucket_refill().

4.13.3.18 ssize_t connection_bucket_write_limit (connection_t * *conn*, time_t *now*)

How many bytes at most can we write onto this connection?

References CELL_NETWORK_SIZE, CONN_TYPE_DIR, connection_bucket_round_robin(), connection_counts_as_relayed_traffic(), connection_is_rate_limited(), OR_CONN_STATE_OPEN, connection_t::outbuf_flushlen, RELAY_PAYLOAD_SIZE, connection_t::state, TO_OR_CONN(), connection_t::type, and or_connection_t::write_bucket.

Referenced by conn_close_if_marked(), and connection_handle_write_impl().

4.13.3.19 static void connection_buckets_decrement (connection_t ∗ *conn*, time_t *now*, size_t *num_read*, size_t *num_written*) `[static]`

We just read **num_read** and wrote **num_written** bytes onto **conn**. Decrement buckets appropriately.

References conn_state_to_string(), CONN_TYPE_EXIT, conn_type_to_string(), connection_counts_as_relayed_traffic(), connection_is_rate_limited(), LD_BUG, OR_CONN_STATE_OPEN, connection_t::port, or_connection_t::read_bucket, rep_hist_note_bytes_read(), rep_hist_note_bytes_written(), rep_hist_note_exit_bytes_read(), rep_hist_note_exit_bytes_written(), connection_t::state, TO_OR_CONN(), tor_fragile_assert, connection_t::type, and or_connection_t::write_bucket.

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), and connection_read_to_buf().

4.13.3.20 void connection_close_immediate (connection_t ∗ *conn*)

Close the underlying socket for **conn**, so we don't try to flush it. Must be used in conjunction with (right before) connection_mark_for_close().

References assert_connection_ok(), buf_clear(), CONN_IS_CLOSED, conn_state_to_string(), conn_type_to_string(), connection_is_listener(), connection_unregister_events(), LD_BUG, LD_NET, connection_t::linked, connection_t::linked_conn_is_closed, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::s, connection_t::state, tor_close_socket(), tor_fragile_assert, and connection_t::type.

Referenced by _connection_write_to_buf_impl(), conn_write_callback(), connection_dir_reached_eof(), connection_exit_connect_dir(), connection_handle_read_impl(), connection_handle_write_impl(), and options_act_reversible().

4.13.3.21 int connection_connect (connection_t ∗ *conn*, const char ∗ *address*, const tor_addr_t ∗ *addr*, uint16_t *port*, int ∗ *socket_error*)

Take conn, make a nonblocking socket; try to connect to addr:port (they arrive in ∗host order∗). If fail, return -1 and if applicable put your best guess about errno into ∗**socket_error**. Else assign s to conn->s: if connected return 1, if EAGAIN return 0.

address is used to make the logs useful.

On success, add conn to the list of polled connections.

References client_check_address_changed(), connection_add(), or_options_t::ConstrainedSockets, or_options_t::ConstrainedSockSize, escaped_safe_str_client(), get_n_open_sockets(), get_options(), LD_CONFIG, LD_NET, or_options_t::OutboundBindAddress, connection_t::s, server_mode(), set_constrained_socket_buffers(), set_socket_nonblocking(), tor_addr_family(), tor_addr_is_loopback(), tor_addr_to_sockaddr(), tor_assert, tor_close_socket(), tor_inet_aton(), tor_open_socket(), and warn_too_many_conns().

Referenced by connection_exit_connect(), connection_or_connect(), and directory_initiate_command_rend().

4.13.3.22 static void connection_consider_empty_read_buckets (connection_t * *conn*) [static]

If we have exhausted our global buckets, or the buckets for conn, stop reading.

References approx_time(), connection_counts_as_relayed_traffic(), connection_stop_reading(), LD_NET, LOG_FN_CONN, OR_CONN_STATE_OPEN, connection_t::read_blocked_on_bw, connection_t::state, and TO_OR_CONN().

Referenced by connection_handle_write_impl(), and connection_read_to_buf().

4.13.3.23 static void connection_consider_empty_write_buckets (connection_t * *conn*) [static]

If we have exhausted our global buckets, or the buckets for conn, stop writing.

References approx_time(), connection_counts_as_relayed_traffic(), connection_stop_writing(), LD_NET, LOG_FN_CONN, OR_CONN_STATE_OPEN, connection_t::state, TO_OR_CONN(), and connection_t::write_blocked_on_bw.

Referenced by connection_handle_write_impl(), and connection_read_to_buf().

4.13.3.24 static int connection_counts_as_relayed_traffic (connection_t * *conn*, time_t *now*) [static]

Return 1 if **conn** should use tokens from the "relayed" bandwidth rates, else 0. Currently, only OR conns with bandwidth class 1, and directory conns that are serving data out, count.

References CLIENT_IDLE_TIME_FOR_PRIORITY, CONN_TYPE_DIR, CONN_TYPE_OR, DIR_CONN_IS_SERVER, TO_OR_CONN(), and connection_t::type.

Referenced by connection_bucket_read_limit(), connection_bucket_refill(), connection_bucket_write_limit(), connection_buckets_decrement(), connection_consider_empty_read_buckets(), and connection_consider_empty_write_buckets().

4.13.3.25 static connection_t * connection_create_listener (struct sockaddr * *listensockaddr*, socklen_t *socklen*, int *type*, char * *address*) [static]

Bind a new non-blocking socket listening to the socket described by **listensockaddr**.

address is only used for logging purposes and to add the information to the conn.

References connection_t::address, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_CONTROL_LISTENER, conn_type_to_string(), connection_add(), connection_free(), connection_new(), connection_start_reading(), dnsserv_configure_listener(), get_n_open_sockets(), get_options(), LD_BUG, LD_NET, LISTENER_STATE_READY, connection_t::port, connection_t::s, set_socket_nonblocking(), connection_t::socket_family, connection_t::state, tor_assert, tor_close_socket(), tor_open_socket(), and warn_too_many_conns().

4.13.3.26 void connection_dump_buffer_mem_stats (int *severity*)

Log how many bytes are used by buffers of different kinds and sizes.

References buf_allocation(), buf_datalen(), and get_connection_array().

Referenced by dumpmemusage().

4.13.3.27 void connection_expire_held_open (void)

Find each connection that has hold_open_until_flushed set to 1 but hasn't written in the past 15 seconds, and set hold_open_until_flushed to 0. This means it will get cleaned up in the next loop through close_if_marked() in **main.c** (p. 690).

References conn_state_to_string(), CONN_TYPE_DIR, CONN_TYPE_EXIT, conn_type_-to_string(), DIR_PURPOSE_SERVER, get_connection_array(), LD_NET, and tor_assert.

Referenced by run_scheduled_events().

4.13.3.28 int connection_fetch_from_buf (char ∗ *string*, size_t *len*, connection_t ∗ *conn*)

A pass-through to fetch_from_buf.

References fetch_from_buf(), and connection_t::inbuf.

Referenced by connection_cpu_process_inbuf(), connection_edge_package_raw_inbuf(), and connection_or_process_cells_from_inbuf().

4.13.3.29 static int connection_finished_connecting (connection_t ∗ *conn*) `[static]`

Called when our attempt to connect() to another server has just succeeded.

This function just passes conn to the connection-specific connection_∗_finished_connecting() function.

References CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_-dir_finished_connecting(), connection_edge_finished_connecting(), connection_or_finished_-connecting(), LD_BUG, TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CONN(), tor_-assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_handle_write_impl().

4.13.3.30 static int connection_finished_flushing (connection_t ∗ *conn*) `[static]`

We just finished flushing bytes from conn->outbuf, and there are no more bytes remaining.

This function just passes conn to the connection-specific connection_∗_finished_flushing() function.

References CONN_IS_CLOSED, CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_-TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_control_finished_flushing(), connection_cpu_finished_flushing(), connection_dir_-finished_flushing(), connection_edge_finished_flushing(), connection_or_finished_flushing(), LD_BUG, TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_-CONN(), tor_assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_handle_read_impl(), and connection_handle_write_impl().

4.13.3.31 static int connection_flushed_some (connection_t * *conn*) `[static]`

Called whenever we've written data on a connection.

References CONN_TYPE_DIR, CONN_TYPE_OR, connection_dirserv_flushed_some(), connection_or_flushed_some(), DIR_CONN_STATE_SERVER_WRITING, connection_t::in_flushed_some, connection_t::state, TO_DIR_CONN(), TO_OR_CONN(), tor_assert, and connection_t::type.

Referenced by connection_handle_read_impl(), and connection_handle_write_impl().

4.13.3.32 void connection_free (connection_t * *conn*)

Make sure **conn** isn't in any of the global conn lists; then free it.

References _connection_free(), CONN_TYPE_CONTROL, connection_in_array(), connection_is_on_closeable_list(), connection_or_remove_from_identity_map(), connection_start_reading(), connection_unregister_events(), control_update_global_event_mask(), control_connection_t::event_mask, LD_BUG, connection_t::linked_conn, connection_t::marked_for_close, connection_t::reading_from_linked_conn, TO_CONTROL_CONN(), TO_OR_CONN(), tor_assert, tor_digest_is_zero(), tor_fragile_assert, and connection_t::type.

Referenced by circuit_free_all(), connection_ap_make_link(), connection_create_listener(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_or_connect(), connection_unlink(), directory_initiate_command_rend(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve(), dnsserv_launch_request(), evdns_server_callback(), purge_expired_resolves(), and spawn_cpuworker().

4.13.3.33 void connection_free_all (void)

Call **_connection_free()** (p. 408) on every connection in our array, and release all storage held by **connection.c** (p. 406). This is used by cpuworkers and dnsworkers when they fork, so they don't keep resources held open (especially sockets).

Don't do the checks in **connection_free()** (p. 417), because they will fail.

References _connection_free(), CONN_TYPE_CONTROL, connection_or_clear_identity_map(), control_update_global_event_mask(), get_connection_array(), smartlist_free(), TO_CONTROL_CONN(), and tor_free.

Referenced by tor_free_all().

4.13.3.34 connection_t* connection_get_by_global_id (uint64_t *id*)

Return the stream with id **id** if it is not already marked for close.

References get_connection_array().

Referenced by connection_cpu_process_inbuf(), and get_stream().

4.13.3.35 connection_t∗ connection_get_by_type (int *type*)

Return a connection of type **type** that is not marked for close.

References get_connection_array().

Referenced by circuit_get_open_circ_or_launch(), hibernate_begin(), and hibernate_go_dormant().

4.13.3.36 connection_t∗ connection_get_by_type_addr_port_purpose (int *type*, const tor_addr_t ∗ *addr*, uint16_t *port*, int *purpose*)

Return a connection with given type, address, port, and purpose; or NULL if no such connection exists.

References get_connection_array(), and tor_addr_eq.

Referenced by consider_testing_reachability(), launch_direct_bridge_descriptor_fetch(), router_pick_trusteddirserver_impl(), and update_v2_networkstatus_cache_downloads().

4.13.3.37 connection_t∗ connection_get_by_type_purpose (int *type*, int *purpose*)

Return an open, non-marked connection of a given type and purpose, or NULL if no such connection exists.

References get_connection_array().

Referenced by update_consensus_networkstatus_downloads(), and update_v2_networkstatus_cache_downloads().

4.13.3.38 connection_t∗ connection_get_by_type_state (int *type*, int *state*)

Return a connection of type **type** that is in state **state**, and that is not marked for close.

References get_connection_array().

Referenced by assign_onionskin_to_cpuworker(), cpuworkers_rotate(), and directory_all_unreachable().

4.13.3.39 connection_t∗ connection_get_by_type_state_rendquery (int *type*, int *state*, const char ∗ *rendquery*)

Return a connection of type **type** that has rendquery equal to **rendquery**, and that is not marked for close. If state is non-zero, conn must be of that state too.

References CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_DIR, CONN_TYPE_EXIT, get_connection_array(), rend_cmp_service_ids(), TO_DIR_CONN(), TO_EDGE_CONN(), and tor_assert.

Referenced by rend_client_remove_intro_point().

4.13.3.40 static int connection_handle_listener_read (connection_t * *conn*, int *new_type*) [static]

The listener connection **conn** told poll() it wanted to read. Call accept() on conn->s, and add the new connection if necessary.

References connection_t::addr, connection_t::address, check_sockaddr(), check_sockaddr_family_match(), CONN_TYPE_AP, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR, connection_add(), connection_free(), connection_init_accepted_conn(), connection_new(), or_options_t::ConstrainedSockets, or_options_t::ConstrainedSockSize, dir_policy_permits_address(), fmt_addr(), get_options(), LD_APP, LD_DIRSERV, LD_NET, connection_t::port, connection_t::s, set_constrained_socket_buffers(), set_socket_nonblocking(), connection_t::socket_family, socks_policy_permits_address(), tor_accept_socket(), tor_addr_copy(), tor_addr_from_sockaddr(), tor_addr_make_unspec(), tor_assert, tor_close_socket(), tor_dup_addr(), connection_t::type, and warn_too_many_conns().

Referenced by connection_handle_read_impl().

4.13.3.41 static int connection_handle_read_impl (connection_t * *conn*) [static]

Read bytes from conn->s and process them.

This function gets called from conn_read() in **main.c** (p. 690), either when poll() has declared that conn wants to read, or (for OR conns) when there are pending TLS bytes.

It calls **connection_read_to_buf()** (p. 423) to bring in any new bytes, and then calls **connection_process_inbuf()** (p. 422) to process them.

Mark the connection and return -1 if you want to close it, else return 0.

References connection_t::active_on_link, approx_time(), buf_datalen(), CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, connection_buckets_decrement(), connection_close_immediate(), connection_edge_end_errno(), connection_finished_flushing(), connection_flushed_some(), connection_handle_listener_read(), connection_is_reading(), connection_or_connect_failed(), connection_process_inbuf(), connection_reached_eof(), connection_read_to_buf(), connection_stop_reading_from_linked_conn(), connection_wants_to_flush(), errno_to_orconn_end_reason(), socks_request_t::has_finished, connection_t::inbuf, connection_t::inbuf_reached_eof, connection_t::linked_conn, connection_t::marked_for_close, OR_CONN_STATE_CONNECTING, connection_t::outbuf, edge_connection_t::socks_request, connection_t::state, connection_t::timestamp_lastread, TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, tor_fragile_assert, and connection_t::type.

4.13.3.42 static int connection_handle_write_impl (connection_t * *conn*, int *force*) [static]

Try to flush more bytes onto conn->s.

This function gets called either from conn_write() in **main.c** (p. 690) when poll() has declared that conn wants to write, or below from connection_write_to_buf() when an entire TLS record is ready.

Update conn->timestamp_lastwritten to now, and call flush_buf or flush_buf_tls appropriately.

If it succeeds and there are no more more bytes on conn->outbuf, then call connection_finished_flushing on it too.

If **force**, then write as many bytes as possible, ignoring bandwidth limits. (Used for flushing messages to controller connections on fatal errors.)

Mark the connection and return -1 if you want to close it, else return 0.

References approx_time(), buf_datalen(), CASE_TOR_TLS_ERROR_ANY, CONN_IS_EDGE, CONN_LOG_PROTECT, CONN_TYPE_AP, CONN_TYPE_OR, connection_bucket_write_limit(), connection_buckets_decrement(), connection_close_immediate(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_edge_end_errno(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_is_listener(), connection_is_reading(), connection_or_connect_failed(), connection_state_is_connecting(), connection_stop_writing(), connection_tls_continue_handshake(), connection_wants_to_flush(), connection_t::dirreq_id, DIRREQ_OR_CONN_BUFFER_FLUSHED, errno_to_orconn_end_reason(), flush_buf(), flush_buf_tls(), geoip_change_dirreq_state(), connection_t::in_flushed_some, LD_BUG, LD_GENERAL, LD_NET, connection_t::marked_for_close, edge_connection_t::n_written, OR_CONN_STATE_PROXY_HANDSHAKING, OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING, OR_CONN_STATE_TLS_HANDSHAKING, OR_CONN_STATE_TLS_SERVER_RENEGOTIATING, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::s, connection_t::state, connection_t::timestamp_lastwritten, or_connection_t::tls, TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, tor_tls_get_n_raw_bytes(), connection_t::type, and connection_t::write_blocked_on_bw.

4.13.3.43 static void connection_init (time_t *now*, connection_t * *conn*, int *type*, int *socket_family*) `[static]`

Initializes conn. (you must call **connection_add()** (p. 694) to link it into the main array).

Set conn->type to **type**. Set conn->s and conn->conn_array_index to -1 to signify they are not yet assigned.

If conn is not a listener type, allocate buffers for it. If it's an AP type, allocate space to store the socks_request.

Assign a pseudorandom next_circ_id between 0 and 2**15.

Initialize conn's timestamps to now.

References buf_new(), connection_t::conn_array_index, CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_is_listener(), connection_t::global_identifier, connection_t::inbuf, connection_t::magic, connection_t::outbuf, connection_t::s, connection_t::socket_family, connection_t::timestamp_created, connection_t::timestamp_lastread, connection_t::timestamp_lastwritten, and connection_t::type.

Referenced by connection_new(), control_connection_new(), dir_connection_new(), edge_connection_new(), and or_connection_new().

4.13.3.44 static int connection_init_accepted_conn (connection_t * *conn*, uint8_t *listener_type*) `[static]`

Initialize states for newly accepted connection **conn**. If conn is an OR, start the TLS handshake. If conn is a transparent AP, get its original destination and place it in circuit_wait.

References AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_NATD_WAIT,

AP_CONN_STATE_SOCKS_WAIT, CONN_TYPE_AP, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_DIR, CONN_TYPE_OR, connection_ap_process_transparent(), connection_start_reading(), connection_tls_start_handshake(), CONTROL_CONN_STATE_NEEDAUTH, control_event_or_conn_status(), DIR_CONN_STATE_SERVER_COMMAND_WAIT, DIR_PURPOSE_SERVER, connection_t::purpose, connection_t::state, TO_EDGE_CONN(), TO_OR_CONN(), and connection_t::type.

Referenced by connection_handle_listener_read().

4.13.3.45 int connection_is_listener (connection_t * *conn*)

Return 1 if **conn** is a listener conn, else return 0.

References CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR_LISTENER, CONN_TYPE_OR_LISTENER, and connection_t::type.

Referenced by _connection_free(), assert_connection_ok(), connection_close_immediate(), connection_handle_write_impl(), connection_init(), and dumpstats().

4.13.3.46 static int connection_is_rate_limited (connection_t * *conn*) `[static]`

Return 1 if we should apply rate limiting to **conn**, and 0 otherwise. Right now this just checks if it's an internal IP address or an internal connection.

References connection_t::addr, connection_t::linked, tor_addr_family(), and tor_addr_is_internal().

Referenced by _connection_write_to_buf_impl(), connection_bucket_read_limit(), connection_bucket_write_limit(), connection_buckets_decrement(), and global_write_bucket_low().

4.13.3.47 void connection_link_connections (connection_t * *conn_a*, connection_t * *conn_b*)

Create a link between **conn_a** and **conn_b**.

References connection_t::linked, connection_t::linked_conn, connection_t::s, and tor_assert.

Referenced by connection_exit_connect_dir(), and directory_initiate_command_rend().

4.13.3.48 connection_t* connection_new (int *type*, int *socket_family*)

Allocate, initialize, and return a new **connection_t** (p. 53) subtype of **type** to make or receive connections of address family **socket_family**. The type should be one of the CONN_TYPE_* constants.

References CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_init(), control_connection_new(), dir_connection_new(), edge_connection_new(), or_connection_new(), and TO_CONN.

Referenced by connection_create_listener(), connection_handle_listener_read(), and spawn_cpuworker().

4.13.3.49 int connection_outbuf_too_full (connection_t ∗ *conn*)

Are there too many bytes on edge connection **conn**'s outbuf to send back a relay-level sendme yet? Return 1 if so, 0 if not. Used by **connection_edge_consider_sending_sendme()** (p. 868).

References CELL_PAYLOAD_SIZE, and connection_t::outbuf_flushlen.

Referenced by connection_edge_consider_sending_sendme().

4.13.3.50 static int connection_process_inbuf (connection_t ∗ *conn*, int *package_partial*) `[static]`

Process new bytes that have arrived on conn->inbuf.

This function just passes conn to the connection-specific connection_∗_process_inbuf() function. It also passes in package_partial if wanted.

References CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_control_process_inbuf(), connection_cpu_process_inbuf(), connection_dir_process_inbuf(), connection_edge_process_inbuf(), connection_or_process_inbuf(), LD_BUG, TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_handle_read_impl().

4.13.3.51 int connection_proxy_connect (connection_t ∗ *conn*, int *type*)

Write a proxy request of **type** (socks4, socks5, https) to conn for conn->addr:conn->port, authenticating with the auth details given in the configuration (if available). SOCKS 5 and HTTP CONNECT proxies support authentication.

Returns -1 if conn->addr is incompatible with the proxy protocol, and 0 otherwise.

Use **connection_read_proxy_handshake()** (p. 423) to complete the handshake.

References connection_t::addr, alloc_http_authenticator(), connection_proxy_state_to_string(), fmt_addr(), get_options(), or_options_t::HttpsProxyAuthenticator, LD_BUG, LD_NET, LD_OR, connection_t::port, connection_t::proxy_state, or_options_t::Socks5ProxyUsername, SOCKS_COMMAND_CONNECT, tor_addr_family(), tor_addr_to_ipv4n(), tor_assert, tor_fragile_assert, tor_free, and tor_snprintf().

Referenced by connection_or_finished_connecting().

4.13.3.52 static const char ∗ connection_proxy_state_to_string (int *state*) `[static]`

Convert state number to string representation for logging purposes.

Referenced by connection_proxy_connect(), and connection_read_proxy_handshake().

4.13.3.53 static int connection_reached_eof (connection_t ∗ *conn*) `[static]`

Callback: invoked when a connection reaches an EOF event.

References CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_control_-

reached_eof(), connection_cpu_reached_eof(), connection_dir_reached_eof(), connection_edge_reached_eof(), connection_or_reached_eof(), LD_BUG, TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CONN(), tor_fragile_assert, and connection_t::type.

Referenced by connection_handle_read_impl().

4.13.3.54 static int connection_read_https_proxy_response (connection_t * *conn*) `[static]`

Read conn's inbuf. If the http response from the proxy is all here, make sure it's good news, then return 1. If it's bad news, return -1. Else return 0 and hope for better luck next time.

References connection_t::address, escaped(), fetch_from_buf_http(), connection_t::inbuf, LD_NET, LD_PROTOCOL, MAX_HEADERS_SIZE, parse_http_response(), and tor_free.

Referenced by connection_read_proxy_handshake().

4.13.3.55 int connection_read_proxy_handshake (connection_t * *conn*)

Call this from connection_*_process_inbuf() to advance the proxy handshake.

No matter what proxy protocol is used, if this function returns 1, the handshake is complete, and the data remaining on inbuf may contain the start of the communication with the requested server.

Returns 0 if the current buffer contains an incomplete response, and -1 on error.

References connection_t::address, connection_proxy_state_to_string(), connection_read_https_proxy_response(), connection_send_socks5_connect(), escaped(), fetch_from_buf_socks_client(), get_options(), connection_t::inbuf, LD_BUG, LD_NET, connection_t::port, connection_t::proxy_state, or_options_t::Socks5ProxyPassword, or_options_t::Socks5ProxyUsername, tor_assert, tor_fragile_assert, and tor_free.

Referenced by connection_or_process_inbuf().

4.13.3.56 static int connection_read_to_buf (connection_t * *conn*, int * *max_to_read*, int * *socket_error*) `[static]`

Pull in new bytes from conn->s or conn->linked_conn onto conn->inbuf, either directly or via TLS. Reduce the token buckets by the number of bytes read.

If *max_to_read is -1, then decide it ourselves, else go with the value passed to us. When returning, if it's changed, subtract the number of bytes we read from *max_to_read.

Return -1 if we want to break conn, else return 0.

References connection_t::address, approx_time(), buf_datalen(), buf_slack(), CASE_TOR_TLS_ERROR_ANY_NONIO, CONN_LOG_PROTECT, CONN_TYPE_AP, connection_bucket_read_limit(), connection_buckets_decrement(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_is_writing(), connection_start_writing(), connection_tls_continue_handshake(), connection_t::inbuf, connection_t::inbuf_reached_eof, LD_BUG, LD_GENERAL, LD_NET, connection_t::linked, connection_t::linked_conn, connection_t::marked_for_close, move_buf_to_buf(), edge_connection_t::n_read, or_connection_t::nickname, OR_CONN_STATE_PROXY_HANDSHAKING, OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING, OR_CONN_STATE_TLS_HANDSHAKING, connection_t::outbuf, connection_t::outbuf_flushlen, read_-

to_buf(), read_to_buf_tls(), connection_t::s, connection_t::state, or_connection_t::tls, or_connection_t::tls_error, TO_EDGE_CONN(), TO_OR_CONN(), tor_tls_err_to_string(), tor_tls_get_n_raw_bytes(), tor_tls_get_pending_bytes(), and connection_t::type.

Referenced by connection_handle_read_impl().

4.13.3.57 static void connection_send_socks5_connect (connection_t * *conn*) `[static]`

Send SOCKS5 CONNECT command to **conn**, copying **conn->addr** and **conn->port** into the request.

References connection_t::addr, connection_t::port, connection_t::proxy_state, SOCKS_COMMAND_CONNECT, tor_addr_family(), tor_addr_to_in6(), and tor_addr_to_ipv4n().

Referenced by connection_read_proxy_handshake().

4.13.3.58 int connection_state_is_connecting (connection_t * *conn*)

Return 1 if conn is in 'connecting' state, else return 0.

References CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, DIR_CONN_STATE_CONNECTING, EXIT_CONN_STATE_CONNECTING, connection_t::marked_for_close, OR_CONN_STATE_CONNECTING, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_handle_write_impl().

4.13.3.59 int connection_state_is_open (connection_t * *conn*)

Return 1 if **conn** is in state "open" and is not marked for close, else return 0.

References AP_CONN_STATE_OPEN, CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_EXIT, CONN_TYPE_OR, CONTROL_CONN_STATE_OPEN, EXIT_CONN_STATE_OPEN, connection_t::marked_for_close, OR_CONN_STATE_OPEN, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_edge_process_relay_cell(), connection_edge_reached_eof(), and run_connection_housekeeping().

4.13.3.60 int connection_wants_to_flush (connection_t * *conn*)

Return conn->outbuf_flushlen: how many bytes conn wants to flush from its outbuf.

References connection_t::outbuf_flushlen.

Referenced by conn_close_if_marked(), connection_edge_finished_connecting(), connection_exit_connect(), connection_handle_read_impl(), and connection_handle_write_impl().

4.13.3.61 control_connection_t* control_connection_new (int *socket_family*)

Allocate and return a new **control_connection_t** (p. 63), initialized as by **connection_init()** (p. 420).

References CONN_TYPE_CONTROL, connection_init(), LD_CONTROL, and TO_CONN.

Referenced by connection_new().

4.13.3.62 static struct sockaddr_in* create_inet_sockaddr (const char * *listenaddress*, uint16_t *listenport*, char ** *readable_address*, socklen_t * *socklen_out*) `[static, read]`

Create an AF_INET listenaddr struct. **listenaddress** provides the host and optionally the port information for the new structure. If no port is provided in **listenaddress** then **listenport** is used.

If not NULL **readable_address** will contain a copy of the host part of **listenaddress**.

The listenaddr struct has to be freed by the caller.

References LD_CONFIG, parse_addr_port(), and tor_free.

4.13.3.63 static struct sockaddr_un* create_unix_sockaddr (const char * *listenaddress*, char ** *readable_address*, socklen_t * *len_out*) `[static, read]`

Create an AF_UNIX listenaddr struct. **listenaddress** provides the path to the Unix socket.

Eventually **listenaddress** will also optionally contain user, group, and file permissions for the new socket. But not yet. XXX Also, since we do not create the socket here the information doesn't help here.

If not NULL **readable_address** will contain a copy of the path part of **listenaddress**.

The listenaddr struct has to be freed by the caller.

4.13.3.64 dir_connection_t* dir_connection_new (int *socket_family*)

Allocate and return a new **dir_connection_t** (p. 79), initialized as by **connection_init()** (p. 420).

References CONN_TYPE_DIR, connection_init(), and TO_CONN.

Referenced by connection_exit_connect_dir(), connection_new(), and directory_initiate_command_rend().

4.13.3.65 edge_connection_t* edge_connection_new (int *type*, int *socket_family*)

Allocate and return a new **edge_connection_t** (p. 90), initialized as by **connection_init()** (p. 420).

References CONN_TYPE_AP, CONN_TYPE_EXIT, connection_init(), edge_connection_t::socks_request, TO_CONN, and tor_assert.

Referenced by connection_ap_make_link(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_new(), dnsserv_launch_request(), and evdns_server_callback().

4.13.3.66 int global_write_bucket_low (connection_t * *conn*, size_t *attempt*, int *priority*)

Return 1 if the global write buckets are low enough that we shouldn't send **attempt** bytes of low-priority directory stuff out to **conn**. Else return 0.

Priority is 1 for v1 requests (directories and running-routers), and 2 for v2 requests (statuses and descriptors). But see FFFF in **directory_handle_command_get()** (p. 575) for why we don't use priority 2 yet.

There are a lot of parameters we could use here:

- global_relayed_write_bucket. Low is bad.
- global_write_bucket. Low is bad.
- bandwidthrate. Low is bad.
- bandwidthburst. Not a big factor?
- attempt. High is bad.
- total bytes queued on outbufs. High is bad. But I'm wary of using this, since a few slow-flushing queues will pump up the number without meaning what we meant to mean. What we really mean is "total directory bytes added to outbufs recently", but that's harder to quantify and harder to keep track of.

References authdir_mode(), or_options_t::BandwidthRate, connection_is_rate_limited(), get_options(), and or_options_t::RelayBandwidthRate.

Referenced by directory_handle_command_get().

4.13.3.67 or_connection_t* or_connection_new (int *socket_family*)

Allocate and return a new **or_connection_t** (p. 139), initialized as by **connection_init()** (p. 420).

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuit_pqueue_last_recalibrated, cell_ewma_get_tick(), CONN_TYPE_OR, connection_init(), crypto_rand_int(), or_connection_t::next_circ_id, smartlist_create(), or_connection_t::timestamp_last_added_nonpadding, and TO_CONN.

Referenced by connection_new(), and connection_or_connect().

4.13.3.68 int retry_all_listeners (smartlist_t * *replaced_conns*, smartlist_t * *new_conns*)

Launch listeners for each port you should have open. Only launch listeners who are not already open, and only close listeners we no longer want.

Add all old conns that should be closed to **replaced_conns**. Add all new connections to **new_conns**.

References or_options_t::ClientOnly, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR_LISTENER, CONN_TYPE_OR_LISTENER, or_options_t::ControlListenAddress, or_options_t::ControlPort, or_options_t::ControlSocket, or_options_t::DirListenAddress, or_options_t::DirPort, or_options_t::DNSListenAddress, or_options_t::DNSPort, get_options(), or_options_t::NatdListenAddress, or_options_t::NatdPort, or_options_t::ORListenAddress, or_options_t::ORPort, retry_listeners(), or_options_t::SocksListenAddress, or_options_t::SocksPort, or_options_t::TransListenAddress, and or_options_t::TransPort.

Referenced by options_act_reversible(), and run_scheduled_events().

4.13.3.69 static int retry_listeners (int *type*, config_line_t * *cfg*, int *port_option*, const char * *default_addr*, smartlist_t * *replaced_conns*, smartlist_t * *new_conns*, int *disable_all_conns*, int *socket_family*) `[static]`

Launch any configured listener connections of type **type**. (A listener is configured if **port_option** is non-zero. If any ListenAddress configuration options are given in **cfg**, create a connection binding to each one. Otherwise, create a single connection binding to the address **default_addr**.)

Only launch the listeners of this type that are not already open, and only close listeners that are no longer wanted. Existing listeners that are still configured are not touched.

If **disable_all_conns** is set, then never open new conns, and close the existing ones.

Add all old conns that should be closed to **replaced_conns**. Add all new connections to **new_conns**.

References connection_t::address, get_connection_array(), config_line_t::key, connection_t::marked_for_close, config_line_t::next, parse_addr_port(), connection_t::port, smartlist_add(), smartlist_create(), connection_t::socket_family, tor_assert, tor_free, connection_t::type, and config_line_t::value.

Referenced by retry_all_listeners().

4.13.3.70 static void set_constrained_socket_buffers (int *sock*, int *size*) `[static]`

Some systems have limited system buffers for recv and xmit on sockets allocated in a virtual server or similar environment. For a Tor server this can produce the "Error creating network socket: No buffer space available" error once all available TCP buffer space is consumed. This method will attempt to constrain the buffers allocated for the socket to the desired size to stay below system TCP buffer limits.

References LD_NET.

Referenced by connection_connect(), and connection_handle_listener_read().

4.13.3.71 static void warn_too_many_conns (void) `[static]`

Warn that an accept or a connect has failed because we're running up against our ulimit. Rate-limit these warnings so that we don't spam the log.

References control_event_general_status(), get_n_open_sockets(), and LD_NET.

Referenced by connection_connect(), connection_create_listener(), and connection_handle_listener_read().

4.13.4 Variable Documentation

4.13.4.1 int global_read_bucket

Max number of bytes I can read this second.

Referenced by do_main_loop(), and second_elapsed_callback().

4.13.4.2 int global_relayed_read_bucket

Max number of relayed (bandwidth class 1) bytes I can read this second.

4.13.4.3 int global_relayed_write_bucket

Max number of relayed (bandwidth class 1) bytes I can write this second.

4.13.4.4 int global_write_bucket

Max number of bytes I can write this second.

Referenced by do_main_loop(), and second_elapsed_callback().

4.13.4.5 uint32_t last_interface_ip = 0 `[static]`

The last IPv4 address that our network interface seemed to have been binding to, in host order. We use this to detect when our IP changes.

Referenced by client_check_address_changed().

4.13.4.6 smartlist_t* outgoing_addrs = NULL `[static]`

A list of uint32_ts for addresses we've used in outgoing connections. Used to detect IP address changes.

4.13.4.7 int write_buckets_empty_last_second = 0 `[static]`

Did either global write bucket run dry last second? If so, we are likely to run dry again this second, so be stingy with the tokens we just put in.

4.14 connection_edge.c File Reference

Handle edge streams.

`#include "or.h"`

`#include <linux/types.h>`

`#include <linux/netfilter_ipv4.h>`

Data Structures

- struct **addressmap_entry_t**
- struct **virtaddress_entry_t**

Defines

- #define **TRANS_NETFILTER**
- #define **SOCKS4_GRANTED** 90
- #define **SOCKS4_REJECT** 91
- #define **TRACKHOSTEXITS_RETRIES** 5

Functions

- static int **connection_ap_handshake_process_socks** (**edge_connection_t** ∗conn)
- static int **connection_ap_process_natd** (**edge_connection_t** ∗conn)
- static int **connection_exit_connect_dir** (**edge_connection_t** ∗exitconn)
- static int **address_is_in_virtual_range** (const char ∗addr)
- static int **consider_plaintext_ports** (**edge_connection_t** ∗conn, uint16_t port)
- static void **clear_trackexithost_mappings** (const char ∗exitname)
- void **_connection_mark_unattached_ap** (**edge_connection_t** ∗conn, int endreason, int line, const char ∗file)
- int **connection_edge_reached_eof** (**edge_connection_t** ∗conn)
- int **connection_edge_process_inbuf** (**edge_connection_t** ∗conn, int package_partial)
- int **connection_edge_destroy** (**circid_t** circ_id, **edge_connection_t** ∗conn)
- static int **relay_send_end_cell_from_edge** (**streamid_t** stream_id, **circuit_t** ∗circ, uint8_t reason, **crypt_path_t** ∗cpath_layer)
- int **connection_edge_end** (**edge_connection_t** ∗conn, uint8_t reason)
- int **connection_edge_end_errno** (**edge_connection_t** ∗conn)
- int **connection_edge_finished_flushing** (**edge_connection_t** ∗conn)
- int **connection_edge_finished_connecting** (**edge_connection_t** ∗edge_conn)
- static int **compute_retry_timeout** (**edge_connection_t** ∗conn)
- void **connection_ap_expire_beginning** (void)
- void **connection_ap_attach_pending** (void)
- void **connection_ap_fail_onehop** (const char ∗failed_digest, **cpath_build_state_t** ∗build_state)
- void **circuit_discard_optional_exit_enclaves** (**extend_info_t** ∗info)
- int **connection_ap_detach_retriable** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ, int reason)
- void **addressmap_init** (void)

- static void **addressmap_ent_free** (void ∗_ent)
- static void **addressmap_virtaddress_ent_free** (void ∗_ent)
- static void **addressmap_virtaddress_remove** (const char ∗address, **addressmap_entry_t** ∗ent)
- static void **addressmap_ent_remove** (const char ∗address, **addressmap_entry_t** ∗ent)
- void **addressmap_clear_configured** (void)
- void **addressmap_clear_transient** (void)
- void **addressmap_clean** (time_t now)
- void **addressmap_free_all** (void)
- int **addressmap_rewrite** (char ∗address, size_t maxlen, time_t ∗expires_out)
- static int **addressmap_rewrite_reverse** (char ∗address, size_t maxlen, time_t ∗expires_out)
- int **addressmap_have_mapping** (const char ∗address, int update_expiry)
- void **addressmap_register** (const char ∗address, char ∗new_address, time_t expires, **addressmap_entry_source_t** source)
- int **client_dns_incr_failures** (const char ∗address)
- void **client_dns_clear_failures** (const char ∗address)
- static void **client_dns_set_addressmap_impl** (const char ∗address, const char ∗name, const char ∗exitname, int ttl)
- void **client_dns_set_addressmap** (const char ∗address, uint32_t val, const char ∗exitname, int ttl)
- static void **client_dns_set_reverse_addressmap** (const char ∗address, const char ∗v, const char ∗exitname, int ttl)
- int **parse_virtual_addr_network** (const char ∗val, int validate_only, char ∗∗msg)
- static char ∗ **addressmap_get_virtual_address** (int type)
- const char ∗ **addressmap_register_virtual_address** (int type, char ∗new_address)
- int **address_is_invalid_destination** (const char ∗address, int client)
- void **addressmap_get_mappings** (**smartlist_t** ∗sl, time_t min_expires, time_t max_expires, int want_expiry)
- int **connection_ap_handshake_rewrite_and_attach** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ, **crypt_path_t** ∗cpath)
- static int **connection_ap_get_original_destination** (**edge_connection_t** ∗conn, **socks_request_t** ∗req)
- int **connection_ap_process_transparent** (**edge_connection_t** ∗conn)
- static **streamid_t** **get_unique_stream_id_by_circ** (**origin_circuit_t** ∗circ)
- int **connection_ap_handshake_send_begin** (**edge_connection_t** ∗ap_conn)
- int **connection_ap_handshake_send_resolve** (**edge_connection_t** ∗ap_conn)
- **edge_connection_t** ∗ **connection_ap_make_link** (char ∗address, uint16_t port, const char ∗digest, int use_begindir, int want_onehop)
- static void **tell_controller_about_resolved_result** (**edge_connection_t** ∗conn, int answer_type, size_t answer_len, const char ∗answer, int ttl, time_t expires)
- void **connection_ap_handshake_socks_resolved** (**edge_connection_t** ∗conn, int answer_type, size_t answer_len, const char ∗answer, int ttl, time_t expires)
- void **connection_ap_handshake_socks_reply** (**edge_connection_t** ∗conn, char ∗reply, size_t replylen, int endreason)
- int **connection_exit_begin_conn** (**cell_t** ∗cell, **circuit_t** ∗circ)
- int **connection_exit_begin_resolve** (**cell_t** ∗cell, **or_circuit_t** ∗circ)
- void **connection_exit_connect** (**edge_connection_t** ∗edge_conn)
- int **connection_edge_is_rendezvous_stream** (**edge_connection_t** ∗conn)
- int **connection_ap_can_use_exit** (**edge_connection_t** ∗conn, **routerinfo_t** ∗exit, int excluded_means_no)
- **hostname_type_t** **parse_extended_hostname** (char ∗address, int allowdotexit)

Variables

- static strmap_t * **addressmap** = NULL
- static strmap_t * **virtaddress_reversemap** = NULL
- static uint32_t **virtual_addr_network** = 0x7fc00000u
- static **maskbits_t** **virtual_addr_netmask_bits** = 10
- static uint32_t **next_virtual_addr** = 0x7fc00000u

4.14.1 Detailed Description

Handle edge streams.

4.14.2 Define Documentation

4.14.2.1 #define TRACKHOSTEXITS_RETRIES 5

How many times do we try connecting with an exit configured via TrackHostExits before concluding that it won't work any more and trying a different one?

4.14.3 Function Documentation

4.14.3.1 void _connection_mark_unattached_ap (edge_connection_t * *conn*, int *endreason*, int *line*, const char * *file*)

An AP stream has failed/finished. If it hasn't already sent back a socks reply, send one now (based on endreason). Also set has_sent_end to 1, and mark the conn.

References edge_connection_t::_base, _connection_mark_for_close(), socks_request_t::command, CONN_TYPE_AP, connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SOCKS_REPLIED, socks_request_t::has_finished, connection_t::hold_open_until_flushed, LD_BUG, connection_t::marked_for_close, edge_connection_t::socks_request, TO_CONN, tor_assert, and connection_t::type.

4.14.3.2 static int address_is_in_virtual_range (const char * *address*) `[static]`

Return true iff **addr** is likely to have been returned by client_dns_get_unused_address.

References addr_mask_cmp_bits(), strcasecmpend(), tor_assert, tor_inet_aton(), virtual_addr_netmask_bits, and virtual_addr_network.

Referenced by addressmap_register(), and addressmap_virtaddress_remove().

4.14.3.3 int address_is_invalid_destination (const char * *address*, int *client*)

Return 1 if **address** has funny characters in it like colons. Return 0 if it's fine, or if we're configured to allow it anyway. **client** should be true if we're using this address as a client; false if we're using it as a server.

References get_options().

Referenced by config_register_addressmaps(), dns_resolve_impl(), and handle_control_mapaddress().

4.14.3.4 void addressmap_clean (time_t *now*)

Clean out entries from the addressmap cache that were added long enough ago that they are no longer valid.

References addressmap_get_mappings().

Referenced by circuit_build_needed_circs().

4.14.3.5 void addressmap_clear_configured (void)

Remove all entries from the addressmap that were set via the configuration file or the command line.

References addressmap_get_mappings().

Referenced by config_register_addressmaps().

4.14.3.6 void addressmap_clear_transient (void)

Remove all entries from the addressmap that are set to expire, ever.

References addressmap_get_mappings().

Referenced by do_hup(), signal_callback(), and signewnym_impl().

4.14.3.7 static void addressmap_ent_free (void * *_ent*) `[static]`

Free the memory associated with the addressmap entry **_ent**.

References addressmap_entry_t::new_address, and tor_free.

Referenced by addressmap_ent_remove(), and addressmap_free_all().

4.14.3.8 static void addressmap_ent_remove (const char * *address*, addressmap_entry_t * *ent*) `[static]`

Remove **ent** (which must be mapped to by **address**) from the client address maps.

References addressmap_ent_free(), and addressmap_virtaddress_remove().

Referenced by addressmap_get_mappings(), addressmap_register(), and clear_trackexithost_mappings().

4.14.3.9 void addressmap_free_all (void)

Free all the elements in the addressmap, and free the addressmap itself.

References addressmap, addressmap_ent_free(), addressmap_virtaddress_ent_free(), strmap_free(), and virtaddress_reversemap.

Referenced by tor_free_all().

4.14.3.10 void addressmap_get_mappings (smartlist_t * *sl*, time_t *min_expires*, time_t *max_expires*, int *want_expiry*)

Iterate over all address mappings which have expiry times between min_expires and max_expires, inclusive. If sl is provided, add an "old-addr new-addr expiry" string to sl for each mapping, omitting the expiry time if want_expiry is false. If sl is NULL, remove the mappings.

References addressmap, addressmap_ent_remove(), addressmap_init(), addressmap_entry_t::expires, format_iso_time(), addressmap_entry_t::new_address, smartlist_add(), strmap_iter_done(), strmap_iter_get(), strmap_iter_init(), strmap_iter_next(), strmap_iter_next_rmv(), and tor_snprintf().

Referenced by addressmap_clean(), addressmap_clear_configured(), addressmap_clear_transient(), and getinfo_helper_events().

4.14.3.11 static char* addressmap_get_virtual_address (int *type*) `[static]`

Return a newly allocated string holding an address of **type** (one of RESOLVED_TYPE_{IPV4|HOSTNAME}) that has not yet been mapped, and that is very unlikely to be the address of any real host.

References addr_mask_cmp_bits(), addressmap, base32_encode(), crypto_rand(), LD_BUG, LD_CONFIG, next_virtual_addr, strmap_get(), tor_assert, tor_inet_ntoa(), virtual_addr_netmask_bits, and virtual_addr_network.

Referenced by addressmap_register_virtual_address().

4.14.3.12 int addressmap_have_mapping (const char * *address*, int *update_expiry*)

Return 1 if **address** is already registered, else return 0. If address is already registered, and **update_expires** is non-zero, then update the expiry time on the mapping with update_expires if it is a mapping created by TrackHostExits.

References addressmap, ADDRMAPSRC_TRACKEXIT, addressmap_entry_t::expires, addressmap_entry_t::source, and strmap_get_lc().

Referenced by consider_recording_trackhost(), and fetch_from_buf_socks().

4.14.3.13 void addressmap_init (void)

Initialize addressmap.

References addressmap, and virtaddress_reversemap.

Referenced by addressmap_get_mappings(), and tor_init().

4.14.3.14 void addressmap_register (const char * *address*, char * *new_address*, time_t *expires*, addressmap_entry_source_t *source*)

Register a request to map **address** to **new_address**, which will expire on **expires** (or 0 if never expires from config file, 1 if never expires from controller, 2 if never expires (virtual address mapping) from the controller.)

new_address should be a newly dup'ed string, which we'll use or free as appropriate. We will leave address alone.

If **new_address** is NULL, or equal to **address**, remove any mappings that exist from **address**.

References address_is_in_virtual_range(), addressmap, addressmap_ent_remove(), addressmap_virtaddress_remove(), control_event_address_mapped(), addressmap_entry_t::expires, LD_APP, LD_CONFIG, addressmap_entry_t::new_address, addressmap_entry_t::num_resolve_failures, safe_str_client(), addressmap_entry_t::source, strmap_get(), strmap_remove(), strmap_set(), and tor_free.

Referenced by addressmap_register_virtual_address(), client_dns_set_addressmap_impl(), config_register_addressmaps(), and handle_control_mapaddress().

4.14.3.15 const char* addressmap_register_virtual_address (int *type*, char * *new_address*)

A controller has requested that we map some address of type **type** to the address **new_address**. Choose an address that is unlikely to be used, and map it, and return it in a newly allocated string. If another address of the same type is already mapped to **new_address**, try to return a copy of that address.

The string in **new_address** may be freed, or inserted into a map as appropriate.

References addressmap, addressmap_get_virtual_address(), addressmap_register(), ADDRMAPSRC_CONTROLLER, virtaddress_entry_t::hostname_address, virtaddress_entry_t::ipv4_address, LD_APP, LD_BUG, addressmap_entry_t::new_address, safe_str_client(), strmap_get(), strmap_set(), tor_assert, tor_free, and virtaddress_reversemap.

Referenced by connection_ap_handshake_rewrite_and_attach(), and handle_control_mapaddress().

4.14.3.16 int addressmap_rewrite (char * *address*, size_t *maxlen*, time_t * *expires_out*)

Look at address, and rewrite it until it doesn't want any more rewrites; but don't get into an infinite loop. Don't write more than maxlen chars into address. Return true if the address changed; false otherwise. Set ∗**expires_out** to the expiry time of the result, or to **time_max** if the result does not expire.

References addressmap, escaped_safe_str_client(), addressmap_entry_t::expires, LD_APP, LD_CONFIG, addressmap_entry_t::new_address, strmap_get(), and tor_free.

Referenced by connection_ap_process_end_not_open().

4.14.3.17 static int addressmap_rewrite_reverse (char * *address*, size_t *maxlen*, time_t * *expires_out*) `[static]`

If we have a cached reverse DNS entry for the address stored in the **maxlen**-byte buffer **address** (typically, a dotted quad) then rewrite to the cached value and return 1. Otherwise return 0. Set ∗**expires_out** to the expiry time of the result, or to **time_max** if the result does not expire.

References addressmap, escaped_safe_str_client(), addressmap_entry_t::expires, LD_APP, addressmap_entry_t::new_address, strmap_get(), tor_free, and tor_snprintf().

4.14.3.18 static void addressmap_virtaddress_ent_free (void * *_ent*) `[static]`

Free storage held by a virtaddress_entry_t∗ entry in **ent**.

References virtaddress_entry_t::hostname_address, virtaddress_entry_t::ipv4_address, and tor_free.

Referenced by addressmap_free_all().

4.14.3.19 static void addressmap_virtaddress_remove (const char * *address*, addressmap_entry_t * *ent*) `[static]`

Free storage held by a virtaddress_entry_t* entry in **ent**.

References address_is_in_virtual_range(), virtaddress_entry_t::hostname_address, virtaddress_entry_t::ipv4_address, addressmap_entry_t::new_address, strmap_get(), strmap_remove(), tor_free, and virtaddress_reversemap.

Referenced by addressmap_ent_remove(), and addressmap_register().

4.14.3.20 void circuit_discard_optional_exit_enclaves (extend_info_t * *info*)

A circuit failed to finish on its last hop **info**. If there are any streams waiting with this exit node in mind, but they don't absolutely require it, make them give up on it.

References socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, edge_connection_t::chosen_exit_retries, clear_trackexithost_mappings(), CONN_TYPE_AP, consider_plaintext_ports(), escaped_safe_str_client(), get_connection_array(), LD_APP, extend_info_t::nickname, socks_request_t::port, router_get_by_nickname(), safe_str_client(), edge_connection_t::socks_request, TO_EDGE_CONN(), tor_assert, and tor_free.

Referenced by circuit_build_failed().

4.14.3.21 static void clear_trackexithost_mappings (const char * *exitname*) `[static]`

Unregister all TrackHostExits mappings from any address to *.exitname.exit.

References addressmap, addressmap_ent_remove(), ADDRMAPSRC_TRACKEXIT, strcmpend(), tor_free, tor_snprintf(), and tor_strlower().

Referenced by circuit_discard_optional_exit_enclaves().

4.14.3.22 void client_dns_clear_failures (const char * *address*)

If **address** is in the client DNS addressmap, reset the number of resolve failures we have on record for it. This is used when we fail a stream because it won't resolve: otherwise future attempts on that address will only try once.

References addressmap, addressmap_entry_t::num_resolve_failures, and strmap_get().

Referenced by connection_ap_process_end_not_open().

4.14.3.23 int client_dns_incr_failures (const char * *address*)

An attempt to resolve **address** failed at some OR. Increment the number of resolve failures we have on record for it, and then return that number.

References addressmap, addressmap_entry_t::expires, LD_APP, MAX_DNS_ENTRY_AGE, addressmap_entry_t::num_resolve_failures, safe_str_client(), strmap_get(), and strmap_set().

Referenced by connection_ap_process_end_not_open().

4.14.3.24 void client_dns_set_addressmap (const char * *address*, uint32_t *val*, const char * *exitname*, int *ttl*)

Record the fact that **address** resolved to **val**. We can now use this in subsequent streams via **addressmap_rewrite()** (p. 434) so we can more correctly choose an exit that will allow **address**.

If **exitname** is defined, then append the addresses with ".exitname.exit" before registering the mapping.

If **ttl** is nonnegative, the mapping will be valid for **ttl**seconds; otherwise, we use the default.

References client_dns_set_addressmap_impl(), INET_NTOA_BUF_LEN, tor_assert, tor_inet_aton(), and tor_inet_ntoa().

Referenced by connection_ap_handshake_socks_resolved(), connection_ap_process_end_not_open(), and connection_edge_process_relay_cell_not_open().

4.14.3.25 static void client_dns_set_addressmap_impl (const char * *address*, const char * *name*, const char * *exitname*, int *ttl*) `[static]`

Record the fact that **address** resolved to **name**. We can now use this in subsequent streams via **addressmap_rewrite()** (p. 434) so we can more correctly choose an exit that will allow **address**.

If **exitname** is defined, then append the addresses with ".exitname.exit" before registering the mapping.

If **ttl** is nonnegative, the mapping will be valid for **ttl**seconds; otherwise, we use the default.

References addressmap_register(), ADDRMAPSRC_DNS, DEFAULT_DNS_TTL, dns_clip_ttl(), INET_NTOA_BUF_LEN, MAX_VERBOSE_NICKNAME_LEN, tor_assert, and tor_snprintf().

Referenced by client_dns_set_addressmap(), and client_dns_set_reverse_addressmap().

4.14.3.26 static void client_dns_set_reverse_addressmap (const char * *address*, const char * *v*, const char * *exitname*, int *ttl*) `[static]`

Add a cache entry noting that **address** (ordinarily a dotted quad) resolved via a RESOLVE_PTR request to the hostname **v**.

If **exitname** is defined, then append the addresses with ".exitname.exit" before registering the mapping.

If **ttl** is nonnegative, the mapping will be valid for **ttl**seconds; otherwise, we use the default.

References client_dns_set_addressmap_impl(), tor_free, and tor_snprintf().

Referenced by connection_ap_handshake_socks_resolved().

4.14.3.27 static int compute_retry_timeout (edge_connection_t * *conn*) `[static]`

Define a schedule for how long to wait between retrying application connections. Rather than waiting a fixed amount of time between each retry, we wait 10 seconds each for the first two tries, and 15 seconds for each retry after that. Hopefully this will improve the expected user experience.

References or_options_t::CircuitStreamTimeout, get_options(), and edge_connection_t::num_socks_retries.

Referenced by connection_ap_expire_beginning().

4.14.3.28 void connection_ap_attach_pending (void)

Tell any AP streams that are waiting for a new circuit to try again, either attaching to an available circ or launching a new one.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, CONN_TYPE_AP, connection_ap_handshake_attach_circuit(), END_STREAM_REASON_CANT_ATTACH, get_connection_array(), connection_t::marked_for_close, and TO_EDGE_CONN().

Referenced by circuit_build_needed_circs(), circuit_has_opened(), rend_client_introcirc_has_opened(), rend_client_receive_rendezvous(), and rend_client_rendezvous_acked().

4.14.3.29 int connection_ap_can_use_exit (edge_connection_t * *conn*, routerinfo_t * *exit*, int *excluded_means_no*)

Return 1 if router **exit** is likely to allow stream **conn** to exit from it, or 0 if it probably will not allow it. (We might be uncertain if conn's destination address has not yet been resolved.)

If **excluded_means_no** is 1 and Exclude*Nodes is set and excludes this relay, return 0.

References edge_connection_t::_base, or_options_t::_ExcludeExitNodesUnion, ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, socks_request_t::address, routerinfo_t::cache_info, edge_connection_t::chosen_exit_name, socks_request_t::command, compare_addr_to_addr_policy(), CONN_TYPE_AP, DIGEST_LEN, routerinfo_t::exit_policy, get_options(), routerinfo_t::has_old_dnsworkers, signed_descriptor_t::identity_digest, policy_is_reject_star(), socks_request_t::port, router_get_by_nickname(), routerset_contains_router(), SOCKS_COMMAND_CONNECT, SOCKS_COMMAND_RESOLVE_PTR, edge_connection_t::socks_request, or_options_t::StrictNodes, tor_assert, tor_inet_aton(), connection_t::type, and edge_connection_t::use_begindir.

Referenced by choose_good_exit_server_general(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_stream_is_being_handled(), and connection_ap_handshake_attach_circuit().

4.14.3.30 int connection_ap_detach_retriable (edge_connection_t * *conn*, origin_circuit_t * *circ*, int *reason*)

The AP connection **conn** has just failed while attaching or sending a BEGIN or resolving on **circ**, but another circuit might work. Detach the circuit, and either reattach it, launch a new circuit, tell the controller, or give up as a appropriate.

Returns -1 on err, 1 on success, 0 on not-yet-sure.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, circuit_detach_stream(), connection_ap_handshake_attach_circuit(), control_event_stream_status(), get_options(), connection_t::state, connection_t::timestamp_lastread, TO_CIRCUIT, and edge_connection_t::use_begindir.

Referenced by connection_ap_expire_beginning(), and connection_ap_process_end_not_open().

4.14.3.31 void connection_ap_expire_beginning (void)

Find all general-purpose AP streams waiting for a response that sent their begin/resolve cell too long ago. Detach from their current circuit, and mark their current circuit as unsuitable for new streams. Then call **connection_ap_handshake_attach_circuit()** (p. 349) to attach to a new circuit (if available) or launch a new one.

For rendezvous streams, simply give up after SocksTimeout seconds (with no retry attempt).

References edge_connection_t::_base, connection_t::addr, socks_request_t::address, AP_CONN_STATE_IS_UNATTACHED, AP_CONN_STATE_OPEN, circuit_get_by_edge_conn(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_REND_JOINED, compute_retry_timeout(), conn_state_to_string(), CONN_TYPE_AP, connection_ap_detach_retriable(), connection_edge_end(), edge_connection_t::cpath_layer, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_CANT_ATTACH, crypt_path_t::extend_info, get_connection_array(), get_options(), LD_APP, LD_REND, connection_t::marked_for_close, or_options_t::MaxCircuitDirtiness, extend_info_t::nickname, edge_connection_t::num_socks_retries, socks_request_t::port, connection_t::port, circuit_t::purpose, safe_str_client(), edge_connection_t::socks_request, or_options_t::SocksTimeout, connection_t::state, connection_t::timestamp_created, circuit_t::timestamp_dirty, connection_t::timestamp_lastread, TO_EDGE_CONN(), TO_ORIGIN_CIRCUIT(), tor_addr_is_null(), and tor_assert.

Referenced by run_scheduled_events().

4.14.3.32 void connection_ap_fail_onehop (const char * *failed_digest*, cpath_build_state_t * *build_state*)

Tell any AP streams that are waiting for a one-hop tunnel to **failed_digest** that they are going to fail.

References extend_info_t::addr, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, cpath_build_state_t::chosen_exit, edge_connection_t::chosen_exit_name, CONN_TYPE_AP, DIGEST_LEN, get_connection_array(), hexdigest_to_digest(), LD_APP, socks_request_t::port, extend_info_t::port, edge_connection_t::socks_request, TO_EDGE_CONN(), tor_addr_eq, tor_addr_from_str(), tor_digest_is_zero(), and edge_connection_t::want_onehop.

Referenced by circuit_build_failed().

4.14.3.33 static int connection_ap_get_original_destination (edge_connection_t * *conn*, socks_request_t * *req*) `[static]`

Fetch the original destination address and port from a system-specific interface and put them into a **socks_request_t** (p. 233) as if they came from a socks request.

Return -1 if an error prevents fetching the destination, else return 0.

References edge_connection_t::_base, connection_t::addr, socks_request_t::address, LD_BUG, LD_NET, connection_t::port, socks_request_t::port, connection_t::s, tor_addr_from_in6(), tor_addr_from_ipv4n(), tor_addr_from_sockaddr(), tor_addr_to_in6(), tor_addr_to_ipv4n(), tor_addr_to_str(), and tor_fragile_assert.

Referenced by connection_ap_process_transparent().

4.14.3.34 static int connection_ap_handshake_process_socks (edge_connection_t * *conn*) `[static]`

connection_edge_process_inbuf() (p. 444) found a conn in state socks_wait. See if conn->inbuf has the right bytes to proceed with the socks handshake.

If the handshake is complete, send it to **connection_ap_handshake_rewrite_and_attach()** (p. 439).

Return -1 if an unexpected error with conn occurs (and mark it for close), else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_SOCKS_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_socks_reply(), control_event_stream_status(), END_STREAM_REASON_FLAG_ALREADY_SOCKS_REPLIED, END_STREAM_REASON_SOCKSPROTOCOL, fetch_from_buf_socks(), get_options(), connection_t::inbuf, LD_APP, or_options_t::LeaveStreamsUnattached, socks_request_t::reply, socks_request_t::replylen, or_options_t::SafeSocks, edge_connection_t::socks_request, connection_t::state, or_options_t::TestSocks, TO_CONN, tor_assert, and connection_t::type.

Referenced by connection_edge_process_inbuf().

4.14.3.35 int connection_ap_handshake_rewrite_and_attach (edge_connection_t * *conn*, origin_circuit_t * *circ*, crypt_path_t * *cpath*)

Connection **conn** just finished its socks handshake, or the controller asked us to take care of it. If **circ** is defined, then that's where we'll want to attach it. Otherwise we have to figure it out ourselves.

First, parse whether it's a .exit address, remap it, and so on. Then if it's for a general circuit, try to attach it to a circuit (or launch one as needed), else if it's for a rendezvous circuit, fetch a rendezvous descriptor first (or attach/launch a circuit if the rendezvous descriptor is already here and fresh enough).

The stream will exit from the hop indicated by **cpath**, or from the last hop in circ's cpath if **cpath** is NULL.

References socks_request_t::address, addressmap_register_virtual_address(), or_options_t::AutomapHostsOnResolve, or_options_t::AutomapHostsSuffixes, socks_request_t::command, escaped_safe_str_client(), get_options(), LD_APP, socks_request_t::port, safe_str_client(), SOCKS_COMMAND_RESOLVE, edge_connection_t::socks_request, strcasecmpend(), tor_assert, tor_inet_aton(), and tor_strlower().

Referenced by connection_ap_handshake_process_socks(), connection_ap_process_natd(), connection_ap_process_transparent(), dnsserv_launch_request(), evdns_server_callback(), and handle_control_attachstream().

4.14.3.36 int connection_ap_handshake_send_begin (edge_connection_t * *ap_conn*)

Write a relay begin cell, using destaddr and destport from ap_conn's socks_request field, and send it down circ.

If ap_conn is broken, mark it for close and return -1. Else return 0.

References origin_circuit_t::_base, edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, origin_circuit_t::build_state, CELL_PAYLOAD_SIZE, CIRCUIT_PURPOSE_C_GENERAL, socks_request_t::command, CONN_TYPE_AP, connection_edge_send_command(), control_event_stream_status(), edge_connection_t::deliver_window, get_unique_stream_id_by_circ(), LD_APP, circuit_t::n_circ_id, edge_connection_t::on_circuit, cpath_build_state_t::onehop_tunnel, edge_connection_t::package_window, socks_request_t::port, circuit_t::purpose, RELAY_PAYLOAD_SIZE, connection_t::s, edge_connection_t::socks_request, connection_t::state, edge_connection_t::stream_id, STREAMWINDOW_START, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), tor_assert, tor_snprintf(), connection_t::type, and edge_connection_t::use_begindir.

Referenced by connection_ap_handshake_attach_chosen_circuit(), and connection_ap_handshake_attach_circuit().

4.14.3.37 int connection_ap_handshake_send_resolve (edge_connection_t * *ap_conn*)

Write a relay resolve cell, using destaddr and destport from ap_conn's socks_request field, and send it down circ.

If ap_conn is broken, mark it for close and return -1. Else return 0.

References origin_circuit_t::_base, edge_connection_t::_base, connection_t::address, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RESOLVE_WAIT, CIRCUIT_PURPOSE_C_GENERAL, socks_request_t::command, CONN_TYPE_AP, connection_edge_send_command(), control_event_stream_status(), get_unique_stream_id_by_circ(), LD_APP, LD_BUG, circuit_t::n_circ_id, edge_connection_t::on_circuit, circuit_t::purpose, REVERSE_LOOKUP_NAME_BUF_LEN, connection_t::s, safe_str_client(), SOCKS_COMMAND_RESOLVE, edge_connection_t::socks_request, connection_t::state, edge_connection_t::stream_id, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), tor_addr_parse_reverse_lookup_name(), tor_addr_to_reverse_lookup_name(), tor_assert, tor_free, and connection_t::type.

Referenced by connection_ap_handshake_attach_chosen_circuit().

4.14.3.38 void connection_ap_handshake_socks_reply (edge_connection_t * *conn*, char * *reply*, size_t *replylen*, int *endreason*)

Send a socks reply to stream **conn**, using the appropriate socks version, etc, and mark **conn** as completed with SOCKS handshaking.

If **reply** is defined, then write **replylen** bytes of it to conn and return, else reply based on **endreason** (one of END_STREAM_REASON_*). If **reply** is undefined, **endreason** can't be 0 or REASON_DONE. Send endreason to the controller, if appropriate.

References control_event_stream_status(), socks_request_t::has_finished, LD_BUG, SOCKS4_NETWORK_LEN, edge_connection_t::socks_request, socks_request_t::socks_version, stream_end_reason_to_socks5_response(), TO_CONN, and tor_assert.

Referenced by _connection_mark_unattached_ap(), connection_ap_handshake_process_socks(), connection_ap_handshake_socks_resolved(), and connection_edge_process_relay_cell_not_open().

4.14.3.39 void connection_ap_handshake_socks_resolved (edge_connection_t * *conn*, int *answer_type*, size_t *answer_len*, const char * *answer*, int *ttl*, time_t *expires*)

Send an answer to an AP connection that has requested a DNS lookup via SOCKS. The type should be one of RESOLVED_TYPE_(IPV4|IPV6|HOSTNAME) or -1 for unreachable; the answer should be in the format specified in the socks extensions document. **ttl** is the ttl for the answer, or -1 on certain errors or for values that didn't come via DNS. **expires** is a time when the answer expires, or -1 or TIME_MAX if there's a good TTL.

References socks_request_t::address, edge_connection_t::chosen_exit_name, client_dns_set_addressmap(), client_dns_set_reverse_addressmap(), connection_ap_handshake_socks_reply(), edge_connection_t::dns_server_request, dnsserv_resolved(), get_uint32(), socks_request_t::has_finished, edge_connection_t::is_dns_request, set_uint16(), SOCKS4_NETWORK_LEN, edge_connection_t::socks_request, socks_request_t::socks_version, tell_controller_about_resolved_result(), and tor_free.

Referenced by _connection_mark_unattached_ap(), and connection_edge_process_relay_cell_not_open().

4.14.3.40 edge_connection_t* connection_ap_make_link (char * *address*, uint16_t *port*, const char * *digest*, int *use_begindir*, int *want_onehop*)

Make an AP **connection_t** (p. 53), make a new linked connection pair, and attach one side to the conn, connection_add it, initialize it to circuit_wait, and call connection_ap_handshake_attach_circuit(conn) on it.

Return the other end of the linked connection pair, or -1 if error.

References edge_connection_t::_base, connection_t::addr, connection_t::address, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, base16_encode(), edge_connection_t::chosen_exit_name, socks_request_t::command, CONN_TYPE_AP, connection_add(), connection_ap_handshake_attach_circuit(), connection_free(), control_event_stream_status(), DIGEST_LEN, edge_connection_new(), END_STREAM_REASON_CANT_ATTACH, socks_request_t::has_finished, HEX_DIGEST_LEN, LD_APP, connection_t::linked, connection_t::marked_for_close, connection_t::port, socks_request_t::port, safe_str_client(), SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, socks_request_t::socks_version, connection_t::state, TO_CONN, tor_addr_make_unspec(), tor_assert, edge_connection_t::use_begindir, and edge_connection_t::want_onehop.

Referenced by directory_initiate_command_rend().

4.14.3.41 static int connection_ap_process_natd (edge_connection_t * *conn*) `[static]`

connection_edge_process_inbuf() (p. 444) found a conn in state natd_wait. See if conn->inbuf has the right bytes to proceed. See FreeBSD's libalias(3) and ProxyEncodeTcpStream() in src/lib/libalias/alias_proxy.c for the encoding form of the original destination.

If the original destination is complete, send it to **connection_ap_handshake_rewrite_and_attach()** (p. 439).

Return -1 if an unexpected error with conn (and it should be marked for close), else return 0.

References edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_ap_handshake_rewrite_and_attach(), control_event_stream_status(), END_STREAM_REASON_INVALID_NATD_DEST, escaped(), fetch_from_buf_line(), get_options(), socks_request_t::has_finished, connection_t::inbuf, LD_APP, or_options_t::LeaveStreamsUnattached, socks_request_t::port, SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, connection_t::state, strcmpstart(), tor_assert, tor_parse_long(), and connection_t::type.

Referenced by connection_edge_process_inbuf().

4.14.3.42 int connection_ap_process_transparent (edge_connection_t * *conn*)

connection_init_accepted_conn() (p. 420) found a new trans AP conn. Get the original destination and send it to **connection_ap_handshake_rewrite_and_attach()** (p. 439).

Return -1 if an unexpected error with conn (and it should be marked for close), else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CONTROLLER_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_ap_get_original_destination(), connection_ap_handshake_rewrite_and_attach(), control_event_stream_status(), END_STREAM_REASON_CANT_FETCH_ORIG_DEST, get_options(), socks_request_t::has_finished, LD_APP, or_options_t::LeaveStreamsUnattached, SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_init_accepted_conn().

4.14.3.43 int connection_edge_destroy (circid_t *circ_id*, edge_connection_t * *conn*)

This edge needs to be closed, because its circuit has closed. Mark it for close and return 0.

References edge_connection_t::_base, CONN_TYPE_AP, control_event_stream_bandwidth(), control_event_stream_status(), edge_connection_t::cpath_layer, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED, connection_t::hold_open_until_flushed, LD_EDGE, connection_t::marked_for_close, edge_connection_t::on_circuit, TO_CONN, and connection_t::type.

Referenced by _circuit_mark_for_close().

4.14.3.44 int connection_edge_end (edge_connection_t * *conn*, uint8_t *reason*)

Send a relay end cell from stream **conn** down conn's circuit, and remember that we've done so. If this is not a client connection, set the relay end cell's reason for closing as **reason**.

Return -1 if this function has already been called on this conn, else return 0.

References edge_connection_t::_base, connection_t::addr, edge_connection_t::address_ttl, circuit_get_by_edge_conn(), CIRCUIT_PURPOSE_IS_CLIENT, connection_edge_is_rendezvous_stream(), connection_edge_send_command(), dns_clip_ttl(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, LD_BUG, LD_EDGE, circuit_t::marked_for_close, connection_t::marked_for_close, connection_t::marked_for_close_file, circuit_t::purpose, RELAY_PAYLOAD_SIZE, connection_t::s, set_uint32(), tor_addr_family(), tor_addr_to_ipv4n(), and tor_fragile_assert.

Referenced by connection_ap_expire_beginning(), connection_edge_end_errno(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_reached_eof(), connection_exit_connect(), connection_exit_connect_dir(), dns_cancel_pending_resolve(), dns_found_answer(), handle_control_attachstream(), hibernate_go_dormant(), and purge_expired_resolves().

4.14.3.45 int connection_edge_end_errno (edge_connection_t * *conn*)

An error has just occurred on an operation on an edge connection **conn**. Extract the errno; convert it to an end reason, and send an appropriate relay end cell to the other end of the connection's circuit.

References edge_connection_t::_base, connection_edge_end(), errno_to_stream_end_reason(), connection_t::s, and tor_assert.

Referenced by conn_read_callback(), connection_exit_connect(), connection_handle_read_impl(), and connection_handle_write_impl().

4.14.3.46 int connection_edge_finished_connecting (edge_connection_t * *edge_conn*)

Connected handler for exit connections: start writing pending data, deliver 'CONNECTED' relay cells as appropriate, and check any pending data that may have been received.

References edge_connection_t::_base, connection_t::addr, connection_t::address, edge_connection_t::address_ttl, CONN_TYPE_EXIT, connection_edge_is_rendezvous_stream(), connection_edge_process_inbuf(), connection_edge_send_command(), connection_start_writing(), connection_wants_to_flush(), connection_watch_events(), dns_clip_ttl(), escaped_safe_str(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, fmt_addr(), edge_connection_t::package_window, connection_t::port, rep_hist_note_exit_stream_opened(), safe_str(), set_uint32(), connection_t::state, TO_CONN, tor_addr_family(), tor_addr_to_ipv4n(), tor_assert, and connection_t::type.

Referenced by connection_finished_connecting().

4.14.3.47 int connection_edge_finished_flushing (edge_connection_t * *conn*)

Connection **conn** has finished writing and has no bytes left on its outbuf.

If it's in state 'open', stop writing, consider responding with a sendme, and return. Otherwise, stop writing and return.

If **conn** is broken, mark it for close and return -1, else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_SOCKS_WAIT, connection_edge_consider_sending_sendme(), connection_stop_writing(), EXIT_CONN_STATE_OPEN, LD_BUG, connection_t::state, TO_CONN, tor_assert, and tor_fragile_assert.

Referenced by connection_finished_flushing().

4.14.3.48 int connection_edge_is_rendezvous_stream (edge_connection_t * *conn*)

Return 1 if **conn** is a rendezvous stream, or 0 if it is a general stream.

References edge_connection_t::rend_data, and tor_assert.

Referenced by ap_stream_wants_exit_attention(), circuit_get_open_circ_or_launch(), connection_ap_handshake_attach_circuit(), connection_ap_process_end_not_open(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_begin_conn(), connection_exit_connect(), relay_lookup_conn(), and write_stream_target_to_buf().

4.14.3.49 int connection_edge_process_inbuf (edge_connection_t * *conn*, int *package_partial*)

Handle new bytes on conn->inbuf based on state:

- If it's waiting for socks info, try to read another step of the socks handshake out of conn->inbuf.
- If it's waiting for the original destination, fetch it.
- If it's open, then package more relay cells from the stream.
- Else, leave the bytes on inbuf alone for now.

Mark and return -1 if there was an unexpected error with the conn, else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_RESOLVE_WAIT, AP_CONN_STATE_SOCKS_WAIT, conn_state_to_string(), connection_ap_handshake_process_socks(), connection_ap_process_natd(), connection_edge_end(), connection_edge_package_raw_inbuf(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, LD_BUG, LD_EDGE, connection_t::state, TO_CONN, tor_assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_edge_finished_connecting(), and connection_process_inbuf().

4.14.3.50 int connection_edge_reached_eof (edge_connection_t * *conn*)

There was an EOF. Send an end and mark the connection for close.

References edge_connection_t::_base, buf_datalen(), connection_edge_end(), connection_state_is_open(), socks_request_t::has_finished, connection_t::inbuf, LD_EDGE, connection_t::marked_for_close, connection_t::s, edge_connection_t::socks_request, and TO_CONN.

Referenced by connection_reached_eof().

4.14.3.51 int connection_exit_begin_conn (cell_t * *cell*, circuit_t * *circ*)

A relay 'begin' or 'begin_dir' cell has arrived, and either we are an exit hop for the circuit, or we are the origin and it is a rendezvous begin.

Launch a new exit connection and initialize things appropriately.

If it's a rendezvous stream, call **connection_exit_connect()** (p. 445) on it.

For general streams, call **dns_resolve()** (p. 635) on it first, and only call **connection_exit_connect()** (p. 445) if the dns answer is already known.

Note that we don't call **connection_add()** (p. 694) on the new stream! We wait for **connection_exit_connect()** (p. 445) to do that.

Return -(some circuit end reason) if we want to tear down **circ**. Else return 0.

References edge_connection_t::_base, or_connection_t::_base, connection_t::addr, connection_t::address, assert_circuit_ok(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_S_REND_JOINED, relay_header_t::command, CONN_TYPE_EXIT, connection_edge_is_rendezvous_stream(), connection_exit_connect(), connection_exit_connect_dir(), connection_free(), connection_or_digest_is_known_relay(), origin_circuit_t::cpath, edge_connection_t::cpath_layer, edge_connection_t::deliver_window, directory_permits_begindir_requests(), circuit_t::dirreq_id, dns_resolve(), edge_connection_new(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_RESOLVEFAILED, EXIT_PURPOSE_CONNECT, get_options(), or_connection_t::identity_digest, or_circuit_t::is_first_hop, LD_BUG, LD_PROTOCOL, LD_REND, relay_header_t::length, edge_connection_t::next_stream, edge_connection_t::on_circuit, or_circuit_t::p_conn, origin_circuit_t::p_streams, edge_connection_t::package_window, parse_addr_port(), cell_t::payload, connection_t::port, crypt_path_t::prev, connection_t::purpose, circuit_t::purpose, or_connection_t::real_addr, RELAY_HEADER_SIZE, relay_header_unpack(), relay_send_end_cell_from_edge(), origin_circuit_t::rend_data, edge_connection_t::rend_data, rend_data_dup(), rend_service_set_connection_addr_port(), safe_str(), server_mode(), connection_t::state, edge_connection_t::stream_id, relay_header_t::stream_id, STREAMWINDOW_START, TO_CONN, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_addr_copy(), tor_addr_is_null(), tor_assert, tor_dup_addr(), tor_free, tor_strlower(), and we_are_hibernating().

Referenced by connection_edge_process_relay_cell().

4.14.3.52 int connection_exit_begin_resolve (cell_t * *cell*, or_circuit_t * *circ*)

Called when we receive a RELAY_COMMAND_RESOLVE cell 'cell' along the circuit **circ**; begin resolving the hostname, and (eventually) reply with a RESOLVED cell.

References edge_connection_t::_base, connection_t::address, assert_circuit_ok(), CONN_TYPE_EXIT, connection_free(), dns_resolve(), edge_connection_new(), EXIT_CONN_STATE_RESOLVEFAILED, EXIT_PURPOSE_RESOLVE, relay_header_t::length, connection_t::marked_for_close, edge_connection_t::on_circuit, cell_t::payload, connection_t::port, connection_t::purpose, RELAY_HEADER_SIZE, relay_header_unpack(), connection_t::state, relay_header_t::stream_id, edge_connection_t::stream_id, TO_CIRCUIT, and TO_CONN.

Referenced by connection_edge_process_relay_cell().

4.14.3.53 void connection_exit_connect (edge_connection_t * *edge_conn*)

Connect to conn's specified addr and port. If it worked, conn has now been added to the connection_array.

Send back a connected cell. Include the resolved IP of the destination address, but *only* if it's a general exit stream. (Rendezvous streams must not reveal what IP they connected to.)

References connection_t::addr, connection_t::address, edge_connection_t::address_ttl, circuit_detach_stream(), circuit_get_by_edge_conn(), connection_connect(), connection_edge_end(), connection_edge_end_errno(), connection_edge_is_rendezvous_stream(), connection_edge_send_command(), connection_free(), connection_wants_to_flush(), connection_watch_events(), dns_clip_ttl(), escaped_safe_str_client(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, LD_BUG, connection_t::port, router_compare_to_my_exit_policy(), set_uint32(), connection_t::state, TO_CONN, tor_addr_family(), and tor_addr_to_ipv4n().

Referenced by connection_exit_begin_conn(), and dns_found_answer().

4.14.3.54 static int connection_exit_connect_dir (edge_connection_t * *exitconn*) [static]

Given an exit conn that should attach to us as a directory server, open a bridge connection with a linked connection pair, create a new directory conn, and join them together. Return 0 on success (or if there was an error we could send back an end cell for). Return -(some circuit end reason) if the circuit needs to be torn down. Either connects **exitconn**, frees it, or marks it, as appropriate.

References dir_connection_t::_base, edge_connection_t::_base, connection_t::addr, connection_t::address, CONN_TYPE_DIR, connection_add(), connection_close_immediate(), connection_edge_end(), connection_edge_send_command(), connection_free(), connection_link_connections(), connection_start_reading(), DIR_CONN_STATE_SERVER_COMMAND_WAIT, dir_connection_new(), DIR_PURPOSE_SERVER, EXIT_CONN_STATE_OPEN, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, connection_t::port, connection_t::purpose, connection_t::state, TO_CONN, TO_OR_CIRCUIT(), tor_addr_copy(), and connection_t::type.

Referenced by connection_exit_begin_conn().

4.14.3.55 static int consider_plaintext_ports (edge_connection_t * *conn*, uint16_t *port*) [static]

Check if **conn** is using a dangerous port. Then warn and/or reject depending on our config options.

References control_event_client_status(), get_options(), LD_APP, or_options_t::RejectPlaintextPorts, smartlist_string_num_isin(), and or_options_t::WarnPlaintextPorts.

Referenced by circuit_discard_optional_exit_enclaves().

4.14.3.56 static streamid_t get_unique_stream_id_by_circ (origin_circuit_t * *circ*) [static]

Iterate over the two bytes of stream_id until we get one that is not already in use; return it. Return 0 if can't get a unique stream_id.

References LD_APP, edge_connection_t::next_stream, origin_circuit_t::next_stream_id, origin_circuit_t::p_streams, and edge_connection_t::stream_id.

Referenced by connection_ap_handshake_send_begin(), and connection_ap_handshake_send_resolve().

4.14.3.57 hostname_type_t parse_extended_hostname (char ∗ *address*, int *allowdotexit*)

If address is of the form "y.onion" with a well-formed handle y: Put a NUL after y, lower-case it, and return ONION_HOSTNAME.

If address is of the form "y.exit" and **allowdotexit** is true: Put a NUL after y and return EXIT_HOSTNAME.

Otherwise: Return NORMAL_HOSTNAME and change nothing.

References LD_APP, REND_SERVICE_ID_LEN_BASE32, and rend_valid_service_id().

4.14.3.58 int parse_virtual_addr_network (const char ∗ *val*, int *validate_only*, char ∗∗ *msg*)

Read a netmask of the form 127.192.0.0/10 from "val", and check whether it's a valid set of virtual addresses to hand out in response to MAPADDRESS requests. Return 0 on success; set ∗msg (if provided) to a newly allocated string and return -1 on failure. If validate_only is false, sets the actual virtual address range to the parsed value.

References addr_mask_cmp_bits(), next_virtual_addr, parse_addr_and_port_range(), virtual_addr_netmask_bits, and virtual_addr_network.

Referenced by options_act().

4.14.3.59 static int relay_send_end_cell_from_edge (streamid_t *stream_id*, circuit_t ∗ *circ*, uint8_t *reason*, crypt_path_t ∗ *cpath_layer*) `[static]`

Send a raw end cell to the stream with ID **stream_id** out over the **circ** towards the hop identified with **cpath_layer**. If this is not a client connection, set the relay end cell's reason for closing as **reason**

References CIRCUIT_PURPOSE_IS_CLIENT, circuit_t::purpose, and relay_send_command_from_edge().

Referenced by connection_exit_begin_conn().

4.14.3.60 static void tell_controller_about_resolved_result (edge_connection_t ∗ *conn*, int *answer_type*, size_t *answer_len*, const char ∗ *answer*, int *ttl*, time_t *expires*) `[static]`

Notify any interested controller connections about a new hostname resolve or resolve error. Takes the same arguments as does **connection_ap_handshake_socks_resolved()** (p. 441).

References socks_request_t::address, control_event_address_mapped(), get_uint32(), INET_NTOA_BUF_LEN, edge_connection_t::socks_request, tor_free, and tor_inet_ntoa().

Referenced by connection_ap_handshake_socks_resolved().

4.14.4 Variable Documentation

4.14.4.1 strmap_t∗ addressmap = NULL `[static]`

A hash table to store client-side address rewrite instructions.

Referenced by addressmap_free_all(), addressmap_get_mappings(), addressmap_get_virtual_address(), addressmap_have_mapping(), addressmap_init(), addressmap_register(), addressmap_register_virtual_address(), addressmap_rewrite(), addressmap_rewrite_reverse(), clear_trackexithost_mappings(), client_dns_clear_failures(), and client_dns_incr_failures().

4.14.4.2 uint32_t next_virtual_addr = 0x7fc00000u `[static]`

What's the next virtual address we will hand out?

Referenced by addressmap_get_virtual_address(), and parse_virtual_addr_network().

4.14.4.3 strmap_t* virtaddress_reversemap = NULL `[static]`

Table mapping addresses to which virtual address, if any, we assigned them to.

We maintain the following invariant: if [A,B] is in virtaddress_reversemap, then B must be a virtual address, and [A,B] must be in addressmap. We do not require that the converse hold: if it fails, then we could end up mapping two virtual addresses to the same address, which is no disaster.

Referenced by addressmap_free_all(), addressmap_init(), addressmap_register_virtual_address(), and addressmap_virtaddress_remove().

4.14.4.4 maskbits_t virtual_addr_netmask_bits = 10 `[static]`

How many bits of **virtual_addr_network** are fixed?

Referenced by address_is_in_virtual_range(), addressmap_get_virtual_address(), and parse_virtual_addr_network().

4.14.4.5 uint32_t virtual_addr_network = 0x7fc00000u `[static]`

Which network should we use for virtual IPv4 addresses? Only the first bits of this value are fixed.

Referenced by address_is_in_virtual_range(), addressmap_get_virtual_address(), and parse_virtual_addr_network().

4.15 connection_or.c File Reference

Functions to handle OR connections, TLS handshaking, and cells on the network.

`#include "or.h"`

Defines

- #define **OR_CONN_HIGHWATER** (32*1024)
- #define **OR_CONN_LOWWATER** (16*1024)
- #define **NEW_CONN_GRACE_PERIOD** (15*60)
- #define **TIME_BEFORE_OR_CONN_IS_TOO_OLD** (60*60*24*7)

Functions

- static int **connection_tls_finish_handshake** (**or_connection_t** *conn)
- static int **connection_or_process_cells_from_inbuf** (**or_connection_t** *conn)
- static int **connection_or_send_versions** (**or_connection_t** *conn)
- static int **connection_init_or_handshake_state** (**or_connection_t** *conn, int started_here)
- static int **connection_or_check_valid_tls_handshake** (**or_connection_t** *conn, int started_here, char *digest_rcvd_out)
- void **connection_or_remove_from_identity_map** (**or_connection_t** *conn)
- void **connection_or_clear_identity_map** (void)
- static void **connection_or_set_identity_digest** (**or_connection_t** *conn, const char *digest)
- void **cell_pack** (**packed_cell_t** *dst, const **cell_t** *src)
- static void **cell_unpack** (**cell_t** *dest, const char *src)
- void **var_cell_pack_header** (const **var_cell_t** *cell, char *hdr_out)
- **var_cell_t** * **var_cell_new** (uint16_t payload_len)
- void **var_cell_free** (**var_cell_t** *cell)
- int **connection_or_reached_eof** (**or_connection_t** *conn)
- int **connection_or_process_inbuf** (**or_connection_t** *conn)
- int **connection_or_flushed_some** (**or_connection_t** *conn)
- int **connection_or_finished_flushing** (**or_connection_t** *conn)
- int **connection_or_finished_connecting** (**or_connection_t** *or_conn)
- int **connection_or_digest_is_known_relay** (const char *id_digest)
- static void **connection_or_init_conn_from_address** (**or_connection_t** *conn, const **tor_addr_t** *addr, uint16_t port, const char *id_digest, int started_here)
- static int **connection_or_is_better** (time_t now, const **or_connection_t** *a, const **or_connection_t** *b, int forgive_new_connections)
- **or_connection_t** * **connection_or_get_for_extend** (const char *digest, const **tor_addr_t** *target_addr, const char **msg_out, int *launch_out)
- static void **connection_or_group_set_badness** (**or_connection_t** *head)
- void **connection_or_set_bad_connections** (void)
- void **connection_or_connect_failed** (**or_connection_t** *conn, int reason, const char *msg)
- **or_connection_t** * **connection_or_connect** (const **tor_addr_t** *_addr, uint16_t port, const char *id_digest)
- int **connection_tls_start_handshake** (**or_connection_t** *conn, int receiving)

- static void **connection_or_tls_renegotiated_cb** (**tor_tls_t** ∗tls, void ∗_conn)
- int **connection_tls_continue_handshake** (**or_connection_t** ∗conn)
- int **connection_or_nonopen_was_started_here** (**or_connection_t** ∗conn)
- void **or_handshake_state_free** (**or_handshake_state_t** ∗state)
- int **connection_or_set_state_open** (**or_connection_t** ∗conn)
- void **connection_or_write_cell_to_buf** (const **cell_t** ∗cell, **or_connection_t** ∗conn)
- void **connection_or_write_var_cell_to_buf** (const **var_cell_t** ∗cell, **or_connection_t** ∗conn)
- static int **connection_fetch_var_cell_from_buf** (**or_connection_t** ∗conn, **var_cell_t** ∗∗out)
- int **connection_or_send_destroy** (**circid_t** circ_id, **or_connection_t** ∗conn, int reason)
- int **is_or_protocol_version_known** (uint16_t v)
- int **connection_or_send_netinfo** (**or_connection_t** ∗conn)

Variables

- static digestmap_t ∗ **orconn_identity_map** = NULL
- static const uint16_t **or_protocol_versions** [ ] = { 1, 2 }
- static const int **n_or_protocol_versions**

4.15.1 Detailed Description

Functions to handle OR connections, TLS handshaking, and cells on the network.

4.15.2 Define Documentation

4.15.2.1 #define OR_CONN_HIGHWATER (32∗1024)

When adding cells to an OR connection's outbuf, keep adding until the outbuf is at least this long, or we run out of cells.

Referenced by connection_or_flushed_some().

4.15.2.2 #define OR_CONN_LOWWATER (16∗1024)

Add cells to an OR connection's outbuf whenever the outbuf's data length drops below this size.

Referenced by connection_or_flushed_some().

4.15.2.3 #define TIME_BEFORE_OR_CONN_IS_TOO_OLD (60∗60∗24∗7)

How old do we let a connection to an OR get before deciding it's too old for new circuits?

Referenced by connection_or_group_set_badness().

4.15.3 Function Documentation

4.15.3.1 void cell_pack (packed_cell_t ∗ *dst*, const cell_t ∗ *src*)

Pack the **cell_t** (p. 30) host-order structure **src** into network-order in the buffer **dest**. See tor-spec.txt for details about the wire format.

Note that this function doesn't touch **dst**->next: the caller should set it or clear it as appropriate.

References packed_cell_t::body, CELL_PAYLOAD_SIZE, cell_t::circ_id, cell_t::command, and cell_t::payload.

Referenced by connection_or_write_cell_to_buf(), and packed_cell_copy().

4.15.3.2 static void cell_unpack (cell_t ∗ *dest*, const char ∗ *src*) `[static]`

Unpack the network-order buffer **src** into a host-order **cell_t** (p. 30) structure **dest**.

References CELL_PAYLOAD_SIZE, cell_t::circ_id, cell_t::command, and cell_t::payload.

Referenced by connection_or_process_cells_from_inbuf().

4.15.3.3 static int connection_fetch_var_cell_from_buf (or_connection_t ∗ *conn*, var_cell_t ∗∗ *out*) `[static]`

See whether there's a variable-length cell waiting on **conn**'s inbuf. Return values as for **fetch_var_cell_from_buf()** (p. 292).

References or_connection_t::_base, fetch_var_cell_from_buf(), connection_t::inbuf, and or_connection_t::link_proto.

Referenced by connection_or_process_cells_from_inbuf().

4.15.3.4 static int connection_init_or_handshake_state (or_connection_t ∗ *conn*, int *started_here*) `[static]`

Allocate a new connection handshake state for the connection **conn**. Return 0 on success, -1 on failure.

References or_connection_t::handshake_state, and or_handshake_state_t::started_here.

Referenced by connection_tls_finish_handshake().

4.15.3.5 static int connection_or_check_valid_tls_handshake (or_connection_t ∗ *conn*, int *started_here*, char ∗ *digest_rcvd_out*) `[static]`

Conn just completed its handshake. Return 0 if all is well, and return -1 if he is lying, broken, or otherwise something is wrong.

If we initiated this connection (**started_here** is true), make sure the other side sent a correctly formed certificate. If I initiated the connection, make sure it's the right guy.

Otherwise (if we _didn't_ initiate this connection), it's okay for the certificate to be weird or absent.

If we return 0, and the certificate is as expected, write a hash of the identity key into digest_rcvd, which must have DIGEST_LEN space in it. (If we return -1 this buffer is undefined.) If the certificate is invalid or missing on an incoming connection, we return 0 and set digest_rcvd to DIGEST_LEN 0 bytes.

As side effects, 1) Set conn->circ_id_type according to tor-spec.txt. 2) If we're an authdirserver and we initiated the connection: drop all descriptors that claim to be on that IP/port but that aren't this guy; and note that this guy is reachable.

References or_connection_t::_base, connection_t::address, authdir_mode_tests_reachability(), base16_encode(), or_connection_t::circ_id_type, CIRC_ID_TYPE_HIGHER, CIRC_ID_TYPE_LOWER, CIRC_ID_TYPE_NEITHER, connection_or_set_identity_digest(), control_event_bootstrap_problem(), control_event_or_conn_status(), crypto_free_pk_env(), crypto_pk_cmp_keys(), crypto_pk_get_digest(), DIGEST_LEN, dirserv_orconn_tls_done(), entry_guard_register_connect_status(), get_identity_key(), get_options(), HEX_DIGEST_LEN, or_connection_t::identity_digest, LD_HANDSHAKE, or_connection_t::nickname, connection_t::port, safe_str_client(), server_mode(), or_connection_t::tls, tor_assert, tor_digest_is_zero(), tor_free, tor_tls_peer_has_cert(), and tor_tls_verify().

Referenced by connection_tls_finish_handshake().

4.15.3.6 void connection_or_clear_identity_map (void)

Remove all entries from the identity-to-orconn map, and clear all identities in OR conns.

References CONN_TYPE_OR, DIGEST_LEN, get_connection_array(), and TO_OR_CONN().

Referenced by connection_free_all().

4.15.3.7 or_connection_t* connection_or_connect (const tor_addr_t * _*addr*, uint16_t *port*, const char * *id_digest*)

Launch a new OR connection to **addr**:**port** and expect to handshake with an OR with identity digest **id_digest**.

If **id_digest** is me, do nothing. If we're already connected to it, return that connection. If the connect() is in progress, set the new conn's state to 'connecting' and return it. If connect() succeeds, call **connection_tls_start_handshake()** (p. 459) on it.

This function is called from router_retry_connections(), for ORs connecting to ORs, and **circuit_establish_circuit()** (p. 312), for OPs connecting to ORs.

Return the launched conn, or NULL if it failed.

References or_connection_t::_base, connection_t::address, connection_connect(), connection_free(), connection_or_connect_failed(), connection_or_finished_connecting(), connection_or_init_conn_from_address(), connection_watch_events(), control_event_or_conn_status(), entry_guard_register_connect_status(), errno_to_orconn_end_reason(), get_options(), or_options_t::HttpsProxy, or_options_t::HttpsProxyAddr, or_options_t::HttpsProxyPort, or_connection_t::identity_digest, LD_PROTOCOL, OR_CONN_STATE_CONNECTING, or_connection_new(), router_digest_is_me(), server_mode(), or_options_t::Socks4Proxy, or_options_t::Socks4ProxyAddr, or_options_t::Socks4ProxyPort, or_options_t::Socks5Proxy, or_options_t::Socks5ProxyAddr, or_options_t::Socks5ProxyPort, connection_t::state, TO_CONN, tor_addr_copy(), and tor_assert.

Referenced by circuit_extend(), circuit_handle_first_hop(), and dirserv_single_reachability_test().

4.15.3.8 void connection_or_connect_failed (or_connection_t * *conn*, int *reason*, const char * *msg*)

conn is in the 'connecting' state, and it failed to complete a TCP connection. Send notifications appropriately.

reason specifies the or_conn_end_reason for the failure; **msg** specifies the strerror-style error message.

References authdir_mode_tests_reachability(), control_event_bootstrap_problem(), control_event_or_conn_status(), and get_options().

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), connection_or_connect(), and run_connection_housekeeping().

4.15.3.9 int connection_or_digest_is_known_relay (const char * *id_digest*)

Return 1 if identity digest **id_digest** is known to be a currently or recently running relay. Otherwise return 0.

References router_get_by_digest(), and router_get_consensus_status_by_id().

Referenced by connection_exit_begin_conn(), and connection_or_init_conn_from_address().

4.15.3.10 int connection_or_finished_connecting (or_connection_t * *or_conn*)

Connected handler for OR connections: begin the TLS handshake.

References connection_t::address, connection_proxy_connect(), connection_start_reading(), connection_tls_start_handshake(), control_event_bootstrap(), get_options(), LD_HANDSHAKE, OR_CONN_STATE_CONNECTING, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::port, connection_t::state, TO_CONN, and tor_assert.

Referenced by connection_finished_connecting(), and connection_or_connect().

4.15.3.11 int connection_or_finished_flushing (or_connection_t * *conn*)

Connection **conn** has finished writing and has no bytes left on its outbuf.

Otherwise it's in state "open": stop writing and return.

If **conn** is broken, mark it for close and return -1, else return 0.

References or_connection_t::_base, assert_connection_ok(), connection_stop_writing(), LD_BUG, OR_CONN_STATE_OPEN, OR_CONN_STATE_OR_HANDSHAKING, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::state, TO_CONN, tor_assert, and tor_fragile_assert.

Referenced by connection_finished_flushing().

4.15.3.12 int connection_or_flushed_some (or_connection_t * *conn*)

Called whenever we have flushed some data on an or_conn: add more data from active circuits.

References or_connection_t::_base, or_connection_t::active_circuits, approx_time(), buf_datalen(), CELL_NETWORK_SIZE, connection_or_flush_from_first_active_circuit(), OR_CONN_HIGHWATER, OR_CONN_LOWWATER, and connection_t::outbuf.

Referenced by connection_flushed_some().

4.15.3.13 or_connection_t* connection_or_get_for_extend (const char * *digest*, const tor_addr_t * *target_addr*, const char ** *msg_out*, int * *launch_out*)

Return the OR connection we should use to extend a circuit to the router whose identity is **digest**, and whose address we believe (or have been told in an extend cell) is **target_addr**. If there is no good connection, set ***msg_out** to a message describing the connection's state and our next action, and set **launch_out** to a boolean for whether we should launch a new connection or not.

References or_connection_t::_base, approx_time(), CONN_TYPE_OR, connection_or_is_better(), DIGEST_LEN, digestmap_get(), or_connection_t::identity_digest, or_connection_t::is_bad_for_new_circs, or_connection_t::is_canonical, or_connection_t::link_proto, connection_t::magic, connection_t::marked_for_close, or_connection_t::next_with_same_id, OR_CONN_STATE_OPEN, orconn_identity_map, or_connection_t::real_addr, connection_t::state, tor_addr_compare(), tor_assert, and connection_t::type.

Referenced by circuit_extend(), and circuit_handle_first_hop().

4.15.3.14 static void connection_or_group_set_badness (or_connection_t * *head*) [static]

Given the head of the linked list for all the or_connections with a given identity, set elements of that list as is_bad_for_new_circs() as appropriate. Helper for **connection_or_set_bad_connections()** (p. 457).

References or_connection_t::_base, connection_t::address, connection_or_is_better(), or_connection_t::is_bad_for_new_circs, or_connection_t::is_canonical, LD_OR, connection_t::marked_for_close, or_connection_t::next_with_same_id, OR_CONN_STATE_OPEN, connection_t::port, or_connection_t::real_addr, connection_t::s, connection_t::state, TIME_BEFORE_OR_CONN_IS_TOO_OLD, connection_t::timestamp_created, and tor_addr_compare().

Referenced by connection_or_set_bad_connections().

4.15.3.15 static void connection_or_init_conn_from_address (or_connection_t * *conn*, const tor_addr_t * *addr*, uint16_t *port*, const char * *id_digest*, int *started_here*) [static]

If we don't necessarily know the router we're connecting to, but we have an addr/port/id_digest, then fill in as much as we can. Start by checking to see if this describes a router we know.

References or_connection_t::_base, routerinfo_t::addr, connection_t::addr, routerinfo_t::address, connection_t::address, or_connection_t::bandwidthburst, or_options_t::BandwidthBurst, or_connection_t::bandwidthrate, or_options_t::BandwidthRate, base16_encode(), connection_or_digest_is_known_relay(), connection_or_set_identity_digest(), DIGEST_LEN, dirserv_get_nickname_by_digest(), get_options(), HEX_DIGEST_LEN, or_connection_t::identity_digest, or_connection_t::is_canonical, networkstatus_get_param(), routerinfo_t::nickname, or_connection_t::nickname, routerinfo_t::or_port, or_options_t::PerConnBWBurst, or_options_t::PerConnBWRate, connection_t::port, or_connection_t::read_bucket, or_connection_t::real_addr, router_get_by_digest(), tor_addr_copy(), tor_addr_eq_ipv4h(), tor_addr_from_ipv4h, tor_dup_addr(), tor_free, and or_connection_t::write_bucket.

Referenced by connection_or_connect(), and connection_tls_finish_handshake().

4.15.3.16 static int connection_or_is_better (time_t *now*, const or_connection_t * *a*, const or_connection_t * *b*, int *forgive_new_connections*) `[static]`

Return true iff **a** is "better" than **b** for new circuits.

A more canonical connection is always better than a less canonical connection. That aside, a connection is better if it has circuits and the other does not, or if it was created more recently.

Requires that both input connections are open; not is_bad_for_new_circs, and not impossibly non-canonical.

If forgive_new_connections is true, then we do not call **a**better than **b** simply because b has no circuits, unless b is also relatively old.

Do not definitively deprecate a new connection with no circuits on it until this much time has passed.

References or_connection_t::_base, or_connection_t::is_canonical, or_connection_t::n_circuits, and connection_t::timestamp_created.

Referenced by connection_or_get_for_extend(), and connection_or_group_set_badness().

4.15.3.17 int connection_or_nonopen_was_started_here (or_connection_t * *conn*)

Return 1 if we initiated this connection, or 0 if it started out as an incoming connection.

References or_connection_t::_base, CONN_TYPE_OR, or_connection_t::handshake_state, or_handshake_state_t::started_here, or_connection_t::tls, tor_assert, tor_tls_is_server(), and connection_t::type.

Referenced by connection_about_to_close_connection(), connection_or_set_state_open(), connection_tls_finish_handshake(), and onionskin_answer().

4.15.3.18 static int connection_or_process_cells_from_inbuf (or_connection_t * *conn*) `[static]`

Process cells from **conn**'s inbuf.

Loop: while inbuf contains a cell, pull it off the inbuf, unpack it, and hand it to **command_process_cell()** (p. 351).

Always return 0.

References or_connection_t::_base, buf_datalen(), CELL_NETWORK_SIZE, cell_unpack(), circ_times, circuit_build_times_network_is_live(), command_process_cell(), command_process_var_cell(), connection_fetch_from_buf(), connection_fetch_var_cell_from_buf(), connection_t::inbuf, LD_OR, connection_t::s, or_connection_t::tls, TO_CONN, tor_tls_get_pending_bytes(), and var_cell_free().

Referenced by connection_or_process_inbuf().

4.15.3.19 int connection_or_process_inbuf (or_connection_t * *conn*)

Handle any new bytes that have come in on connection **conn**. If conn is in 'open' state, hand it to **connection_or_process_cells_from_inbuf()** (p. 455) (else do nothing).

References or_connection_t::_base, connection_or_process_cells_from_inbuf(), connection_read_proxy_handshake(), connection_tls_start_handshake(), OR_CONN_STATE_OPEN, OR_CONN_STATE_OR_HANDSHAKING, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::state, TO_CONN, and tor_assert.

Referenced by connection_process_inbuf().

4.15.3.20 int connection_or_reached_eof (or_connection_t * *conn*)

We've received an EOF from **conn**. Mark it for close and return.

References LD_OR, and TO_CONN.

Referenced by connection_reached_eof().

4.15.3.21 void connection_or_remove_from_identity_map (or_connection_t * *conn*)

If conn is listed in orconn_identity_map, remove it, and clear conn->identity_digest. Otherwise do nothing.

References DIGEST_LEN, digestmap_get(), digestmap_remove(), digestmap_set(), or_connection_t::identity_digest, LD_BUG, or_connection_t::next_with_same_id, or_connection_t::nickname, orconn_identity_map, tor_assert, and tor_digest_is_zero().

Referenced by _connection_free(), connection_free(), connection_or_set_identity_digest(), and connection_unlink().

4.15.3.22 int connection_or_send_destroy (circid_t *circ_id*, or_connection_t * *conn*, int *reason*)

Write a destroy cell with circ ID **circ_id** and reason **reason** onto OR connection **conn**. Don't perform range-checking on reason: we may want to propagate reasons from other cells.

Return 0.

References cell_t::circ_id, cell_t::command, connection_or_write_cell_to_buf(), LD_OR, cell_t::payload, and tor_assert.

Referenced by _circuit_mark_for_close(), command_process_create_cell(), and connection_edge_process_relay_cell().

4.15.3.23 int connection_or_send_netinfo (or_connection_t * *conn*)

Send a NETINFO cell on **conn**, telling the other server what we know about their address, our address, and the current time.

References or_connection_t::_base, routerinfo_t::addr, connection_t::addr, append_address_to_payload(), cell_t::command, connection_or_write_cell_to_buf(), cell_t::payload, router_get_my_routerinfo(), set_uint32(), and tor_addr_from_ipv4h.

Referenced by command_process_versions_cell().

4.15.3.24 static int connection_or_send_versions (or_connection_t * *conn*) `[static]`

Send a VERSIONS cell on **conn**, telling the other host about the link protocol versions that this Tor can support.

References var_cell_t::command, connection_or_write_var_cell_to_buf(), or_connection_t::handshake_state, n_or_protocol_versions, or_protocol_versions, var_cell_t::payload, or_handshake_state_t::sent_versions_at, set_uint16(), tor_assert, var_cell_free(), and var_cell_new().

Referenced by connection_tls_finish_handshake().

4.15.3.25 void connection_or_set_bad_connections (void)

Go through all the OR connections, and set the is_bad_for_new_circs flag on:

- all connections that are too old.
- all open non-canonical connections for which a canonical connection exists to the same router.
- all open canonical connections for which a 'better' canonical connection exists to the same router.
- all open non-canonical connections for which a 'better' non-canonical connection exists to the same router at the same address.

See **connection_or_is_better()** (p. 455) for our idea of what makes one OR connection better than another.

References connection_or_group_set_badness(), and orconn_identity_map.

Referenced by run_scheduled_events().

4.15.3.26 static void connection_or_set_identity_digest (or_connection_t * *conn*, const char * *digest*) `[static]`

Change conn->identity_digest to digest, and add conn into orconn_digest_map.

References connection_or_remove_from_identity_map(), DIGEST_LEN, digestmap_new(), digestmap_set(), or_connection_t::identity_digest, or_connection_t::next_with_same_id, orconn_identity_map, tor_assert, and tor_digest_is_zero().

Referenced by connection_or_check_valid_tls_handshake(), and connection_or_init_conn_from_address().

4.15.3.27 int connection_or_set_state_open (or_connection_t * *conn*)

Set **conn**'s state to OR_CONN_STATE_OPEN, and tell other subsystems as appropriate. Called when we are done with all TLS and OR handshaking.

References or_connection_t::_base, circ_times, circuit_build_times_network_is_live(), circuit_n_conn_done(), connection_or_nonopen_was_started_here(), connection_start_reading(), control_event_or_conn_status(), entry_guard_register_connect_status(), GEOIP_CLIENT_CONNECT, geoip_note_client_seen(), or_connection_t::handshake_state, or_connection_t::identity_digest, LD_OR, OR_CONN_STATE_OPEN, or_handshake_state_free(), rep_hist_note_connect_succeeded(), router_get_by_digest(), router_set_status(), connection_t::state, TO_CONN, tor_addr_family(), and tor_addr_to_ipv4h().

Referenced by command_process_netinfo_cell(), and connection_tls_finish_handshake().

4.15.3.28 static void connection_or_tls_renegotiated_cb (tor_tls_t * *tls*, void * _*conn*) `[static]`

Invoked on the server side from inside **tor_tls_read()** (p. 1227) when the server gets a successful TLS renegotiation from the client.

References connection_tls_finish_handshake(), TO_CONN, tor_tls_block_renegotiation(), and tor_tls_set_renegotiate_callback().

Referenced by connection_tls_continue_handshake().

4.15.3.29 void connection_or_write_cell_to_buf (const cell_t * *cell*, or_connection_t * *conn*)

Pack **cell** into wire-format, and write it onto **conn**'s outbuf. For cells that use or affect a circuit, this should only be called by **connection_or_flush_from_first_active_circuit()** (p. 877).

References approx_time(), packed_cell_t::body, CELL_NETWORK_SIZE, cell_pack(), cell_t::command, or_connection_t::timestamp_last_added_nonpadding, TO_CONN, and tor_assert.

Referenced by connection_or_send_destroy(), connection_or_send_netinfo(), and run_connection_housekeeping().

4.15.3.30 void connection_or_write_var_cell_to_buf (const var_cell_t * *cell*, or_connection_t * *conn*)

Pack a variable-length **cell** into wire-format, and write it onto **conn**'s outbuf. Right now, this *DOES NOT* support cells that affect a circuit.

References approx_time(), var_cell_t::command, var_cell_t::payload, var_cell_t::payload_len, or_connection_t::timestamp_last_added_nonpadding, TO_CONN, tor_assert, VAR_CELL_HEADER_SIZE, and var_cell_pack_header().

Referenced by connection_or_send_versions().

4.15.3.31 int connection_tls_continue_handshake (or_connection_t * *conn*)

Move forward with the tls handshake. If it finishes, hand **conn** to **connection_tls_finish_handshake()** (p. 459).

Return -1 if **conn** is broken, else return 0.

References or_connection_t::_base, CASE_TOR_TLS_ERROR_ANY, connection_or_tls_renegotiated_cb(), connection_start_reading(), connection_start_writing(), connection_stop_writing(), connection_tls_finish_handshake(), LD_OR, OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING, OR_CONN_STATE_TLS_HANDSHAKING, OR_CONN_STATE_TLS_SERVER_RENEGOTIATING, connection_t::state, or_connection_t::tls, TO_CONN, tor_assert, tor_tls_err_to_string(), tor_tls_handshake(), tor_tls_is_server(), tor_tls_renegotiate(), tor_tls_set_renegotiate_callback(), and tor_tls_used_v1_handshake().

Referenced by connection_handle_write_impl(), connection_read_to_buf(), and connection_tls_start_handshake().

4.15.3.32 static int connection_tls_finish_handshake (or_connection_t * *conn*) [static]

The tls handshake is finished.

Make sure we are happy with the person we just handshaked with.

If he initiated the connection, make sure he's not already connected, then initialize conn from the information in router.

If all is successful, call **circuit_n_conn_done()** (p. 315) to handle events that have been pending on the <tls handshake completion. Also set the directory to be dirty (only matters if I'm an authdirserver).

References or_connection_t::_base, connection_t::addr, connection_t::address, circ_times, circuit_build_times_network_is_live(), connection_init_or_handshake_state(), connection_or_check_valid_tls_handshake(), connection_or_init_conn_from_address(), connection_or_nonopen_was_started_here(), connection_or_send_versions(), connection_or_set_state_open(), DIGEST_LEN, directory_set_dirty(), LD_HANDSHAKE, or_connection_t::link_proto, OR_CONN_STATE_OR_HANDSHAKING, connection_t::port, safe_str_client(), connection_t::state, or_connection_t::tls, tor_tls_block_renegotiation(), and tor_tls_used_v1_handshake().

Referenced by connection_or_tls_renegotiated_cb(), and connection_tls_continue_handshake().

4.15.3.33 int connection_tls_start_handshake (or_connection_t * *conn*, int *receiving*)

Begin the tls handshake with **conn**. **receiving** is 0 if we initiated the connection, else it's 1.

Assign a new tls object to conn->tls, begin reading on **conn**, and pass **conn** to **connection_tls_continue_handshake()** (p. 458).

Return -1 if **conn** is broken, else return 0.

References or_connection_t::_base, connection_t::address, connection_start_reading(), connection_tls_continue_handshake(), escaped_safe_str(), LD_BUG, LD_HANDSHAKE, note_crypto_pk_op(), OR_CONN_STATE_TLS_HANDSHAKING, connection_t::s, connection_t::state, or_connection_t::tls, TO_CONN, tor_tls_new(), and tor_tls_set_logged_address().

Referenced by connection_init_accepted_conn(), connection_or_finished_connecting(), and connection_or_process_inbuf().

4.15.3.34 int is_or_protocol_version_known (uint16_t *v*)

Return true iff **v** is a link protocol version that this Tor implementation believes it can support.

References n_or_protocol_versions, and or_protocol_versions.

Referenced by command_process_versions_cell().

4.15.3.35 void or_handshake_state_free (or_handshake_state_t ∗ *state*)

Free all storage held by **state**.

References tor_free.

Referenced by _connection_free(), and connection_or_set_state_open().

4.15.3.36 void var_cell_free (var_cell_t ∗ *cell*)

Release all space held by **cell**.

References tor_free.

Referenced by connection_or_process_cells_from_inbuf(), and connection_or_send_versions().

4.15.3.37 var_cell_t∗ var_cell_new (uint16_t *payload_len*)

Allocate and return a new **var_cell_t** (p. 250) with **payload_len** bytes of payload space.

References var_cell_t::circ_id, var_cell_t::command, and var_cell_t::payload_len.

Referenced by connection_or_send_versions(), and fetch_var_cell_from_buf().

4.15.3.38 void var_cell_pack_header (const var_cell_t ∗ *cell*, char ∗ *hdr_out*)

Write the header of **cell** into the first VAR_CELL_HEADER_SIZE bytes of **hdr_out**.

References var_cell_t::circ_id, var_cell_t::command, var_cell_t::payload_len, and set_uint16().

Referenced by connection_or_write_var_cell_to_buf().

4.15.4 Variable Documentation

4.15.4.1 const int n_or_protocol_versions `[static]`

Initial value:

```
(int)( sizeof(or_protocol_versions)/sizeof(uint16_t) )
```

Number of versions in **or_protocol_versions**.

Referenced by connection_or_send_versions(), and is_or_protocol_version_known().

4.15.4.2 const uint16_t or_protocol_versions[] = { 1, 2 } `[static]`

Array of recognized link protocol versions.

Referenced by connection_or_send_versions(), and is_or_protocol_version_known().

4.15.4.3 digestmap_t* orconn_identity_map = NULL `[static]`

Map from identity digest of connected OR or desired OR to a **connection_t** (p. 53) with that identity digest. If there is more than one such **connection_t** (p. 53), they form a linked list, with next_with_same_id as the next pointer.

Referenced by connection_or_get_for_extend(), connection_or_remove_from_identity_-map(), connection_or_set_bad_connections(), and connection_or_set_identity_digest().

4.16 container.c File Reference

Implements a smartlist (a resizable array) along with helper functions to use smartlists. Also includes hash table implementations of a string-to-void∗ map, and of a digest-to-void∗ map.

`#include "compat.h"`

`#include "util.h"`

`#include "log.h"`

`#include "container.h"`

`#include "crypto.h"`

`#include <stdlib.h>`

`#include <string.h>`

`#include <assert.h>`

`#include "ht.h"`

Defines

- #define **SMARTLIST_DEFAULT_CAPACITY** 16
- #define **LEFT_CHILD**(i) (2∗(i) + 1)
- #define **RIGHT_CHILD**(i) (2∗(i) + 2)
- #define **PARENT**(i) (((i)-1) / 2)
- #define **IDXP**(p) ((int∗)STRUCT_VAR_P(p, idx_field_offset))
- #define **UPDATE_IDX**(i)
- #define **IDX_OF_ITEM**(p) (∗IDXP(p))
- #define **DEFINE_MAP_STRUCTS**(maptype, keydecl, prefix)
- #define **OPTIMIZED_DIGESTMAP_SET**
- #define **IMPLEMENT_ORDER_FUNC**(funcname, elt_t)

Functions

- **smartlist_t** ∗ **smartlist_create** (void)
- void **smartlist_free** (**smartlist_t** ∗sl)
- void **smartlist_clear** (**smartlist_t** ∗sl)
- static INLINE void **smartlist_ensure_capacity** (**smartlist_t** ∗sl, int size)
- void **smartlist_add** (**smartlist_t** ∗sl, void ∗element)
- void **smartlist_add_all** (**smartlist_t** ∗s1, const **smartlist_t** ∗s2)
- void **smartlist_remove** (**smartlist_t** ∗sl, const void ∗element)
- void ∗ **smartlist_pop_last** (**smartlist_t** ∗sl)
- void **smartlist_reverse** (**smartlist_t** ∗sl)
- void **smartlist_string_remove** (**smartlist_t** ∗sl, const char ∗element)
- int **smartlist_isin** (const **smartlist_t** ∗sl, const void ∗element)
- int **smartlist_string_isin** (const **smartlist_t** ∗sl, const char ∗element)
- int **smartlist_string_pos** (const **smartlist_t** ∗sl, const char ∗element)
- int **smartlist_string_isin_case** (const **smartlist_t** ∗sl, const char ∗element)
- int **smartlist_string_num_isin** (const **smartlist_t** ∗sl, int num)
- int **smartlist_digest_isin** (const **smartlist_t** ∗sl, const char ∗element)
- int **smartlist_overlap** (const **smartlist_t** ∗sl1, const **smartlist_t** ∗sl2)

- void **smartlist_intersect** (**smartlist_t** ∗sl1, const **smartlist_t** ∗sl2)
- void **smartlist_subtract** (**smartlist_t** ∗sl1, const **smartlist_t** ∗sl2)
- void **smartlist_del** (**smartlist_t** ∗sl, int idx)
- void **smartlist_del_keeporder** (**smartlist_t** ∗sl, int idx)
- void **smartlist_insert** (**smartlist_t** ∗sl, int idx, void ∗val)
- int **smartlist_split_string** (**smartlist_t** ∗sl, const char ∗str, const char ∗sep, int flags, int max)
- char ∗ **smartlist_join_strings** (**smartlist_t** ∗sl, const char ∗join, int terminate, size_t ∗len_out)
- char ∗ **smartlist_join_strings2** (**smartlist_t** ∗sl, const char ∗join, size_t join_len, int terminate, size_t ∗len_out)
- void **smartlist_sort** (**smartlist_t** ∗sl, int(∗compare)(const void ∗∗a, const void ∗∗b))
- void ∗ **smartlist_get_most_frequent** (const **smartlist_t** ∗sl, int(∗compare)(const void ∗∗a, const void ∗∗b))
- void **smartlist_uniq** (**smartlist_t** ∗sl, int(∗compare)(const void ∗∗a, const void ∗∗b), void(∗free_fn)(void ∗a))
- void ∗ **smartlist_bsearch** (**smartlist_t** ∗sl, const void ∗key, int(∗compare)(const void ∗key, const void ∗∗member))
- int **smartlist_bsearch_idx** (const **smartlist_t** ∗sl, const void ∗key, int(∗compare)(const void ∗key, const void ∗∗member), int ∗found_out)
- static int **_compare_string_ptrs** (const void ∗∗_a, const void ∗∗_b)
- void **smartlist_sort_strings** (**smartlist_t** ∗sl)
- char ∗ **smartlist_get_most_frequent_string** (**smartlist_t** ∗sl)
- void **smartlist_uniq_strings** (**smartlist_t** ∗sl)
- static INLINE void **smartlist_heapify** (**smartlist_t** ∗sl, int(∗compare)(const void ∗a, const void ∗b), int idx_field_offset, int idx)
- void **smartlist_pqueue_add** (**smartlist_t** ∗sl, int(∗compare)(const void ∗a, const void ∗b), int idx_field_offset, void ∗item)
- void ∗ **smartlist_pqueue_pop** (**smartlist_t** ∗sl, int(∗compare)(const void ∗a, const void ∗b), int idx_field_offset)
- void **smartlist_pqueue_remove** (**smartlist_t** ∗sl, int(∗compare)(const void ∗a, const void ∗b), int idx_field_offset, void ∗item)
- void **smartlist_pqueue_assert_ok** (**smartlist_t** ∗sl, int(∗compare)(const void ∗a, const void ∗b), int idx_field_offset)
- static int **_compare_digests** (const void ∗∗_a, const void ∗∗_b)
- void **smartlist_sort_digests** (**smartlist_t** ∗sl)
- void **smartlist_uniq_digests** (**smartlist_t** ∗sl)
- static int **_compare_digests256** (const void ∗∗_a, const void ∗∗_b)
- void **smartlist_sort_digests256** (**smartlist_t** ∗sl)
- char ∗ **smartlist_get_most_frequent_digest256** (**smartlist_t** ∗sl)
- void **smartlist_uniq_digests256** (**smartlist_t** ∗sl)
- **DEFINE_MAP_STRUCTS** (strmap_t, char ∗key, strmap_)
- **DEFINE_MAP_STRUCTS** (digestmap_t, char key[DIGEST_LEN], digestmap_)
- static INLINE int **strmap_entries_eq** (const strmap_entry_t ∗a, const strmap_entry_t ∗b)
- static INLINE unsigned int **strmap_entry_hash** (const strmap_entry_t ∗a)
- static INLINE int **digestmap_entries_eq** (const digestmap_entry_t ∗a, const digestmap_entry_t ∗b)
- static INLINE unsigned int **digestmap_entry_hash** (const digestmap_entry_t ∗a)
- **HT_PROTOTYPE** (HT_GENERATE(strmap_impl, HT_GENERATE(strmap_entry_t, HT_GENERATE(node, HT_GENERATE(strmap_entry_hash, HT_GENERATE(strmap_entries_eq)

- digestmap_t * **digestmap_new** (void)
- void * **strmap_set** (strmap_t *map, const char *key, void *val)
- void * **digestmap_set** (digestmap_t *map, const char *key, void *val)
- void * **strmap_get** (const strmap_t *map, const char *key)
- void * **digestmap_get** (const digestmap_t *map, const char *key)
- void * **strmap_remove** (strmap_t *map, const char *key)
- void * **digestmap_remove** (digestmap_t *map, const char *key)
- void * **strmap_set_lc** (strmap_t *map, const char *key, void *val)
- void * **strmap_get_lc** (const strmap_t *map, const char *key)
- void * **strmap_remove_lc** (strmap_t *map, const char *key)
- strmap_iter_t * **strmap_iter_init** (strmap_t *map)
- digestmap_iter_t * **digestmap_iter_init** (digestmap_t *map)
- strmap_iter_t * **strmap_iter_next** (strmap_t *map, strmap_iter_t *iter)
- digestmap_iter_t * **digestmap_iter_next** (digestmap_t *map, digestmap_iter_t *iter)
- strmap_iter_t * **strmap_iter_next_rmv** (strmap_t *map, strmap_iter_t *iter)
- digestmap_iter_t * **digestmap_iter_next_rmv** (digestmap_t *map, digestmap_iter_t *iter)
- void **strmap_iter_get** (strmap_iter_t *iter, const char **keyp, void **valp)
- void **digestmap_iter_get** (digestmap_iter_t *iter, const char **keyp, void **valp)
- int **strmap_iter_done** (strmap_iter_t *iter)
- int **digestmap_iter_done** (digestmap_iter_t *iter)
- void **strmap_free** (strmap_t *map, void(*free_val)(void *))
- void **digestmap_free** (digestmap_t *map, void(*free_val)(void *))
- void **strmap_assert_ok** (const strmap_t *map)
- void **digestmap_assert_ok** (const digestmap_t *map)
- int **strmap_isempty** (const strmap_t *map)
- int **digestmap_isempty** (const digestmap_t *map)
- int **strmap_size** (const strmap_t *map)
- int **digestmap_size** (const digestmap_t *map)
- **digestset_t** * **digestset_new** (int max_elements)
- void **digestset_free** (**digestset_t** *set)

4.16.1 Detailed Description

Implements a smartlist (a resizable array) along with helper functions to use smartlists. Also includes hash table implementations of a string-to-void* map, and of a digest-to-void* map.

4.16.2 Define Documentation

4.16.2.1 #define DEFINE_MAP_STRUCTS(maptype, keydecl, prefix)

Value:

```
typedef struct prefix ## entry_t {                          \
    HT_ENTRY(prefix ## entry_t) node;                         \
    void *val;                                                \
    keydecl;                                                  \
  } prefix ## entry_t;                                        \
  struct maptype {                                            \
    HT_HEAD(prefix ## impl, prefix ## entry_t) head;          \
  }
```

Helper: Declare an entry type and a map type to implement a mapping using **ht.h** (p. ??). The map type will be called **maptype**. The key part of each entry is declared using the C declaration **keydecl**. All functions and types associated with the map get prefixed with **prefix**

4.16.2.2 #define IMPLEMENT_ORDER_FUNC(funcname, elt_t)

Value:

```
static int                                                     \
  _cmp_ ## elt_t(const void *_a, const void *_b)                 \
  {                                                              \
    const elt_t *a = _a, *b = _b;                                \
    if (*a<*b)                                                   \
      return -1;                                                 \
    else if (*a>*b)                                              \
      return 1;                                                  \
    else                                                         \
      return 0;                                                  \
  }                                                              \
  elt_t                                                          \
  funcname(elt_t *array, int n_elements, int nth)                \
  {                                                              \
    tor_assert(nth >= 0);                                        \
    tor_assert(nth < n_elements);                                \
    qsort(array, n_elements, sizeof(elt_t), _cmp_ ##elt_t);      \
    return array[nth];                                           \
  }
```

Declare a function called **funcname** that acts as a find_nth_FOO function for an array of type **elt_t***.

NOTE: The implementation kind of sucks: It's O(n log n), whereas finding the kth element of an n-element list can be done in O(n). Then again, this implementation is not in critical path, and it is obviously correct.

4.16.2.3 #define SMARTLIST_DEFAULT_CAPACITY 16

All newly allocated smartlists have this capacity.

Referenced by smartlist_create().

4.16.2.4 #define UPDATE_IDX(i)

Value:

```
do {                             \
    void *updated = sl->list[i];                        \
    *IDXP(updated) = i;                                 \
  } while (0)
```

4.16.3 Function Documentation

4.16.3.1 static int _compare_digests (const void ** _*a*, const void ** _*b*) [static]

Helper: compare two DIGEST_LEN digests.

References DIGEST_LEN.

Referenced by smartlist_sort_digests(), and smartlist_uniq_digests().

4.16.3.2 static int _compare_digests256 (const void ** _ *a*, const void ** _ *b*) [static]

Helper: compare two DIGEST256_LEN digests.

References DIGEST256_LEN.

Referenced by smartlist_get_most_frequent_digest256(), smartlist_sort_digests256(), and smartlist_uniq_digests256().

4.16.3.3 static int _compare_string_ptrs (const void ** _ *a*, const void ** _ *b*) [static]

Helper: compare two const char **s.

Referenced by smartlist_get_most_frequent_string(), smartlist_sort_strings(), and smartlist_uniq_strings().

4.16.3.4 void digestmap_assert_ok (const digestmap_t * *map*)

Fail with an assertion error if anything has gone wrong with the internal representation of **map**.

References tor_assert.

4.16.3.5 static INLINE int digestmap_entries_eq (const digestmap_entry_t * *a*, const digestmap_entry_t * *b*) [static]

Helper: compare digestmap_entry_t objects by key value.

References DIGEST_LEN.

4.16.3.6 static INLINE unsigned int digestmap_entry_hash (const digestmap_entry_t * *a*) [static]

Helper: return a hash value for a digest_map_t.

Referenced by digestmap_set().

4.16.3.7 void digestmap_free (digestmap_t * *map*, void(*)(void *) *free_val*)

Remove all entries from **map**, and deallocate storage for those entries. If free_val is provided, it is invoked on every value in **map**.

References tor_assert, and tor_free.

Referenced by dirserv_free_all(), dirserv_free_fingerprint_list(), download_status_map_update_from_v2_networkstatus(), entry_guards_compute_status(), entry_guards_parse_state(), free_or_history(), networkstatus_free_all(), networkstatus_vote_free(), rend_cache_free_all(), rend_service_free(), rep_hist_free_all(), routerlist_free_all(), routerset_free(), and update_extrainfo_downloads().

4.16.3.8 void* digestmap_get (const digestmap_t * *map*, const char * *key*)

Like **strmap_get()** (p. 481) above but for digestmaps.

References DIGEST_LEN, and tor_assert.

Referenced by add_fingerprint_to_dir(), authority_cert_dl_failed(), authority_cert_get_by_digests(), authority_cert_get_newest_by_id(), authority_certs_fetch_missing(), connection_or_get_for_extend(), connection_or_remove_from_identity_map(), dirserv_get_networkstatus_v2(), dirserv_get_nickname_by_digest(), dirserv_get_status_impl(), dirserv_set_cached_networkstatus_v2(), download_status_map_update_from_v2_networkstatus(), entry_guards_compute_status(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), get_cert_list(), get_link_history(), get_or_history(), lookup_cached_dir_by_fp(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store_v2_desc_as_dir(), rend_service_introduce(), router_get_dl_status_by_descriptor_digest(), routerset_contains(), signed_descs_update_status_from_consensus_networkstatus(), update_extrainfo_downloads(), and update_router_descriptor_cache_downloads_v2().

4.16.3.9 int digestmap_isempty (const digestmap_t * *map*)

Return true iff **map** has no entries.

Referenced by rep_hist_dump_stats().

4.16.3.10 int digestmap_iter_done (digestmap_iter_t * *iter*)

Return true iff **iter** has advanced past the last entry of **map**.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), rend_cache_clean_v2_descs_as_dir(), rep_hist_downrate_old_runs(), rep_hist_dump_stats(), rep_hist_record_mtbf_data(), and rep_history_clean().

4.16.3.11 void digestmap_iter_get (digestmap_iter_t * *iter*, const char ** *keyp*, void ** *valp*)

Set ***keyp** and ***valp** to the current entry pointed to by iter.

References tor_assert.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), rend_cache_clean_v2_descs_as_dir(), rep_hist_downrate_old_runs(), rep_hist_dump_stats(), rep_hist_record_mtbf_data(), and rep_history_clean().

4.16.3.12 digestmap_iter_t* digestmap_iter_init (digestmap_t * *map*)

Start iterating through **map**. See **strmap_iter_init()** (p. 482) for example.

References tor_assert.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), rend_cache_clean_v2_descs_as_dir(), rep_hist_downrate_old_runs(), rep_hist_dump_stats(), rep_hist_record_mtbf_data(), and rep_history_clean().

4.16.3.13 digestmap_iter_t* digestmap_iter_next (digestmap_t * *map*, digestmap_iter_t * *iter*)

Advance the iterator **iter** for map a single step to the next entry, and return its new value.

References tor_assert.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), rend_cache_clean_v2_descs_as_dir(), rep_hist_downrate_old_runs(), rep_hist_dump_stats(), rep_hist_record_mtbf_data(), and rep_history_clean().

4.16.3.14 digestmap_iter_t* digestmap_iter_next_rmv (digestmap_t * *map*, digestmap_iter_t * *iter*)

Advance the iterator **iter** a single step to the next entry, removing the current entry, and return its new value.

References tor_assert, and tor_free.

Referenced by rend_cache_clean_v2_descs_as_dir(), and rep_history_clean().

4.16.3.15 digestmap_t* digestmap_new (void)

Constructor to create a new empty map from digests to void*'s.

Referenced by authdir_config_new(), authority_certs_fetch_missing(), connection_or_set_identity_digest(), dirserv_get_networkstatus_v2(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), download_status_map_update_from_v2_networkstatus(), dump_distinct_digest_count(), entry_guards_compute_status(), entry_guards_parse_state(), get_cert_list(), get_or_history(), get_possible_sybil_list(), rend_cache_init(), rend_service_introduce(), rep_hist_downrate_old_runs(), rep_hist_init(), router_get_consensus_status_by_descriptor_digest(), router_parse_entry_from_string(), routerset_new(), update_consensus_router_descriptor_downloads(), update_extrainfo_downloads(), and update_router_descriptor_cache_downloads_v2().

4.16.3.16 void* digestmap_remove (digestmap_t * *map*, const char * *key*)

Like **strmap_remove()** (p. 483) above but for digestmaps.

References DIGEST_LEN, tor_assert, and tor_free.

Referenced by connection_or_remove_from_identity_map(), dirserv_set_cached_networkstatus_v2(), and download_status_map_update_from_v2_networkstatus().

4.16.3.17 void* digestmap_set (digestmap_t * *map*, const char * *key*, void * *val*)

Like **strmap_set()** (p. 483) above but for digestmaps.

References DIGEST_LEN, digestmap_entry_hash(), and tor_assert.

Referenced by add_fingerprint_to_dir(), connection_or_remove_from_identity_map(), connection_or_set_identity_digest(), dirserv_set_cached_networkstatus_v2(), download_status_map_update_from_v2_networkstatus(), entry_guards_compute_status(), entry_guards_parse_state(), get_cert_list(), get_link_history(), get_or_history(), get_possible_sybil_list(), rend_cache_store_v2_desc_as_dir(), rend_service_introduce(), router_get_-

consensus_status_by_descriptor_digest(), router_parse_entry_from_string(), and routerset_parse().

4.16.3.18 int digestmap_size (const digestmap_t * *map*)

Return the number of items in **map**.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_status_impl(), dirserv_set_cached_networkstatus_v2(), and dump_distinct_digest_count().

4.16.3.19 void digestset_free (digestset_t * *set*)

Free all storage held in **set**.

References tor_free.

Referenced by routerlist_remove_old_routers().

4.16.3.20 digestset_t* digestset_new (int *max_elements*)

Return a newly allocated **digestset_t** (p. 78), optimized to hold a total of **max_elements** digests with a reasonably low false positive weight.

References tor_log2().

Referenced by routerlist_remove_old_routers().

4.16.3.21 HT_PROTOTYPE (HT_GENERATE(*strmap_impl*, HT_GENERATE(*strmap_entry_t*, HT_GENERATE(*node*, HT_GENERATE(*strmap_entry_hash*, HT_GENERATE(*strmap_entries_eq*)

Constructor to create a new empty map from strings to void*'s.

4.16.3.22 void smartlist_add (smartlist_t * *sl*, void * *element*)

Append element to the end of the list.

References smartlist_t::list, smartlist_t::num_used, and smartlist_ensure_capacity().

Referenced by add_an_entry_guard(), add_connection_to_closeable_list(), add_nickname_list_to_smartlist(), add_predicted_port(), add_trusted_dir_server(), add_wildcarded_test_address(), addressmap_get_mappings(), append_bytes_to_file(), authority_cert_get_all(), authority_certs_fetch_missing(), authority_type_to_string(), bridge_add_from_config(), choose_good_entry_server(), choose_good_exit_server_general(), choose_good_middle_server(), choose_random_entry(), circuit_get_all_pending_on_or_conn(), circuit_get_unhandled_ports(), circuit_list_path_impl(), circuit_set_state(), client_check_address_changed(), config_dump(), connection_add(), connection_dir_client_reached_eof(), connection_start_reading_from_linked_conn(), control_event_descriptors_changed(), control_event_networkstatus_changed_helper(), control_event_networkstatus_changed_single(), control_setconf_helper(), decode_hashed_passwords(), dir_split_resource_into_fingerprint_pairs(), directory_get_consensus_url(), directory_handle_command_get(), directory_post_to_hs_dir(), dirserv_add_descriptor(), dirserv_get_networkstatus_v2(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirvote_add_signatures(), dirvote_compute_consensuses(), dirvote_compute_params(), dirvote_fetch_missing_votes(), entry_guards_parse_state(), entry_guards_prepend_from_config(), find_all_exitpolicy(), geoip_add_entry(), geoip_get_client_history(), geoip_get_dirreq_history(), geoip_get_request_history(), geoip_load_file(), get_detached_signatures_from_pending_consensuses(), get_frequent_members(), getinfo_helper_config(), getinfo_helper_dir(), getinfo_helper_entry_guards(), getinfo_helper_events(), getinfo_helper_networkstatus(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_mapaddress(), handle_control_protocolinfo(), handle_control_resolve(), hid_serv_get_responsible_directories(), list_getinfo_options(), list_server_status_v1(), list_v3_auth_ids(), log_credential_status(), log_entry_guards(), make_consensus_method_list(), microdesc_cache_rebuild(), microdescs_parse_from_string(), networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), networkstatus_getinfo_by_purpose(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), notify_control_networkstatus_changed(), options_init_logs(), options_validate(), parse_addr_policy(), policies_set_router_exitpolicy_to_reject_all(), policy_expand_private(), policy_summarize(), policy_summary_create(), relay_send_command_from_edge(), rend_add_service(), rend_config_services(), rend_encode_v2_descriptors(), rend_encrypt_v2_intro_points_basic(), rend_parse_introduction_points(), rend_parse_service_descriptor(), rend_service_set_connection_addr_port(), rend_service_update_descriptor(), rend_services_introduce(), rep_hist_buffer_stats_add_circ(), rep_hist_buffer_stats_write(), rep_hist_get_router_stability_doc(), retry_listeners(), router_add_exit_policy(), router_add_running_routers_to_smartlist(), router_choose_random_node(), router_load_routers_from_string(), router_load_single_router(), router_parse_entry_from_string(), router_parse_list_from_string(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), router_rebuild_descriptor(), router_rebuild_store(), router_set_networkstatus_v2(), routerlist_add_family(), routerlist_add_network_family(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_replace(), routerset_get_all_routers(), routerset_parse(), routersets_get_disjunction(), smartlist_insert(), smartlist_pqueue_add(), smartlist_split_string(), tokenize_string(), tor_listdir(), tor_tls_client_is_using_v2_ciphers(), trusted_dirs_flush_certs_to_disk(), trusted_dirs_load_certs_from_string(), update_consensus_router_descriptor_downloads(), update_extrainfo_downloads(), upload_service_descriptor(), wildcard_increment_answer(), wrap_string(), and write_bytes_to_file().

4.16.3.23 void smartlist_add_all (smartlist_t * *s1*, const smartlist_t * *s2*)

Append each element from S2 to the end of S1.

References smartlist_t::list, smartlist_t::num_used, smartlist_ensure_capacity(), and tor_assert.

Referenced by compute_consensus_method(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), dirvote_clear_votes(), dirvote_compute_params(), get_possible_sybil_list(), handle_control_authenticate(), networkstatus_compute_consensus(), parse_addr_policy(), rend_config_services(), and routerset_parse().

4.16.3.24 void* smartlist_bsearch (smartlist_t * *sl*, const void * *key*, int(*)(const void *key, const void **member) *compare*)

Assuming the members of **sl** are in order, return a pointer to the member that matches **key**. Ordering and matching are defined by a **compare** function that returns 0 on a match; less than 0 if key is less than member, and greater than 0 if key is greater then member.

References smartlist_bsearch_idx().

Referenced by geoip_get_country_by_ip(), measured_bw_line_apply(), networkstatus_v2_find_entry(), networkstatus_vote_find_entry(), and router_get_consensus_status_by_id().

4.16.3.25 int smartlist_bsearch_idx (const smartlist_t * *sl*, const void * *key*, int(*)(const void *key, const void **member) *compare*, int * *found_out*)

Assuming the members of **sl** are in order, return the index of the member that matches **key**. If no member matches, return the index of the first member greater than **key**, or smartlist_len(sl) if no member is greater than **key**. Set **found_out** to true on a match, to false otherwise. Ordering and matching are defined by a **compare** function that returns 0 on a match; less than 0 if key is less than member, and greater than 0 if key is greater then member.

References smartlist_t::list, and tor_assert.

Referenced by networkstatus_vote_find_entry_idx(), and smartlist_bsearch().

4.16.3.26 void smartlist_clear (smartlist_t * *sl*)

Remove all elements from the list.

References smartlist_t::num_used.

Referenced by choose_good_exit_server_general(), choose_random_entry(), clear_bridge_list(), clear_trusted_dir_servers(), client_check_address_changed(), compute_consensus_method(), config_assign_value(), config_register_addressmaps(), connection_or_unlink_all_active_circs(), directory_post_to_hs_dir(), dirserv_add_multiple_descriptors(), dirvote_clear_votes(), dirvote_compute_consensuses(), dns_reset_correctness_checks(), handle_control_mapaddress(), microdescs_parse_from_string(), networkstatus_v2_parse_from_string(), options_init_logs(), options_validate(), parse_addr_policy(), rend_config_services(), rend_encode_v2_descriptors(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_service_authorization(), rend_services_introduce(), rep_hist_buffer_stats_write(), router_load_routers_from_string(), router_parse_directory(), router_reset_warnings(), routerlist_reset_warnings(), routerstatus_parse_entry_from_string(), and upload_service_descriptor().

4.16.3.27 smartlist_t* smartlist_create (void)

Allocate and return an empty smartlist.

References smartlist_t::capacity, smartlist_t::list, smartlist_t::num_used, and SMARTLIST_DEFAULT_CAPACITY.

Referenced by accounting_parse_options(), add_nickname_list_to_smartlist(), add_trusted_dir_server(), add_wildcarded_test_address(), append_bytes_to_file(), authority_cert_parse_from_string(), authority_certs_fetch_missing(), authority_type_to_string(), bridge_add_from_config(), bridges_retry_helper(), check_nickname_list(), choose_good_entry_server(), choose_good_exit_server_general(), choose_good_middle_server(), choose_-

random_entry(), circuit_build_times_parse_state(), circuit_count_pending_on_or_conn(), circuit_get_unhandled_ports(), circuit_list_path_impl(), circuit_n_conn_done(), circuit_set_state(), clear_bridge_list(), clear_trusted_dir_servers(), client_check_address_changed(), client_likes_consensus(), compute_consensus_method(), compute_consensus_versions_list(), config_assign_value(), config_dump(), config_register_addressmaps(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), control_event_descriptors_changed(), control_event_networkstatus_changed_helper(), control_event_networkstatus_changed_single(), control_setconf_helper(), decode_hashed_passwords(), detached_get_signatures(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_get_consensus_url(), directory_get_from_hs_dir(), directory_handle_command_get(), directory_post_to_hs_dir(), dirserv_add_descriptor(), dirserv_add_multiple_descriptors(), dirserv_get_networkstatus_v2(), dirserv_get_routerdescs(), dirvote_add_signatures(), dirvote_add_vote(), dirvote_clear_votes(), dirvote_compute_consensuses(), dirvote_compute_params(), dirvote_fetch_missing_votes(), entry_guards_parse_state(), entry_guards_prepend_from_config(), entry_guards_update_state(), extrainfo_parse_entry_from_string(), find_all_exitpolicy(), format_versions_list(), geoip_get_client_history(), geoip_get_dirreq_history(), geoip_get_request_history(), geoip_load_file(), geoip_note_client_seen(), geoip_parse_entry(), get_cert_list(), get_connection_array(), get_detached_signatures_from_pending_consensuses(), get_possible_sybil_list(), getargs_helper(), getinfo_helper_config(), getinfo_helper_dir(), getinfo_helper_entry_guards(), getinfo_helper_events(), getinfo_helper_networkstatus(), handle_control_authenticate(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_mapaddress(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_resolve(), handle_control_setevents(), handle_control_usefeature(), hid_serv_responsible_for_desc_id(), list_getinfo_options(), list_pending_downloads(), list_server_status_v1(), list_torrc_options(), list_v3_auth_ids(), log_credential_status(), log_entry_guards(), make_consensus_method_list(), microdesc_cache_rebuild(), microdescs_parse_from_string(), networkstatus_check_consensus_signature(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), networkstatus_get_v2_list(), networkstatus_getinfo_by_purpose(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), notify_control_networkstatus_changed(), options_act_reversible(), options_init_logs(), options_validate(), or_connection_new(), parse_addr_policy(), parse_bridge_line(), parse_dir_server_line(), parse_http_response(), parse_log_severity_config(), parse_port_config(), policies_set_router_exitpolicy_to_reject_all(), policy_expand_private(), policy_summarize(), policy_summary_create(), predicted_ports_init(), relay_send_command_from_edge(), rend_add_service(), rend_cache_store_v2_desc_as_client(), rend_config_services(), rend_encrypt_v2_intro_points_basic(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_service_authorization(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_set_connection_addr_port(), rend_service_update_descriptor(), rend_services_introduce(), rep_hist_buffer_stats_add_circ(), rep_hist_buffer_stats_write(), rep_hist_get_router_stability_doc(), rep_hist_load_mtbf_data(), rep_hist_update_state(), retry_listeners(), router_add_exit_policy(), router_choose_random_node(), router_get_consensus_status_by_nickname(), router_get_routerlist(), router_get_trusted_dir_servers(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_load_single_router(), router_nickname_is_in_list(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), router_rebuild_descriptor(), router_rebuild_store(), router_reload_v2_networkstatus(), router_set_networkstatus_v2(), routerlist_reset_warnings(), routers_update_status_from_consensus_networkstatus(), routerset_get_all_routers(), routerset_new(), routerset_parse(), set_expiry(), tor_init(), tor_listdir(), tor_tls_client_is_using_v2_ciphers(), tor_version_is_obsolete(), trusted_dirs_flush_certs_to_disk(), update_consensus_router_descriptor_downloads(), update_extrainfo_downloads(), update_router_descriptor_cache_downloads_v2(), upload_service_descriptor(), wildcard_increment_answer(), and write_bytes_to_file().

4.16.3.28 void smartlist_del (smartlist_t * *sl*, int *idx*)

Remove the **idx**th element of sl; if idx is not the last element, swap the last element of sl into the **idx**th space. Return the old value of the **idx**th element.

References smartlist_t::list, smartlist_t::num_used, and tor_assert.

Referenced by circuit_remove_handled_ports(), connection_remove(), exit_policy_remove_redundancies(), networkstatus_v2_list_clean(), rend_client_get_random_intro(), rend_client_remove_intro_point(), rend_services_introduce(), rep_hist_get_predicted_ports(), routerlist_remove(), and routerlist_remove_old().

4.16.3.29 void smartlist_del_keeporder (smartlist_t * *sl*, int *idx*)

Remove the **idx**th element of sl; if idx is not the last element, moving all subsequent elements back one space. Return the old value of the **idx**th element.

References smartlist_t::list, smartlist_t::num_used, and tor_assert.

Referenced by dir_split_resource_into_fingerprints(), exit_policy_remove_redundancies(), parse_bridge_line(), parse_dir_server_line(), remove_dead_entry_guards(), remove_obsolete_entry_guards(), and smartlist_uniq().

4.16.3.30 int smartlist_digest_isin (const smartlist_t * *sl*, const char * *element*)

Return true iff **sl** has some element E such that !memcmp(E,**element**,DIGEST_LEN)

References DIGEST_LEN, smartlist_t::list, and smartlist_t::num_used.

Referenced by authority_certs_fetch_missing(), directory_post_to_hs_dir(), and entry_guards_prepend_from_config().

4.16.3.31 static INLINE void smartlist_ensure_capacity (smartlist_t * *sl*, int *size*) [static]

Make sure that **sl** can hold at least **size** entries.

References smartlist_t::capacity, smartlist_t::list, and tor_assert.

Referenced by smartlist_add(), smartlist_add_all(), and smartlist_insert().

4.16.3.32 void smartlist_free (smartlist_t * *sl*)

Deallocate a smartlist. Does not release storage associated with the list's elements.

References smartlist_t::list, and tor_free.

Referenced by _connection_free(), accounting_parse_options(), add_nickname_list_to_smartlist(), addr_policy_list_free(), append_bytes_to_file(), authority_cert_parse_from_string(), authority_type_to_string(), check_nickname_list(), choose_good_exit_server_general(), choose_good_middle_server(), choose_random_entry(), circuit_all_predicted_ports_handled(), circuit_build_times_parse_state(), circuit_count_pending_on_or_conn(), circuit_free_all(), circuit_list_path_impl(), circuit_n_conn_done(), clear_geoip_db(), client_likes_consensus(), compute_consensus_method(), compute_consensus_versions_list(), config_dump(), config_register_addressmaps(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_free_all(), control_event_descriptors_changed(), control_event_networkstatus_changed_helper(), control_event_networkstatus_changed_single(), control_setconf_helper(), decode_hashed_passwords(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_get_consensus_url(), directory_get_from_hs_dir(), directory_handle_command_get(), directory_post_to_hs_dir(), dirserv_add_descriptor(), dirserv_add_multiple_descriptors(), dirserv_get_networkstatus_v2(), dirserv_get_routerdescs(), dirvote_compute_consensuses(), dirvote_compute_params(), dirvote_create_microdescriptor(), dirvote_free_all(), dns_free_all(), entry_guards_free_all(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), format_versions_list(), geoip_get_client_history(), geoip_get_dirreq_history(), geoip_get_request_history(), geoip_load_file(), get_detached_signatures_from_pending_consensuses(), getargs_helper(), getinfo_helper_config(), getinfo_helper_dir(), getinfo_helper_entry_guards(), getinfo_helper_events(), getinfo_helper_networkstatus(), handle_control_attachstream(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_closestream(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_mapaddress(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_redirectstream(), handle_control_resolve(), handle_control_setcircuitpurpose(), handle_control_setevents(), handle_control_usefeature(), hid_serv_responsible_for_desc_id(), list_getinfo_options(), list_server_status_v1(), list_torrc_options(), list_v3_auth_ids(), log_credential_status(), log_entry_guards(), make_consensus_method_list(), microdesc_cache_rebuild(), microdesc_cache_reload(), microdesc_free(), microdescs_add_to_cache(), microdescs_parse_from_string(), networkstatus_check_consensus_signature(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_free_all(), networkstatus_get_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_free(), networkstatus_v2_parse_from_string(), networkstatus_vote_free(), notify_control_networkstatus_changed(), ns_detached_signatures_free(), option_clear(), options_act_reversible(), options_init_logs(), options_validate(), parse_addr_policy(), parse_bridge_line(), parse_dir_server_line(), parse_http_response(), parse_log_severity_config(), parse_port_config(), policy_summarize(), predicted_ports_free(), relay_send_command_from_edge(), rend_config_services(), rend_encrypt_v2_intro_points_basic(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_service_authorization(), rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), rend_service_free(), rend_service_free_all(), rep_hist_buffer_stats_write(), rep_hist_get_router_stability_doc(), rep_hist_load_mtbf_data(), rep_hist_update_state(), router_free_all(), router_get_consensus_status_by_nickname(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_load_single_router(), router_nickname_is_in_list(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), router_rebuild_descriptor(), router_reload_v2_networkstatus(), routerinfo_free(), routerlist_free(), routerlist_free_all(), routerset_free(), routerset_parse(), tor_free_all(), tor_tls_client_is_using_v2_ciphers(), tor_version_is_obsolete(), update_extrainfo_downloads(), upload_service_descriptor(), and write_bytes_to_file().

4.16.3.33 void* smartlist_get_most_frequent (const smartlist_t * *sl*, int(*)(const void **a, const void **b) *compare*)

Given a smartlist **sl** sorted with the function **compare**, return the most frequent member in the list. Break ties in favor of later elements. If the list is empty, return NULL.

References smartlist_t::list, and smartlist_t::num_used.

Referenced by smartlist_get_most_frequent_digest256(), and smartlist_get_most_frequent_string().

4.16.3.34 char* smartlist_get_most_frequent_digest256 (smartlist_t * *sl*)

Return the most frequent member of the sorted list of DIGEST256_LEN digests in **sl**

References _compare_digests256(), and smartlist_get_most_frequent().

4.16.3.35 char* smartlist_get_most_frequent_string (smartlist_t * *sl*)

Return the most frequent string in the sorted list **sl**

References _compare_string_ptrs(), and smartlist_get_most_frequent().

4.16.3.36 static INLINE void smartlist_heapify (smartlist_t * *sl*, int(*)(const void *a, const void *b) *compare*, int *idx_field_offset*, int *idx*) `[static]`

Helper. **sl** may have at most one violation of the heap property: the item at **idx** may be greater than one or both of its children. Restore the heap property.

References smartlist_t::list, and smartlist_t::num_used.

Referenced by smartlist_pqueue_pop(), and smartlist_pqueue_remove().

4.16.3.37 void smartlist_insert (smartlist_t * *sl*, int *idx*, void * *val*)

Insert the value **val** as the new **idx**th element of **sl**, moving all items previously at **idx** or later forward one space.

References smartlist_t::list, smartlist_t::num_used, smartlist_add(), smartlist_ensure_capacity(), and tor_assert.

Referenced by add_an_entry_guard(), and policy_summary_split().

4.16.3.38 void smartlist_intersect (smartlist_t * *sl1*, const smartlist_t * *sl2*)

Remove every element E of sl1 such that !smartlist_isin(sl2,E). Does not preserve the order of sl1.

References smartlist_t::list, smartlist_t::num_used, and smartlist_isin().

4.16.3.39 int smartlist_isin (const smartlist_t * *sl*, const void * *element*)

Return true iff some element E of sl has E==element.

References smartlist_t::list, and smartlist_t::num_used.

Referenced by add_connection_to_closeable_list(), assert_circuit_ok(), choose_random_entry(), connection_in_array(), connection_is_on_closeable_list(), connection_start_reading_from_linked_conn(), connection_stop_reading_from_linked_conn(), smartlist_intersect(), and smartlist_overlap().

4.16.3.40 char* smartlist_join_strings (smartlist_t * *sl*, const char * *join*, int *terminate*, size_t * *len_out*)

Allocate and return a new string containing the concatenation of the elements of **sl**, in order, separated by **join**. If **terminate** is true, also terminate the string with **join**. If **len_out** is not NULL, set **len_out** to the length of the returned string. Requires that every element of **sl** is NUL-terminated string.

References smartlist_join_strings2().

Referenced by authority_type_to_string(), circuit_list_path_impl(), compute_consensus_versions_list(), config_dump(), control_event_descriptors_changed(), control_event_networkstatus_changed_helper(), control_setconf_helper(), directory_get_consensus_url(), directory_handle_command_get(), dirvote_compute_params(), dirvote_create_microdescriptor(), format_networkstatus_vote(), format_versions_list(), geoip_get_client_history(), geoip_get_request_history(), get_assigned_option(), getinfo_helper_config(), getinfo_helper_dir(), getinfo_helper_entry_guards(), getinfo_helper_events(), getinfo_helper_networkstatus(), handle_control_getconf(), handle_control_mapaddress(), handle_control_protocolinfo(), list_getinfo_options(), list_server_status_v1(), list_v3_auth_ids(), log_credential_status(), log_entry_guards(), make_consensus_method_list(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), options_validate(), parse_bridge_line(), parse_dir_server_line(), policy_summarize(), relay_send_command_from_edge(), rep_hist_buffer_stats_write(), rep_hist_get_router_stability_doc(), router_dump_router_to_string(), router_load_routers_from_string(), router_set_networkstatus_v2(), routerset_to_string(), and tor_tls_client_is_using_v2_ciphers().

4.16.3.41 char* smartlist_join_strings2 (smartlist_t * *sl*, const char * *join*, size_t *join_len*, int *terminate*, size_t * *len_out*)

As smartlist_join_strings, but instead of separating/terminated with a NUL-terminated string **join**, uses the **join_len**-byte sequence at **join**. (Useful for generating a sequence of NUL-terminated strings.)

References smartlist_t::list, smartlist_t::num_used, and tor_assert.

Referenced by smartlist_join_strings().

4.16.3.42 int smartlist_overlap (const smartlist_t * *sl1*, const smartlist_t * *sl2*)

Return true iff some element E of sl2 has smartlist_isin(sl1,E).

References smartlist_t::list, smartlist_t::num_used, and smartlist_isin().

4.16.3.43 void* smartlist_pop_last (smartlist_t * *sl*)

If **sl** is nonempty, remove and return the final element. Otherwise, return NULL.

References smartlist_t::list, smartlist_t::num_used, and tor_assert.

Referenced by connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), and connection_dirserv_add_servers_to_outbuf().

4.16.3.44 void smartlist_pqueue_add (smartlist_t ∗ *sl*, int(∗)(const void ∗a, const void ∗b) *compare*, int *idx_field_offset*, void ∗ *item*)

Insert **item** into the heap stored in **sl**, where order is determined by **compare** and the offset of the item in the heap is stored in an int-typed field at position **idx_field_offset** within item.

References smartlist_t::list, smartlist_t::num_used, and smartlist_add().

Referenced by add_cell_ewma_to_conn(), and set_expiry().

4.16.3.45 void smartlist_pqueue_assert_ok (smartlist_t ∗ *sl*, int(∗)(const void ∗a, const void ∗b) *compare*, int *idx_field_offset*)

Assert that the heap property is correctly maintained by the heap stored in **sl**, where order is determined by **compare**.

References smartlist_t::list, smartlist_t::num_used, and tor_assert.

4.16.3.46 void∗ smartlist_pqueue_pop (smartlist_t ∗ *sl*, int(∗)(const void ∗a, const void ∗b) *compare*, int *idx_field_offset*)

Remove and return the top-priority item from the heap stored in **sl**, where order is determined by **compare** and the item's position is stored at position **idx_field_offset** within the item. **sl** must not be empty.

References smartlist_t::list, smartlist_t::num_used, smartlist_heapify(), and tor_assert.

Referenced by pop_first_cell_ewma_from_conn(), and purge_expired_resolves().

4.16.3.47 void smartlist_pqueue_remove (smartlist_t ∗ *sl*, int(∗)(const void ∗a, const void ∗b) *compare*, int *idx_field_offset*, void ∗ *item*)

Remove the item **item** from the heap stored in **sl**, where order is determined by **compare** and the item's position is stored at position **idx_field_offset** within the item. **sl** must not be empty.

References smartlist_t::list, smartlist_t::num_used, smartlist_heapify(), and tor_assert.

Referenced by remove_cell_ewma_from_conn().

4.16.3.48 void smartlist_remove (smartlist_t ∗ *sl*, const void ∗ *element*)

Remove all elements E from sl such that E==element. Preserve the order of any elements before E, but elements after E can be rearranged.

References smartlist_t::list, and smartlist_t::num_used.

Referenced by _circuit_mark_for_close(), circuit_set_state(), connection_stop_reading_from_linked_conn(), connection_unlink(), networkstatus_add_detached_signatures(), and smartlist_subtract().

4.16.3.49 void smartlist_reverse (smartlist_t ∗ *sl*)

Reverse the order of the items in **sl**.

References smartlist_t::list, smartlist_t::num_used, and tor_assert.

4.16.3.50 void smartlist_sort (smartlist_t ∗ *sl*, int(∗)(const void ∗∗a, const void ∗∗b) *compare*)

Sort the members of **sl** into an order defined by the ordering function **compare**, which returns less then 0 if a precedes b, greater than 0 if b precedes a, and 0 if a 'equals' b.

References smartlist_t::list, and smartlist_t::num_used.

Referenced by compute_consensus_method(), compute_routerstatus_consensus(), dir_split_resource_into_fingerprint_pairs(), directory_get_consensus_url(), dirserv_read_measured_bandwidths(), geoip_get_client_history(), geoip_get_request_history(), geoip_load_file(), get_possible_sybil_list(), networkstatus_compute_consensus(), networkstatus_v2_parse_from_string(), rep_hist_buffer_stats_write(), router_rebuild_store(), routerlist_remove_old_routers(), routers_sort_by_identity(), smartlist_sort_digests(), smartlist_sort_digests256(), smartlist_sort_strings(), and sort_version_list().

4.16.3.51 void smartlist_sort_digests (smartlist_t ∗ *sl*)

Sort the list of DIGEST_LEN-byte digests into ascending order.

References _compare_digests(), and smartlist_sort().

Referenced by dir_split_resource_into_fingerprints(), dirserv_get_networkstatus_v2_fingerprints(), launch_router_descriptor_downloads(), and rend_encrypt_v2_intro_points_basic().

4.16.3.52 void smartlist_sort_digests256 (smartlist_t ∗ *sl*)

Sort the list of DIGEST256_LEN-byte digests into ascending order.

References _compare_digests256(), and smartlist_sort().

Referenced by dir_split_resource_into_fingerprints().

4.16.3.53 void smartlist_sort_strings (smartlist_t ∗ *sl*)

Sort a smartlist **sl** containing strings into lexically ascending order.

References _compare_string_ptrs(), and smartlist_sort().

Referenced by dir_split_resource_into_fingerprints(), dirvote_compute_params(), list_getinfo_options(), networkstatus_compute_consensus(), rend_config_services(), and router_rebuild_descriptor().

4.16.3.54 int smartlist_split_string (smartlist_t ∗ *sl*, const char ∗ *str*, const char ∗ *sep*, int *flags*, int *max*)

Split a string **str** along all occurrences of **sep**, adding the split strings, in order, to **sl**.

If **flags**&SPLIT_SKIP_SPACE is true, remove initial and trailing space from each entry. If **flags**&SPLIT_IGNORE_BLANK is true, remove any entries of length 0. If **flags**&SPLIT_STRIP_SPACE is true, strip spaces from each split string.

If max>0, divide the string into no more than **max** pieces. If **sep** is NULL, split on any sequence of horizontal space.

References smartlist_add(), tor_assert, and tor_strstrip().

Referenced by accounting_parse_options(), add_nickname_list_to_smartlist(), check_nickname_list(), circuit_build_times_parse_state(), config_assign_value(), config_register_addressmaps(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), entry_guards_parse_state(), format_versions_list(), getargs_helper(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_mapaddress(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_resolve(), handle_control_setevents(), handle_control_usefeature(), networkstatus_compute_consensus(), options_init_logs(), parse_addr_policy(), parse_bridge_line(), parse_dir_server_line(), parse_http_response(), parse_log_severity_config(), parse_port_config(), rend_config_services(), rend_parse_service_authorization(), rend_parse_v2_service_descriptor(), rep_hist_load_mtbf_data(), rep_hist_update_state(), router_nickname_is_in_list(), router_rebuild_descriptor(), routerset_parse(), and tor_version_is_obsolete().

4.16.3.55 int smartlist_string_isin (const smartlist_t * *sl*, const char * *element*)

Return true iff **sl** has some element E such that !strcmp(E,**element**)

References smartlist_t::list, and smartlist_t::num_used.

Referenced by add_nickname_list_to_smartlist(), answer_is_wildcarded(), dirvote_compute_consensuses(), router_load_routers_from_string(), router_rebuild_descriptor(), router_set_networkstatus_v2(), smartlist_string_num_isin(), and wildcard_increment_answer().

4.16.3.56 int smartlist_string_isin_case (const smartlist_t * *sl*, const char * *element*)

Return true iff **sl** has some element E such that !strcasecmp(E,**element**)

References smartlist_t::list, and smartlist_t::num_used.

Referenced by add_wildcarded_test_address(), and is_test_address().

4.16.3.57 int smartlist_string_num_isin (const smartlist_t * *sl*, int *num*)

Return true iff **sl** has some element E such that E is equal to the decimal encoding of **num**.

References smartlist_string_isin(), and tor_snprintf().

Referenced by circuit_all_predicted_ports_handled(), circuit_get_open_circ_or_launch(), circuit_stream_is_being_handled(), consider_plaintext_ports(), and rend_service_requires_uptime().

4.16.3.58 int smartlist_string_pos (const smartlist_t * *sl*, const char * *element*)

If **element** is equal to an element of **sl**, return that element's index. Otherwise, return -1.

References smartlist_t::list, and smartlist_t::num_used.

Referenced by networkstatus_compute_consensus(), and routerstatus_parse_entry_from_string().

4.16.3.59 void smartlist_string_remove (smartlist_t * *sl*, const char * *element*)

If there are any strings in sl equal to element, remove and free them. Does not preserve order.

References smartlist_t::list, smartlist_t::num_used, tor_assert, and tor_free.

Referenced by router_load_extrainfo_from_string(), router_load_routers_from_string(), router_rebuild_descriptor(), and router_set_networkstatus_v2().

4.16.3.60 void smartlist_subtract (smartlist_t ∗ *sl1*, const smartlist_t ∗ *sl2*)

Remove every element E of sl1 such that smartlist_isin(sl2,E). Does not preserve the order of sl1.

References smartlist_t::list, smartlist_t::num_used, and smartlist_remove().

4.16.3.61 void smartlist_uniq (smartlist_t ∗ *sl*, int(∗)(const void ∗∗a, const void ∗∗b) *compare*, void(∗)(void ∗a) *free_fn*)

Given a sorted smartlist **sl** and the comparison function used to sort it, remove all duplicate members. If free_fn is provided, calls free_fn on each duplicate. Otherwise, just removes them. Preserves order.

References smartlist_t::list, smartlist_t::num_used, and smartlist_del_keeporder().

Referenced by compute_consensus_method(), dir_split_resource_into_fingerprint_pairs(), networkstatus_v2_parse_from_string(), smartlist_uniq_digests(), smartlist_uniq_digests256(), smartlist_uniq_strings(), and sort_version_list().

4.16.3.62 void smartlist_uniq_digests (smartlist_t ∗ *sl*)

Remove duplicate digests from a sorted list, and free them with **tor_free()** (p. 1271).

References _compare_digests(), _tor_free(), and smartlist_uniq().

Referenced by dir_split_resource_into_fingerprints().

4.16.3.63 void smartlist_uniq_digests256 (smartlist_t ∗ *sl*)

Remove duplicate 256-bit digests from a sorted list, and free them with **tor_free()** (p. 1271).

References _compare_digests256(), _tor_free(), and smartlist_uniq().

Referenced by dir_split_resource_into_fingerprints().

4.16.3.64 void smartlist_uniq_strings (smartlist_t ∗ *sl*)

Remove duplicate strings from a sorted list, and free them with **tor_free()** (p. 1271).

References _compare_string_ptrs(), _tor_free(), and smartlist_uniq().

Referenced by dir_split_resource_into_fingerprints(), networkstatus_compute_consensus(), rend_config_services(), and router_rebuild_descriptor().

4.16.3.65 void strmap_assert_ok (const strmap_t ∗ *map*)

Fail with an assertion error if anything has gone wrong with the internal representation of **map**.

References tor_assert.

4.16.3.66 static INLINE int strmap_entries_eq (const strmap_entry_t * *a*, const strmap_entry_t * *b*) [static]

Helper: compare strmap_entry_t objects by key value.

4.16.3.67 static INLINE unsigned int strmap_entry_hash (const strmap_entry_t * *a*) [static]

Helper: return a hash value for a strmap_entry_t.

4.16.3.68 void strmap_free (strmap_t * *map*, void(*)(void *) *free_val*)

Remove all entries from **map**, and deallocate storage for those entries. If free_val is provided, it is invoked on every value in **map**.

References tor_assert, and tor_free.

Referenced by addressmap_free_all(), clear_geoip_db(), dirserv_free_all(), dirserv_free_fingerprint_list(), dns_reset_correctness_checks(), networkstatus_free_all(), ns_detached_signatures_free(), rend_cache_free_all(), rend_parse_service_authorization(), rend_service_authorization_free_all(), rend_service_load_keys(), routerset_free(), and routerstatus_list_update_named_server_map().

4.16.3.69 void* strmap_get (const strmap_t * *map*, const char * *key*)

Return the current value associated with **key**, or NULL if no value is set.

References tor_assert.

Referenced by addressmap_get_virtual_address(), addressmap_register(), addressmap_register_virtual_address(), addressmap_rewrite(), addressmap_rewrite_reverse(), addressmap_virtaddress_remove(), client_dns_clear_failures(), client_dns_incr_failures(), detached_get_digests(), detached_get_signatures(), dirserv_get_consensus(), dirvote_add_signatures_to_pending_consensus(), lookup_cached_dir_by_fp(), networkstatus_add_detached_signatures(), rend_client_lookup_service_authorization(), rend_parse_client_keys(), rend_parse_service_authorization(), rend_service_load_keys(), strmap_get_lc(), and wildcard_increment_answer().

4.16.3.70 void* strmap_get_lc (const strmap_t * *map*, const char * *key*)

Same as strmap_get, but first converts **key** to lowercase.

References strmap_get(), tor_free, and tor_strlower().

Referenced by add_fingerprint_to_dir(), addressmap_have_mapping(), dirserv_get_name_status(), geoip_add_entry(), geoip_get_country(), lookup_last_hid_serv_request(), networkstatus_compute_consensus(), networkstatus_get_router_digest_by_nickname(), networkstatus_nickname_is_unnamed(), rend_cache_lookup_entry(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), router_get_consensus_status_by_nickname(), and routerset_contains().

4.16.3.71 int strmap_isempty (const strmap_t * *map*)

Return true iff **map** has no entries.

4.16.3.72 int strmap_iter_done (strmap_iter_t * *iter*)

Return true iff **iter** has advanced past the last entry of **map**.

Referenced by addressmap_get_mappings(), directory_clean_last_hid_serv_requests(), and rend_cache_clean().

4.16.3.73 void strmap_iter_get (strmap_iter_t * *iter*, const char ** *keyp*, void ** *valp*)

Set ***keyp** and ***valp** to the current entry pointed to by iter.

References tor_assert.

Referenced by addressmap_get_mappings(), directory_clean_last_hid_serv_requests(), and rend_cache_clean().

4.16.3.74 strmap_iter_t* strmap_iter_init (strmap_t * *map*)

return an **iterator** pointer to the front of a map.

Iterator example:

```
// uppercase values in "map", removing empty values.

strmap_iter_t *iter;
const char *key;
void *val;
char *cp;

for (iter = strmap_iter_init(map); !strmap_iter_done(iter); ) {
   strmap_iter_get(iter, &key, &val);
   cp = (char*)val;
   if (!*cp) {
      iter = strmap_iter_next_rmv(map,iter);
      free(val);
   } else {
      for (;*cp;cp++) *cp = TOR_TOUPPER(*cp);
      iter = strmap_iter_next(map,iter);
   }
}
```

References tor_assert.

Referenced by addressmap_get_mappings(), directory_clean_last_hid_serv_requests(), and rend_cache_clean().

4.16.3.75 strmap_iter_t* strmap_iter_next (strmap_t * *map*, strmap_iter_t * *iter*)

Advance the iterator **iter** for **map** a single step to the next entry, and return its new value.

References tor_assert.

Referenced by addressmap_get_mappings(), directory_clean_last_hid_serv_requests(), and rend_cache_clean().

4.16.3.76 strmap_iter_t∗ strmap_iter_next_rmv (strmap_t ∗ *map*, strmap_iter_t ∗ *iter*)

Advance the iterator **iter** a single step to the next entry, removing the current entry, and return its new value.

References tor_assert, and tor_free.

Referenced by addressmap_get_mappings(), directory_clean_last_hid_serv_requests(), and rend_cache_clean().

4.16.3.77 void∗ strmap_remove (strmap_t ∗ *map*, const char ∗ *key*)

Remove the value currently associated with **key** from the map. Return the value if one was set, or NULL if there was no entry for **key**.

Note: you must free any storage associated with the returned value.

References tor_assert, and tor_free.

Referenced by addressmap_register(), addressmap_virtaddress_remove(), and strmap_remove_lc().

4.16.3.78 void∗ strmap_remove_lc (strmap_t ∗ *map*, const char ∗ *key*)

Same as strmap_remove, but first converts **key** to lowercase

References strmap_remove(), tor_free, and tor_strlower().

4.16.3.79 void∗ strmap_set (strmap_t ∗ *map*, const char ∗ *key*, void ∗ *val*)

Set the current value for **key** to **val**. Returns the previous value for **key** if one was set, or NULL if one was not.

This function makes a copy of **key** if necessary, but not of **val**.

References tor_assert.

Referenced by addressmap_register(), addressmap_register_virtual_address(), client_dns_incr_failures(), detached_get_digests(), detached_get_signatures(), dirserv_set_cached_consensus_networkstatus(), lookup_last_hid_serv_request(), rend_parse_client_keys(), rend_parse_service_authorization(), strmap_set_lc(), and wildcard_increment_answer().

4.16.3.80 void∗ strmap_set_lc (strmap_t ∗ *map*, const char ∗ *key*, void ∗ *val*)

Same as strmap_set, but first converts **key** to lowercase.

References strmap_set(), tor_free, and tor_strlower().

Referenced by add_fingerprint_to_dir(), geoip_add_entry(), networkstatus_compute_consensus(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), routerset_parse(), and routerstatus_list_update_named_server_map().

4.16.3.81 int strmap_size (const strmap_t ∗ *map*)

Return the number of items in **map**.

Referenced by dirserv_get_status_impl(), evdns_wildcard_check_callback(), rend_cache_size(), rend_parse_client_keys(), and rend_service_load_keys().

4.17 control.c File Reference

Implementation for Tor's control-socket interface. See doc/spec/control-spec.txt for full details on protocol.

`#include "or.h"`

Data Structures

- struct **getinfo_item_t**

Defines

- #define **CONTROL_PRIVATE**
- #define **STATE_IS_OPEN**(s) ((s) == CONTROL_CONN_STATE_OPEN)
- #define **_EVENT_MIN** 0x0001
- #define **EVENT_CIRCUIT_STATUS** 0x0001
- #define **EVENT_STREAM_STATUS** 0x0002
- #define **EVENT_OR_CONN_STATUS** 0x0003
- #define **EVENT_BANDWIDTH_USED** 0x0004
- #define **EVENT_LOG_OBSOLETE** 0x0005
- #define **EVENT_NEW_DESC** 0x0006
- #define **EVENT_DEBUG_MSG** 0x0007
- #define **EVENT_INFO_MSG** 0x0008
- #define **EVENT_NOTICE_MSG** 0x0009
- #define **EVENT_WARN_MSG** 0x000A
- #define **EVENT_ERR_MSG** 0x000B
- #define **EVENT_ADDRMAP** 0x000C
- #define **EVENT_DESCCHANGED** 0x000E
- #define **EVENT_STATUS_CLIENT** 0x0010
- #define **EVENT_STATUS_SERVER** 0x0011
- #define **EVENT_STATUS_GENERAL** 0x0012
- #define **EVENT_GUARD** 0x0013
- #define **EVENT_STREAM_BANDWIDTH_USED** 0x0014
- #define **EVENT_CLIENTS_SEEN** 0x0015
- #define **EVENT_NEWCONSENSUS** 0x0016
- #define **EVENT_BUILDTIMEOUT_SET** 0x0017
- #define **_EVENT_MAX** 0x0017
- #define **EVENT_IS_INTERESTING**(e) (**global_event_mask** & (1<<(e)))
- #define **AUTHENTICATION_COOKIE_LEN** 32
- #define **BOOTSTRAP_MSG_LEN** 1024
- #define **ALL_FORMATS** 1
- #define **CONNECTION_PRINTF_TO_BUF_BUFFERSIZE** 1024
- #define **SEND_CONTROL1_EVENT_BUFFERSIZE** 10064
- #define **ITEM**(name, fn, desc) { name, getinfo_helper_##fn, desc, 0 }
- #define **PREFIX**(name, fn, desc) { name, getinfo_helper_##fn, desc, 1 }
- #define **DOC**(name, desc) { name, NULL, desc, 0 }
- #define **MAX_COMMAND_LINE_LENGTH** (1024*1024)
- #define **BOOTSTRAP_PROBLEM_THRESHOLD** 10

Typedefs

- typedef uint32_t **event_mask_t**
- typedef int **event_format_t**
- typedef int(* **getinfo_helper_t**)(**control_connection_t** *, const char *q, char **a)

Functions

- static void **connection_printf_to_buf** (**control_connection_t** *conn, const char *format,...) CHECK_PRINTF(2
- static void static void **send_control_done** (**control_connection_t** *conn)
- static void **send_control_event** (uint16_t event, **event_format_t** which, const char *format,...) CHECK_PRINTF(3
- static void static int **handle_control_setconf** (**control_connection_t** *conn, uint32_t len, char *body)
- static int **handle_control_resetconf** (**control_connection_t** *conn, uint32_t len, char *body)
- static int **handle_control_getconf** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_loadconf** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_setevents** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_authenticate** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_signal** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_mapaddress** (**control_connection_t** *conn, uint32_t len, const char *body)
- static char * **list_getinfo_options** (void)
- static int **handle_control_getinfo** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_extendcircuit** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_setcircuitpurpose** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_attachstream** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_postdescriptor** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_redirectstream** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_closestream** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_closecircuit** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_resolve** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **handle_control_usefeature** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **write_stream_target_to_buf** (**edge_connection_t** *conn, char *buf, size_t len)

- static void **orconn_target_get_name** (char *buf, size_t len, **or_connection_t** *conn)
- static char * **get_cookie_file** (void)
- static INLINE int **event_to_log_severity** (int event)
- static INLINE int **log_severity_to_event** (int severity)
- void **control_update_global_event_mask** (void)
- void **control_adjust_event_log_severity** (void)
- int **control_event_is_interesting** (int event)
- static INLINE void **connection_write_str_to_buf** (const char *s, **control_connection_t** *conn)
- size_t **write_escaped_data** (const char *data, size_t len, char **out)
- size_t **read_escaped_data** (const char *data, size_t len, char **out)
- static INLINE int **get_escaped_string_length** (const char *start, size_t in_len_max, int *chars_out)
- static const char * **extract_escaped_string** (const char *start, size_t in_len_max, char **out, size_t *out_len)
- static const char * **decode_escaped_string** (const char *start, size_t in_len_max, char **out, size_t *out_len)
- static void **send_control_event_string** (uint16_t event, **event_format_t** which, const char *msg)
- static void **send_control_event_impl** (uint16_t event, **event_format_t** which, const char *format, va_list ap)
- static **origin_circuit_t** * **get_circ** (const char *id)
- static **edge_connection_t** * **get_stream** (const char *id)
- static int **control_setconf_helper** (**control_connection_t** *conn, uint32_t len, char *body, int use_defaults)
- **smartlist_t** * **decode_hashed_passwords** (**config_line_t** *passwords)
- static int **handle_control_saveconf** (**control_connection_t** *conn, uint32_t len, const char *body)
- static int **getinfo_helper_misc** (**control_connection_t** *conn, const char *question, char **answer)
- static char * **munge_extrainfo_into_routerinfo** (const char *ri_body, **signed_descriptor_t** *ri, **signed_descriptor_t** *ei)
- static int **getinfo_helper_dir** (**control_connection_t** *control_conn, const char *question, char **answer)
- static int **getinfo_helper_events** (**control_connection_t** *control_conn, const char *question, char **answer)
- static int **handle_getinfo_helper** (**control_connection_t** *control_conn, const char *question, char **answer)
- static uint8_t **circuit_purpose_from_string** (const char *string)
- static **smartlist_t** * **getargs_helper** (const char *command, **control_connection_t** *conn, const char *body, int min_args, int max_args)
- static const char * **find_element_starting_with** (**smartlist_t** *sl, int start_at, const char *prefix)
- static int **is_keyval_pair** (const char *s)
- static int **handle_control_protocolinfo** (**control_connection_t** *conn, uint32_t len, const char *body)
- int **connection_control_finished_flushing** (**control_connection_t** *conn)
- int **connection_control_reached_eof** (**control_connection_t** *conn)
- static int **is_valid_initial_command** (**control_connection_t** *conn, const char *cmd)
- int **connection_control_process_inbuf** (**control_connection_t** *conn)

- int **control_event_circuit_status** (**origin_circuit_t** *circ, **circuit_status_event_t** tp, int reason_code)
- int **control_event_stream_status** (**edge_connection_t** *conn, **stream_status_event_t** tp, int reason_code)
- int **control_event_or_conn_status** (**or_connection_t** *conn, **or_conn_status_event_t** tp, int reason)
- int **control_event_stream_bandwidth** (**edge_connection_t** *edge_conn)
- int **control_event_stream_bandwidth_used** (void)
- int **control_event_bandwidth_used** (uint32_t n_read, uint32_t n_written)
- void **disable_control_logging** (void)
- void **enable_control_logging** (void)
- void **control_event_logmsg** (int severity, uint32_t domain, const char *msg)
- int **control_event_descriptors_changed** (**smartlist_t** *routers)
- int **control_event_address_mapped** (const char *from, const char *to, time_t expires, const char *error)
- int **control_event_or_authdir_new_descriptor** (const char *action, const char *desc, size_t desclen, const char *msg)
- static int **control_event_networkstatus_changed_helper** (**smartlist_t** *statuses, uint16_t event, const char *event_string)
- int **control_event_networkstatus_changed** (**smartlist_t** *statuses)
- int **control_event_newconsensus** (const **networkstatus_t** *consensus)
- int **control_event_buildtimeout_set** (const **circuit_build_times_t** *cbt, **buildtimeout_set_event_t** type)
- int **control_event_networkstatus_changed_single** (**routerstatus_t** *rs)
- int **control_event_my_descriptor_changed** (void)
- static int **control_event_status** (int type, int severity, const char *format, va_list args)
- int **control_event_general_status** (int severity, const char *format,...)
- int **control_event_client_status** (int severity, const char *format,...)
- int **control_event_server_status** (int severity, const char *format,...)
- int **control_event_guard** (const char *nickname, const char *digest, const char *status)
- int **init_cookie_authentication** (int enabled)
- static int **bootstrap_status_to_string** (**bootstrap_status_t** s, const char **tag, const char **summary)
- void **control_event_bootstrap** (**bootstrap_status_t** status, int progress)
- void **control_event_bootstrap_problem** (const char *warn, int reason)
- void **control_event_clients_seen** (const char *controller_str)

Variables

- static **event_mask_t global_event_mask** = 0
- static int **disable_log_messages** = 0
- static int **authentication_cookie_is_set** = 0
- static char **authentication_cookie** [AUTHENTICATION_COOKIE_LEN]
- static char **last_sent_bootstrap_message** [BOOTSTRAP_MSG_LEN]
- static const **getinfo_item_t getinfo_items** [ ]
- static int **bootstrap_percent** = BOOTSTRAP_STATUS_UNDEF
- static int **bootstrap_problems** = 0

4.17.1 Detailed Description

Implementation for Tor's control-socket interface. See doc/spec/control-spec.txt for full details on protocol.

4.17.2 Define Documentation

4.17.2.1 #define ALL_FORMATS 1

Flag for event_format_t. Indicates that we should use the one standard format.

Referenced by control_event_address_mapped(), control_event_bandwidth_used(), control_event_buildtimeout_set(), control_event_circuit_status(), control_event_descriptors_changed(), control_event_guard(), control_event_logmsg(), control_event_my_descriptor_changed(), control_event_networkstatus_changed_helper(), control_event_or_authdir_new_descriptor(), control_event_or_conn_status(), control_event_status(), control_event_stream_bandwidth(), control_event_stream_bandwidth_used(), control_event_stream_status(), and send_control_event_impl().

4.17.2.2 #define AUTHENTICATION_COOKIE_LEN 32

If we're using cookie-type authentication, how long should our cookies be?

Referenced by handle_control_authenticate(), and init_cookie_authentication().

4.17.2.3 #define BOOTSTRAP_MSG_LEN 1024

A sufficiently large size to record the last bootstrap phase string.

Referenced by control_event_bootstrap(), and control_event_bootstrap_problem().

4.17.2.4 #define EVENT_IS_INTERESTING(e) (global_event_mask & (1<<(e)))

Macro: true if any control connection is interested in events of type **e**.

Referenced by control_adjust_event_log_severity(), control_event_address_mapped(), control_event_bandwidth_used(), control_event_circuit_status(), control_event_client_status(), control_event_descriptors_changed(), control_event_general_status(), control_event_guard(), control_event_is_interesting(), control_event_logmsg(), control_event_networkstatus_changed_helper(), control_event_networkstatus_changed_single(), control_event_or_authdir_new_descriptor(), control_event_or_conn_status(), control_event_server_status(), control_event_stream_bandwidth(), control_event_stream_bandwidth_used(), and control_event_stream_status().

4.17.2.5 #define MAX_COMMAND_LINE_LENGTH (1024*1024)

Do not accept any control command of more than 1MB in length. Anything that needs to be anywhere near this long probably means that one of our interfaces is broken.

Referenced by connection_control_process_inbuf().

4.17.2.6 #define STATE_IS_OPEN(s) ((s) == CONTROL_CONN_STATE_-OPEN)

Yield true iff **s** is the state of a **control_connection_t** (p. 63) that has finished authentication and is accepting commands.

Referenced by control_update_global_event_mask(), and handle_control_protocolinfo().

4.17.3 Typedef Documentation

4.17.3.1 typedef int event_format_t

Bit field of flags to select how to format a controller event. Recognized flag is ALL_FORMATS.

4.17.3.2 typedef uint32_t event_mask_t

Bitfield: The bit 1<<e is set if **any** open control connection is interested in events of type **e**. We use this so that we can decide to skip generating event messages that nobody has interest in without having to walk over the global connection list to find out.

4.17.3.3 typedef int(∗ getinfo_helper_t)(control_connection_t ∗, const char ∗q, char ∗∗a)

Callback function for GETINFO: on a given control connection, try to answer the question **q** and store the newly-allocated answer in ∗**a**. If an internal error occurs, return -1. On success, or if the key is not recognized, return 0. Do not set **a** if the key is not recognized.

4.17.4 Function Documentation

4.17.4.1 static int bootstrap_status_to_string (bootstrap_status_t *s*, const char ∗∗ *tag*, const char ∗∗ *summary*) `[static]`

Convert the name of a bootstrapping phase **s** into strings **tag** and **summary** suitable for display by the controller.

Referenced by control_event_bootstrap(), and control_event_bootstrap_problem().

4.17.4.2 static uint8_t circuit_purpose_from_string (const char ∗ *string*) `[static]`

Given a string, convert it to a circuit purpose.

References CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_CONTROLLER, CIRCUIT_PURPOSE_UNKNOWN, and strcasecmpstart().

Referenced by handle_control_extendcircuit(), and handle_control_setcircuitpurpose().

4.17.4.3 int connection_control_finished_flushing (control_connection_t ∗ *conn*)

Called when **conn** has no more bytes left on its outbuf.

References connection_stop_writing(), TO_CONN, and tor_assert.

Referenced by connection_finished_flushing().

4.17.4.4 int connection_control_process_inbuf (control_connection_t ∗ *conn*)

Called when data has arrived on a v1 control connection: Try to fetch commands from conn->inbuf, and execute them.

References control_connection_t::_base, connection_printf_to_buf(), connection_stop_reading(), connection_write_str_to_buf(), CONTROL_CONN_STATE_NEEDAUTH, CONTROL_CONN_STATE_OPEN, fetch_from_buf_line(), handle_control_attachstream(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_closestream(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_loadconf(), handle_control_mapaddress(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_redirectstream(), handle_control_resetconf(), handle_control_resolve(), handle_control_saveconf(), handle_control_setcircuitpurpose(), handle_control_setconf(), handle_control_setevents(), handle_control_signal(), handle_control_usefeature(), connection_t::hold_open_until_flushed, connection_t::inbuf, control_connection_t::incoming_cmd, control_connection_t::incoming_cmd_cur_len, control_connection_t::incoming_cmd_len, is_valid_initial_command(), MAX_COMMAND_LINE_LENGTH, peek_buf_has_control0_command(), set_uint16(), connection_t::state, TO_CONN, and tor_assert.

Referenced by connection_process_inbuf().

4.17.4.5 int connection_control_reached_eof (control_connection_t ∗ *conn*)

Called when **conn** has gotten its socket closed.

References LD_CONTROL, TO_CONN, and tor_assert.

Referenced by connection_reached_eof().

4.17.4.6 static void connection_printf_to_buf (control_connection_t ∗ *conn*, const char ∗ *format*, *...*) `[static]`

Acts like sprintf, but writes its formatted string to the end of **conn**->outbuf. The message may be truncated if it is too long, but it will always end with a CRLF sequence.

Currently the length of the message is limited to 1024 (including the ending CR LF NUL ("\\r\\n\\0").

References LD_BUG, TO_CONN, and tor_vsnprintf().

Referenced by connection_control_process_inbuf(), control_setconf_helper(), getargs_helper(), handle_control_attachstream(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_closestream(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_loadconf(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_redirectstream(), handle_control_setcircuitpurpose(), handle_control_setevents(), handle_control_signal(), and handle_control_usefeature().

4.17.4.7 static INLINE void connection_write_str_to_buf (const char ∗ *s*, control_connection_t ∗ *conn*) `[static]`

Append a NUL-terminated string **s** to the end of **conn**->outbuf.

References TO_CONN.

Referenced by connection_control_process_inbuf(), control_setconf_helper(), handle_control_attachstream(), handle_control_authenticate(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_saveconf(), handle_control_setcircuitpurpose(), and send_control_done().

4.17.4.8 void control_adjust_event_log_severity (void)

Adjust the log severities that result in control_event_logmsg being called to match the severity of log messages that any controllers are interested in.

References change_callback_log_severity(), control_event_logmsg(), EVENT_IS_INTERESTING, and event_to_log_severity().

Referenced by options_act_reversible().

4.17.4.9 int control_event_address_mapped (const char ∗ *from*, const char ∗ *to*, time_t *expires*, const char ∗ *error*)

Called when an address mapping on **from** from changes to **to**. **expires** values less than 3 are special; see **connection_edge.c** (p. 429). If **error** is non-NULL, it is an error code describing the failure mode of the mapping.

References ALL_FORMATS, EVENT_IS_INTERESTING, format_iso_time(), format_local_iso_time(), and send_control_event().

Referenced by addressmap_register(), handle_control_resolve(), and tell_controller_about_resolved_result().

4.17.4.10 int control_event_bandwidth_used (uint32_t *n_read*, uint32_t *n_written*)

A second or more has elapsed: tell any interested control connections how much bandwidth we used.

References ALL_FORMATS, EVENT_IS_INTERESTING, and send_control_event().

Referenced by second_elapsed_callback().

4.17.4.11 void control_event_bootstrap (bootstrap_status_t *status*, int *progress*)

Called when Tor has made progress at bootstrapping its directory information and initial circuits.

status is the new status, that is, what task we will be doing next. **percent** is zero if we just started this task, else it represents progress on the task.

References BOOTSTRAP_MSG_LEN, bootstrap_status_to_string(), control_event_client_status(), last_sent_bootstrap_message, LD_CONTROL, and tor_snprintf().

Referenced by circuit_handle_first_hop(), circuit_send_next_onion_skin(), connection_edge_process_relay_cell_not_open(), connection_or_finished_connecting(), do_main_loop(), load_downloaded_routers(), and update_router_have_minimum_dir_info().

4.17.4.12 void control_event_bootstrap_problem (const char ∗ *warn*, int *reason*)

Called when Tor has failed to make bootstrapping progress in a way that indicates a problem. **warn** gives a hint as to why, and **reason** provides an "or_conn_end_reason" tag.

References any_bridge_descriptors_known(), any_pending_bridge_descriptor_fetches(), BOOTSTRAP_MSG_LEN, bootstrap_status_to_string(), control_event_client_status(), get_options(), last_sent_bootstrap_message, LD_CONTROL, orconn_end_reason_to_control_string(), and tor_snprintf().

Referenced by connection_about_to_close_connection(), connection_or_check_valid_tls_handshake(), and connection_or_connect_failed().

4.17.4.13 int control_event_buildtimeout_set (const circuit_build_times_t ∗ *cbt*, buildtimeout_set_event_t *type*)

Called when we compute a new circuitbuildtimeout

References ALL_FORMATS, circuit_build_times_t::alpha, control_event_is_interesting(), send_control_event(), circuit_build_times_t::timeout_ms, circuit_build_times_t::total_build_times, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_init(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), and circuit_build_times_set_timeout().

4.17.4.14 int control_event_circuit_status (origin_circuit_t ∗ *circ*, circuit_status_event_t *tp*, int *reason_code*)

Something has happened to circuit **circ**: tell any interested control connections.

References origin_circuit_t::_base, ALL_FORMATS, circuit_end_reason_to_control_string(), circuit_list_path_for_controller(), circuit_purpose_to_controller_string(), END_CIRC_REASON_FLAG_REMOTE, EVENT_IS_INTERESTING, origin_circuit_t::global_identifier, LD_BUG, circuit_t::purpose, send_control_event(), tor_assert, tor_free, and tor_snprintf().

Referenced by _circuit_mark_for_close(), circuit_establish_circuit(), circuit_finish_handshake(), circuit_has_opened(), and handle_control_extendcircuit().

4.17.4.15 int control_event_client_status (int *severity*, const char ∗ *format*, ...)

Format and send an EVENT_STATUS_CLIENT event whose main text is obtained by formatting the arguments using the printf-style **format**.

References control_event_status(), and EVENT_IS_INTERESTING.

Referenced by circuit_note_clock_jumped(), circuit_send_next_onion_skin(), consider_plaintext_ports(), control_event_bootstrap(), control_event_bootstrap_problem(), fetch_from_buf_socks(), networkstatus_set_current_consensus(), and update_router_have_minimum_dir_info().

4.17.4.16 void control_event_clients_seen (const char ∗ *controller_str*)

We just generated a new summary of which countries we've seen clients from recently. Send a copy to the controller in case it wants to display it for the user.

References send_control_event().

Referenced by geoip_bridge_stats_write().

4.17.4.17 int control_event_descriptors_changed (smartlist_t * *routers*)

Called whenever we receive new router descriptors: tell any interested control connections. **routers** is a list of routerinfo_t's.

References ALL_FORMATS, EVENT_IS_INTERESTING, MAX_VERBOSE_NICKNAME_LEN, router_get_verbose_nickname(), send_control_event_string(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

Referenced by dirserv_add_descriptor(), and routerlist_descriptors_added().

4.17.4.18 int control_event_general_status (int *severity*, const char * *format*, *...*)

Format and send an EVENT_STATUS_GENERAL event whose main text is obtained by formatting the arguments using the printf-style **format**.

References control_event_status(), and EVENT_IS_INTERESTING.

Referenced by circuit_note_clock_jumped(), command_process_netinfo_cell(), connection_dir_client_reached_eof(), control_event_logmsg(), directory_all_unreachable(), init_libevent(), networkstatus_set_current_consensus(), router_set_networkstatus_v2(), routers_update_all_from_networkstatus(), and warn_too_many_conns().

4.17.4.19 int control_event_guard (const char * *nickname*, const char * *digest*, const char * *status*)

Called when the status of an entry guard with the given **nickname** and identity **digest** has changed to **status**: tells any controllers that care.

References ALL_FORMATS, base16_encode(), DIGEST_LEN, EVENT_IS_INTERESTING, HEX_DIGEST_LEN, MAX_VERBOSE_NICKNAME_LEN, router_get_by_digest(), router_get_verbose_nickname(), send_control_event(), and tor_snprintf().

Referenced by add_an_entry_guard(), control_event_guard_deferred(), entry_guard_set_status(), remove_dead_entry_guards(), and remove_obsolete_entry_guards().

4.17.4.20 int control_event_is_interesting (int *event*)

Return true iff the event with code **c** is being sent to any current control connection. This is useful if the amount of work needed to prepare to call the appropriate control_event_...() function is high.

References EVENT_IS_INTERESTING.

Referenced by control_event_buildtimeout_set(), control_event_newconsensus(), and notify_control_networkstatus_changed().

4.17.4.21 void control_event_logmsg (int *severity*, uint32_t *domain*, const char * *msg*)

We got a log message: tell any interested control connections.

References ALL_FORMATS, control_event_general_status(), disable_log_messages, esc_for_log(), EVENT_IS_INTERESTING, in_main_thread(), LD_BUG, log_severity_to_event(), send_control_event(), and tor_free.

Referenced by control_adjust_event_log_severity(), and options_act_reversible().

4.17.4.22 int control_event_my_descriptor_changed (void)

Our own router descriptor has changed; tell any controllers that care.

References ALL_FORMATS, and send_control_event().

Referenced by router_rebuild_descriptor().

4.17.4.23 int control_event_networkstatus_changed (smartlist_t * *statuses*)

Called when the routerstatus_ts **statuses** have changed: sends an NS event to any controller that cares.

References control_event_networkstatus_changed_helper().

Referenced by control_event_networkstatus_changed_single(), and notify_control_networkstatus_changed().

4.17.4.24 static int control_event_networkstatus_changed_helper (smartlist_t * *statuses*, uint16_t *event*, const char * *event_string*) `[static]`

Helper function for NS-style events. Constructs and sends an event of type **event** with string **event_string** out of the set of networkstatuses **statuses**. Currently it is used for NS events and NEWCONSENSUS events.

References ALL_FORMATS, EVENT_IS_INTERESTING, networkstatus_getinfo_helper_single(), send_control_event_string(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and write_escaped_data().

Referenced by control_event_networkstatus_changed(), and control_event_newconsensus().

4.17.4.25 int control_event_networkstatus_changed_single (routerstatus_t * *rs*)

Called when a single local_routerstatus_t has changed: Sends an NS event to any controller that cares.

References control_event_networkstatus_changed(), EVENT_IS_INTERESTING, smartlist_add(), smartlist_create(), and smartlist_free().

Referenced by router_set_status().

4.17.4.26 int control_event_newconsensus (const networkstatus_t * *consensus*)

Called when we get a new consensus networkstatus. Sends a NEWCONSENSUS event consisting of an NS-style line for each relay in the consensus.

References control_event_is_interesting(), control_event_networkstatus_changed_helper(), and networkstatus_t::routerstatus_list.

Referenced by notify_control_networkstatus_changed().

4.17.4.27 int control_event_or_authdir_new_descriptor (const char ∗ *action*, const char ∗ *desc*, size_t *desclen*, const char ∗ *msg*)

The authoritative dirserver has received a new descriptor that has passed basic syntax checks and is properly self-signed.

Notify any interested party of the new descriptor and what has been done with it, and also optionally give an explanation/reason.

References ALL_FORMATS, EVENT_IS_INTERESTING, send_control_event_string(), tor_free, tor_snprintf(), and write_escaped_data().

Referenced by dirserv_add_descriptor().

4.17.4.28 int control_event_or_conn_status (or_connection_t ∗ *conn*, or_conn_status_event_t *tp*, int *reason*)

Called when the status of an OR connection **conn** changes: tell any interested control connections. **tp** is the new status for the connection. If **conn** has just closed or failed, then **reason** may be the reason why.

References ALL_FORMATS, circuit_count_pending_on_or_conn(), EVENT_IS_INTERESTING, LD_BUG, or_connection_t::n_circuits, orconn_end_reason_to_control_string(), orconn_target_get_name(), send_control_event(), and tor_snprintf().

Referenced by connection_about_to_close_connection(), connection_init_accepted_conn(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_connect_failed(), and connection_or_set_state_open().

4.17.4.29 int control_event_server_status (int *severity*, const char ∗ *format*, *...*)

Format and send an EVENT_STATUS_SERVER event whose main text is obtained by formatting the arguments using the printf-style **format**.

References control_event_status(), and EVENT_IS_INTERESTING.

Referenced by add_wildcarded_test_address(), connection_dir_client_reached_eof(), evdns_log_cb(), inform_testing_reachability(), resolve_my_address(), router_dirport_found_reachable(), router_new_address_suggestion(), router_orport_found_reachable(), second_elapsed_callback(), and wildcard_increment_answer().

4.17.4.30 static int control_event_status (int *type*, int *severity*, const char ∗ *format*, va_list *args*) `[static]`

Helper: sends a status event where **type** is one of EVENT_STATUS_{GENERAL,CLIENT,SERVER}, where **severity** is one of LOG_{NOTICE,WARN,ERR}, and where **format** is a printf-style format string corresponding to **args**.

References ALL_FORMATS, LD_BUG, send_control_event_impl(), and tor_snprintf().

Referenced by control_event_client_status(), control_event_general_status(), and control_event_server_status().

4.17.4.31 int control_event_stream_bandwidth (edge_connection_t * *edge_conn*)

Print out STREAM_BW event for a single conn

References edge_connection_t::_base, ALL_FORMATS, EVENT_IS_INTERESTING, connection_t::global_identifier, edge_connection_t::n_read, edge_connection_t::n_written, and send_control_event().

Referenced by connection_about_to_close_connection(), and connection_edge_destroy().

4.17.4.32 int control_event_stream_bandwidth_used (void)

A second or more has elapsed: tell any interested control connections how much bandwidth streams have used.

References edge_connection_t::_base, ALL_FORMATS, CONN_TYPE_AP, EVENT_IS_INTERESTING, get_connection_array(), connection_t::global_identifier, edge_connection_t::n_read, edge_connection_t::n_written, send_control_event(), and TO_EDGE_CONN().

Referenced by second_elapsed_callback().

4.17.4.33 int control_event_stream_status (edge_connection_t * *conn*, stream_status_event_t *tp*, int *reason_code*)

Something has happened to the stream associated with AP connection **conn**: tell any interested control connections.

References edge_connection_t::_base, ALL_FORMATS, circuit_get_by_edge_conn(), CIRCUIT_IS_ORIGIN, socks_request_t::command, CONN_TYPE_DIR, DIR_PURPOSE_IS_UPLOAD, END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED, END_STREAM_REASON_FLAG_REMOTE, EVENT_IS_INTERESTING, origin_circuit_t::global_identifier, connection_t::global_identifier, edge_connection_t::is_dns_request, LD_BUG, connection_t::purpose, REMAP_STREAM_SOURCE_CACHE, REMAP_STREAM_SOURCE_EXIT, send_control_event(), edge_connection_t::socks_request, stream_end_reason_to_control_string(), TO_CONN, TO_ORIGIN_CIRCUIT(), tor_assert, tor_free, tor_snprintf(), connection_t::type, edge_connection_t::use_begindir, and write_stream_target_to_buf().

Referenced by connection_about_to_close_connection(), connection_ap_detach_retriable(), connection_ap_handshake_process_socks(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_reply(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_destroy(), evdns_server_callback(), and remap_event_helper().

4.17.4.34 static int control_setconf_helper (control_connection_t * *conn*, uint32_t *len*, char * *body*, int *use_defaults*) `[static]`

Helper for setconf and resetconf. Acts like setconf, except it passes **use_defaults** on to **options_trial_assign()** (p. 397). Modifies the contents of body.

References config_free_lines(), config_get_lines(), connection_printf_to_buf(), connection_write_str_to_buf(), extract_escaped_string(), LD_CONTROL, options_trial_assign(), send_control_done(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

Referenced by handle_control_resetconf(), and handle_control_setconf().

4.17.4.35 void control_update_global_event_mask (void)

Set **global_event_mask**∗ to the bitwise OR of each live control connection's event_mask field.

References CONN_TYPE_CONTROL, get_connection_array(), global_event_mask, STATE_IS_OPEN, and TO_CONTROL_CONN().

Referenced by connection_free(), connection_free_all(), and handle_control_setevents().

4.17.4.36 static const char∗ decode_escaped_string (const char ∗ *start*, size_t *in_len_max*, char ∗∗ *out*, size_t ∗ *out_len*) `[static]`

Given a pointer to a string starting at **start** containing **in_len_max** characters, decode a string beginning with one double quote, containing any number of non-quote characters or characters escaped with a backslash, and ending with a final double quote. Place the resulting string (unquoted, unescaped) into a newly allocated string in ∗**out**; store its length in **out_len**. On success, return a pointer to the character immediately following the escaped string. On failure, return NULL.

References get_escaped_string_length(), and tor_assert.

Referenced by handle_control_authenticate().

4.17.4.37 smartlist_t∗ decode_hashed_passwords (config_line_t ∗ *passwords*)

Decode the hashed, base64'd passwords stored in **passwords**. Return a smartlist of acceptable passwords (unterminated strings of length S2K_SPECIFIER_LEN+DIGEST_LEN) on success, or NULL on failure.

References base16_decode(), base64_decode(), DIGEST_LEN, config_line_t::next, S2K_SPECIFIER_LEN, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), tor_assert, tor_free, and config_line_t::value.

Referenced by handle_control_authenticate().

4.17.4.38 void disable_control_logging (void)

Called when we are sending a log message to the controllers: suspend sending further log messages to the controllers until we're done. Used by CONN_LOG_PROTECT.

References disable_log_messages.

4.17.4.39 void enable_control_logging (void)

We're done sending a log message to the controllers: re-enable controller logging. Used by CONN_LOG_PROTECT.

References disable_log_messages, and tor_assert.

4.17.4.40 static INLINE int event_to_log_severity (int *event*) `[static]`

Given a control event code for a message event, return the corresponding log severity.

Referenced by control_adjust_event_log_severity().

4.17.4.41 static const char* extract_escaped_string (const char * *start*, size_t *in_len_max*, char ** *out*, size_t * *out_len*) [static]

As decode_escaped_string, but does not decode the string: copies the entire thing, including quotation marks.

References get_escaped_string_length().

Referenced by control_setconf_helper().

4.17.4.42 static const char* find_element_starting_with (smartlist_t * *sl*, int *start_at*, const char * *prefix*) [static]

Helper. Return the first element of **sl** at index **start_at** or higher that starts with **prefix**, case-insensitive. Return NULL if no such element exists.

References strcasecmpstart().

Referenced by handle_control_attachstream(), handle_control_extendcircuit(), handle_control_resolve(), and handle_control_setcircuitpurpose().

4.17.4.43 static origin_circuit_t* get_circ (const char * *id*) [static]

Given a text circuit **id**, return the corresponding circuit.

References circuit_get_by_global_id(), and tor_parse_ulong().

Referenced by handle_control_attachstream(), handle_control_closecircuit(), handle_control_extendcircuit(), and handle_control_setcircuitpurpose().

4.17.4.44 static char * get_cookie_file (void) [static]

Helper: Return a newly allocated string containing a path to the file where we store our authentication cookie.

References or_options_t::CookieAuthFile, get_datadir_fname, and get_options().

Referenced by handle_control_protocolinfo(), and init_cookie_authentication().

4.17.4.45 static INLINE int get_escaped_string_length (const char * *start*, size_t *in_len_max*, int * *chars_out*) [static]

If the first **in_len_max** characters in **start** contain a double-quoted string with escaped characters, return the length of that string (as encoded, including quotes). Otherwise return -1.

Referenced by decode_escaped_string(), and extract_escaped_string().

4.17.4.46 static edge_connection_t* get_stream (const char * *id*) [static]

Given a text stream **id**, return the corresponding AP connection.

References CONN_TYPE_AP, connection_get_by_global_id(), connection_t::marked_for_close, TO_EDGE_CONN(), tor_parse_uint64(), and connection_t::type.

Referenced by handle_control_attachstream(), handle_control_closestream(), and handle_control_redirectstream().

4.17.4.47 static smartlist_t∗ getargs_helper (const char ∗ *command*, control_connection_t ∗ *conn*, const char ∗ *body*, int *min_args*, int *max_args*) `[static]`

Return a newly allocated smartlist containing the arguments to the command waiting in **body**. If there are fewer than **min_args** arguments, or if **max_args** is nonnegative and there are more than **max_args** arguments, send a 512 error to the controller, using **command** as the command name in the error message.

References connection_printf_to_buf(), smartlist_create(), smartlist_free(), smartlist_split_string(), and tor_free.

Referenced by handle_control_attachstream(), handle_control_closecircuit(), handle_control_closestream(), handle_control_extendcircuit(), handle_control_redirectstream(), and handle_control_setcircuitpurpose().

4.17.4.48 static int getinfo_helper_dir (control_connection_t ∗ *control_conn*, const char ∗ *question*, char ∗∗ *answer*) `[static]`

Implementation helper for GETINFO: knows the answers for questions about directory information.

References routerinfo_t::cache_info, router_get_by_hexdigest(), router_get_by_nickname(), router_get_routerlist(), routerlist, routerlist_t::routers, signed_descriptor_get_body(), signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), strcmpstart(), and tor_free.

4.17.4.49 static int getinfo_helper_events (control_connection_t ∗ *control_conn*, const char ∗ *question*, char ∗∗ *answer*) `[static]`

Implementation helper for GETINFO: knows how to generate summaries of the current states of things we send events about.

References or_connection_t::_base, edge_connection_t::_base, _circuit_get_global_list(), addressmap_get_mappings(), AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_RESOLVE_WAIT, AP_CONN_STATE_SOCKS_WAIT, check_whether_dirport_reachable(), check_whether_orport_reachable(), circuit_get_by_edge_conn(), CIRCUIT_IS_ORIGIN, circuit_list_path_for_controller(), circuit_purpose_to_controller_string(), CIRCUIT_STATE_OPEN, networkstatus_t::client_versions, socks_request_t::command, CONN_TYPE_AP, CONN_TYPE_OR, directories_have_accepted_server_descriptor(), geoip_get_bridge_stats_controller(), get_connection_array(), get_n_authorities(), get_options(), origin_circuit_t::global_identifier, connection_t::global_identifier, has_completed_circuit, last_sent_bootstrap_message, LD_BUG, LD_GENERAL, circuit_t::marked_for_close, networkstatus_get_latest_consensus(), circuit_t::next, or_connection_t::nickname, OR_CONN_STATE_OPEN, orconn_target_get_name(), circuit_t::purpose, router_have_minimum_dir_info(), server_mode(), networkstatus_t::server_versions, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), edge_connection_t::socks_request, connection_t::state, circuit_t::state, strcmpstart(), TO_EDGE_CONN(), TO_OR_CONN(), TO_ORIGIN_CIRCUIT(), tor_fragile_assert, tor_free, tor_snprintf(), tor_version_is_obsolete(), V3_AUTHORITY, VS_EMPTY, VS_NEW, VS_NEW_IN_SERIES, VS_OLD, VS_RECOMMENDED, VS_UNKNOWN, VS_UNRECOMMENDED, and write_stream_target_to_buf().

4.17.4.50 static int getinfo_helper_misc (control_connection_t * *conn*, const char * *question*, char ** *answer*) [static]

Implementation helper for GETINFO: knows the answers for various trivial-to-implement questions.

References base16_encode(), routerinfo_t::cache_info, DIGEST_LEN, get_options(), get_torrc_fname(), get_version(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, list_getinfo_options(), options_dump(), router_get_my_routerinfo(), router_pick_published_address(), and tor_dup_ip().

4.17.4.51 static int handle_control_attachstream (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get an ATTACHSTREAM message. Try to attach the requested stream, and report success or failure.

References origin_circuit_t::_base, edge_connection_t::_base, routerinfo_t::allow_single_hop_exits, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_RESOLVE_WAIT, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, edge_connection_t::chosen_exit_name, circuit_detach_stream(), circuit_get_by_edge_conn(), circuit_get_cpath_hop(), circuit_get_cpath_len(), CIRCUIT_STATE_OPEN, connection_ap_handshake_rewrite_and_attach(), connection_edge_end(), connection_printf_to_buf(), connection_write_str_to_buf(), DIGEST_LEN, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, find_element_starting_with(), get_circ(), get_options(), get_stream(), getargs_helper(), hex_str(), extend_info_t::identity_digest, router_get_by_digest(), send_control_done(), smartlist_free(), circuit_t::state, connection_t::state, tor_digest_is_zero(), tor_free, and tor_parse_ulong().

Referenced by connection_control_process_inbuf().

4.17.4.52 static int handle_control_authenticate (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get an AUTHENTICATE message. Check whether the authentication is valid, and if so, update the connection's state to OPEN. Reply with DONE or ERROR.

We only get here if both kinds of authentication failed.

References control_connection_t::_base, authentication_cookie, AUTHENTICATION_COOKIE_LEN, base16_decode(), connection_printf_to_buf(), connection_write_str_to_buf(), CONTROL_CONN_STATE_OPEN, or_options_t::CookieAuthentication, decode_escaped_string(), decode_hashed_passwords(), DIGEST_LEN, get_options(), or_options_t::HashedControlPassword, or_options_t::HashedControlSessionPassword, LD_CONTROL, connection_t::s, S2K_SPECIFIER_LEN, secret_to_key(), send_control_done(), smartlist_add_all(), smartlist_create(), smartlist_free(), connection_t::state, TO_CONN, tor_assert, and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.53 static int handle_control_closecircuit (control_connection_t ∗ *conn*, uint32_t *len*, const char ∗ *body*) [static]

Called when we get a CLOSECIRCUIT command; try to close the named circuit and report success or failure.

References connection_printf_to_buf(), get_circ(), getargs_helper(), LD_CONTROL, origin_circuit_t::p_streams, send_control_done(), smartlist_free(), TO_CIRCUIT, and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.54 static int handle_control_closestream (control_connection_t ∗ *conn*, uint32_t *len*, const char ∗ *body*) [static]

Called when we get a CLOSESTREAM command; try to close the named stream and report success or failure.

References connection_printf_to_buf(), get_stream(), getargs_helper(), send_control_done(), smartlist_free(), tor_free, and tor_parse_ulong().

Referenced by connection_control_process_inbuf().

4.17.4.55 static int handle_control_extendcircuit (control_connection_t ∗ *conn*, uint32_t *len*, const char ∗ *body*) [static]

Called when we get an EXTENDCIRCUIT message. Try to extend the listed circuit, and report success or failure.

References origin_circuit_t::_base, CIRCLAUNCH_NEED_CAPACITY, circuit_append_new_exit(), circuit_handle_first_hop(), circuit_launch_by_router(), CIRCUIT_PURPOSE_C_GENERAL, circuit_purpose_from_string(), CIRCUIT_PURPOSE_UNKNOWN, circuit_send_next_onion_skin(), circuit_set_state(), CIRCUIT_STATE_BUILDING, CIRCUIT_STATE_OPEN, connection_printf_to_buf(), connection_write_str_to_buf(), control_event_circuit_status(), extend_info_free(), extend_info_from_router(), find_element_starting_with(), get_circ(), getargs_helper(), origin_circuit_t::global_identifier, is_keyval_pair(), LD_CONTROL, origin_circuit_init(), router_get_by_nickname(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), circuit_t::state, TO_CIRCUIT, and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.56 static int handle_control_getconf (control_connection_t ∗ *conn*, uint32_t *body_len*, const char ∗ *body*) [static]

Called when we receive a GETCONF message. Parse the request, and reply with a CONFVALUE or an ERROR message

References connection_printf_to_buf(), connection_write_str_to_buf(), get_options(), config_line_t::key, config_line_t::next, option_get_assignment(), option_get_canonical_name(), option_is_recognized(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_split_string(), TO_CONN, tor_assert, tor_free, tor_snprintf(), and config_line_t::value.

Referenced by connection_control_process_inbuf().

4.17.4.57 static int handle_control_getinfo (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we receive a GETINFO command. Try to fetch all requested information, and reply with information or error message.

References connection_printf_to_buf(), connection_write_str_to_buf(), handle_getinfo_helper(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), TO_CONN, tor_free, and write_escaped_data().

Referenced by connection_control_process_inbuf().

4.17.4.58 static int handle_control_loadconf (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a +LOADCONF message.

References connection_printf_to_buf(), LD_CONTROL, options_init_from_string(), send_control_done(), and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.59 static int handle_control_mapaddress (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a MAPADDRESS command; try to bind all listed addresses, and report success or failure.

References address_is_invalid_destination(), addressmap_register(), addressmap_register_virtual_address(), ADDRMAPSRC_CONTROLLER, LD_CONTROL, safe_str_client(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_split_string(), TO_CONN, tor_free, tor_snprintf(), and tor_strlower().

Referenced by connection_control_process_inbuf().

4.17.4.60 static int handle_control_postdescriptor (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a POSTDESCRIPTOR message. Try to learn the provided descriptor, and report success or failure.

References connection_printf_to_buf(), read_escaped_data(), router_load_single_router(), router_purpose_from_string(), ROUTER_PURPOSE_GENERAL, ROUTER_PURPOSE_UNKNOWN, send_control_done(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcasecmpstart(), tor_assert, and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.61 static int handle_control_protocolinfo (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a PROTOCOLINFO command: send back a reply.

References connection_printf_to_buf(), or_options_t::CookieAuthentication, esc_for_log(), escaped(), get_cookie_file(), get_options(), or_options_t::HashedControlPassword, or_options_t::HashedControlSessionPassword, control_connection_t::have_sent_protocolinfo, smartlist_-

add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_split_string(), STATE_IS_OPEN, TO_CONN, tor_free, and tor_parse_long().

Referenced by connection_control_process_inbuf().

4.17.4.62 static int handle_control_redirectstream (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we receive a REDIRECTSTERAM command. Try to change the target address of the named AP stream, and report success or failure.

References socks_request_t::address, connection_printf_to_buf(), get_stream(), getargs_helper(), socks_request_t::port, send_control_done(), smartlist_free(), edge_connection_t::socks_request, tor_free, and tor_parse_ulong().

Referenced by connection_control_process_inbuf().

4.17.4.63 static int handle_control_resetconf (control_connection_t * *conn*, uint32_t *len*, char * *body*) [static]

Called when we receive a RESETCONF message: parse the body and try to update our configuration. Reply with a DONE or ERROR message. Modifies the contents of body.

References control_setconf_helper().

Referenced by connection_control_process_inbuf().

4.17.4.64 static int handle_control_resolve (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a RESOLVE command: start trying to resolve the listed addresses.

References control_event_address_mapped(), dnsserv_launch_request(), control_connection_t::event_mask, find_element_starting_with(), is_keyval_pair(), LD_CONTROL, send_control_done(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.65 static int handle_control_saveconf (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a SAVECONF command. Try to flush the current options to disk, and report success or failure.

References connection_write_str_to_buf(), options_save_current(), and send_control_done().

Referenced by connection_control_process_inbuf().

4.17.4.66 static int handle_control_setcircuitpurpose (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a SETCIRCUITPURPOSE message. If we can find the circuit and it's a valid purpose, change it.

References origin_circuit_t::_base, circuit_purpose_from_string(), CIRCUIT_PURPOSE_UNKNOWN, connection_printf_to_buf(), connection_write_str_to_buf(), find_element_starting_with(), get_circ(), getargs_helper(), circuit_t::purpose, smartlist_free(), and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.67 static int handle_control_setconf (control_connection_t * *conn*, uint32_t *len*, char * *body*) [static]

Called when we receive a SETCONF message: parse the body and try to update our configuration. Reply with a DONE or ERROR message. Modifies the contents of body.

References control_setconf_helper().

Referenced by connection_control_process_inbuf().

4.17.4.68 static int handle_control_setevents (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a SETEVENTS message: update conn->event_mask, and reply with DONE or ERROR.

References connection_printf_to_buf(), control_update_global_event_mask(), control_connection_t::event_mask, send_control_done(), smartlist_create(), smartlist_free(), smartlist_split_string(), and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.69 static int handle_control_signal (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a SIGNAL command. React to the provided signal, and report success or failure. (If the signal results in a shutdown, success may not be reported.)

References connection_printf_to_buf(), control_signal_act(), send_control_done(), TO_CONN, and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.70 static int handle_control_usefeature (control_connection_t * *conn*, uint32_t *len*, const char * *body*) [static]

Called when we get a USEFEATURE command: parse the feature list, and set up the control_connection's options properly.

References connection_printf_to_buf(), send_control_done(), smartlist_create(), smartlist_free(), smartlist_split_string(), and tor_free.

Referenced by connection_control_process_inbuf().

4.17.4.71 static int handle_getinfo_helper (control_connection_t * *control_conn*, const char * *question*, char ** *answer*) [static]

Lookup the 'getinfo' entry **question**, and return the answer in ***answer** (or NULL if key not recognized). Return 0 if success or unrecognized, or -1 if recognized but internal error.

References getinfo_item_t::fn, strcmpstart(), tor_assert, and getinfo_item_t::varname.

Referenced by handle_control_getinfo().

4.17.4.72 int init_cookie_authentication (int *enabled*)

Choose a random authentication cookie and write it to disk. Anybody who can read the cookie from disk will be considered authorized to use the control connection. Return -1 if we can't write the file, or 0 on success.

References authentication_cookie, authentication_cookie_is_set, AUTHENTICATION_COOKIE_LEN, crypto_rand(), escaped(), get_cookie_file(), get_options(), LD_FS, tor_free, and write_bytes_to_file().

Referenced by options_act().

4.17.4.73 static int is_keyval_pair (const char ∗ *s*) `[static]`

Helper. Return true iff s is an argument that we should treat as a key-value pair.

Referenced by handle_control_extendcircuit(), and handle_control_resolve().

4.17.4.74 static int is_valid_initial_command (control_connection_t ∗ *conn*, const char ∗ *cmd*) `[static]`

Return true iff **cmd** is allowable (or at least forgivable) at this stage of the protocol.

References control_connection_t::_base, CONTROL_CONN_STATE_OPEN, control_connection_t::have_sent_protocolinfo, and connection_t::state.

Referenced by connection_control_process_inbuf().

4.17.4.75 static char ∗ list_getinfo_options (void) `[static]`

Allocate and return a list of recognized GETINFO options.

References smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort_strings(), tor_asprintf(), tor_free, and getinfo_item_t::varname.

Referenced by getinfo_helper_misc().

4.17.4.76 static INLINE int log_severity_to_event (int *severity*) `[static]`

Given a log severity, return the corresponding control event code.

Referenced by control_event_logmsg().

4.17.4.77 static char∗ munge_extrainfo_into_routerinfo (const char ∗ *ri_body*, signed_descriptor_t ∗ *ri*, signed_descriptor_t ∗ *ei*) `[static]`

Awful hack: return a newly allocated string based on a routerinfo and (possibly) an extrainfo, sticking the read-history and write-history from **ei** into the resulting string. The thing you get back won't necessarily have a valid signature.

New code should never use this; it's for backward compatibility.

NOTE: **ri_body** is as returned by signed_descriptor_get_body: it might not be NUL-terminated.

References signed_descriptor_get_body(), signed_descriptor_t::signed_descriptor_len, tor_assert, and tor_free.

4.17.4.78 static void orconn_target_get_name (char * *name*, size_t *len*, or_connection_t * *conn*) `[static]`

Figure out the best name for the target router of an OR connection **conn**, and write it into the **len**-character buffer **name**.

References or_connection_t::_base, connection_t::address, base16_encode(), DIGEST_LEN, or_connection_t::identity_digest, MAX_VERBOSE_NICKNAME_LEN, connection_t::port, router_get_by_digest(), router_get_verbose_nickname(), tor_assert, tor_digest_is_zero(), and tor_snprintf().

Referenced by control_event_or_conn_status(), and getinfo_helper_events().

4.17.4.79 size_t read_escaped_data (const char * *data*, size_t *len*, char ** *out*)

Given a **len**-character string in **data**, made of lines terminated by CRLF, allocate a new string in *__out__, and copy the contents of **data** into *__out__, removing any period that appears at the start of a line, and replacing all CRLF sequences with LF. Return the number of bytes in *__out__.

Referenced by handle_control_postdescriptor().

4.17.4.80 static void send_control_done (control_connection_t * *conn*) `[static]`

Send a "DONE" message down the control connection **conn**.

References connection_write_str_to_buf().

Referenced by control_setconf_helper(), handle_control_attachstream(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_closestream(), handle_control_loadconf(), handle_control_postdescriptor(), handle_control_redirectstream(), handle_control_resolve(), handle_control_saveconf(), handle_control_setevents(), handle_control_signal(), and handle_control_usefeature().

4.17.4.81 static void send_control_event (uint16_t *event*, event_format_t *which*, const char * *format*, *...*) `[static]`

Send an event to all v1 controllers that are listening for code **event**. The event's body is created by the printf-style format in **format**, and other arguments as provided.

Currently the length of the message is limited to 1024 (including the ending \n\r\0.

References send_control_event_impl().

Referenced by control_event_address_mapped(), control_event_bandwidth_used(), control_event_buildtimeout_set(), control_event_circuit_status(), control_event_clients_seen(), control_event_guard(), control_event_logmsg(), control_event_my_descriptor_changed(), control_event_or_conn_status(), control_event_stream_bandwidth(), control_event_stream_bandwidth_used(), and control_event_stream_status().

4.17.4.82 static void send_control_event_impl (uint16_t *event*, event_format_t *which*, const char ∗ *format*, va_list *ap*) `[static]`

Helper for send_control1_event and send_control1_event_extended: Send an event to all v1 controllers that are listening for code **event**. The event's body is created by the printf-style format in **format**, and other arguments as provided.

Currently the length of the message is limited to 1024 (including the ending \r\n\0).

References ALL_FORMATS, LD_BUG, send_control_event_string(), and tor_vsnprintf().

Referenced by control_event_status(), and send_control_event().

4.17.4.83 static void send_control_event_string (uint16_t *event*, event_format_t *which*, const char ∗ *msg*) `[static]`

Send an event to all v1 controllers that are listening for code **event**. The event's body is given by **msg**.

If **which** & SHORT_NAMES, the event contains short-format names: send it to controllers that haven't enabled the VERBOSE_NAMES feature. If **which** & LONG_NAMES, the event contains long-format names: send it to controllers that *have* enabled VERBOSE_NAMES.

The EXTENDED_FORMAT and NONEXTENDED_FORMAT flags behave similarly with respect to the EXTENDED_EVENTS feature.

References CONN_TYPE_CONTROL, CONTROL_CONN_STATE_OPEN, control_connection_t::event_mask, get_connection_array(), strcmpstart(), TO_CONN, TO_CONTROL_CONN(), and tor_assert.

Referenced by control_event_descriptors_changed(), control_event_networkstatus_changed_helper(), control_event_or_authdir_new_descriptor(), and send_control_event_impl().

4.17.4.84 size_t write_escaped_data (const char ∗ *data*, size_t *len*, char ∗∗ *out*)

Given a **len**-character string in **data**, made of lines terminated by CRLF, allocate a new string in ∗**out**, and copy the contents of **data** into ∗**out**, adding a period before any period that appears at the start of a line, and adding a period-CRLF line at the end. Replace all LF characters sequences with CRLF. Return the number of bytes in ∗**out**.

References tor_assert.

Referenced by control_event_networkstatus_changed_helper(), control_event_or_authdir_new_descriptor(), and handle_control_getinfo().

4.17.4.85 static int write_stream_target_to_buf (edge_connection_t ∗ *conn*, char ∗ *buf*, size_t *len*) `[static]`

Given an AP connection **conn** and a **len**-character buffer **buf**, determine the address:port combination requested on **conn**, and write it to **buf**. Return 0 on success, -1 on failure.

References socks_request_t::address, edge_connection_t::chosen_exit_name, connection_edge_is_rendezvous_stream(), socks_request_t::port, edge_connection_t::socks_request, and tor_snprintf().

Referenced by control_event_stream_status(), and getinfo_helper_events().

4.17.5 Variable Documentation

4.17.5.1 char authentication_cookie[AUTHENTICATION_COOKIE_LEN] `[static]`

If authentication_cookie_is_set, a secret cookie that we've stored to disk and which we're using to authenticate controllers. (If the controller can read it off disk, it has permission to connect.

Referenced by handle_control_authenticate(), and init_cookie_authentication().

4.17.5.2 int authentication_cookie_is_set = 0 `[static]`

If true, we've set authentication_cookie to a secret code and stored it to disk.

Referenced by init_cookie_authentication().

4.17.5.3 int bootstrap_percent = BOOTSTRAP_STATUS_UNDEF `[static]`

What percentage through the bootstrap process are we? We remember this so we can avoid sending redundant bootstrap status events, and so we can guess context for the bootstrap messages which are ambiguous. It starts at 'undef', but gets set to 'starting' while Tor initializes.

4.17.5.4 int bootstrap_problems = 0 `[static]`

How many problems have we had getting to the next bootstrapping phase? These include failure to establish a connection to a Tor relay, failures to finish the TLS handshake, failures to validate the consensus document, etc.

4.17.5.5 int disable_log_messages = 0 `[static]`

True iff we have disabled log messages from being sent to the controller

Referenced by control_event_logmsg(), disable_control_logging(), and enable_control_logging().

4.17.5.6 const getinfo_item_t getinfo_items[ ] `[static]`

Table mapping questions accepted by GETINFO to the functions that know how to answer them.

4.17.5.7 event_mask_t global_event_mask = 0 `[static]`

An event mask of all the events that any controller is interested in receiving.

Referenced by control_update_global_event_mask().

4.17.5.8 char last_sent_bootstrap_message[BOOTSTRAP_MSG_LEN] `[static]`

What was the last bootstrap phase message we sent? We keep track of this so we can respond to getinfo status/bootstrap-phase queries.

Referenced by control_event_bootstrap(), control_event_bootstrap_problem(), and getinfo_helper_events().

4.18 cpuworker.c File Reference

Implements a farm of 'CPU worker' processes to perform CPU-intensive tasks in another thread or process, to not interrupt the main thread.

`#include "or.h"`

Defines

- #define **MAX_CPUWORKERS** 16
- #define **MIN_CPUWORKERS** 1
- #define **TAG_LEN** 10
- #define **LEN_ONION_RESPONSE** (1+TAG_LEN+ONIONSKIN_REPLY_LEN+CPATH_KEY_MATERIAL_LEN)
- #define **CPUWORKER_BUSY_TIMEOUT** (60*60*12)

Functions

- static void **cpuworker_main** (void *data) ATTR_NORETURN
- static int **spawn_cpuworker** (void)
- static void **spawn_enough_cpuworkers** (void)
- static void **process_pending_task** (**connection_t** *cpuworker)
- void **cpu_init** (void)
- int **connection_cpu_finished_flushing** (**connection_t** *conn)
- static void **tag_pack** (char *tag, uint64_t conn_id, **circid_t** circ_id)
- static void **tag_unpack** (const char *tag, uint64_t *conn_id, **circid_t** *circ_id)
- void **cpuworkers_rotate** (void)
- int **connection_cpu_reached_eof** (**connection_t** *conn)
- int **connection_cpu_process_inbuf** (**connection_t** *conn)
- static void **cull_wedged_cpuworkers** (void)
- int **assign_onionskin_to_cpuworker** (**connection_t** *cpuworker, **or_circuit_t** *circ, char *onionskin)

Variables

- static int **num_cpuworkers** = 0
- static int **num_cpuworkers_busy** = 0
- static time_t **last_rotation_time** = 0

4.18.1 Detailed Description

Implements a farm of 'CPU worker' processes to perform CPU-intensive tasks in another thread or process, to not interrupt the main thread.

Right now, we only use this for processing onionskins.

4.18.2 Define Documentation

4.18.2.1 #define CPUWORKER_BUSY_TIMEOUT (60*60*12)

How long should we let a cpuworker stay busy before we give up on it and decide that we have a bug or infinite loop? This value is high because some servers with low memory/cpu sometimes spend an hour or more swapping, and Tor starves.

Referenced by cull_wedged_cpuworkers().

4.18.2.2 #define LEN_ONION_RESPONSE (1+TAG_LEN+ONIONSKIN_REPLY_LEN+CPATH_KEY_MATERIAL_LEN)

How many bytes are sent from the cpuworker back to tor?

Referenced by connection_cpu_process_inbuf(), and cpuworker_main().

4.18.2.3 #define MAX_CPUWORKERS 16

The maximum number of cpuworker processes we will keep around.

Referenced by spawn_enough_cpuworkers().

4.18.2.4 #define MIN_CPUWORKERS 1

The minimum number of cpuworker processes we will keep around.

Referenced by spawn_enough_cpuworkers().

4.18.2.5 #define TAG_LEN 10

The tag specifies which circuit this onionskin was from.

Referenced by assign_onionskin_to_cpuworker(), connection_cpu_process_inbuf(), and cpuworker_main().

4.18.3 Function Documentation

4.18.3.1 int assign_onionskin_to_cpuworker (connection_t * *cpuworker*, or_circuit_t * *circ*, char * *onionskin*)

Try to tell a cpuworker to perform the public key operations necessary to respond to **onionskin** for the circuit **circ**.

If **cpuworker** is defined, assert that he's idle, and use him. Else, look for an idle cpuworker and use him. If none idle, queue task onto the pending onion list and return. Return 0 if we successfully assign the task, or -1 on failure.

References or_connection_t::_base, CONN_TYPE_CPUWORKER, connection_get_by_type_state(), CPUWORKER_STATE_BUSY_ONION, CPUWORKER_STATE_IDLE, cull_wedged_cpuworkers(), connection_t::global_identifier, LD_OR, num_cpuworkers, num_cpuworkers_busy, onion_pending_add(), or_circuit_t::p_circ_id, or_circuit_t::p_conn, spawn_enough_cpuworkers(), connection_t::state, TAG_LEN, tag_pack(), connection_t::timestamp_lastwritten, tor_assert, and tor_free.

Referenced by command_process_create_cell(), and process_pending_task().

4.18.3.2 int connection_cpu_finished_flushing (connection_t ∗ *conn*)

Called when we're done sending a request to a cpuworker.

References CONN_TYPE_CPUWORKER, connection_stop_writing(), tor_assert, and connection_t::type.

Referenced by connection_finished_flushing().

4.18.3.3 int connection_cpu_process_inbuf (connection_t ∗ *conn*)

Called when we get data from a cpuworker. If the answer is not complete, wait for a complete answer. If the answer is complete, process it as appropriate.

References buf_datalen(), circuit_get_by_circid_orconn(), CIRCUIT_IS_ORIGIN, CONN_TYPE_CPUWORKER, CONN_TYPE_OR, connection_fetch_from_buf(), connection_get_by_global_id(), CPUWORKER_STATE_BUSY_ONION, CPUWORKER_STATE_IDLE, connection_t::inbuf, last_rotation_time, LD_OR, LEN_ONION_RESPONSE, connection_t::marked_for_close, num_cpuworkers, num_cpuworkers_busy, onionskin_answer(), process_pending_task(), spawn_enough_cpuworkers(), connection_t::state, TAG_LEN, tag_unpack(), connection_t::timestamp_created, TO_OR_CIRCUIT(), TO_OR_CONN(), tor_assert, and connection_t::type.

Referenced by connection_process_inbuf().

4.18.3.4 int connection_cpu_reached_eof (connection_t ∗ *conn*)

If the cpuworker closes the connection, mark it as closed and spawn a new one as needed.

References CPUWORKER_STATE_IDLE, LD_GENERAL, num_cpuworkers, num_cpuworkers_busy, spawn_enough_cpuworkers(), and connection_t::state.

Referenced by connection_reached_eof().

4.18.3.5 void cpu_init (void)

Initialize the cpuworker subsystem.

References cpuworkers_rotate().

Referenced by do_main_loop().

4.18.3.6 static void cpuworker_main (void ∗ *data*) `[static]`

Implement a cpuworker. 'data' is an fdarray as returned by socketpair. Read and writes from fdarray[1]. Reads requests, writes answers.

Request format: Task type [1 byte, always CPUWORKER_TASK_ONION] Opaque tag TAG_LEN Onionskin challenge ONIONSKIN_CHALLENGE_LEN Response format: Success/failure [1 byte, boolean.] Opaque tag TAG_LEN Onionskin challenge ONIONSKIN_REPLY_LEN Negotiated keys KEY_LEN∗2+DIGEST_LEN∗2

(Note: this _should_ be by addr/port, since we're concerned with specific connections, not with routers (where we'd use identity).)

References crypto_free_pk_env(), crypto_thread_cleanup(), dup_onion_keys(), handle_signals(), LD_BUG, LD_OR, LEN_ONION_RESPONSE, onion_skin_server_handshake(), read_all(), spawn_exit(), TAG_LEN, tor_assert, tor_close_socket(), tor_free, tor_free_all(), and write_all().

Referenced by spawn_cpuworker().

4.18.3.7 void cpuworkers_rotate (void)

Called when the onion key has changed and we need to spawn new cpuworkers. Close all currently idle cpuworkers, and mark the last rotation time as now.

References CONN_TYPE_CPUWORKER, connection_get_by_type_state(), CPUWORKER_STATE_IDLE, get_options(), last_rotation_time, num_cpuworkers, server_mode(), and spawn_enough_cpuworkers().

Referenced by cpu_init(), do_hup(), options_act(), and run_scheduled_events().

4.18.3.8 static void cull_wedged_cpuworkers (void) `[static]`

We have a bug that I can't find. Sometimes, very rarely, cpuworkers get stuck in the 'busy' state, even though the cpuworker process thinks of itself as idle. I don't know why. But here's a workaround to kill any cpuworker that's been busy for more than CPUWORKER_BUSY_TIMEOUT.

References CONN_TYPE_CPUWORKER, CPUWORKER_BUSY_TIMEOUT, CPUWORKER_STATE_BUSY_ONION, get_connection_array(), LD_BUG, num_cpuworkers, and num_cpuworkers_busy.

Referenced by assign_onionskin_to_cpuworker().

4.18.3.9 static void process_pending_task (connection_t * *cpuworker*) `[static]`

Take a pending task from the queue and assign it to 'cpuworker'.

References assign_onionskin_to_cpuworker(), LD_OR, onion_next_task(), and tor_assert.

Referenced by connection_cpu_process_inbuf().

4.18.3.10 static int spawn_cpuworker (void) `[static]`

Launch a new cpuworker. Return 0 if we're happy, -1 if we failed.

References connection_t::address, CONN_TYPE_CPUWORKER, connection_add(), connection_free(), connection_new(), connection_start_reading(), cpuworker_main(), CPUWORKER_STATE_IDLE, LD_NET, LD_OR, connection_t::s, set_socket_nonblocking(), spawn_func(), connection_t::state, tor_assert, tor_close_socket(), tor_free, and tor_socketpair().

Referenced by spawn_enough_cpuworkers().

4.18.3.11 static void spawn_enough_cpuworkers (void) `[static]`

If we have too few or too many active cpuworkers, try to spawn new ones or kill idle ones.

References get_options(), LD_GENERAL, MAX_CPUWORKERS, MIN_CPUWORKERS, num_cpuworkers, or_options_t::NumCpus, and spawn_cpuworker().

Referenced by assign_onionskin_to_cpuworker(), connection_cpu_process_inbuf(), connection_cpu_reached_eof(), and cpuworkers_rotate().

4.18.3.12 static void tag_pack (char * *tag*, uint64_t *conn_id*, circid_t *circ_id*) `[static]`

Pack global_id and circ_id; set *tag to the result. (See note on cpuworker_main for wire format.)

References set_uint16(), and set_uint64().

Referenced by assign_onionskin_to_cpuworker().

4.18.3.13 static void tag_unpack (const char * *tag*, uint64_t * *conn_id*, circid_t * *circ_id*) `[static]`

Unpack **tag** into addr, port, and circ_id.

References get_uint16(), and get_uint64().

Referenced by connection_cpu_process_inbuf().

4.18.4 Variable Documentation

4.18.4.1 time_t last_rotation_time = 0 `[static]`

We need to spawn new cpuworkers whenever we rotate the onion keys on platforms where execution contexts==processes. This variable stores the last time we got a key rotation event.

Referenced by connection_cpu_process_inbuf(), and cpuworkers_rotate().

4.18.4.2 int num_cpuworkers = 0 `[static]`

How many cpuworkers we have running right now.

Referenced by assign_onionskin_to_cpuworker(), connection_cpu_process_inbuf(), connection_cpu_reached_eof(), cpuworkers_rotate(), cull_wedged_cpuworkers(), and spawn_enough_cpuworkers().

4.18.4.3 int num_cpuworkers_busy = 0 `[static]`

How many of the running cpuworkers have an assigned task right now.

Referenced by assign_onionskin_to_cpuworker(), connection_cpu_process_inbuf(), connection_cpu_reached_eof(), and cull_wedged_cpuworkers().

4.19 crypto.c File Reference

Wrapper functions to present a consistent interface to public-key and symmetric cryptography operations from OpenSSL.

`#include "orconfig.h"`

`#include <openssl/err.h>`

`#include <openssl/rsa.h>`

`#include <openssl/pem.h>`

`#include <openssl/evp.h>`

`#include <openssl/engine.h>`

`#include <openssl/rand.h>`

`#include <openssl/opensslv.h>`

`#include <openssl/bn.h>`

`#include <openssl/dh.h>`

`#include <openssl/conf.h>`

`#include <openssl/hmac.h>`

`#include <unistd.h>`

`#include <fcntl.h>`

`#include <sys/fcntl.h>`

`#include "crypto.h"`

`#include "../common/log.h"`

`#include "aes.h"`

`#include "../common/util.h"`

`#include "container.h"`

`#include "compat.h"`

`#include "sha256.c"`

Data Structures

- struct **crypto_pk_env_t**
- struct **crypto_cipher_env_t**
- struct **crypto_dh_env_t**
- struct **crypto_digest_env_t**
- struct **CRYPTO_dynlock_value**

Defines

- #define **CRYPTO_PRIVATE**
- #define **SHA256_CTX** sha256_state
- #define **SHA256_Init** sha256_init
- #define **SHA256_Update** sha256_process

- #define **LTC_ARGCHK**(x) tor_assert(x)
- #define **SHA256_Final**(a, b) sha256_done(b,a)
- #define **PUBLIC_KEY_OK**(k) ((k) && (k) → key && (k) → key → n)
- #define **PRIVATE_KEY_OK**(k) ((k) && (k) → key && (k) → key → p)
- #define **DH_PRIVATE_KEY_BITS** 320
- #define **MIN**(a, b) ((a)<(b)?(a):(b))
- #define **ADD_ENTROPY** 32
- #define **HAVE_RAND_POLL** (OPENSSL_VERSION_NUMBER >= 0x0090600fl)
- #define **RAND_POLL_IS_SAFE**
- #define **X** 255
- #define **SP** 64
- #define **PAD** 65
- #define **EXPBIAS** 6

Functions

- static unsigned char ∗ **SHA256** (const unsigned char ∗m, size_t len, unsigned char ∗d)
- static int **setup_openssl_threading** (void)
- static int **tor_check_dh_key** (int severity, BIGNUM ∗bn)
- static INLINE int **crypto_get_rsa_padding_overhead** (int padding)
- static INLINE int **crypto_get_rsa_padding** (int padding)
- static void **crypto_log_errors** (int severity, const char ∗doing)
- static void **log_engine** (const char ∗fn, ENGINE ∗e)
- static ENGINE ∗ **try_load_engine** (const char ∗path, const char ∗engine)
- int **crypto_global_init** (int useAccel, const char ∗accelName, const char ∗accelDir)
- void **crypto_thread_cleanup** (void)
- int **crypto_global_cleanup** (void)
- **crypto_pk_env_t** ∗ **_crypto_new_pk_env_rsa** (RSA ∗rsa)
- **crypto_pk_env_t** ∗ **_crypto_new_pk_env_evp_pkey** (EVP_PKEY ∗pkey)
- RSA ∗ **_crypto_pk_env_get_rsa** (**crypto_pk_env_t** ∗env)
- EVP_PKEY ∗ **_crypto_pk_env_get_evp_pkey** (**crypto_pk_env_t** ∗env, int private)
- DH ∗ **_crypto_dh_env_get_dh** (**crypto_dh_env_t** ∗dh)
- **crypto_pk_env_t** ∗ **crypto_new_pk_env** (void)
- void **crypto_free_pk_env** (**crypto_pk_env_t** ∗env)
- **crypto_cipher_env_t** ∗ **crypto_create_init_cipher** (const char ∗key, int encrypt_mode)
- **crypto_cipher_env_t** ∗ **crypto_new_cipher_env** (void)
- void **crypto_free_cipher_env** (**crypto_cipher_env_t** ∗env)
- int **crypto_pk_generate_key_with_bits** (**crypto_pk_env_t** ∗env, int bits)
- int **crypto_pk_read_private_key_from_string** (**crypto_pk_env_t** ∗env, const char ∗s)
- int **crypto_pk_read_private_key_from_filename** (**crypto_pk_env_t** ∗env, const char ∗keyfile)
- static int **crypto_pk_write_key_to_string_impl** (**crypto_pk_env_t** ∗env, char ∗∗dest, size_t ∗len, int is_public)
- int **crypto_pk_write_public_key_to_string** (**crypto_pk_env_t** ∗env, char ∗∗dest, size_t ∗len)
- int **crypto_pk_write_private_key_to_string** (**crypto_pk_env_t** ∗env, char ∗∗dest, size_t ∗len)

- int **crypto_pk_read_public_key_from_string** (**crypto_pk_env_t** *env, const char *src, size_t len)
- int **crypto_pk_write_private_key_to_filename** (**crypto_pk_env_t** *env, const char *fname)
- int **crypto_pk_check_key** (**crypto_pk_env_t** *env)
- int **crypto_pk_key_is_private** (const **crypto_pk_env_t** *key)
- int **crypto_pk_cmp_keys** (**crypto_pk_env_t** *a, **crypto_pk_env_t** *b)
- size_t **crypto_pk_keysize** (**crypto_pk_env_t** *env)
- **crypto_pk_env_t** * **crypto_pk_dup_key** (**crypto_pk_env_t** *env)
- **crypto_pk_env_t** * **crypto_pk_copy_full** (**crypto_pk_env_t** *env)
- int **crypto_pk_public_encrypt** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen, int padding)
- int **crypto_pk_private_decrypt** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen, int padding, int warnOnFailure)
- int **crypto_pk_public_checksig** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_pk_public_checksig_digest** (**crypto_pk_env_t** *env, const char *data, size_t datalen, const char *sig, size_t siglen)
- int **crypto_pk_private_sign** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_pk_private_sign_digest** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_pk_public_hybrid_encrypt** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen, int padding, int force)
- int **crypto_pk_private_hybrid_decrypt** (**crypto_pk_env_t** *env, char *to, const char *from, size_t fromlen, int padding, int warnOnFailure)
- int **crypto_pk_asn1_encode** (**crypto_pk_env_t** *pk, char *dest, size_t dest_len)
- **crypto_pk_env_t** * **crypto_pk_asn1_decode** (const char *str, size_t len)
- int **crypto_pk_get_digest** (**crypto_pk_env_t** *pk, char *digest_out)
- void **add_spaces_to_fp** (char *out, size_t outlen, const char *in)
- int **crypto_pk_get_fingerprint** (**crypto_pk_env_t** *pk, char *fp_out, int add_space)
- int **crypto_pk_check_fingerprint_syntax** (const char *s)
- int **crypto_cipher_generate_key** (**crypto_cipher_env_t** *env)
- void **crypto_cipher_set_key** (**crypto_cipher_env_t** *env, const char *key)
- void **crypto_cipher_generate_iv** (char *iv_out)
- int **crypto_cipher_set_iv** (**crypto_cipher_env_t** *env, const char *iv)
- const char * **crypto_cipher_get_key** (**crypto_cipher_env_t** *env)
- int **crypto_cipher_encrypt_init_cipher** (**crypto_cipher_env_t** *env)
- int **crypto_cipher_decrypt_init_cipher** (**crypto_cipher_env_t** *env)
- int **crypto_cipher_encrypt** (**crypto_cipher_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_cipher_decrypt** (**crypto_cipher_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_cipher_crypt_inplace** (**crypto_cipher_env_t** *env, char *buf, size_t len)
- int **crypto_cipher_encrypt_with_iv** (**crypto_cipher_env_t** *cipher, char *to, size_t tolen, const char *from, size_t fromlen)
- int **crypto_cipher_decrypt_with_iv** (**crypto_cipher_env_t** *cipher, char *to, size_t tolen, const char *from, size_t fromlen)
- int **crypto_digest** (char *digest, const char *m, size_t len)

- int **crypto_digest256** (char *digest, const char *m, size_t len, digest_algorithm_t algorithm)
- int **crypto_digest_all** (**digests_t** *ds_out, const char *m, size_t len)
- const char * **crypto_digest_algorithm_get_name** (digest_algorithm_t alg)
- int **crypto_digest_algorithm_parse_name** (const char *name)
- **crypto_digest_env_t** * **crypto_new_digest_env** (void)
- **crypto_digest_env_t** * **crypto_new_digest256_env** (digest_algorithm_t algorithm)
- void **crypto_free_digest_env** (**crypto_digest_env_t** *digest)
- void **crypto_digest_add_bytes** (**crypto_digest_env_t** *digest, const char *data, size_t len)
- void **crypto_digest_get_digest** (**crypto_digest_env_t** *digest, char *out, size_t out_len)
- **crypto_digest_env_t** * **crypto_digest_dup** (const **crypto_digest_env_t** *digest)
- void **crypto_digest_assign** (**crypto_digest_env_t** *into, const **crypto_digest_env_t** *from)
- void **crypto_hmac_sha1** (char *hmac_out, const char *key, size_t key_len, const char *msg, size_t msg_len)
- static void **init_dh_param** (void)
- **crypto_dh_env_t** * **crypto_dh_new** (void)
- int **crypto_dh_get_bytes** (**crypto_dh_env_t** *dh)
- int **crypto_dh_generate_public** (**crypto_dh_env_t** *dh)
- int **crypto_dh_get_public** (**crypto_dh_env_t** *dh, char *pubkey, size_t pubkey_len)
- ssize_t **crypto_dh_compute_secret** (int severity, **crypto_dh_env_t** *dh, const char *pubkey, size_t pubkey_len, char *secret_out, size_t secret_bytes_out)
- int **crypto_expand_key_material** (const char *key_in, size_t key_in_len, char *key_out, size_t key_out_len)
- void **crypto_dh_free** (**crypto_dh_env_t** *dh)
- int **crypto_seed_rng** (int startup)
- int **crypto_rand** (char *to, size_t n)
- int **crypto_rand_int** (unsigned int max)
- uint64_t **crypto_rand_uint64** (uint64_t max)
- char * **crypto_random_hostname** (int min_rand_len, int max_rand_len, const char *prefix, const char *suffix)
- void * **smartlist_choose** (const **smartlist_t** *sl)
- void **smartlist_shuffle** (**smartlist_t** *sl)
- int **base64_encode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **base64_decode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **digest_to_base64** (char *d64, const char *digest)
- int **digest_from_base64** (char *digest, const char *d64)
- int **digest256_to_base64** (char *d64, const char *digest)
- int **digest256_from_base64** (char *digest, const char *d64)
- void **base32_encode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **base32_decode** (char *dest, size_t destlen, const char *src, size_t srclen)
- void **secret_to_key** (char *key_out, size_t key_out_len, const char *secret, size_t secret_len, const char *s2k_specifier)
- static void **_openssl_locking_cb** (int mode, int n, const char *file, int line)
- static struct **CRYPTO_dynlock_value** * **_openssl_dynlock_create_cb** (const char *file, int line)

- static void **_openssl_dynlock_lock_cb** (int mode, struct **CRYPTO_dynlock_value** ∗v, const char ∗file, int line)
- static void **_openssl_dynlock_destroy_cb** (struct **CRYPTO_dynlock_value** ∗v, const char ∗file, int line)

Variables

- static **tor_mutex_t** ∗∗ **_openssl_mutexes** = NULL
- static int **_n_openssl_mutexes** = 0
- static int **_crypto_global_initialized** = 0
- static BIGNUM ∗ **dh_param_p** = NULL
- static BIGNUM ∗ **dh_param_g** = NULL
- static const uint8_t **base64_decode_table** [256]

4.19.1 Detailed Description

Wrapper functions to present a consistent interface to public-key and symmetric cryptography operations from OpenSSL.

4.19.2 Define Documentation

4.19.2.1 #define PRIVATE_KEY_OK(k) ((k) && (k) → key && (k) → key → p)

Macro: is k a valid RSA private key?

Referenced by crypto_pk_copy_full(), crypto_pk_key_is_private(), and crypto_pk_write_private_key_to_filename().

4.19.2.2 #define PUBLIC_KEY_OK(k) ((k) && (k) → key && (k) → key → n)

Macro: is k a valid RSA public or private key?

Referenced by crypto_pk_cmp_keys().

4.19.2.3 #define RAND_POLL_IS_SAFE

Value:

```
((OPENSSL_VERSION_NUMBER >= 0x009070afl &&    \
    OPENSSL_VERSION_NUMBER <= 0x00907fffl) ||   \
   (OPENSSL_VERSION_NUMBER >= 0x0090803fl))
```

4.19.3 Function Documentation

4.19.3.1 DH∗ _crypto_dh_env_get_dh (crypto_dh_env_t ∗ *dh*)

Used by **tortls.c** (p. 1219): Get the DH∗ from a **crypto_dh_env_t** (p. 71).

References crypto_dh_env_t::dh.

Referenced by tor_tls_context_new().

4.19.3.2 crypto_pk_env_t∗ _crypto_new_pk_env_evp_pkey (EVP_PKEY ∗ *pkey*)

used by **tortls.c** (p. 1219): wrap the RSA from an evp_pkey in a **crypto_pk_env_t** (p. 74). returns NULL if this isn't an RSA key.

References _crypto_new_pk_env_rsa().

4.19.3.3 crypto_pk_env_t∗ _crypto_new_pk_env_rsa (RSA ∗ *rsa*)

used by **tortls.c** (p. 1219): wrap an RSA∗ in a **crypto_pk_env_t** (p. 74).

References crypto_pk_env_t::key, crypto_pk_env_t::refs, and tor_assert.

Referenced by _crypto_new_pk_env_evp_pkey(), crypto_new_pk_env(), crypto_pk_asn1_decode(), crypto_pk_copy_full(), and tor_tls_verify().

4.19.3.4 EVP_PKEY∗ _crypto_pk_env_get_evp_pkey (crypto_pk_env_t ∗ *env*, int *private*)

used by **tortls.c** (p. 1219): get an equivalent EVP_PKEY∗ for a **crypto_pk_env_t** (p. 74). Iff private is set, include the private-key portion of the key.

References crypto_pk_env_t::key, and tor_assert.

Referenced by tor_tls_context_new(), and tor_tls_create_certificate().

4.19.3.5 RSA∗ _crypto_pk_env_get_rsa (crypto_pk_env_t ∗ *env*)

Helper, used by tor-checkkey.c and tor-gencert.c. Return the RSA from a **crypto_pk_env_t** (p. 74).

References crypto_pk_env_t::key.

4.19.3.6 static struct CRYPTO_dynlock_value∗ _openssl_dynlock_create_cb (const char ∗ *file*, int *line*) `[static, read]`

OpenSSL callback function to allocate a lock: see CRYPTO_set_dynlock_∗ documentation in OpenSSL's docs for more info.

References CRYPTO_dynlock_value::lock, and tor_mutex_new().

Referenced by setup_openssl_threading().

4.19.3.7 static void _openssl_dynlock_destroy_cb (struct CRYPTO_dynlock_value ∗ *v*, const char ∗ *file*, int *line*) `[static]`

OpenSSL callback function to free a lock: see CRYPTO_set_dynlock_∗ documentation in OpenSSL's docs for more info.

References CRYPTO_dynlock_value::lock, tor_free, and tor_mutex_free().

Referenced by setup_openssl_threading().

4.19.3.8 static void _openssl_dynlock_lock_cb (int *mode*, struct CRYPTO_dynlock_value * *v*, const char * *file*, int *line*) `[static]`

OpenSSL callback function to acquire or release a lock: see CRYPTO_set_dynlock_* documentation in OpenSSL's docs for more info.

References CRYPTO_dynlock_value::lock, tor_mutex_acquire(), and tor_mutex_release().

Referenced by setup_openssl_threading().

4.19.3.9 static void _openssl_locking_cb (int *mode*, int *n*, const char * *file*, int *line*) `[static]`

Helper: OpenSSL uses this callback to manipulate mutexes.

References tor_mutex_acquire(), and tor_mutex_release().

Referenced by setup_openssl_threading().

4.19.3.10 void add_spaces_to_fp (char * *out*, size_t *outlen*, const char * *in*)

Copy **in** to the **outlen**-byte buffer **out**, adding spaces every four spaces.

References tor_assert.

Referenced by crypto_pk_get_fingerprint().

4.19.3.11 int base32_decode (char * *dest*, size_t *destlen*, const char * *src*, size_t *srclen*)

Implements base32 decoding as in rfc3548. Limitation: Requires that srclen*5 is a multiple of 8. Returns 0 if successful, -1 otherwise.

References LD_BUG, tor_assert, and tor_free.

Referenced by rend_cache_lookup_v2_desc_as_dir(), rend_compute_v2_desc_id(), rend_parse_introduction_points(), and rend_parse_v2_service_descriptor().

4.19.3.12 void base32_encode (char * *dest*, size_t *destlen*, const char * *src*, size_t *srclen*)

Implements base32 encoding as in rfc3548. Limitation: Requires that srclen*8 is a multiple of 5.

References BASE32_CHARS, and tor_assert.

Referenced by addressmap_get_virtual_address(), crypto_random_hostname(), directory_get_from_hs_dir(), directory_post_to_hs_dir(), lookup_last_hid_serv_request(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store_v2_desc_as_dir(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_get_service_id(), rend_mid_establish_intro(), rend_mid_introduce(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

4.19.3.13 int base64_decode (char ∗ *dest*, size_t *destlen*, const char ∗ *src*, size_t *srclen*)

Base-64 decode **srclen** bytes of data from **src**. Write the result into **dest**, if it will fit within **destlen** bytes. Return the number of bytes written on success; -1 if destlen is too short, or other failure.

NOTE 1: destlen is checked conservatively, as though srclen contained no spaces or padding.

NOTE 2: This implementation does not check for the correct number of padding "=" characters at the end of the string, and does not check for internal padding characters.

References tor_assert.

Referenced by decode_hashed_passwords(), digest256_from_base64(), digest_from_base64(), dir_split_resource_into_fingerprints(), get_next_token(), rend_parse_client_keys(), and rend_parse_service_authorization().

4.19.3.14 int base64_encode (char ∗ *dest*, size_t *destlen*, const char ∗ *src*, size_t *srclen*)

Base-64 encode **srclen** bytes of data from **src**. Write the result into **dest**, if it will fit within **destlen** bytes. Return the number of bytes written on success; -1 if destlen is too short, or other failure.

References tor_assert.

Referenced by alloc_http_authenticator(), digest256_to_base64(), digest_to_base64(), directory_get_from_hs_dir(), networkstatus_format_signatures(), rend_encode_v2_descriptors(), rend_service_load_keys(), and router_append_dirobj_signature().

4.19.3.15 int crypto_cipher_crypt_inplace (crypto_cipher_env_t ∗ *env*, char ∗ *buf*, size_t *len*)

Encrypt **len** bytes on **from** using the cipher in **env**; on success, return 0. On failure, return -1.

References aes_crypt_inplace(), and crypto_cipher_env_t::cipher.

Referenced by relay_crypt_one_payload().

4.19.3.16 int crypto_cipher_decrypt (crypto_cipher_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Decrypt **fromlen** bytes from **from** using the cipher **env**; on success, store the result to **to** and return 0. On failure, return -1.

References aes_crypt(), crypto_cipher_env_t::cipher, and tor_assert.

Referenced by crypto_pk_private_hybrid_decrypt(), and rend_decrypt_introduction_points().

4.19.3.17 int crypto_cipher_decrypt_init_cipher (crypto_cipher_env_t ∗ *env*)

Initialize the cipher in **env** for decryption. Return 0 on success, -1 on failure.

References aes_set_key(), crypto_cipher_env_t::cipher, CIPHER_KEY_LEN, crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_create_init_cipher().

4.19.3.18 int crypto_cipher_decrypt_with_iv (crypto_cipher_env_t * *cipher*, char * *to*, size_t *tolen*, const char * *from*, size_t *fromlen*)

Decrypt **fromlen** bytes (at least 1+CIPHER_IV_LEN) from **from** with the key in **cipher** to the buffer in **to** of length **tolen**. **tolen** must be at least **fromlen** minus CIPHER_IV_LEN bytes for the initialization vector. On success, return the number of bytes written, on failure, return -1.

This function adjusts the current position of the counter in **cipher** to immediately after the decrypted data.

References CIPHER_IV_LEN, crypto_cipher_encrypt(), crypto_cipher_set_iv(), and tor_assert.

Referenced by rend_decrypt_introduction_points().

4.19.3.19 int crypto_cipher_encrypt (crypto_cipher_env_t * *env*, char * *to*, const char * *from*, size_t *fromlen*)

Encrypt **fromlen** bytes from **from** using the cipher **env**; on success, store the result to **to** and return 0. On failure, return -1.

References aes_crypt(), crypto_cipher_env_t::cipher, and tor_assert.

Referenced by crypto_cipher_decrypt_with_iv(), crypto_cipher_encrypt_with_iv(), crypto_pk_public_hybrid_encrypt(), and rend_encrypt_v2_intro_points_basic().

4.19.3.20 int crypto_cipher_encrypt_init_cipher (crypto_cipher_env_t * *env*)

Initialize the cipher in **env** for encryption. Return 0 on success, -1 on failure.

References aes_set_key(), crypto_cipher_env_t::cipher, CIPHER_KEY_LEN, crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_create_init_cipher(), and crypto_pk_public_hybrid_encrypt().

4.19.3.21 int crypto_cipher_encrypt_with_iv (crypto_cipher_env_t * *cipher*, char * *to*, size_t *tolen*, const char * *from*, size_t *fromlen*)

Encrypt **fromlen** bytes (at least 1) from **from** with the key in **cipher** to the buffer in **to** of length **tolen**. **tolen** must be at least **fromlen** plus CIPHER_IV_LEN bytes for the initialization vector. On success, return the number of bytes written, on failure, return -1.

This function adjusts the current position of the counter in **cipher** to immediately after the encrypted data.

References CIPHER_IV_LEN, crypto_cipher_encrypt(), crypto_cipher_generate_iv(), crypto_cipher_set_iv(), and tor_assert.

Referenced by rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.19.3.22 void crypto_cipher_generate_iv (char ∗ *iv_out*)

Generate an initialization vector for our AES-CTR cipher; store it in the first CIPHER_IV_LEN bytes of **iv_out**.

References CIPHER_IV_LEN, and crypto_rand().

Referenced by crypto_cipher_encrypt_with_iv().

4.19.3.23 int crypto_cipher_generate_key (crypto_cipher_env_t ∗ *env*)

Generate a new random key for the symmetric cipher in **env**. Return 0 on success, -1 on failure. Does not initialize the cipher.

References CIPHER_KEY_LEN, crypto_rand(), crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_pk_public_hybrid_encrypt().

4.19.3.24 const char∗ crypto_cipher_get_key (crypto_cipher_env_t ∗ *env*)

Return a pointer to the key set for the cipher in **env**.

References crypto_cipher_env_t::key.

4.19.3.25 int crypto_cipher_set_iv (crypto_cipher_env_t ∗ *env*, const char ∗ *iv*)

Adjust the counter of **env** to point to the first byte of the block corresponding to the encryption of the CIPHER_IV_LEN bytes at **iv**.

References aes_set_iv(), crypto_cipher_env_t::cipher, and tor_assert.

Referenced by crypto_cipher_decrypt_with_iv(), and crypto_cipher_encrypt_with_iv().

4.19.3.26 void crypto_cipher_set_key (crypto_cipher_env_t ∗ *env*, const char ∗ *key*)

Set the symmetric key for the cipher in **env** to the first CIPHER_KEY_LEN bytes of **key**. Does not initialize the cipher.

References CIPHER_KEY_LEN, crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_create_init_cipher().

4.19.3.27 crypto_cipher_env_t∗ crypto_create_init_cipher (const char ∗ *key*, int *encrypt_mode*)

Create a new symmetric cipher for a given key and encryption flag (1=encrypt, 0=decrypt). Return the crypto object on success; NULL on failure.

References crypto_cipher_decrypt_init_cipher(), crypto_cipher_encrypt_init_cipher(), crypto_cipher_set_key(), crypto_free_cipher_env(), crypto_new_cipher_env(), and LD_CRYPTO.

Referenced by circuit_init_cpath_crypto(), crypto_pk_private_hybrid_decrypt(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.19.3.28 ssize_t crypto_dh_compute_secret (int *severity*, crypto_dh_env_t * *dh*, const char * *pubkey*, size_t *pubkey_len*, char * *secret_out*, size_t *secret_bytes_out*)

Given a DH key exchange object, and our peer's value of g^y (as a **pubkey_len**-byte value in **pubkey**) generate **secret_bytes_out** bytes of shared key material and write them to **secret_out**. Return the number of bytes generated on success, or -1 on failure.

(We generate key material by computing SHA1(g^xy || "\x00") || SHA1(g^xy || "\x01") || ... where || is concatenation.)

References crypto_dh_get_bytes(), crypto_expand_key_material(), crypto_log_errors(), crypto_dh_env_t::dh, DIGEST_LEN, LD_CRYPTO, tor_assert, tor_check_dh_key(), and tor_free.

Referenced by onion_skin_client_handshake(), onion_skin_server_handshake(), rend_client_receive_rendezvous(), and rend_service_introduce().

4.19.3.29 void crypto_dh_free (crypto_dh_env_t * *dh*)

Free a DH key exchange object.

References crypto_dh_env_t::dh, tor_assert, and tor_free.

Referenced by circuit_finish_handshake(), circuit_free_cpath_node(), onion_skin_create(), onion_skin_server_handshake(), rend_client_receive_rendezvous(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and tor_tls_context_new().

4.19.3.30 int crypto_dh_generate_public (crypto_dh_env_t * *dh*)

Generate <x,g^x> for our part of the key exchange. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_dh_env_t::dh, LD_CRYPTO, and tor_check_dh_key().

Referenced by crypto_dh_get_public(), and rend_service_introduce().

4.19.3.31 int crypto_dh_get_bytes (crypto_dh_env_t * *dh*)

Return the length of the DH key in **dh**, in bytes.

References crypto_dh_env_t::dh, and tor_assert.

Referenced by crypto_dh_compute_secret(), onion_skin_client_handshake(), and onion_skin_create().

4.19.3.32 int crypto_dh_get_public (crypto_dh_env_t * *dh*, char * *pubkey*, size_t *pubkey_len*)

Generate g^x as necessary, and write the g^x for the key exchange as a **pubkey_len**-byte value into **pubkey**. Return 0 on success, -1 on failure. **pubkey_len** must be >= DH_BYTES.

References crypto_dh_generate_public(), crypto_dh_env_t::dh, LD_CRYPTO, and tor_assert.

Referenced by onion_skin_create(), onion_skin_server_handshake(), and rend_service_rendezvous_has_opened().

4.19.3.33 crypto_dh_env_t* crypto_dh_new (void)

Allocate and return a new DH object for a key exchange.

References crypto_log_errors(), crypto_dh_env_t::dh, init_dh_param(), and tor_free.

Referenced by onion_skin_create(), onion_skin_server_handshake(), rend_service_introduce(), and tor_tls_context_new().

4.19.3.34 int crypto_digest (char * *digest*, const char * *m*, size_t *len*)

Compute the SHA1 digest of **len** bytes in data stored in **m**. Write the DIGEST_LEN byte result into **digest**. Return 0 on success, -1 on failure.

References tor_assert.

Referenced by crypto_digest_all(), crypto_expand_key_material(), crypto_pk_get_digest(), crypto_pk_private_sign_digest(), crypto_pk_public_checksig_digest(), rend_encode_v2_descriptors(), rend_mid_establish_intro(), rend_service_intro_has_opened(), and router_get_hash_impl().

4.19.3.35 void crypto_digest_add_bytes (crypto_digest_env_t * *digest*, const char * *data*, size_t *len*)

Add **len** bytes from **data** to the digest object.

References crypto_digest_env_t::algorithm, crypto_digest_env_t::d, crypto_digest_env_t::sha1, crypto_digest_env_t::sha2, tor_assert, and tor_fragile_assert.

Referenced by accounting_set_wakeup_time(), circuit_init_cpath_crypto(), get_secret_id_part_bytes(), hash_list_members(), relay_digest_matches(), relay_set_digest(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.19.3.36 const char* crypto_digest_algorithm_get_name (digest_algorithm_t *alg*)

Return the name of an algorithm, as used in directory documents.

References tor_fragile_assert.

Referenced by networkstatus_add_detached_signatures(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), and vote_routerstatus_find_microdesc_hash().

4.19.3.37 int crypto_digest_algorithm_parse_name (const char * *name*)

Given the name of a digest algorithm, return its integer value, or -1 if the name is not recognized.

Referenced by networkstatus_parse_detached_signatures(), and networkstatus_parse_vote_from_string().

4.19.3.38 int crypto_digest_all (digests_t * *ds_out*, const char * *m*, size_t *len*)

Set the **digests_t** (p. 77) in **ds_out** to contain every digest on the **len** bytes in **m** that we know how to compute. Return 0 on success, -1 on failure.

References crypto_digest(), digests_t::d, and tor_assert.

Referenced by router_get_hashes_impl().

4.19.3.39 void crypto_digest_assign (crypto_digest_env_t * *into*, const crypto_digest_env_t * *from*)

Replace the state of the digest object **into** with the state of the digest object **from**.

References tor_assert.

Referenced by relay_digest_matches().

4.19.3.40 crypto_digest_env_t* crypto_digest_dup (const crypto_digest_env_t * *digest*)

Allocate and return a new digest object with the same state as **digest**

References tor_assert.

Referenced by relay_digest_matches().

4.19.3.41 void crypto_digest_get_digest (crypto_digest_env_t * *digest*, char * *out*, size_t *out_len*)

Compute the hash of the data that has been passed to the digest object; write the first out_len bytes of the result to **out**. **out_len** must be <= DIGEST256_LEN.

References crypto_digest_env_t::algorithm, crypto_digest_env_t::d, DIGEST256_LEN, DIGEST_LEN, crypto_digest_env_t::sha1, crypto_digest_env_t::sha2, tor_assert, and tor_fragile_assert.

Referenced by accounting_set_wakeup_time(), get_secret_id_part_bytes(), hash_list_members(), relay_digest_matches(), relay_set_digest(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.19.3.42 int crypto_expand_key_material (const char * *key_in*, size_t *key_in_len*, char * *key_out*, size_t *key_out_len*)

Given **key_in_len** bytes of negotiated randomness in **key_in** ("K"), expand it into **key_out_len** bytes of negotiated key material in **key_out** by taking the first **key_out_len** bytes of H(K | [00]) | H(K | [01]) | ....

Return 0 on success, -1 on failure.

References crypto_digest(), DIGEST_LEN, tor_assert, and tor_free.

Referenced by crypto_dh_compute_secret(), fast_client_handshake(), and fast_server_handshake().

4.19.3.43 void crypto_free_cipher_env (crypto_cipher_env_t * *env*)

Free a symmetric cipher.

References aes_free_cipher(), crypto_cipher_env_t::cipher, tor_assert, and tor_free.

Referenced by circuit_free(), circuit_free_cpath_node(), crypto_create_init_cipher(), crypto_pk_private_hybrid_decrypt(), crypto_pk_public_hybrid_encrypt(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.19.3.44 void crypto_free_digest_env (crypto_digest_env_t * *digest*)

Deallocate a digest object.

References tor_free.

Referenced by accounting_set_wakeup_time(), circuit_free(), circuit_free_cpath_node(), get_secret_id_part_bytes(), hash_list_members(), relay_digest_matches(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.19.3.45 void crypto_free_pk_env (crypto_pk_env_t * *env*)

Release a reference to an asymmetric key; when all the references are released, free the key.

References crypto_pk_env_t::key, crypto_pk_env_t::refs, and tor_free.

Referenced by authority_cert_free(), circuit_free(), connection_or_check_valid_tls_handshake(), cpuworker_main(), extend_info_free(), init_key_from_file(), init_keys(), load_authority_keyset(), microdesc_free(), networkstatus_v2_free(), rend_authorized_client_free(), rend_intro_point_free(), rend_mid_establish_intro(), rend_service_descriptor_free(), rend_service_free(), rend_service_load_keys(), rotate_onion_key(), router_free_all(), router_parse_directory(), router_parse_runningrouters(), routerinfo_free(), set_identity_key(), set_onion_key(), token_clear(), tor_tls_context_decref(), and tor_tls_context_new().

4.19.3.46 static INLINE int crypto_get_rsa_padding (int *padding*) `[static]`

Given a padding method **padding**, return the correct OpenSSL constant.

References PK_NO_PADDING, PK_PKCS1_OAEP_PADDING, PK_PKCS1_PADDING, and tor_assert.

Referenced by crypto_pk_private_decrypt(), crypto_pk_public_encrypt(), and crypto_pk_public_hybrid_encrypt().

4.19.3.47 static INLINE int crypto_get_rsa_padding_overhead (int *padding*) `[static]`

Return the number of bytes added by padding method **padding**.

References tor_assert.

Referenced by crypto_pk_public_hybrid_encrypt().

4.19.3.48 int crypto_global_cleanup (void)

Uninitialize the crypto library. Return 0 on success, -1 on failure.

References _n_openssl_mutexes, tor_free, and tor_mutex_free().

Referenced by tor_cleanup().

4.19.3.49 int crypto_global_init (int *useAccel*, const char ∗ *accelName*, const char ∗ *accelDir*)

Initialize the crypto library. Return 0 on success, -1 on failure.

References _crypto_global_initialized, crypto_seed_rng(), LD_CRYPTO, log_engine(), setup_openssl_threading(), and try_load_engine().

Referenced by init_keys(), and tor_init().

4.19.3.50 void crypto_hmac_sha1 (char ∗ *hmac_out*, const char ∗ *key*, size_t *key_len*, const char ∗ *msg*, size_t *msg_len*)

Compute the HMAC-SHA-1 of the **msg_len** bytes in **msg**, using the **key** of length **key_len**. Store the DIGEST_LEN-byte result in **hmac_out**.

References tor_assert.

4.19.3.51 static void crypto_log_errors (int *severity*, const char ∗ *doing*) `[static]`

Log all pending crypto errors at level **severity**. Use **doing** to describe our current activities.

References LD_CRYPTO.

Referenced by crypto_dh_compute_secret(), crypto_dh_generate_public(), crypto_dh_new(), crypto_pk_asn1_decode(), crypto_pk_asn1_encode(), crypto_pk_check_key(), crypto_pk_generate_key_with_bits(), crypto_pk_get_digest(), crypto_pk_private_decrypt(), crypto_pk_private_sign(), crypto_pk_public_checksig(), crypto_pk_public_encrypt(), crypto_pk_read_private_key_from_string(), crypto_pk_read_public_key_from_string(), crypto_pk_write_key_to_string_impl(), crypto_pk_write_private_key_to_filename(), and crypto_rand().

4.19.3.52 crypto_cipher_env_t∗ crypto_new_cipher_env (void)

Allocate and return a new symmetric cipher.

References aes_new_cipher(), and crypto_cipher_env_t::cipher.

Referenced by crypto_create_init_cipher(), and crypto_pk_public_hybrid_encrypt().

4.19.3.53 crypto_digest_env_t∗ crypto_new_digest_env (void)

Allocate and return a new digest object.

References crypto_digest_env_t::algorithm, crypto_digest_env_t::d, and crypto_digest_env_t::sha1.

Referenced by accounting_set_wakeup_time(), circuit_init_cpath_crypto(), get_secret_id_part_bytes(), hash_list_members(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.19.3.54 crypto_pk_env_t∗ crypto_new_pk_env (void)

Allocate and return storage for a public key. The key itself will not yet be set.

References _crypto_new_pk_env_rsa().

Referenced by get_next_token(), init_key_from_file(), init_keys(), rend_service_load_keys(), rotate_onion_key(), and tor_tls_context_new().

4.19.3.55 crypto_pk_env_t* crypto_pk_asn1_decode (const char * *str*, size_t *len*)

Decode an ASN.1-encoded public key from **str**; return the result on success and NULL on failure.

References _crypto_new_pk_env_rsa(), crypto_log_errors(), and tor_free.

Referenced by rend_mid_establish_intro(), rend_parse_service_descriptor(), and rend_service_introduce().

4.19.3.56 int crypto_pk_asn1_encode (crypto_pk_env_t * *pk*, char * *dest*, size_t *dest_len*)

ASN.1-encode the public portion of **pk** into **dest**. Return -1 on error, or the number of characters used on success.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_free.

Referenced by rend_service_intro_has_opened().

4.19.3.57 int crypto_pk_check_fingerprint_syntax (const char * *s*)

Return true iff **s** is in the correct format for a fingerprint.

References FINGERPRINT_LEN.

4.19.3.58 int crypto_pk_check_key (crypto_pk_env_t * *env*)

Return true iff **env** has a valid key.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_read_private_key_from_filename(), init_key_from_file(), and rend_service_load_keys().

4.19.3.59 int crypto_pk_cmp_keys (crypto_pk_env_t * *a*, crypto_pk_env_t * *b*)

Compare the public-key components of a and b. Return -1 if a<b, 0 if a==b, and 1 if a>b.

References crypto_pk_env_t::key, PUBLIC_KEY_OK, and tor_assert.

Referenced by connection_or_check_valid_tls_handshake(), load_authority_keyset(), router_differences_are_cosmetic(), and router_dump_router_to_string().

4.19.3.60 crypto_pk_env_t* crypto_pk_copy_full (crypto_pk_env_t * *env*)

Make a real honest-to-goodness copy of **env**, and return it.

References _crypto_new_pk_env_rsa(), crypto_pk_env_t::key, PRIVATE_KEY_OK, and tor_assert.

Referenced by dup_onion_keys().

4.19.3.61 crypto_pk_env_t∗ crypto_pk_dup_key (crypto_pk_env_t ∗ *env*)

Increase the reference count of **env**, and return it.

References crypto_pk_env_t::key, crypto_pk_env_t::refs, and tor_assert.

Referenced by authority_cert_dup(), extend_info_alloc(), extend_info_dup(), rend_service_launch_establish_intro(), rend_service_load_keys(), rend_service_update_descriptor(), router_rebuild_descriptor(), and tor_tls_context_new().

4.19.3.62 int crypto_pk_generate_key_with_bits (crypto_pk_env_t ∗ *env*, int *bits*)

Generate a **bits**-bit new public/private keypair in **env**. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

4.19.3.63 int crypto_pk_get_digest (crypto_pk_env_t ∗ *pk*, char ∗ *digest_out*)

Given a private or public key **pk**, put a SHA1 hash of the public key into **digest_out** (must have DIGEST_LEN bytes of space). Return 0 on success, -1 on failure.

References crypto_digest(), crypto_log_errors(), crypto_pk_env_t::key, and tor_free.

Referenced by accounting_set_wakeup_time(), authority_cert_parse_from_string(), connection_or_check_valid_tls_handshake(), crypto_pk_get_fingerprint(), dir_signing_key_is_trusted(), dirserv_generate_networkstatus_vote_obj(), dirserv_router_get_status(), dirvote_compute_consensuses(), init_keys(), networkstatus_check_document_signature(), networkstatus_v2_parse_from_string(), rend_encode_v2_descriptors(), rend_get_service_id(), rend_mid_establish_intro(), rend_parse_v2_service_descriptor(), rend_service_introduce(), rend_service_load_keys(), router_parse_entry_from_string(), router_rebuild_descriptor(), and set_identity_key().

4.19.3.64 int crypto_pk_get_fingerprint (crypto_pk_env_t ∗ *pk*, char ∗ *fp_out*, int *add_space*)

Given a private or public key **pk**, put a fingerprint of the public key into **fp_out** (must have at least FINGERPRINT_LEN+1 bytes of space). Return 0 on success, -1 on failure.

Fingerprints are computed as the SHA1 digest of the ASN.1 encoding of the public key, converted to hexadecimal, in upper case, with a space after every four digits.

If **add_space** is false, omit the spaces.

References add_spaces_to_fp(), base16_encode(), crypto_pk_get_digest(), DIGEST_LEN, FINGERPRINT_LEN, and HEX_DIGEST_LEN.

Referenced by dirserv_add_own_fingerprint(), do_list_fingerprint(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), init_keys(), and router_dump_router_to_string().

4.19.3.65 int crypto_pk_key_is_private (const crypto_pk_env_t ∗ *key*)

Return true iff **key** contains the private-key portion of the RSA key.

References PRIVATE_KEY_OK, and tor_assert.

Referenced by token_check_object().

4.19.3.66 size_t crypto_pk_keysize (crypto_pk_env_t ∗ *env*)

Return the size of the public key modulus in **env**, in bytes.

References crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_private_hybrid_decrypt(), crypto_pk_public_checksig_digest(), crypto_pk_public_hybrid_encrypt(), networkstatus_check_document_signature(), onion_skin_create(), rend_parse_service_descriptor(), rend_service_introduce(), router_append_dirobj_signature(), and token_check_object().

4.19.3.67 int crypto_pk_private_decrypt (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*, int *padding*, int *warnOnFailure*)

Decrypt **fromlen** bytes from **from** with the private key in **env**, using the padding method **padding**. On success, write the result to **to**, and return the number of bytes written. On failure, return -1.

References crypto_get_rsa_padding(), crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_private_hybrid_decrypt().

4.19.3.68 int crypto_pk_private_hybrid_decrypt (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*, int *padding*, int *warnOnFailure*)

Invert crypto_pk_public_hybrid_encrypt.

References CIPHER_KEY_LEN, crypto_cipher_decrypt(), crypto_create_init_cipher(), crypto_free_cipher_env(), crypto_pk_keysize(), crypto_pk_private_decrypt(), LD_CRYPTO, tor_assert, and tor_free.

Referenced by onion_skin_server_handshake(), and rend_service_introduce().

4.19.3.69 int crypto_pk_private_sign (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Sign **fromlen** bytes of data from **from** with the private key in **env**, using PKCS1 padding. On success, write the signature to **to**, and return the number of bytes written. On failure, return -1.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_private_sign_digest(), and router_append_dirobj_signature().

4.19.3.70 int crypto_pk_private_sign_digest (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Compute a SHA1 digest of **fromlen** bytes of data stored at **from**; sign the data with the private key in **env**, and store it in **to**. Return the number of bytes written on success, and -1 on failure.

References crypto_digest(), crypto_pk_private_sign(), and DIGEST_LEN.

Referenced by rend_service_intro_has_opened().

4.19.3.71 int crypto_pk_public_checksig (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Check the signature in **from** (**fromlen** bytes long) with the public key in **env**, using PKCS1 padding. On success, write the signed data to **to**, and return the number of bytes written. On failure, return -1.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by check_signature_token(), crypto_pk_public_checksig_digest(), networkstatus_check_document_signature(), and routerinfo_incompatible_with_extrainfo().

4.19.3.72 int crypto_pk_public_checksig_digest (crypto_pk_env_t ∗ *env*, const char ∗ *data*, size_t *datalen*, const char ∗ *sig*, size_t *siglen*)

Check a siglen-byte long signature at **sig** against **datalen** bytes of data at **data**, using the public key in **env**. Return 0 if **sig** is a correct signature for SHA1(data). Else return -1.

References crypto_digest(), crypto_pk_keysize(), crypto_pk_public_checksig(), DIGEST_LEN, LD_BUG, LD_CRYPTO, tor_assert, and tor_free.

Referenced by rend_mid_establish_intro(), and rend_parse_service_descriptor().

4.19.3.73 int crypto_pk_public_encrypt (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*, int *padding*)

Encrypt **fromlen** bytes from **from** with the public key in **env**, using the padding method **padding**. On success, write the result to **to**, and return the number of bytes written. On failure, return -1.

References crypto_get_rsa_padding(), crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_public_hybrid_encrypt().

4.19.3.74 int crypto_pk_public_hybrid_encrypt (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*, int *padding*, int *force*)

Perform a hybrid (public/secret) encryption on **fromlen** bytes of data from **from**, with padding type 'padding', storing the results on **to**.

If no padding is used, the public key must be at least as large as **from**.

Returns the number of bytes written on success, -1 on failure.

The encrypted data consists of:

- The source data, padded and encrypted with the public key, if the padded source data is no longer than the public key, and **force** is false, OR
- The beginning of the source data prefixed with a 16-byte symmetric key, padded and encrypted with the public key; followed by the rest of the source data encrypted in AES-CTR mode with the symmetric key.

References CIPHER_KEY_LEN, crypto_cipher_encrypt(), crypto_cipher_encrypt_init_-cipher(), crypto_cipher_generate_key(), crypto_free_cipher_env(), crypto_get_rsa_-padding(), crypto_get_rsa_padding_overhead(), crypto_new_cipher_env(), crypto_pk_-keysize(), crypto_pk_public_encrypt(), crypto_cipher_env_t::key, PK_NO_PADDING, tor_assert, and tor_free.

Referenced by onion_skin_create().

4.19.3.75 int crypto_pk_read_private_key_from_filename (crypto_pk_env_t * *env*, const char * *keyfile*)

Read a PEM-encoded private key from the file named by **keyfile** into **env**. Return 0 on success, -1 on failure.

References crypto_pk_check_key(), crypto_pk_read_private_key_from_string(), LD_-CRYPTO, read_file_to_str(), and tor_free.

Referenced by init_key_from_file().

4.19.3.76 int crypto_pk_read_private_key_from_string (crypto_pk_env_t * *env*, const char * *s*)

Read a PEM-encoded private key from the string **s** into **env**. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_read_private_key_from_filename(), and get_next_token().

4.19.3.77 int crypto_pk_read_public_key_from_string (crypto_pk_env_t * *env*, const char * *src*, size_t *len*)

Read a PEM-encoded public key from the first **len** characters of **src**, and store the result in **env**. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by get_next_token().

4.19.3.78 static int crypto_pk_write_key_to_string_impl (crypto_pk_env_t * *env*, char ** *dest*, size_t * *len*, int *is_public*) `[static]`

Helper function to implement crypto_pk_write_*_key_to_string.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_write_private_key_to_string(), and crypto_pk_write_public_-key_to_string().

4.19.3.79 int crypto_pk_write_private_key_to_filename (crypto_pk_env_t * *env*, const char * *fname*)

Write the private key from **env** into the file named by **fname**, PEM-encoded. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, PRIVATE_KEY_OK, tor_assert, tor_free, and write_str_to_file().

Referenced by init_key_from_file(), and rotate_onion_key().

4.19.3.80 int crypto_pk_write_private_key_to_string (crypto_pk_env_t * *env*, char ** *dest*, size_t * *len*)

PEM-encode the private key portion of **env** and write it to a newly allocated string. On success, set *****dest** to the new string, *****len** to the string's length, and return 0. On failure, return -1.

References crypto_pk_write_key_to_string_impl().

Referenced by rend_service_load_keys().

4.19.3.81 int crypto_pk_write_public_key_to_string (crypto_pk_env_t * *env*, char ** *dest*, size_t * *len*)

PEM-encode the public key portion of **env** and write it to a newly allocated string. On success, set *****dest** to the new string, *****len** to the string's length, and return 0. On failure, return -1.

References crypto_pk_write_key_to_string_impl().

Referenced by dirserv_dump_directory_to_string(), dirvote_create_microdescriptor(), generate_runningrouters(), generate_v2_networkstatus_opinion(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), and router_dump_router_to_string().

4.19.3.82 int crypto_rand (char * *to*, size_t *n*)

Write **n** bytes of strong random data to **to**. Return 0 on success, -1 on failure.

References crypto_log_errors(), and tor_assert.

Referenced by _dns_randfn(), addressmap_get_virtual_address(), circuit_send_next_onion_skin(), crypto_cipher_generate_iv(), crypto_cipher_generate_key(), crypto_rand_int(), crypto_rand_uint64(), crypto_random_hostname(), do_hash_password(), fast_server_handshake(), init_cookie_authentication(), rend_client_send_establish_rendezvous(), rend_encrypt_v2_intro_points_basic(), and rend_service_load_keys().

4.19.3.83 int crypto_rand_int (unsigned int *max*)

Return a pseudorandom integer, chosen uniformly from the values between 0 and **max**-1.

References crypto_rand(), and tor_assert.

Referenced by add_an_entry_guard(), circuit_build_times_shuffle_and_store_array(), crypto_random_hostname(), entry_guards_parse_state(), or_connection_new(), origin_circuit_new(), rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_consider_services_upload(), run_scheduled_events(), smartlist_choose(), smartlist_shuffle(), and update_consensus_networkstatus_fetch_time().

4.19.3.84 uint64_t crypto_rand_uint64 (uint64_t *max*)

Return a pseudorandom 64-bit integer, chosen uniformly from the values between 0 and **max**-1.

References crypto_rand(), and tor_assert.

Referenced by circuit_build_times_generate_sample(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.19.3.85 char∗ crypto_random_hostname (int *min_rand_len*, int *max_rand_len*, const char ∗ *prefix*, const char ∗ *suffix*)

Generate and return a new random hostname starting with **prefix**, ending with **suffix**, and containing no less than **min_rand_len** and no more than **max_rand_len** random base32 characters between.

References base32_encode(), crypto_rand(), crypto_rand_int(), tor_assert, and tor_free.

Referenced by launch_wildcard_check(), tor_tls_context_new(), and tor_tls_new().

4.19.3.86 int crypto_seed_rng (int *startup*)

Seed OpenSSL's random number generator with bytes from the operating system. **startup** should be true iff we have just started Tor and have not yet allocated a bunch of fds. Return 0 on success, -1 on failure.

References LD_CRYPTO, and read_all().

Referenced by crypto_global_init(), and run_scheduled_events().

4.19.3.87 void crypto_thread_cleanup (void)

Free crypto resources held by this thread.

Referenced by cpuworker_main().

4.19.3.88 int digest256_from_base64 (char ∗ *digest*, const char ∗ *d64*)

Given a base-64 encoded, nul-terminated digest in **d64** (without trailing newline or = characters), decode it and store the result in the first DIGEST256_LEN bytes at **digest**.

References base64_decode(), BASE64_DIGEST256_LEN, and DIGEST256_LEN.

Referenced by vote_routerstatus_find_microdesc_hash().

4.19.3.89 int digest256_to_base64 (char ∗ *d64*, const char ∗ *digest*)

Base-64 encode DIGEST256_LINE bytes from **digest**, remove the trailing = and newline characters, and store the nul-terminated result in the first BASE64_DIGEST256_LEN+1 bytes of **d64**.

References BASE64_DIGEST256_LEN, base64_encode(), and DIGEST256_LEN.

Referenced by dirvote_format_microdesc_vote_line().

4.19.3.90 int digest_from_base64 (char * *digest*, const char * *d64*)

Given a base-64 encoded, nul-terminated digest in **d64** (without trailing newline or = characters), decode it and store the result in the first DIGEST_LEN bytes at **digest**.

References base64_decode(), BASE64_DIGEST_LEN, and DIGEST_LEN.

Referenced by routerstatus_parse_entry_from_string().

4.19.3.91 int digest_to_base64 (char * *d64*, const char * *digest*)

Base-64 encode DIGEST_LINE bytes from **digest**, remove the trailing = and newline characters, and store the nul-terminated result in the first BASE64_DIGEST_LEN+1 bytes of **d64**.

References BASE64_DIGEST_LEN, base64_encode(), and DIGEST_LEN.

Referenced by rep_hist_get_router_stability_doc(), and routerstatus_format_entry().

4.19.3.92 static void init_dh_param (void) `[static]`

Initialize dh_param_p and dh_param_g if they are not already set.

References tor_assert.

Referenced by crypto_dh_new(), and tor_check_dh_key().

4.19.3.93 static void log_engine (const char * *fn*, ENGINE * *e*) `[static]`

Log any OpenSSL engines we're using at NOTICE.

References LD_CRYPTO.

Referenced by crypto_global_init().

4.19.3.94 void secret_to_key (char * *key_out*, size_t *key_out_len*, const char * *secret*, size_t *secret_len*, const char * *s2k_specifier*)

Implement RFC2440-style iterated-salted S2K conversion: convert the **secret_len**-byte **secret** into a **key_out_len** byte **key_out**. As in RFC2440, the first 8 bytes of s2k_specifier are a salt; the 9th byte describes how much iteration to do. Does not support **key_out_len** > DIGEST_LEN.

References crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), DIGEST_LEN, tor_assert, and tor_free.

Referenced by do_hash_password(), and handle_control_authenticate().

4.19.3.95 static int setup_openssl_threading (void) `[static]`

Helper: Construct mutexes, and set callbacks to help OpenSSL handle being multithreaded.

References _n_openssl_mutexes, _openssl_dynlock_create_cb(), _openssl_dynlock_destroy_cb(), _openssl_dynlock_lock_cb(), _openssl_locking_cb(), tor_get_thread_id(), and tor_mutex_new().

Referenced by crypto_global_init().

4.19.3.96 void∗ smartlist_choose (const smartlist_t ∗ *sl*)

Return a randomly chosen element of **sl**; or NULL if **sl** is empty.

References crypto_rand_int().

Referenced by choose_random_entry(), directory_get_from_hs_dir(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.19.3.97 void smartlist_shuffle (smartlist_t ∗ *sl*)

Scramble the elements of **sl** into a random order.

References crypto_rand_int().

Referenced by update_extrainfo_downloads().

4.19.3.98 static int tor_check_dh_key (int *severity*, BIGNUM ∗ *bn*) `[static]`

Check for bad Diffie-Hellman public keys (g^x). Return 0 if the key is okay (in the subgroup [2,p-2]), or -1 if it's bad. See `http://www.cl.cam.ac.uk/ftp/users/rja14/psandqs.ps.gz` for some tips.

References init_dh_param(), LD_CRYPTO, and tor_assert.

Referenced by crypto_dh_compute_secret(), and crypto_dh_generate_public().

4.19.3.99 static ENGINE∗ try_load_engine (const char ∗ *path*, const char ∗ *engine*) `[static]`

Try to load an engine in a shared library via fully qualified path.

Referenced by crypto_global_init().

4.19.4 Variable Documentation

4.19.4.1 int _crypto_global_initialized = 0 `[static]`

Boolean: has OpenSSL's crypto been initialized?

Referenced by crypto_global_init().

4.19.4.2 int _n_openssl_mutexes = 0 `[static]`

How many mutexes have we allocated for use by OpenSSL?

Referenced by crypto_global_cleanup(), and setup_openssl_threading().

4.19.4.3 tor_mutex_t∗∗ _openssl_mutexes = NULL `[static]`

A number of preallocated mutexes for use by OpenSSL.

4.19.4.4 const uint8_t base64_decode_table[256] `[static]`

Initial value:

```
 {
  X, X, X, X, X, X, X, X, X, SP, SP, SP, X, SP, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  SP, X, X, X, X, X, X, X, X, X, X, 62, X, X, X, 63,
  52, 53, 54, 55, 56, 57, 58, 59, 60, 61, X, X, X, PAD, X, X,
  X, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
  15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, X, X, X, X, X,
  X, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
  41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
  X, X, X, X, X, X, X, X, X, X, X, X, X, X, X, X,
}
```

Internal table mapping byte values to what they represent in base64. Numbers 0..63 are 6-bit integers. SPs are spaces, and should be skipped. Xs are invalid and must not appear in base64. PAD indicates end-of-string.

4.19.4.5 BIGNUM∗ dh_param_g = NULL `[static]`

Shared G parameter for our DH key exchanges.

4.19.4.6 BIGNUM∗ dh_param_p = NULL `[static]`

Shared P parameter for our DH key exchanged.

4.20 crypto.h File Reference

Headers for **crypto.c** (p. 516).

`#include <stdio.h>`

`#include "torint.h"`

Data Structures

- struct **digests_t**

Defines

- #define **DIGEST_LEN** 20
- #define **DIGEST256_LEN** 32
- #define **CIPHER_KEY_LEN** 16
- #define **CIPHER_IV_LEN** 16
- #define **PK_BYTES** (1024/8)
- #define **DH_BYTES** (1024/8)
- #define **BASE64_DIGEST_LEN** 27
- #define **BASE64_DIGEST256_LEN** 43
- #define **PK_NO_PADDING** 60000
- #define **PK_PKCS1_PADDING** 60001
- #define **PK_PKCS1_OAEP_PADDING** 60002
- #define **PKCS1_PADDING_OVERHEAD** 11
- #define **PKCS1_OAEP_PADDING_OVERHEAD** 42
- #define **FINGERPRINT_LEN** 49
- #define **HEX_DIGEST_LEN** 40
- #define **HEX_DIGEST256_LEN** 64
- #define **N_DIGEST_ALGORITHMS** (DIGEST_SHA256+1)
- #define **crypto_pk_generate_key**(env) crypto_pk_generate_key_with_bits((env), (PK_BYTES*8))
- #define **BASE32_CHARS** "abcdefghijklmnopqrstuvwxyz234567"
- #define **S2K_SPECIFIER_LEN** 9

Typedefs

- typedef struct **crypto_pk_env_t crypto_pk_env_t**
- typedef struct **crypto_cipher_env_t crypto_cipher_env_t**
- typedef struct **crypto_digest_env_t crypto_digest_env_t**
- typedef struct **crypto_dh_env_t crypto_dh_env_t**

Enumerations

- enum **digest_algorithm_t** { **DIGEST_SHA1** = 0, **DIGEST_SHA256** = 1 }

Functions

- int **crypto_global_init** (int hardwareAccel, const char ∗accelName, const char ∗accelPath)
- void **crypto_thread_cleanup** (void)
- int **crypto_global_cleanup** (void)
- **crypto_pk_env_t** ∗ **crypto_new_pk_env** (void)
- void **crypto_free_pk_env** (**crypto_pk_env_t** ∗env)
- **crypto_cipher_env_t** ∗ **crypto_create_init_cipher** (const char ∗key, int encrypt_mode)
- **crypto_cipher_env_t** ∗ **crypto_new_cipher_env** (void)
- void **crypto_free_cipher_env** (**crypto_cipher_env_t** ∗env)
- int **crypto_pk_generate_key_with_bits** (**crypto_pk_env_t** ∗env, int bits)
- int **crypto_pk_read_private_key_from_filename** (**crypto_pk_env_t** ∗env, const char ∗keyfile)
- int **crypto_pk_write_public_key_to_string** (**crypto_pk_env_t** ∗env, char ∗∗dest, size_t ∗len)
- int **crypto_pk_write_private_key_to_string** (**crypto_pk_env_t** ∗env, char ∗∗dest, size_t ∗len)
- int **crypto_pk_read_public_key_from_string** (**crypto_pk_env_t** ∗env, const char ∗src, size_t len)
- int **crypto_pk_read_private_key_from_string** (**crypto_pk_env_t** ∗env, const char ∗s)
- int **crypto_pk_write_private_key_to_filename** (**crypto_pk_env_t** ∗env, const char ∗fname)
- int **crypto_pk_check_key** (**crypto_pk_env_t** ∗env)
- int **crypto_pk_cmp_keys** (**crypto_pk_env_t** ∗a, **crypto_pk_env_t** ∗b)
- size_t **crypto_pk_keysize** (**crypto_pk_env_t** ∗env)
- **crypto_pk_env_t** ∗ **crypto_pk_dup_key** (**crypto_pk_env_t** ∗orig)
- **crypto_pk_env_t** ∗ **crypto_pk_copy_full** (**crypto_pk_env_t** ∗orig)
- int **crypto_pk_key_is_private** (const **crypto_pk_env_t** ∗key)
- int **crypto_pk_public_encrypt** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen, int padding)
- int **crypto_pk_private_decrypt** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen, int padding, int warnOnFailure)
- int **crypto_pk_public_checksig** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen)
- int **crypto_pk_public_checksig_digest** (**crypto_pk_env_t** ∗env, const char ∗data, size_t datalen, const char ∗sig, size_t siglen)
- int **crypto_pk_private_sign** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen)
- int **crypto_pk_private_sign_digest** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen)
- int **crypto_pk_public_hybrid_encrypt** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen, int padding, int force)
- int **crypto_pk_private_hybrid_decrypt** (**crypto_pk_env_t** ∗env, char ∗to, const char ∗from, size_t fromlen, int padding, int warnOnFailure)
- int **crypto_pk_asn1_encode** (**crypto_pk_env_t** ∗pk, char ∗dest, size_t dest_len)
- **crypto_pk_env_t** ∗ **crypto_pk_asn1_decode** (const char ∗str, size_t len)
- int **crypto_pk_get_digest** (**crypto_pk_env_t** ∗pk, char ∗digest_out)
- int **crypto_pk_get_fingerprint** (**crypto_pk_env_t** ∗pk, char ∗fp_out, int add_space)

- int **crypto_pk_check_fingerprint_syntax** (const char *s)
- int **crypto_cipher_generate_key** (**crypto_cipher_env_t** *env)
- void **crypto_cipher_set_key** (**crypto_cipher_env_t** *env, const char *key)
- void **crypto_cipher_generate_iv** (char *iv_out)
- int **crypto_cipher_set_iv** (**crypto_cipher_env_t** *env, const char *iv)
- const char * **crypto_cipher_get_key** (**crypto_cipher_env_t** *env)
- int **crypto_cipher_encrypt_init_cipher** (**crypto_cipher_env_t** *env)
- int **crypto_cipher_decrypt_init_cipher** (**crypto_cipher_env_t** *env)
- int **crypto_cipher_encrypt** (**crypto_cipher_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_cipher_decrypt** (**crypto_cipher_env_t** *env, char *to, const char *from, size_t fromlen)
- int **crypto_cipher_crypt_inplace** (**crypto_cipher_env_t** *env, char *d, size_t len)
- int **crypto_cipher_encrypt_with_iv** (**crypto_cipher_env_t** *env, char *to, size_t tolen, const char *from, size_t fromlen)
- int **crypto_cipher_decrypt_with_iv** (**crypto_cipher_env_t** *env, char *to, size_t tolen, const char *from, size_t fromlen)
- int **crypto_digest** (char *digest, const char *m, size_t len)
- int **crypto_digest256** (char *digest, const char *m, size_t len, digest_algorithm_t algorithm)
- int **crypto_digest_all** (**digests_t** *ds_out, const char *m, size_t len)
- const char * **crypto_digest_algorithm_get_name** (digest_algorithm_t alg)
- int **crypto_digest_algorithm_parse_name** (const char *name)
- **crypto_digest_env_t** * **crypto_new_digest_env** (void)
- **crypto_digest_env_t** * **crypto_new_digest256_env** (digest_algorithm_t algorithm)
- void **crypto_free_digest_env** (**crypto_digest_env_t** *digest)
- void **crypto_digest_add_bytes** (**crypto_digest_env_t** *digest, const char *data, size_t len)
- void **crypto_digest_get_digest** (**crypto_digest_env_t** *digest, char *out, size_t out_len)
- **crypto_digest_env_t** * **crypto_digest_dup** (const **crypto_digest_env_t** *digest)
- void **crypto_digest_assign** (**crypto_digest_env_t** *into, const **crypto_digest_env_t** *from)
- void **crypto_hmac_sha1** (char *hmac_out, const char *key, size_t key_len, const char *msg, size_t msg_len)
- **crypto_dh_env_t** * **crypto_dh_new** (void)
- int **crypto_dh_get_bytes** (**crypto_dh_env_t** *dh)
- int **crypto_dh_generate_public** (**crypto_dh_env_t** *dh)
- int **crypto_dh_get_public** (**crypto_dh_env_t** *dh, char *pubkey_out, size_t pubkey_out_len)
- ssize_t **crypto_dh_compute_secret** (int severity, **crypto_dh_env_t** *dh, const char *pubkey, size_t pubkey_len, char *secret_out, size_t secret_out_len)
- void **crypto_dh_free** (**crypto_dh_env_t** *dh)
- int **crypto_expand_key_material** (const char *key_in, size_t in_len, char *key_out, size_t key_out_len)
- int **crypto_seed_rng** (int startup)
- int **crypto_rand** (char *to, size_t n)
- int **crypto_rand_int** (unsigned int max)
- uint64_t **crypto_rand_uint64** (uint64_t max)

- char * **crypto_random_hostname** (int min_rand_len, int max_rand_len, const char *prefix, const char *suffix)
- void * **smartlist_choose** (const struct **smartlist_t** *sl)
- void **smartlist_shuffle** (struct **smartlist_t** *sl)
- int **base64_encode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **base64_decode** (char *dest, size_t destlen, const char *src, size_t srclen)
- void **base32_encode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **base32_decode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **digest_to_base64** (char *d64, const char *digest)
- int **digest_from_base64** (char *digest, const char *d64)
- int **digest256_to_base64** (char *d64, const char *digest)
- int **digest256_from_base64** (char *digest, const char *d64)
- void **secret_to_key** (char *key_out, size_t key_out_len, const char *secret, size_t secret_len, const char *s2k_specifier)

4.20.1 Detailed Description

Headers for **crypto.c** (p. 516).

4.20.2 Define Documentation

4.20.2.1 #define BASE32_CHARS "abcdefghijklmnopqrstuvwxyz234567"

Characters that can appear (case-insensitively) in a base-32 encoding.

Referenced by base32_encode(), rend_parse_v2_service_descriptor(), and rend_valid_service_id().

4.20.2.2 #define BASE64_DIGEST256_LEN 43

Length of a sha256 message digest when encoded in base64 with trailing = signs removed.

Referenced by digest256_from_base64(), digest256_to_base64(), dir_split_resource_into_fingerprints(), dirvote_format_microdesc_vote_line(), and vote_routerstatus_find_microdesc_hash().

4.20.2.3 #define BASE64_DIGEST_LEN 27

Length of a sha1 message digest when encoded in base64 with trailing = signs removed.

Referenced by digest_from_base64(), digest_to_base64(), dir_split_resource_into_fingerprints(), rep_hist_get_router_stability_doc(), and routerstatus_format_entry().

4.20.2.4 #define CIPHER_IV_LEN 16

Length of our symmetric cipher's IV.

Referenced by crypto_cipher_decrypt_with_iv(), crypto_cipher_encrypt_with_iv(), crypto_cipher_generate_iv(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.20.2.5 #define CIPHER_KEY_LEN 16

Length of our symmetric cipher's keys.

Referenced by circuit_finish_handshake(), circuit_init_cpath_crypto(), crypto_cipher_decrypt_init_cipher(), crypto_cipher_encrypt_init_cipher(), crypto_cipher_generate_key(), crypto_cipher_set_key(), crypto_pk_private_hybrid_decrypt(), crypto_pk_public_hybrid_encrypt(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), and rend_mid_introduce().

4.20.2.6 #define DH_BYTES (1024/8)

Length of our DH keys.

4.20.2.7 #define DIGEST256_LEN 32

Length of the output of our second (improved) message digests. (For now this is just sha256, but any it can be any other 256-byte digest).

Referenced by _compare_digests256(), crypto_digest_get_digest(), digest256_from_base64(), digest256_to_base64(), dir_split_resource_into_fingerprints(), microdesc_cache_lookup_by_digest256(), networkstatus_add_detached_signatures(), networkstatus_check_document_signature(), networkstatus_get_detached_signatures(), networkstatus_parse_detached_signatures(), and tor_digest256_is_zero().

4.20.2.8 #define DIGEST_LEN 20

Length of the output of our message digest.

Referenced by _compare_digests(), _compare_dir_src_ents_by_authority_id(), _compare_old_routers_by_identity(), _compare_pairs(), _compare_routerinfo_by_id_digest(), _compare_routerinfo_by_ip_and_bw(), _compare_votes_by_authority_id(), accounting_set_wakeup_time(), add_an_entry_guard(), add_fingerprint_to_dir(), add_trusted_dir_server(), already_have_cert(), authority_cert_get_by_digests(), authority_cert_get_by_sk_digest(), authority_cert_parse_from_string(), authority_certs_fetch_missing(), bridge_add_from_config(), circuit_dump_by_conn(), circuit_extend(), circuit_find_to_cannibalize(), circuit_finish_handshake(), circuit_get_all_pending_on_or_conn(), circuit_get_intro_point(), circuit_get_next_by_pk_and_purpose(), circuit_get_open_circ_or_launch(), circuit_init_cpath_crypto(), circuit_is_acceptable(), circuit_list_path_impl(), circuit_n_conn_done(), circuit_send_next_onion_skin(), client_likes_consensus(), command_process_create_cell(), compare_digest_to_routerstatus_entry(), compare_routerstatus_entries(), compare_vote_rs(), connection_ap_can_use_exit(), connection_ap_fail_onehop(), connection_ap_make_link(), connection_dir_client_reached_eof(), connection_or_check_valid_tls_handshake(), connection_or_clear_identity_map(), connection_or_get_for_extend(), connection_or_init_conn_from_address(), connection_or_remove_from_identity_map(), connection_or_set_identity_digest(), connection_tls_finish_handshake(), control_event_guard(), crypto_dh_compute_secret(), crypto_digest_get_digest(), crypto_expand_key_material(), crypto_pk_get_fingerprint(), crypto_pk_private_sign_digest(), crypto_pk_public_checksig_digest(), decode_hashed_passwords(), digest_from_base64(), digest_to_base64(), digestmap_entries_eq(), digestmap_get(), digestmap_remove(), digestmap_set(), dir_networkstatus_download_failed(), dir_routerdesc_download_failed(), dir_signing_key_is_trusted(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_get_from_hs_dir(), directory_handle_command_get(), directory_-

initiate_command_rend(), directory_post_to_hs_dir(), dirserv_dump_directory_to_string(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_name_status(), dirserv_get_networkstatus_v2(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_load_fingerprint_file(), dirserv_orconn_tls_done(), dirserv_router_get_status(), dirvote_act(), dirvote_add_vote(), dirvote_compute_consensuses(), dirvote_fetch_missing_votes(), dirvote_get_vote(), do_hash_password(), entry_guard_register_connect_status(), entry_guard_set_status(), entry_guards_parse_state(), entry_guards_update_state(), extend_info_alloc(), extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), fast_client_handshake(), fast_server_handshake(), fetch_bridge_descriptors(), find_bridge_by_digest(), find_intro_circuit(), format_networkstatus_vote(), generate_runningrouters(), generate_v2_networkstatus_opinion(), get_link_history(), get_or_history(), get_secret_id_part_bytes(), getinfo_helper_entry_guards(), getinfo_helper_misc(), handle_control_attachstream(), handle_control_authenticate(), hex_digest_matches(), hexdigest_to_digest(), init_keys(), initiate_descriptor_downloads(), is_an_entry_guard(), list_single_server_status(), list_v3_auth_ids(), lookup_cached_dir_by_fp(), lookup_last_hid_serv_request(), measured_bw_line_parse(), networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), networkstatus_check_document_signature(), networkstatus_compute_consensus(), networkstatus_copy_old_consensus_info(), networkstatus_format_signatures(), networkstatus_get_cache_filename(), networkstatus_get_detached_signatures(), networkstatus_get_voter_by_id(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_get_by_digest(), networkstatus_v2_parse_from_string(), notify_control_networkstatus_changed(), onion_skin_client_handshake(), onion_skin_server_handshake(), onionskin_answer(), orconn_target_get_name(), parse_bridge_line(), parse_dir_server_line(), remove_dead_entry_guards(), remove_obsolete_entry_guards(), rend_cache_clean_v2_descs_as_dir(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_client_receive_rendezvous(), rend_client_refetch_v2_renddesc(), rend_client_remove_intro_point(), rend_client_send_introduction(), rend_compute_v2_desc_id(), rend_config_services(), rend_desc_v2_is_parsable(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_get_descriptor_id_bytes(), rend_get_service_id(), rend_id_is_in_interval(), rend_mid_establish_intro(), rend_mid_introduce(), rend_mid_rendezvous(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rend_service_get_by_pk_digest(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_rendezvous_has_opened(), rend_services_introduce(), rep_hist_dump_stats(), rep_hist_load_mtbf_data(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), rep_hist_record_mtbf_data(), router_add_to_routerlist(), router_digest_is_me(), router_digest_is_trusted_dir_type(), router_dump_router_to_string(), router_extrainfo_digest_is_me(), router_fingerprint_is_me(), router_get_by_hexdigest(), router_get_by_nickname(), router_get_consensus_status_by_nickname(), router_get_trusteddirserver_by_digest(), router_get_verbose_nickname(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), router_rebuild_descriptor(), router_reload_v2_networkstatus(), router_set_networkstatus_v2(), router_set_status(), routerinfo_get_configured_bridge(), routerinfo_incompatible_with_extrainfo(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), routerlist_replace(), routers_update_status_from_consensus_networkstatus(), routerset_parse(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), routerstatus_has_changed(), routerstatus_list_update_named_server_map(), secret_to_key(), set_routerstatus_from_routerinfo(), signed_desc_digest_is_recognized(), signed_descs_update_status_from_consensus_networkstatus(), smartlist_digest_isin(), tor_digest_is_zero(), tor_version_parse(), trusted_dirs_load_certs_from_string(), trusteddirserver_get_by_v3_auth_digest(), update_consensus_router_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.20.2.9 #define FINGERPRINT_LEN 49

Length of encoded public key fingerprints, including space; but not including terminating NUL.

Referenced by crypto_pk_check_fingerprint_syntax(), crypto_pk_get_fingerprint(), dirserv_add_own_fingerprint(), do_list_fingerprint(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), init_keys(), and router_dump_router_to_string().

4.20.2.10 #define HEX_DIGEST256_LEN 64

Length of hex encoding of SHA256 digest, not including final NUL.

Referenced by dir_split_resource_into_fingerprints(), networkstatus_get_detached_signatures(), and networkstatus_parse_detached_signatures().

4.20.2.11 #define HEX_DIGEST_LEN 40

Length of hex encoding of SHA1 digest, not including final NUL.

Referenced by circuit_get_open_circ_or_launch(), circuit_list_path_impl(), connection_ap_make_link(), connection_dir_client_reached_eof(), connection_or_check_valid_tls_handshake(), connection_or_init_conn_from_address(), consider_recording_trackhost(), control_event_guard(), crypto_pk_get_fingerprint(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), dirserv_get_name_status(), dirserv_get_networkstatus_v2(), dirserv_load_fingerprint_file(), dirvote_fetch_missing_votes(), entry_guard_register_connect_status(), entry_guard_set_status(), entry_guards_parse_state(), entry_guards_update_state(), extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), fetch_bridge_descriptors(), format_networkstatus_vote(), getinfo_helper_misc(), hex_digest_matches(), hexdigest_to_digest(), initiate_descriptor_downloads(), is_legal_hexdigest(), list_single_server_status(), measured_bw_line_parse(), networkstatus_add_detached_signatures(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_cache_filename(), networkstatus_get_detached_signatures(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), parse_bridge_line(), parse_dir_server_line(), remove_dead_entry_guards(), remove_obsolete_entry_guards(), rend_service_launch_establish_intro(), rep_hist_dump_stats(), rep_hist_load_mtbf_data(), rep_hist_record_mtbf_data(), router_dump_router_to_string(), router_fingerprint_is_me(), router_get_by_hexdigest(), router_get_by_nickname(), router_get_consensus_status_by_nickname(), router_get_verbose_nickname(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), router_reload_v2_networkstatus(), router_set_networkstatus_v2(), routerset_parse(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), tor_version_parse(), and update_v2_networkstatus_cache_downloads().

4.20.2.12 #define PK_BYTES (1024/8)

Length of our public keys.

Referenced by token_check_object().

4.20.2.13 #define PK_NO_PADDING 60000

Constants used to indicate no padding for public-key encryption

Referenced by crypto_get_rsa_padding(), and crypto_pk_public_hybrid_encrypt().

4.20.2.14 #define PK_PKCS1_OAEP_PADDING 60002

Constants used to indicate OAEP padding for public-key encryption

Referenced by crypto_get_rsa_padding(), onion_skin_create(), onion_skin_server_handshake(), and rend_service_introduce().

4.20.2.15 #define PK_PKCS1_PADDING 60001

Constants used to indicate PKCS1 padding for public-key encryption

Referenced by crypto_get_rsa_padding().

4.20.2.16 #define PKCS1_OAEP_PADDING_OVERHEAD 42

Number of bytes added for PKCS1-OAEP padding.

Referenced by rend_mid_introduce().

4.20.2.17 #define PKCS1_PADDING_OVERHEAD 11

Number of bytes added for PKCS1 padding.

4.20.2.18 #define S2K_SPECIFIER_LEN 9

Length of RFC2440-style S2K specifier: the first 8 bytes are a salt, the 9th describes how much iteration to do.

Referenced by decode_hashed_passwords(), do_hash_password(), and handle_control_authenticate().

4.20.3 Function Documentation

4.20.3.1 int base32_decode (char * *dest*, size_t *destlen*, const char * *src*, size_t *srclen*)

Implements base32 decoding as in rfc3548. Limitation: Requires that srclen*5 is a multiple of 8. Returns 0 if successful, -1 otherwise.

References LD_BUG, tor_assert, and tor_free.

Referenced by rend_cache_lookup_v2_desc_as_dir(), rend_compute_v2_desc_id(), rend_parse_introduction_points(), and rend_parse_v2_service_descriptor().

4.20.3.2 void base32_encode (char * *dest*, size_t *destlen*, const char * *src*, size_t *srclen*)

Implements base32 encoding as in rfc3548. Limitation: Requires that srclen*8 is a multiple of 5.

References BASE32_CHARS, and tor_assert.

Referenced by addressmap_get_virtual_address(), crypto_random_hostname(), directory_get_from_hs_dir(), directory_post_to_hs_dir(), lookup_last_hid_serv_request(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store_v2_desc_as_dir(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_get_service_id(), rend_mid_establish_intro(), rend_mid_introduce(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

4.20.3.3 int base64_decode (char ∗ *dest*, size_t *destlen*, const char ∗ *src*, size_t *srclen*)

Base-64 decode **srclen** bytes of data from **src**. Write the result into **dest**, if it will fit within **destlen** bytes. Return the number of bytes written on success; -1 if destlen is too short, or other failure.

NOTE 1: destlen is checked conservatively, as though srclen contained no spaces or padding.

NOTE 2: This implementation does not check for the correct number of padding "=" characters at the end of the string, and does not check for internal padding characters.

References tor_assert.

Referenced by decode_hashed_passwords(), digest256_from_base64(), digest_from_base64(), dir_split_resource_into_fingerprints(), get_next_token(), rend_parse_client_keys(), and rend_parse_service_authorization().

4.20.3.4 int base64_encode (char ∗ *dest*, size_t *destlen*, const char ∗ *src*, size_t *srclen*)

Base-64 encode **srclen** bytes of data from **src**. Write the result into **dest**, if it will fit within **destlen** bytes. Return the number of bytes written on success; -1 if destlen is too short, or other failure.

References tor_assert.

Referenced by alloc_http_authenticator(), digest256_to_base64(), digest_to_base64(), directory_get_from_hs_dir(), networkstatus_format_signatures(), rend_encode_v2_descriptors(), rend_service_load_keys(), and router_append_dirobj_signature().

4.20.3.5 int crypto_cipher_crypt_inplace (crypto_cipher_env_t ∗ *env*, char ∗ *buf*, size_t *len*)

Encrypt **len** bytes on **from** using the cipher in **env**; on success, return 0. On failure, return -1.

References aes_crypt_inplace(), and crypto_cipher_env_t::cipher.

Referenced by relay_crypt_one_payload().

4.20.3.6 int crypto_cipher_decrypt (crypto_cipher_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Decrypt **fromlen** bytes from **from** using the cipher **env**; on success, store the result to **to** and return 0. On failure, return -1.

References aes_crypt(), crypto_cipher_env_t::cipher, and tor_assert.

Referenced by crypto_pk_private_hybrid_decrypt(), and rend_decrypt_introduction_points().

4.20.3.7 int crypto_cipher_decrypt_init_cipher (crypto_cipher_env_t ∗ *env*)

Initialize the cipher in **env** for decryption. Return 0 on success, -1 on failure.

References aes_set_key(), crypto_cipher_env_t::cipher, CIPHER_KEY_LEN, crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_create_init_cipher().

4.20.3.8 int crypto_cipher_decrypt_with_iv (crypto_cipher_env_t ∗ *cipher*, char ∗ *to*, size_t *tolen*, const char ∗ *from*, size_t *fromlen*)

Decrypt **fromlen** bytes (at least 1+CIPHER_IV_LEN) from **from** with the key in **cipher** to the buffer in **to** of length **tolen**. **tolen** must be at least **fromlen** minus CIPHER_IV_LEN bytes for the initialization vector. On success, return the number of bytes written, on failure, return -1.

This function adjusts the current position of the counter in **cipher** to immediately after the decrypted data.

References CIPHER_IV_LEN, crypto_cipher_encrypt(), crypto_cipher_set_iv(), and tor_assert.

Referenced by rend_decrypt_introduction_points().

4.20.3.9 int crypto_cipher_encrypt (crypto_cipher_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Encrypt **fromlen** bytes from **from** using the cipher **env**; on success, store the result to **to** and return 0. On failure, return -1.

References aes_crypt(), crypto_cipher_env_t::cipher, and tor_assert.

Referenced by crypto_cipher_decrypt_with_iv(), crypto_cipher_encrypt_with_iv(), crypto_pk_public_hybrid_encrypt(), and rend_encrypt_v2_intro_points_basic().

4.20.3.10 int crypto_cipher_encrypt_init_cipher (crypto_cipher_env_t ∗ *env*)

Initialize the cipher in **env** for encryption. Return 0 on success, -1 on failure.

References aes_set_key(), crypto_cipher_env_t::cipher, CIPHER_KEY_LEN, crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_create_init_cipher(), and crypto_pk_public_hybrid_encrypt().

4.20.3.11 int crypto_cipher_encrypt_with_iv (crypto_cipher_env_t ∗ *cipher*, char ∗ *to*, size_t *tolen*, const char ∗ *from*, size_t *fromlen*)

Encrypt **fromlen** bytes (at least 1) from **from** with the key in **cipher** to the buffer in **to** of length **tolen**. **tolen** must be at least **fromlen** plus CIPHER_IV_LEN bytes for the initialization vector. On success, return the number of bytes written, on failure, return -1.

This function adjusts the current position of the counter in **cipher** to immediately after the encrypted data.

References CIPHER_IV_LEN, crypto_cipher_encrypt(), crypto_cipher_generate_iv(), crypto_cipher_set_iv(), and tor_assert.

Referenced by rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.20.3.12 void crypto_cipher_generate_iv (char * *iv_out*)

Generate an initialization vector for our AES-CTR cipher; store it in the first CIPHER_IV_LEN bytes of **iv_out**.

References CIPHER_IV_LEN, and crypto_rand().

Referenced by crypto_cipher_encrypt_with_iv().

4.20.3.13 int crypto_cipher_generate_key (crypto_cipher_env_t * *env*)

Generate a new random key for the symmetric cipher in **env**. Return 0 on success, -1 on failure. Does not initialize the cipher.

References CIPHER_KEY_LEN, crypto_rand(), crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_pk_public_hybrid_encrypt().

4.20.3.14 const char* crypto_cipher_get_key (crypto_cipher_env_t * *env*)

Return a pointer to the key set for the cipher in **env**.

References crypto_cipher_env_t::key.

4.20.3.15 int crypto_cipher_set_iv (crypto_cipher_env_t * *env*, const char * *iv*)

Adjust the counter of **env** to point to the first byte of the block corresponding to the encryption of the CIPHER_IV_LEN bytes at **iv**.

References aes_set_iv(), crypto_cipher_env_t::cipher, and tor_assert.

Referenced by crypto_cipher_decrypt_with_iv(), and crypto_cipher_encrypt_with_iv().

4.20.3.16 void crypto_cipher_set_key (crypto_cipher_env_t * *env*, const char * *key*)

Set the symmetric key for the cipher in **env** to the first CIPHER_KEY_LEN bytes of **key**. Does not initialize the cipher.

References CIPHER_KEY_LEN, crypto_cipher_env_t::key, and tor_assert.

Referenced by crypto_create_init_cipher().

4.20.3.17 crypto_cipher_env_t* crypto_create_init_cipher (const char * *key*, int *encrypt_mode*)

Create a new symmetric cipher for a given key and encryption flag (1=encrypt, 0=decrypt). Return the crypto object on success; NULL on failure.

References crypto_cipher_decrypt_init_cipher(), crypto_cipher_encrypt_init_cipher(), crypto_cipher_set_key(), crypto_free_cipher_env(), crypto_new_cipher_env(), and LD_CRYPTO.

Referenced by circuit_init_cpath_crypto(), crypto_pk_private_hybrid_decrypt(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.20.3.18 ssize_t crypto_dh_compute_secret (int *severity*, crypto_dh_env_t ∗ *dh*, const char ∗ *pubkey*, size_t *pubkey_len*, char ∗ *secret_out*, size_t *secret_bytes_out*)

Given a DH key exchange object, and our peer's value of g^y (as a **pubkey_len**-byte value in **pubkey**) generate **secret_bytes_out** bytes of shared key material and write them to **secret_out**. Return the number of bytes generated on success, or -1 on failure.

(We generate key material by computing SHA1(g^xy || "\x00") || SHA1(g^xy || "\x01") || ... where || is concatenation.)

References crypto_dh_get_bytes(), crypto_expand_key_material(), crypto_log_errors(), crypto_dh_env_t::dh, DIGEST_LEN, LD_CRYPTO, tor_assert, tor_check_dh_key(), and tor_free.

Referenced by onion_skin_client_handshake(), onion_skin_server_handshake(), rend_client_receive_rendezvous(), and rend_service_introduce().

4.20.3.19 void crypto_dh_free (crypto_dh_env_t ∗ *dh*)

Free a DH key exchange object.

References crypto_dh_env_t::dh, tor_assert, and tor_free.

Referenced by circuit_finish_handshake(), circuit_free_cpath_node(), onion_skin_create(), onion_skin_server_handshake(), rend_client_receive_rendezvous(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and tor_tls_context_new().

4.20.3.20 int crypto_dh_generate_public (crypto_dh_env_t ∗ *dh*)

Generate <x,g^x> for our part of the key exchange. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_dh_env_t::dh, LD_CRYPTO, and tor_check_dh_key().

Referenced by crypto_dh_get_public(), and rend_service_introduce().

4.20.3.21 int crypto_dh_get_bytes (crypto_dh_env_t ∗ *dh*)

Return the length of the DH key in **dh**, in bytes.

References crypto_dh_env_t::dh, and tor_assert.

Referenced by crypto_dh_compute_secret(), onion_skin_client_handshake(), and onion_skin_create().

4.20.3.22 int crypto_dh_get_public (crypto_dh_env_t * *dh*, char * *pubkey*, size_t *pubkey_len*)

Generate g^x as necessary, and write the g^x for the key exchange as a **pubkey_len**-byte value into **pubkey**. Return 0 on success, -1 on failure. **pubkey_len** must be >= DH_BYTES.

References crypto_dh_generate_public(), crypto_dh_env_t::dh, LD_CRYPTO, and tor_assert.

Referenced by onion_skin_create(), onion_skin_server_handshake(), and rend_service_rendezvous_has_opened().

4.20.3.23 crypto_dh_env_t* crypto_dh_new (void)

Allocate and return a new DH object for a key exchange.

References crypto_log_errors(), crypto_dh_env_t::dh, init_dh_param(), and tor_free.

Referenced by onion_skin_create(), onion_skin_server_handshake(), rend_service_introduce(), and tor_tls_context_new().

4.20.3.24 int crypto_digest (char * *digest*, const char * *m*, size_t *len*)

Compute the SHA1 digest of **len** bytes in data stored in **m**. Write the DIGEST_LEN byte result into **digest**. Return 0 on success, -1 on failure.

References tor_assert.

Referenced by crypto_digest_all(), crypto_expand_key_material(), crypto_pk_get_digest(), crypto_pk_private_sign_digest(), crypto_pk_public_checksig_digest(), rend_encode_v2_descriptors(), rend_mid_establish_intro(), rend_service_intro_has_opened(), and router_get_hash_impl().

4.20.3.25 void crypto_digest_add_bytes (crypto_digest_env_t * *digest*, const char * *data*, size_t *len*)

Add **len** bytes from **data** to the digest object.

References crypto_digest_env_t::algorithm, crypto_digest_env_t::d, crypto_digest_env_t::sha1, crypto_digest_env_t::sha2, tor_assert, and tor_fragile_assert.

Referenced by accounting_set_wakeup_time(), circuit_init_cpath_crypto(), get_secret_id_part_bytes(), hash_list_members(), relay_digest_matches(), relay_set_digest(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.20.3.26 const char* crypto_digest_algorithm_get_name (digest_algorithm_t *alg*)

Return the name of an algorithm, as used in directory documents.

References tor_fragile_assert.

Referenced by networkstatus_add_detached_signatures(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), and vote_routerstatus_find_microdesc_hash().

4.20.3.27 int crypto_digest_algorithm_parse_name (const char ∗ *name*)

Given the name of a digest algorithm, return its integer value, or -1 if the name is not recognized.

Referenced by networkstatus_parse_detached_signatures(), and networkstatus_parse_vote_from_string().

4.20.3.28 int crypto_digest_all (digests_t ∗ *ds_out*, const char ∗ *m*, size_t *len*)

Set the **digests_t** (p. 77) in **ds_out** to contain every digest on the **len** bytes in **m** that we know how to compute. Return 0 on success, -1 on failure.

References crypto_digest(), digests_t::d, and tor_assert.

Referenced by router_get_hashes_impl().

4.20.3.29 void crypto_digest_assign (crypto_digest_env_t ∗ *into*, const crypto_digest_env_t ∗ *from*)

Replace the state of the digest object **into** with the state of the digest object **from**.

References tor_assert.

Referenced by relay_digest_matches().

4.20.3.30 crypto_digest_env_t∗ crypto_digest_dup (const crypto_digest_env_t ∗ *digest*)

Allocate and return a new digest object with the same state as **digest**

References tor_assert.

Referenced by relay_digest_matches().

4.20.3.31 void crypto_digest_get_digest (crypto_digest_env_t ∗ *digest*, char ∗ *out*, size_t *out_len*)

Compute the hash of the data that has been passed to the digest object; write the first out_len bytes of the result to **out**. **out_len** must be <= DIGEST256_LEN.

References crypto_digest_env_t::algorithm, crypto_digest_env_t::d, DIGEST256_LEN, DIGEST_LEN, crypto_digest_env_t::sha1, crypto_digest_env_t::sha2, tor_assert, and tor_fragile_assert.

Referenced by accounting_set_wakeup_time(), get_secret_id_part_bytes(), hash_list_members(), relay_digest_matches(), relay_set_digest(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.20.3.32 int crypto_expand_key_material (const char ∗ *key_in*, size_t *key_in_len*, char ∗ *key_out*, size_t *key_out_len*)

Given **key_in_len** bytes of negotiated randomness in **key_in** ("K"), expand it into **key_out_len** bytes of negotiated key material in **key_out** by taking the first **key_out_len** bytes of H(K | [00]) | H(K | [01]) | ....

Return 0 on success, -1 on failure.

References crypto_digest(), DIGEST_LEN, tor_assert, and tor_free.

Referenced by crypto_dh_compute_secret(), fast_client_handshake(), and fast_server_handshake().

4.20.3.33 void crypto_free_cipher_env (crypto_cipher_env_t * *env*)

Free a symmetric cipher.

References aes_free_cipher(), crypto_cipher_env_t::cipher, tor_assert, and tor_free.

Referenced by circuit_free(), circuit_free_cpath_node(), crypto_create_init_cipher(), crypto_pk_private_hybrid_decrypt(), crypto_pk_public_hybrid_encrypt(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), and rend_encrypt_v2_intro_points_stealth().

4.20.3.34 void crypto_free_digest_env (crypto_digest_env_t * *digest*)

Deallocate a digest object.

References tor_free.

Referenced by accounting_set_wakeup_time(), circuit_free(), circuit_free_cpath_node(), get_secret_id_part_bytes(), hash_list_members(), relay_digest_matches(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.20.3.35 void crypto_free_pk_env (crypto_pk_env_t * *env*)

Release a reference to an asymmetric key; when all the references are released, free the key.

References crypto_pk_env_t::key, crypto_pk_env_t::refs, and tor_free.

Referenced by authority_cert_free(), circuit_free(), connection_or_check_valid_tls_handshake(), cpuworker_main(), extend_info_free(), init_key_from_file(), init_keys(), load_authority_keyset(), microdesc_free(), networkstatus_v2_free(), rend_authorized_client_free(), rend_intro_point_free(), rend_mid_establish_intro(), rend_service_descriptor_free(), rend_service_free(), rend_service_load_keys(), rotate_onion_key(), router_free_all(), router_parse_directory(), router_parse_runningrouters(), routerinfo_free(), set_identity_key(), set_onion_key(), token_clear(), tor_tls_context_decref(), and tor_tls_context_new().

4.20.3.36 int crypto_global_cleanup (void)

Uninitialize the crypto library. Return 0 on success, -1 on failure.

References _n_openssl_mutexes, tor_free, and tor_mutex_free().

Referenced by tor_cleanup().

4.20.3.37 int crypto_global_init (int *useAccel*, const char * *accelName*, const char * *accelDir*)

Initialize the crypto library. Return 0 on success, -1 on failure.

References _crypto_global_initialized, crypto_seed_rng(), LD_CRYPTO, log_engine(), setup_openssl_threading(), and try_load_engine().

Referenced by init_keys(), and tor_init().

4.20.3.38 void crypto_hmac_sha1 (char ∗ *hmac_out*, const char ∗ *key*, size_t *key_len*, const char ∗ *msg*, size_t *msg_len*)

Compute the HMAC-SHA-1 of the **msg_len** bytes in **msg**, using the **key** of length **key_len**. Store the DIGEST_LEN-byte result in **hmac_out**.

References tor_assert.

4.20.3.39 crypto_cipher_env_t∗ crypto_new_cipher_env (void)

Allocate and return a new symmetric cipher.

References aes_new_cipher(), and crypto_cipher_env_t::cipher.

Referenced by crypto_create_init_cipher(), and crypto_pk_public_hybrid_encrypt().

4.20.3.40 crypto_digest_env_t∗ crypto_new_digest_env (void)

Allocate and return a new digest object.

References crypto_digest_env_t::algorithm, crypto_digest_env_t::d, and crypto_digest_env_t::sha1.

Referenced by accounting_set_wakeup_time(), circuit_init_cpath_crypto(), get_secret_id_part_bytes(), hash_list_members(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_get_descriptor_id_bytes(), rend_service_introduce(), and secret_to_key().

4.20.3.41 crypto_pk_env_t∗ crypto_new_pk_env (void)

Allocate and return storage for a public key. The key itself will not yet be set.

References _crypto_new_pk_env_rsa().

Referenced by get_next_token(), init_key_from_file(), init_keys(), rend_service_load_keys(), rotate_onion_key(), and tor_tls_context_new().

4.20.3.42 crypto_pk_env_t∗ crypto_pk_asn1_decode (const char ∗ *str*, size_t *len*)

Decode an ASN.1-encoded public key from **str**; return the result on success and NULL on failure.

References _crypto_new_pk_env_rsa(), crypto_log_errors(), and tor_free.

Referenced by rend_mid_establish_intro(), rend_parse_service_descriptor(), and rend_service_introduce().

4.20.3.43 int crypto_pk_asn1_encode (crypto_pk_env_t * *pk*, char * *dest*, size_t *dest_len*)

ASN.1-encode the public portion of **pk** into **dest**. Return -1 on error, or the number of characters used on success.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_free.

Referenced by rend_service_intro_has_opened().

4.20.3.44 int crypto_pk_check_fingerprint_syntax (const char * *s*)

Return true iff **s** is in the correct format for a fingerprint.

References FINGERPRINT_LEN.

4.20.3.45 int crypto_pk_check_key (crypto_pk_env_t * *env*)

Return true iff **env** has a valid key.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_read_private_key_from_filename(), init_key_from_file(), and rend_service_load_keys().

4.20.3.46 int crypto_pk_cmp_keys (crypto_pk_env_t * *a*, crypto_pk_env_t * *b*)

Compare the public-key components of a and b. Return -1 if a<b, 0 if a==b, and 1 if a>b.

References crypto_pk_env_t::key, PUBLIC_KEY_OK, and tor_assert.

Referenced by connection_or_check_valid_tls_handshake(), load_authority_keyset(), router_differences_are_cosmetic(), and router_dump_router_to_string().

4.20.3.47 crypto_pk_env_t* crypto_pk_copy_full (crypto_pk_env_t * *env*)

Make a real honest-to-goodness copy of **env**, and return it.

References _crypto_new_pk_env_rsa(), crypto_pk_env_t::key, PRIVATE_KEY_OK, and tor_assert.

Referenced by dup_onion_keys().

4.20.3.48 crypto_pk_env_t* crypto_pk_dup_key (crypto_pk_env_t * *env*)

Increase the reference count of **env**, and return it.

References crypto_pk_env_t::key, crypto_pk_env_t::refs, and tor_assert.

Referenced by authority_cert_dup(), extend_info_alloc(), extend_info_dup(), rend_service_launch_establish_intro(), rend_service_load_keys(), rend_service_update_descriptor(), router_rebuild_descriptor(), and tor_tls_context_new().

4.20.3.49 int crypto_pk_generate_key_with_bits (crypto_pk_env_t ∗ *env*, int *bits*)

Generate a **bits**-bit new public/private keypair in **env**. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

4.20.3.50 int crypto_pk_get_digest (crypto_pk_env_t ∗ *pk*, char ∗ *digest_out*)

Given a private or public key **pk**, put a SHA1 hash of the public key into **digest_out** (must have DIGEST_LEN bytes of space). Return 0 on success, -1 on failure.

References crypto_digest(), crypto_log_errors(), crypto_pk_env_t::key, and tor_free.

Referenced by accounting_set_wakeup_time(), authority_cert_parse_from_string(), connection_or_check_valid_tls_handshake(), crypto_pk_get_fingerprint(), dir_signing_key_is_trusted(), dirserv_generate_networkstatus_vote_obj(), dirserv_router_get_status(), dirvote_compute_consensuses(), init_keys(), networkstatus_check_document_signature(), networkstatus_v2_parse_from_string(), rend_encode_v2_descriptors(), rend_get_service_id(), rend_mid_establish_intro(), rend_parse_v2_service_descriptor(), rend_service_introduce(), rend_service_load_keys(), router_parse_entry_from_string(), router_rebuild_descriptor(), and set_identity_key().

4.20.3.51 int crypto_pk_get_fingerprint (crypto_pk_env_t ∗ *pk*, char ∗ *fp_out*, int *add_space*)

Given a private or public key **pk**, put a fingerprint of the public key into **fp_out** (must have at least FINGERPRINT_LEN+1 bytes of space). Return 0 on success, -1 on failure.

Fingerprints are computed as the SHA1 digest of the ASN.1 encoding of the public key, converted to hexadecimal, in upper case, with a space after every four digits.

If **add_space** is false, omit the spaces.

References add_spaces_to_fp(), base16_encode(), crypto_pk_get_digest(), DIGEST_LEN, FINGERPRINT_LEN, and HEX_DIGEST_LEN.

Referenced by dirserv_add_own_fingerprint(), do_list_fingerprint(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), init_keys(), and router_dump_router_to_string().

4.20.3.52 int crypto_pk_key_is_private (const crypto_pk_env_t ∗ *key*)

Return true iff **key** contains the private-key portion of the RSA key.

References PRIVATE_KEY_OK, and tor_assert.

Referenced by token_check_object().

4.20.3.53 size_t crypto_pk_keysize (crypto_pk_env_t ∗ *env*)

Return the size of the public key modulus in **env**, in bytes.

References crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_private_hybrid_decrypt(), crypto_pk_public_checksig_digest(), crypto_pk_public_hybrid_encrypt(), networkstatus_check_document_signature(), onion_-

skin_create(), rend_parse_service_descriptor(), rend_service_introduce(), router_append_dirobj_signature(), and token_check_object().

4.20.3.54 int crypto_pk_private_decrypt (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*, int *padding*, int *warnOnFailure*)

Decrypt **fromlen** bytes from **from** with the private key in **env**, using the padding method **padding**. On success, write the result to **to**, and return the number of bytes written. On failure, return -1.

References crypto_get_rsa_padding(), crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_private_hybrid_decrypt().

4.20.3.55 int crypto_pk_private_hybrid_decrypt (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*, int *padding*, int *warnOnFailure*)

Invert crypto_pk_public_hybrid_encrypt.

References CIPHER_KEY_LEN, crypto_cipher_decrypt(), crypto_create_init_cipher(), crypto_free_cipher_env(), crypto_pk_keysize(), crypto_pk_private_decrypt(), LD_CRYPTO, tor_assert, and tor_free.

Referenced by onion_skin_server_handshake(), and rend_service_introduce().

4.20.3.56 int crypto_pk_private_sign (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Sign **fromlen** bytes of data from **from** with the private key in **env**, using PKCS1 padding. On success, write the signature to **to**, and return the number of bytes written. On failure, return -1.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_private_sign_digest(), and router_append_dirobj_signature().

4.20.3.57 int crypto_pk_private_sign_digest (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Compute a SHA1 digest of **fromlen** bytes of data stored at **from**; sign the data with the private key in **env**, and store it in **to**. Return the number of bytes written on success, and -1 on failure.

References crypto_digest(), crypto_pk_private_sign(), and DIGEST_LEN.

Referenced by rend_service_intro_has_opened().

4.20.3.58 int crypto_pk_public_checksig (crypto_pk_env_t ∗ *env*, char ∗ *to*, const char ∗ *from*, size_t *fromlen*)

Check the signature in **from** (**fromlen** bytes long) with the public key in **env**, using PKCS1 padding. On success, write the signed data to **to**, and return the number of bytes written. On failure, return -1.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by check_signature_token(), crypto_pk_public_checksig_digest(), networkstatus_check_document_signature(), and routerinfo_incompatible_with_extrainfo().

4.20.3.59 int crypto_pk_public_checksig_digest (crypto_pk_env_t * *env*, const char * *data*, size_t *datalen*, const char * *sig*, size_t *siglen*)

Check a siglen-byte long signature at **sig** against **datalen** bytes of data at **data**, using the public key in **env**. Return 0 if **sig** is a correct signature for SHA1(data). Else return -1.

References crypto_digest(), crypto_pk_keysize(), crypto_pk_public_checksig(), DIGEST_LEN, LD_BUG, LD_CRYPTO, tor_assert, and tor_free.

Referenced by rend_mid_establish_intro(), and rend_parse_service_descriptor().

4.20.3.60 int crypto_pk_public_encrypt (crypto_pk_env_t * *env*, char * *to*, const char * *from*, size_t *fromlen*, int *padding*)

Encrypt **fromlen** bytes from **from** with the public key in **env**, using the padding method **padding**. On success, write the result to **to**, and return the number of bytes written. On failure, return -1.

References crypto_get_rsa_padding(), crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_public_hybrid_encrypt().

4.20.3.61 int crypto_pk_public_hybrid_encrypt (crypto_pk_env_t * *env*, char * *to*, const char * *from*, size_t *fromlen*, int *padding*, int *force*)

Perform a hybrid (public/secret) encryption on **fromlen** bytes of data from **from**, with padding type 'padding', storing the results on **to**.

If no padding is used, the public key must be at least as large as **from**.

Returns the number of bytes written on success, -1 on failure.

The encrypted data consists of:

- The source data, padded and encrypted with the public key, if the padded source data is no longer than the public key, and **force** is false, OR
- The beginning of the source data prefixed with a 16-byte symmetric key, padded and encrypted with the public key; followed by the rest of the source data encrypted in AES-CTR mode with the symmetric key.

References CIPHER_KEY_LEN, crypto_cipher_encrypt(), crypto_cipher_encrypt_init_cipher(), crypto_cipher_generate_key(), crypto_free_cipher_env(), crypto_get_rsa_padding(), crypto_get_rsa_padding_overhead(), crypto_new_cipher_env(), crypto_pk_keysize(), crypto_pk_public_encrypt(), crypto_cipher_env_t::key, PK_NO_PADDING, tor_assert, and tor_free.

Referenced by onion_skin_create().

4.20.3.62 int crypto_pk_read_private_key_from_filename (crypto_pk_env_t * *env*, const char * *keyfile*)

Read a PEM-encoded private key from the file named by **keyfile** into **env**. Return 0 on success, -1 on failure.

References crypto_pk_check_key(), crypto_pk_read_private_key_from_string(), LD_CRYPTO, read_file_to_str(), and tor_free.

Referenced by init_key_from_file().

4.20.3.63 int crypto_pk_read_private_key_from_string (crypto_pk_env_t * *env*, const char * *s*)

Read a PEM-encoded private key from the string **s** into **env**. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by crypto_pk_read_private_key_from_filename(), and get_next_token().

4.20.3.64 int crypto_pk_read_public_key_from_string (crypto_pk_env_t * *env*, const char * *src*, size_t *len*)

Read a PEM-encoded public key from the first **len** characters of **src**, and store the result in **env**. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, and tor_assert.

Referenced by get_next_token().

4.20.3.65 int crypto_pk_write_private_key_to_filename (crypto_pk_env_t * *env*, const char * *fname*)

Write the private key from **env** into the file named by **fname**, PEM-encoded. Return 0 on success, -1 on failure.

References crypto_log_errors(), crypto_pk_env_t::key, PRIVATE_KEY_OK, tor_assert, tor_free, and write_str_to_file().

Referenced by init_key_from_file(), and rotate_onion_key().

4.20.3.66 int crypto_pk_write_private_key_to_string (crypto_pk_env_t * *env*, char ** *dest*, size_t * *len*)

PEM-encode the private key portion of **env** and write it to a newly allocated string. On success, set *__dest__ to the new string, *__len__ to the string's length, and return 0. On failure, return -1.

References crypto_pk_write_key_to_string_impl().

Referenced by rend_service_load_keys().

4.20.3.67 int crypto_pk_write_public_key_to_string (crypto_pk_env_t * *env*, char ** *dest*, size_t * *len*)

PEM-encode the public key portion of **env** and write it to a newly allocated string. On success, set *__dest__ to the new string, *__len__ to the string's length, and return 0. On failure, return -1.

References crypto_pk_write_key_to_string_impl().

Referenced by dirserv_dump_directory_to_string(), dirvote_create_microdescriptor(), generate_runningrouters(), generate_v2_networkstatus_opinion(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), and router_dump_router_to_string().

4.20.3.68 int crypto_rand (char * *to*, size_t *n*)

Write **n** bytes of strong random data to **to**. Return 0 on success, -1 on failure.

References crypto_log_errors(), and tor_assert.

Referenced by _dns_randfn(), addressmap_get_virtual_address(), circuit_send_next_onion_skin(), crypto_cipher_generate_iv(), crypto_cipher_generate_key(), crypto_rand_int(), crypto_rand_uint64(), crypto_random_hostname(), do_hash_password(), fast_server_handshake(), init_cookie_authentication(), rend_client_send_establish_rendezvous(), rend_encrypt_v2_intro_points_basic(), and rend_service_load_keys().

4.20.3.69 int crypto_rand_int (unsigned int *max*)

Return a pseudorandom integer, chosen uniformly from the values between 0 and **max**-1.

References crypto_rand(), and tor_assert.

Referenced by add_an_entry_guard(), circuit_build_times_shuffle_and_store_array(), crypto_random_hostname(), entry_guards_parse_state(), or_connection_new(), origin_circuit_new(), rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_consider_services_upload(), run_scheduled_events(), smartlist_choose(), smartlist_shuffle(), and update_consensus_networkstatus_fetch_time().

4.20.3.70 uint64_t crypto_rand_uint64 (uint64_t *max*)

Return a pseudorandom 64-bit integer, chosen uniformly from the values between 0 and **max**-1.

References crypto_rand(), and tor_assert.

Referenced by circuit_build_times_generate_sample(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.20.3.71 char* crypto_random_hostname (int *min_rand_len*, int *max_rand_len*, const char * *prefix*, const char * *suffix*)

Generate and return a new random hostname starting with **prefix**, ending with **suffix**, and containing no less than **min_rand_len** and no more than **max_rand_len** random base32 characters between.

References base32_encode(), crypto_rand(), crypto_rand_int(), tor_assert, and tor_free.

Referenced by launch_wildcard_check(), tor_tls_context_new(), and tor_tls_new().

4.20.3.72 int crypto_seed_rng (int *startup*)

Seed OpenSSL's random number generator with bytes from the operating system. **startup** should be true iff we have just started Tor and have not yet allocated a bunch of fds. Return 0 on success, -1 on failure.

References LD_CRYPTO, and read_all().

Referenced by crypto_global_init(), and run_scheduled_events().

4.20.3.73 void crypto_thread_cleanup (void)

Free crypto resources held by this thread.

Referenced by cpuworker_main().

4.20.3.74 int digest256_from_base64 (char ∗ *digest*, const char ∗ *d64*)

Given a base-64 encoded, nul-terminated digest in **d64** (without trailing newline or = characters), decode it and store the result in the first DIGEST256_LEN bytes at **digest**.

References base64_decode(), BASE64_DIGEST256_LEN, and DIGEST256_LEN.

Referenced by vote_routerstatus_find_microdesc_hash().

4.20.3.75 int digest256_to_base64 (char ∗ *d64*, const char ∗ *digest*)

Base-64 encode DIGEST256_LINE bytes from **digest**, remove the trailing = and newline characters, and store the nul-terminated result in the first BASE64_DIGEST256_LEN+1 bytes of **d64**.

References BASE64_DIGEST256_LEN, base64_encode(), and DIGEST256_LEN.

Referenced by dirvote_format_microdesc_vote_line().

4.20.3.76 int digest_from_base64 (char ∗ *digest*, const char ∗ *d64*)

Given a base-64 encoded, nul-terminated digest in **d64** (without trailing newline or = characters), decode it and store the result in the first DIGEST_LEN bytes at **digest**.

References base64_decode(), BASE64_DIGEST_LEN, and DIGEST_LEN.

Referenced by routerstatus_parse_entry_from_string().

4.20.3.77 int digest_to_base64 (char ∗ *d64*, const char ∗ *digest*)

Base-64 encode DIGEST_LINE bytes from **digest**, remove the trailing = and newline characters, and store the nul-terminated result in the first BASE64_DIGEST_LEN+1 bytes of **d64**.

References BASE64_DIGEST_LEN, base64_encode(), and DIGEST_LEN.

Referenced by rep_hist_get_router_stability_doc(), and routerstatus_format_entry().

4.20.3.78 void secret_to_key (char ∗ *key_out*, size_t *key_out_len*, const char ∗ *secret*, size_t *secret_len*, const char ∗ *s2k_specifier*)

Implement RFC2440-style iterated-salted S2K conversion: convert the **secret_len**-byte **secret** into a **key_out_len** byte **key_out**. As in RFC2440, the first 8 bytes of s2k_specifier are a salt; the 9th byte describes how much iteration to do. Does not support **key_out_len** > DIGEST_LEN.

References crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), DIGEST_LEN, tor_assert, and tor_free.

Referenced by do_hash_password(), and handle_control_authenticate().

4.20.3.79 void smartlist_shuffle (smartlist_t * *sl*)

Scramble the elements of **sl** into a random order.

References crypto_rand_int().

Referenced by update_extrainfo_downloads().

4.21 directory.c File Reference

Code to send and fetch directories and router descriptors via HTTP. Directories use **dirserv.c** (p. 583) to generate the results; clients use routers.c to parse them.

`#include "or.h"`

Defines

- #define **ALLOW_DIRECTORY_TIME_SKEW** (30*60)
- #define **X_ADDRESS_HEADER** "X-Your-Address-Is: "
- #define **FULL_DIR_CACHE_LIFETIME** (60*60)
- #define **RUNNINGROUTERS_CACHE_LIFETIME** (20*60)
- #define **DIRPORTFRONTPAGE_CACHE_LIFETIME** (20*60)
- #define **NETWORKSTATUS_CACHE_LIFETIME** (5*60)
- #define **ROUTERDESC_CACHE_LIFETIME** (30*60)
- #define **ROUTERDESC_BY_DIGEST_CACHE_LIFETIME** (48*60*60)
- #define **ROBOTS_CACHE_LIFETIME** (24*60*60)
- #define **MICRODESC_CACHE_LIFETIME** (48*60*60)
- #define **CONDITIONAL_CONSENSUS_FPR_LEN** 3
- #define **MAX_DIRECTORY_OBJECT_SIZE** (10*(1<<20))
- #define **CONSENSUS_URL_PREFIX** "/tor/status-vote/current/consensus/"
- #define **CONSENSUS_FLAVORED_PREFIX** "/tor/status-vote/current/consensus-"

Functions

- static void **directory_send_command** (**dir_connection_t** *conn, int purpose, int direct, const char *resource, const char *payload, size_t payload_len, int supports_conditional_consensus, time_t if_modified_since)
- static int **directory_handle_command** (**dir_connection_t** *conn)
- static int **body_is_plausible** (const char *body, size_t body_len, int purpose)
- static int **purpose_needs_anonymity** (uint8_t dir_purpose, uint8_t router_purpose)
- static char * **http_get_header** (const char *headers, const char *which)
- static void **http_set_address_origin** (const char *headers, **connection_t** *conn)
- static void **connection_dir_download_networkstatus_failed** (**dir_connection_t** *conn, int status_code)
- static void **connection_dir_download_routerdesc_failed** (**dir_connection_t** *conn)
- static void **connection_dir_download_cert_failed** (**dir_connection_t** *conn, int status_code)
- static void **dir_networkstatus_download_failed** (**smartlist_t** *failed, int status_code)
- static void **dir_routerdesc_download_failed** (**smartlist_t** *failed, int status_code, int router_purpose, int was_extrainfo, int was_descriptor_digests)
- static void **note_client_request** (int purpose, int compressed, size_t bytes)
- static int **client_likes_consensus** (**networkstatus_t** *v, const char *want_url)
- static void **directory_initiate_command_rend** (const char *address, const **tor_addr_t** *addr, uint16_t or_port, uint16_t dir_port, int supports_conditional_consensus, int supports_begindir, const char *digest, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since, const **rend_data_t** *rend_query)

- char * **authority_type_to_string** (**authority_type_t** auth)
- static const char * **dir_conn_purpose_to_string** (int purpose)
- int **router_supports_extrainfo** (const char *identity_digest, int is_authority)
- int **directories_have_accepted_server_descriptor** (void)
- void **directory_post_to_dirservers** (uint8_t dir_purpose, uint8_t router_purpose, **authority_type_t** type, const char *payload, size_t payload_len, size_t extrainfo_len)
- void **directory_get_from_dirserver** (uint8_t dir_purpose, uint8_t router_purpose, const char *resource, int pds_flags)
- void **directory_get_from_all_authorities** (uint8_t dir_purpose, uint8_t router_purpose, const char *resource)
- void **directory_initiate_command_routerstatus_rend** (**routerstatus_t** *status, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since, const **rend_data_t** *rend_query)
- void **directory_initiate_command_routerstatus** (**routerstatus_t** *status, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since)
- static int **directory_conn_is_self_reachability_test** (**dir_connection_t** *conn)
- void **connection_dir_request_failed** (**dir_connection_t** *conn)
- static int **directory_command_should_use_begindir** (**or_options_t** *options, const **tor_addr_t** *addr, int or_port, uint8_t router_purpose, int anonymized_connection)
- void **directory_initiate_command** (const char *address, const **tor_addr_t** *_addr, uint16_t or_port, uint16_t dir_port, int supports_conditional_consensus, int supports_begindir, const char *digest, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since)
- int **connection_dir_is_encrypted** (**dir_connection_t** *conn)
- static int **_compare_strs** (const void **a, const void **b)
- static char * **directory_get_consensus_url** (int supports_conditional_consensus)
- static int **parse_http_url** (const char *headers, char **url)
- int **parse_http_response** (const char *headers, int *code, time_t *date, **compress_method_t** *compression, char **reason)
- static int **load_downloaded_routers** (const char *body, **smartlist_t** *which, int descriptor_digests, int router_purpose, const char *source)
- static int **connection_dir_client_reached_eof** (**dir_connection_t** *conn)
- int **connection_dir_reached_eof** (**dir_connection_t** *conn)
- int **connection_dir_process_inbuf** (**dir_connection_t** *conn)
- static void **write_http_status_line** (**dir_connection_t** *conn, int status, const char *reason_phrase)
- static void **write_http_response_header_impl** (**dir_connection_t** *conn, ssize_t length, const char *type, const char *encoding, const char *extra_headers, long cache_lifetime)
- static void **write_http_response_header** (**dir_connection_t** *conn, ssize_t length, int compressed, long cache_lifetime)
- void **note_request** (const char *key, size_t bytes)
- char * **directory_dump_request_log** (void)
- static int **directory_handle_command_get** (**dir_connection_t** *conn, const char *headers, const char *body, size_t body_len)
- static int **directory_handle_command_post** (**dir_connection_t** *conn, const char *headers, const char *body, size_t body_len)

- int **connection_dir_finished_flushing** (**dir_connection_t** *conn)
- int **connection_dir_finished_connecting** (**dir_connection_t** *conn)
- static void **find_dl_schedule_and_len** (**download_status_t** *dls, int server, const int **schedule, size_t *schedule_len)
- time_t **download_status_increment_failure** (**download_status_t** *dls, int status_code, const char *item, int server, time_t now)
- void **download_status_reset** (**download_status_t** *dls)
- static int **_compare_pairs** (const void **a, const void **b)
- int **dir_split_resource_into_fingerprint_pairs** (const char *res, **smartlist_t** *pairs_out)
- int **dir_split_resource_into_fingerprints** (const char *resource, **smartlist_t** *fp_out, int *compressed_out, int flags)

Variables

- static const int **server_dl_schedule** []
- static const int **client_dl_schedule** []
- static const int **server_consensus_dl_schedule** []
- static const int **client_consensus_dl_schedule** []
- static const int **bridge_dl_schedule** []

4.21.1 Detailed Description

Code to send and fetch directories and router descriptors via HTTP. Directories use **dirserv.c** (p. 583) to generate the results; clients use routers.c to parse them.

4.21.2 Define Documentation

4.21.2.1 #define ALLOW_DIRECTORY_TIME_SKEW (30*60)

How far in the future do we allow a directory server to tell us it is before deciding that one of us has the wrong time?

Referenced by connection_dir_client_reached_eof().

4.21.2.2 #define FULL_DIR_CACHE_LIFETIME (60*60)

HTTP cache control: how long do we tell proxies they can cache each kind of document we serve?

Referenced by directory_handle_command_get().

4.21.2.3 #define MAX_DIRECTORY_OBJECT_SIZE (10*(1<<20))

If any directory object is arriving, and it's over 10MB large, we're getting DoS'd. (As of 0.1.2.x, raw directories are about 1MB, and we never ask for more than 96 router descriptors at a time.)

Referenced by connection_dir_process_inbuf().

4.21.3 Function Documentation

4.21.3.1 static int _compare_pairs (const void ** *a*, const void ** *b*) `[static]`

Helper. Compare two **fp_pair_t** (p. 100) objects, and return -1, 0, or 1 as appropriate.

References DIGEST_LEN, fp_pair_t::first, and fp_pair_t::second.

Referenced by dir_split_resource_into_fingerprint_pairs().

4.21.3.2 static int _compare_strs (const void ** *a*, const void ** *b*) `[static]`

Helper for sorting

sort strings alphabetically

Referenced by directory_get_consensus_url().

4.21.3.3 char* authority_type_to_string (authority_type_t *auth*)

Return a newly allocated string describing **auth**.

References BRIDGE_AUTHORITY, HIDSERV_AUTHORITY, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), V1_AUTHORITY, V2_AUTHORITY, and V3_AUTHORITY.

Referenced by directory_post_to_dirservers().

4.21.3.4 static int body_is_plausible (const char * *body*, size_t *len*, int *purpose*) `[static]`

Return true iff **body** doesn't start with a plausible router or running-list or directory opening. This is a sign of possible compression.

References DIR_PURPOSE_FETCH_RENDDESC, and strcmpstart().

Referenced by connection_dir_client_reached_eof().

4.21.3.5 int client_likes_consensus (networkstatus_t * *v*, const char * *want_url*) `[static]`

Decide whether a client would accept the consensus we have.

Clients can say they only want a consensus if it's signed by more than half the authorities in a list. They pass this list in the url as "...consensus/fpr+fpr+fpr".

fpr may be an abbreviated fingerprint, i.e. only a left substring of the full authority identity digest. (Only strings of even length, i.e. encodings of full bytes, are handled correctly. In the case of an odd number of hex digits the last one is silently ignored.)

Returns 1 if more than half of the requested authorities signed the consensus, 0 otherwise.

References base16_decode(), DIGEST_LEN, dir_split_resource_into_fingerprints(), LD_DIR, smartlist_create(), smartlist_free(), tor_free, and networkstatus_t::voters.

Referenced by directory_handle_command_get().

4.21.3.6 static int connection_dir_client_reached_eof (dir_connection_t * *conn*) [static]

We are a client, and we've finished reading the server's response. Parse it and act appropriately.

If we're still happy with using this directory server in the future, return 0. Otherwise return -1; and the caller should consider trying the request again.

The caller will take care of marking the connection for close.

References dir_connection_t::_base, connection_t::address, ALLOW_DIRECTORY_TIME_SKEW, base16_encode(), body_is_plausible(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), connection_dir_download_routerdesc_failed(), control_event_general_status(), control_event_server_status(), detect_compression_method(), DIGEST_LEN, dir_networkstatus_download_failed(), DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_RENDDESC, DIR_PURPOSE_FETCH_RENDDESC_V2, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, DIR_PURPOSE_HAS_FETCHED_RENDDESC, DIR_PURPOSE_UPLOAD_DIR, DIR_PURPOSE_UPLOAD_RENDDESC, DIR_PURPOSE_UPLOAD_RENDDESC_V2, DIR_PURPOSE_UPLOAD_SIGNATURES, DIR_PURPOSE_UPLOAD_VOTE, dir_routerdesc_download_failed(), dir_split_resource_into_fingerprints(), dir_connection_t::dirconn_direct, directories_have_accepted_server_descriptor(), directory_conn_is_self_reachability_test(), directory_info_has_arrived(), dirvote_add_signatures(), dirvote_add_vote(), escaped(), trusted_dir_server_t::fake_status, fetch_from_buf_http(), format_time_interval(), get_options(), trusted_dir_server_t::has_accepted_serverdesc, HEX_DIGEST_LEN, http_get_header(), dir_connection_t::identity_digest, connection_t::inbuf, routerstatus_t::last_dir_503_at, LD_BUG, LD_DIR, LD_GENERAL, LD_HTTP, LD_PROTOCOL, LD_REND, load_downloaded_routers(), MAX_DIR_DL_SIZE, MAX_HEADERS_SIZE, networkstatus_consensus_download_failed(), networkstatus_set_current_consensus(), trusted_dir_server_t::nickname, rend_data_t::onion_address, parse_http_response(), connection_t::port, connection_t::purpose, rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_client_desc_trynow(), dir_connection_t::rend_data, dir_connection_t::requested_resource, router_dirport_found_reachable(), router_get_consensus_status_by_id(), router_get_trusted_dir_servers(), router_get_trusteddirserver_by_digest(), router_load_extrainfo_from_string(), router_new_address_suggestion(), dir_connection_t::router_purpose, router_set_networkstatus_v2(), routers_update_all_from_networkstatus(), SAVED_NOWHERE, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), connection_t::timestamp_lastwritten, tor_assert, tor_free, tor_gzip_uncompress(), and trusted_dirs_load_certs_from_string().

Referenced by connection_dir_reached_eof().

4.21.3.7 static void connection_dir_download_cert_failed (dir_connection_t * *conn*, int *status*) [static]

Called when an attempt to fetch a certificate fails.

References dir_connection_t::_base, authority_cert_dl_failed(), DIR_PURPOSE_FETCH_CERTIFICATE, dir_split_resource_into_fingerprints(), connection_t::purpose, dir_connection_t::requested_resource, smartlist_create(), smartlist_free(), tor_assert, tor_free, and update_certificate_downloads().

Referenced by connection_dir_client_reached_eof(), and connection_dir_request_failed().

4.21.3.8 static void connection_dir_download_networkstatus_failed (dir_connection_t * *conn*, int *status_code*) [static]

Called when an attempt to download one or more network status documents on connection **conn** failed. Decide whether to retry the fetch now, later, or never.

References dir_connection_t::_base, dir_networkstatus_download_failed(), dir_split_resource_into_fingerprints(), directory_get_from_dirserver(), download_status_failed, connection_t::purpose, dir_connection_t::requested_resource, router_get_trusted_dir_servers(), dir_connection_t::router_purpose, smartlist_create(), smartlist_free(), strcmpstart(), and tor_free.

Referenced by connection_dir_client_reached_eof(), and connection_dir_request_failed().

4.21.3.9 static void connection_dir_download_routerdesc_failed (dir_connection_t * *conn*) [static]

Called when an attempt to download one or more router descriptors or extra-info documents on connection **conn** failed.

References dir_connection_t::_base, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_SERVERDESC, connection_t::purpose, and tor_assert.

Referenced by connection_dir_client_reached_eof(), and connection_dir_request_failed().

4.21.3.10 int connection_dir_finished_connecting (dir_connection_t * *conn*)

Connected handler for directory connections: begin sending data to the server

References dir_connection_t::_base, connection_t::address, CONN_TYPE_DIR, DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_CONNECTING, LD_HTTP, connection_t::port, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_finished_connecting().

4.21.3.11 int connection_dir_finished_flushing (dir_connection_t * *conn*)

Write handler for directory connections; called when all data has been flushed. Close the connection or wait for a response as appropriate.

References dir_connection_t::_base, CONN_TYPE_DIR, connection_stop_writing(), DIR_CONN_STATE_CLIENT_READING, DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_SERVER_WRITING, DIRREQ_FLUSHING_DIR_CONN_FINISHED, geoip_change_dirreq_state(), LD_BUG, LD_DIR, LD_DIRSERV, connection_t::state, TO_CONN, tor_assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_finished_flushing().

4.21.3.12 int connection_dir_is_encrypted (dir_connection_t * *conn*)

Return true iff anything we say on **conn** is being encrypted before we send it to the client/server.

References TO_CONN.

Referenced by connection_dirserv_add_servers_to_outbuf(), and directory_handle_command_get().

4.21.3.13 int connection_dir_process_inbuf (dir_connection_t * *conn*)

Read handler for directory connections. (That's connections *to* directory servers and connections *at* directory servers.)

References dir_connection_t::_base, buf_datalen(), CONN_TYPE_DIR, DIR_CONN_STATE_SERVER_COMMAND_WAIT, directory_handle_command(), connection_t::inbuf, connection_t::inbuf_reached_eof, LD_HTTP, MAX_DIRECTORY_OBJECT_SIZE, connection_t::state, TO_CONN, tor_assert, and connection_t::type.

Referenced by connection_process_inbuf().

4.21.3.14 int connection_dir_reached_eof (dir_connection_t * *conn*)

Called when a directory connection reaches EOF.

References dir_connection_t::_base, connection_close_immediate(), connection_dir_client_reached_eof(), DIR_CONN_STATE_CLIENT_FINISHED, DIR_CONN_STATE_CLIENT_READING, LD_HTTP, connection_t::state, and TO_CONN.

Referenced by connection_reached_eof(), and run_connection_housekeeping().

4.21.3.15 void connection_dir_request_failed (dir_connection_t * *conn*)

Called when we are unable to complete the client's request to a directory server due to a network error: Mark the router as down and try again if possible.

References dir_connection_t::_base, connection_t::address, connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), connection_dir_download_routerdesc_failed(), DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, directory_conn_is_self_reachability_test(), entry_list_is_constrained(), get_options(), dir_connection_t::identity_digest, LD_DIR, networkstatus_consensus_download_failed(), connection_t::purpose, and router_set_status().

Referenced by connection_about_to_close_connection(), and directory_initiate_command_rend().

4.21.3.16 static const char* dir_conn_purpose_to_string (int *purpose*) `[static]`

Return a string describing a given directory connection purpose.

References DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_RENDDESC, DIR_PURPOSE_FETCH_RENDDESC_V2, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, DIR_PURPOSE_UPLOAD_DIR, DIR_PURPOSE_UPLOAD_RENDDESC, DIR_PURPOSE_UPLOAD_RENDDESC_V2, DIR_PURPOSE_UPLOAD_SIGNATURES, DIR_PURPOSE_UPLOAD_VOTE, and LD_BUG.

Referenced by directory_get_from_dirserver(), and directory_initiate_command_rend().

4.21.3.17 static void dir_networkstatus_download_failed (smartlist_t * *failed*, int *status_code*) [static]

Called when one or more networkstatus fetches have failed (with uppercase fingerprints listed in **failed**). Mark those fingerprints as having failed once, unless they failed with status code 503.

References base16_decode(), DIGEST_LEN, download_status_failed, escaped(), LD_BUG, router_get_trusteddirserver_by_digest(), and trusted_dir_server_t::v2_ns_dl_status.

Referenced by connection_dir_client_reached_eof(), and connection_dir_download_networkstatus_failed().

4.21.3.18 static void dir_routerdesc_download_failed (smartlist_t * *failed*, int *status_code*, int *router_purpose*, int *was_extrainfo*, int *was_descriptor_digests*) [static]

Called when one or more routerdesc (or extrainfo, if **was_extrainfo**) fetches have failed (with uppercase fingerprints listed in **failed**, either as descriptor digests or as identity digests based on **was_descriptor_digests**).

References base16_decode(), DIGEST_LEN, directory_fetches_from_authorities(), download_status_increment_failure(), signed_descriptor_t::ei_dl_status, escaped(), get_options(), LD_BUG, MAX_ROUTERDESC_DOWNLOAD_FAILURES, download_status_t::n_download_failures, retry_bridge_descriptor_fetch_directly(), router_get_by_extrainfo_digest(), router_get_dl_status_by_descriptor_digest(), ROUTER_PURPOSE_BRIDGE, and tor_assert.

Referenced by connection_dir_client_reached_eof().

4.21.3.19 int dir_split_resource_into_fingerprint_pairs (const char * *res*, smartlist_t * *pairs_out*)

Divide a string **res** of the form FP1-FP2+FP3-FP4...[.z], where each FP is a hex-encoded fingerprint, into a sequence of distinct sorted **fp_pair_t** (p. 100). Skip malformed pairs. On success, return 0 and add those **fp_pair_t** (p. 100) into **pairs_out**. On failure, return -1.

References _compare_pairs(), _tor_free(), base16_decode(), DIGEST_LEN, escaped(), fp_pair_t::first, HEX_DIGEST_LEN, LD_DIR, fp_pair_t::second, smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_sort(), smartlist_split_string(), smartlist_uniq(), and tor_free.

Referenced by directory_handle_command_get().

4.21.3.20 int dir_split_resource_into_fingerprints (const char * *resource*, smartlist_t * *fp_out*, int * *compressed_out*, int *flags*)

Given a directory **resource** request, containing zero or more strings separated by plus signs, followed optionally by ".z", store the strings, in order, into **fp_out**. If **compressed_out** is non-NULL, set it to 1 if the resource ends in ".z", else set it to 0.

If (flags & DSR_HEX), then delete all elements that aren't hex digests, and decode the rest. If (flags & DSR_BASE64), then use "-" rather than "+" as a separator, delete all the elements that aren't base64-encoded digests, and decode the rest. If (flags & DSR_DIGEST256), these digests should be 256 bits long; else they should be 160.

If (flags & DSR_SORT_UNIQ), then sort the list and remove all duplicates.

References base16_decode(), base64_decode(), BASE64_DIGEST256_LEN, BASE64_DIGEST_LEN, DIGEST256_LEN, DIGEST_LEN, escaped(), HEX_DIGEST256_LEN, HEX_DIGEST_LEN, LD_DIR, smartlist_add_all(), smartlist_create(), smartlist_del_keeporder(), smartlist_free(), smartlist_sort_digests(), smartlist_sort_digests256(), smartlist_sort_strings(), smartlist_split_string(), smartlist_uniq_digests(), smartlist_uniq_digests256(), smartlist_uniq_strings(), tor_assert, and tor_free.

Referenced by client_likes_consensus(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), directory_handle_command_get(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), and list_pending_downloads().

4.21.3.21 int directories_have_accepted_server_descriptor (void)

Return true iff any trusted directory authority has accepted our server descriptor.

We consider any authority sufficient because waiting for all of them means it never happens while any authority is down; we don't go for something more complex in the middle (like >1/3 or >1/2 or >=1/2) because that doesn't seem necessary yet.

References or_options_t::_PublishServerDescriptor, get_options(), and router_get_trusted_dir_servers().

Referenced by connection_dir_client_reached_eof(), and getinfo_helper_events().

4.21.3.22 static int directory_command_should_use_begindir (or_options_t * *options*, const tor_addr_t * *addr*, int *or_port*, uint8_t *router_purpose*, int *anonymized_connection*) `[static]`

Evaluate the situation and decide if we should use an encrypted "begindir-style" connection for this directory request. 1) If or_port is 0, or it's a direct conn and or_port is firewalled or we're a dir mirror, no. 2) If we prefer to avoid begindir conns, and we're not fetching or publishing a bridge relay descriptor, no. 3) Else yes.

References or_options_t::Address, directory_fetches_from_authorities(), fascist_firewall_allows_address_or(), ROUTER_PURPOSE_BRIDGE, server_mode(), and or_options_t::TunnelDirConns.

Referenced by directory_initiate_command_rend().

4.21.3.23 static int directory_conn_is_self_reachability_test (dir_connection_t * *conn*) `[static]`

Return true iff **conn** is the client side of a directory connection we launched to ourself in order to determine the reachability of our dir_port.

References dir_connection_t::_base, routerinfo_t::addr, connection_t::addr, routerinfo_t::dir_port, dir_connection_t::identity_digest, connection_t::port, dir_connection_t::requested_resource, router_digest_is_me(), router_get_my_routerinfo(), strcmpstart(), and tor_addr_eq_ipv4h().

Referenced by connection_dir_client_reached_eof(), and connection_dir_request_failed().

4.21.3.24 static char* directory_get_consensus_url (int *supports_conditional_consensus*) [static]

Return the URL we should use for a consensus download.

This url depends on whether or not the server we go to is sufficiently new to support conditional consensus downloading, i.e. GET .../consensus/**fpr**+**fpr**+**fpr**

References _compare_strs(), base16_encode(), router_get_trusted_dir_servers(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), tor_free, tor_snprintf(), and V3_AUTHORITY.

Referenced by directory_send_command().

4.21.3.25 void directory_get_from_all_authorities (uint8_t *dir_purpose*, uint8_t *router_purpose*, const char * *resource*)

As directory_get_from_dirserver, but initiates a request to *every* directory authority other than ourself. Only for use by authorities when searching for missing information while voting.

References DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_STATUS_VOTE, directory_initiate_command_routerstatus(), router_digest_is_me(), router_get_trusted_dir_servers(), tor_assert, and V3_AUTHORITY.

Referenced by dirvote_fetch_missing_signatures().

4.21.3.26 void directory_get_from_dirserver (uint8_t *dir_purpose*, uint8_t *router_purpose*, const char * *resource*, int *pds_flags*)

Start a connection to a random running directory server, using connection purpose **dir_purpose**, intending to fetch descriptors of purpose **router_purpose**, and requesting **resource**. Use **pds_flags** as arguments to **router_pick_directory_server()** (p. 1001) or **router_pick_trusteddirserver()** (p. 1001).

References routerinfo_t::addr, routerinfo_t::address, BRIDGE_AUTHORITY, routerinfo_t::cache_info, choose_random_entry(), dir_conn_purpose_to_string(), DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_RENDDESC, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, directory_all_unreachable(), directory_fetches_from_authorities(), directory_initiate_command(), directory_initiate_command_routerstatus(), EXTRAINFO_CACHE, or_options_t::FetchServerDescriptors, get_options(), HIDSERV_AUTHORITY, signed_descriptor_t::identity_digest, LD_BUG, LD_DIR, networkstatus_get_latest_consensus(), routerinfo_t::or_port, PDS_ALLOW_SELF, PDS_IGNORE_FASCISTFIREWALL, PDS_NO_EXISTING_SERVERDESC_FETCH, purpose_needs_anonymity(), router_pick_directory_server(), router_pick_trusteddirserver(), ROUTER_PURPOSE_BRIDGE, tor_addr_from_ipv4h, or_options_t::UseBridges, V2_AUTHORITY, V3_AUTHORITY, and networkstatus_t::valid_after.

Referenced by connection_dir_download_networkstatus_failed(), fetch_bridge_descriptors(), initiate_descriptor_downloads(), update_consensus_networkstatus_downloads(), update_router_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.21.3.27 static int directory_handle_command (dir_connection_t ∗ *conn*) [static]

Called when a dirserver receives data on a directory connection; looks for an HTTP request. If the request is complete, remove it from the inbuf, try to process it; otherwise, leave it on the buffer. Return a 0 on success, or -1 on error.

References dir_connection_t::_base, connection_t::address, CONN_TYPE_DIR, directory_handle_command_get(), directory_handle_command_post(), escaped(), fetch_from_buf_http(), http_set_address_origin(), connection_t::inbuf, LD_DIRSERV, LD_PROTOCOL, MAX_DIR_UL_SIZE, MAX_HEADERS_SIZE, safe_str(), TO_CONN, tor_assert, tor_free, and connection_t::type.

Referenced by connection_dir_process_inbuf().

4.21.3.28 static int directory_handle_command_get (dir_connection_t ∗ *conn*, const char ∗ *headers*, const char ∗ *body*, size_t *body_len*) [static]

Helper function: called when a dirserver gets a complete HTTP GET request. Look for a request for a directory or for a rendezvous service descriptor. On finding one, write a response into conn->outbuf. If the request is unrecognized, send a 400. Always return 0.

References dir_connection_t::_base, connection_t::addr, alloc_http_authenticator(), authdir_mode_tests_reachability(), authority_cert_get_all(), authority_cert_get_by_digests(), authority_cert_get_by_sk_digest(), authority_cert_get_newest_by_id(), or_options_t::BridgeAuthoritativeDir, or_options_t::BridgePassword, or_options_t::BridgeRelay, dir_connection_t::cached_dir, dir_connection_t::cached_dir_offset, client_likes_consensus(), connection_dir_is_encrypted(), connection_dirserv_flushed_some(), DIGEST_LEN, cached_dir_t::dir, DIR_CONN_STATE_SERVER_WRITING, cached_dir_t::dir_len, dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), dir_connection_t::dir_spool_src, cached_dir_t::dir_z, cached_dir_t::dir_z_len, dirserv_estimate_data_size(), dirserv_estimate_microdesc_size(), dirserv_get_directory(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_runningrouters(), dirserv_have_any_microdesc(), dirserv_have_any_serverdesc(), dirserv_remove_old_statuses(), dirvote_get_pending_consensus(), dirvote_get_pending_detached_signatures(), dirvote_get_vote(), dir_connection_t::fingerprint_stack, networkstatus_t::fresh_until, FULL_DIR_CACHE_LIFETIME, GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, geoip_note_client_seen(), geoip_note_ns_response(), GEOIP_REJECT_BUSY, GEOIP_REJECT_NOT_ENOUGH_SIGS, GEOIP_REJECT_NOT_FOUND, GEOIP_REJECT_NOT_MODIFIED, GEOIP_REJECT_UNAVAILABLE, geoip_start_dirreq(), GEOIP_SUCCESS, get_dirportfrontpage(), get_my_v3_authority_cert(), get_options(), global_write_bucket_low(), or_options_t::HidServDirectoryV2, or_options_t::HSAuthoritativeDir, http_get_header(), LD_DIRSERV, LD_REND, networkstatus_get_latest_consensus(), networkstatus_getinfo_by_purpose(), parse_http_time(), parse_http_url(), cached_dir_t::published, cached_dir_t::refcnt, rend_cache_lookup_desc(), rend_cache_lookup_v2_desc_as_dir(), REND_DESC_ID_V2_LEN_BASE32, rep_hist_get_router_stability_doc(), safe_str(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), connection_t::state, strcmpstart(), TO_CONN, tor_addr_eq_ipv4h(), tor_assert, tor_free, tor_inet_aton(), tor_timegm(), tor_zlib_new(), write_http_response_header(), write_http_response_header_impl(), write_http_status_line(), and dir_connection_t::zlib_state.

Referenced by directory_handle_command().

4.21.3.29 static int directory_handle_command_post (dir_connection_t ∗ *conn*, const char ∗ *headers*, const char ∗ *body*, size_t *body_len*) [static]

Helper function: called when a dirserver gets a complete HTTP POST request. Look for an uploaded server descriptor or rendezvous service descriptor. On finding one, process it and write a response into conn->outbuf. If the request is unrecognized, send a 400. Always return 0.

References dir_connection_t::_base, connection_t::address, authdir_mode(), authdir_mode_bridge(), authdir_mode_handles_descs(), authdir_mode_v3(), DIR_CONN_STATE_SERVER_WRITING, dirserv_add_multiple_descriptors(), dirserv_get_directory(), dirvote_add_signatures(), dirvote_add_vote(), get_options(), or_options_t::HidServDirectoryV2, or_options_t::HSAuthoritativeDir, LD_DIR, LD_DIRSERV, LD_REND, parse_http_url(), rend_cache_store(), rend_cache_store_v2_desc_as_dir(), ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_GENERAL, connection_t::state, strcmpstart(), tor_assert, tor_free, WRA_WAS_ADDED(), WRA_WAS_OUTDATED(), write_http_response_header_impl(), and write_http_status_line().

Referenced by directory_handle_command().

4.21.3.30 void directory_initiate_command (const char ∗ *address*, const tor_addr_t ∗ *_addr*, uint16_t *or_port*, uint16_t *dir_port*, int *supports_conditional_consensus*, int *supports_begindir*, const char ∗ *digest*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char ∗ *resource*, const char ∗ *payload*, size_t *payload_len*, time_t *if_modified_since*)

Helper for directory_initiate_command_routerstatus: send the command to a server whose address is **address**, whose IP is **addr**, whose directory port is **dir_port**, whose tor version **supports_begindir**, and whose identity key digest is **digest**.

References directory_initiate_command_rend().

Referenced by consider_testing_reachability(), directory_get_from_dirserver(), and launch_direct_bridge_descriptor_fetch().

4.21.3.31 static void directory_initiate_command_rend (const char ∗ *address*, const tor_addr_t ∗ *_addr*, uint16_t *or_port*, uint16_t *dir_port*, int *supports_conditional_consensus*, int *supports_begindir*, const char ∗ *digest*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char ∗ *resource*, const char ∗ *payload*, size_t *payload_len*, time_t *if_modified_since*, const rend_data_t ∗ *rend_query*) [static]

Same as **directory_initiate_command()** (p. 576), but accepts rendezvous data to fetch a hidden service descriptor.

References dir_connection_t::_base, connection_t::addr, connection_t::address, connection_add(), connection_ap_make_link(), connection_connect(), connection_dir_request_failed(), connection_free(), connection_link_connections(), connection_start_reading(), connection_watch_events(), DIGEST_LEN, dir_conn_purpose_to_string(), DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_CONNECTING, dir_connection_new(), dir_connection_t::dirconn_direct, directory_command_should_use_begindir(), directory_send_command(), get_options(), or_options_t::HttpProxy, or_options_t::HttpProxyAddr, or_options_t::HttpProxyPort, dir_connection_t::identity_digest, LD_DIR, LD_NET, connection_t::port, connection_t::purpose, dir_connection_t::rend_data, rend_data_dup(), rep_hist_note_used_internal(), rep_hist_note_used_port(), dir_connection_t::router_purpose, or_options_t::Socks4Proxy, or_options_t::Socks5Proxy, connection_t::state, TO_CONN, tor_addr_copy(), and tor_assert.

Referenced by directory_initiate_command(), and directory_initiate_command_routerstatus_rend().

4.21.3.32 void directory_initiate_command_routerstatus (routerstatus_t * *status*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char * *resource*, const char * *payload*, size_t *payload_len*, time_t *if_modified_since*)

Launch a new connection to the directory server **status** to upload or download a server or rendezvous descriptor. **dir_purpose** determines what kind of directory connection we're launching, and must be one of DIR_PURPOSE_{FETCH|UPLOAD}_{DIR|RENDDESC|RENDDESC_V2}. **router_purpose** specifies the descriptor purposes we have in mind (currently only used for FETCH_DIR).

When uploading, **payload** and **payload_len** determine the content of the HTTP post. Otherwise, **payload** should be NULL.

When fetching a rendezvous descriptor, **resource** is the service ID we want to fetch.

References directory_initiate_command_routerstatus_rend().

Referenced by directory_get_from_all_authorities(), directory_get_from_dirserver(), directory_post_to_dirservers(), directory_post_to_hs_dir(), initiate_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.21.3.33 void directory_initiate_command_routerstatus_rend (routerstatus_t * *status*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char * *resource*, const char * *payload*, size_t *payload_len*, time_t *if_modified_since*, const rend_data_t * *rend_query*)

Same as **directory_initiate_command_routerstatus()** (p. 577), but accepts rendezvous data to fetch a hidden service descriptor.

References routerstatus_t::addr, routerinfo_t::address, routerstatus_t::dir_port, directory_initiate_command_rend(), routerstatus_t::identity_digest, INET_NTOA_BUF_LEN, LD_DIR, routerstatus_t::nickname, routerstatus_t::or_port, router_get_by_digest(), tor_addr_from_ipv4h, tor_inet_ntoa(), routerstatus_t::version_supports_begindir, and routerstatus_t::version_supports_conditional_consensus.

Referenced by directory_get_from_hs_dir(), and directory_initiate_command_routerstatus().

4.21.3.34 void directory_post_to_dirservers (uint8_t *dir_purpose*, uint8_t *router_purpose*, authority_type_t *type*, const char * *payload*, size_t *payload_len*, size_t *extrainfo_len*)

Start a connection to every suitable directory authority, using connection purpose 'purpose' and uploading the payload 'payload' (length 'payload_len'). dir_purpose should be one of 'DIR_PURPOSE_UPLOAD_DIR' or 'DIR_PURPOSE_UPLOAD_RENDDESC'.

type specifies what sort of dir authorities (V1, V2, HIDSERV, BRIDGE) we should upload to.

If **extrainfo_len** is nonzero, the first **payload_len** bytes of **payload** hold a router descriptor, and the next **extrainfo_len** bytes of **payload** hold an extra-info document. Upload the descriptor to all authorities, and the extra-info document to all authorities that support it.

References tor_addr_t::addr, authority_type_to_string(), DIR_PURPOSE_UPLOAD_DIR, directory_initiate_command_routerstatus(), fascist_firewall_allows_address_dir(), LD_DIR, purpose_needs_anonymity(), router_get_trusted_dir_servers(), router_supports_extrainfo(), tor_addr_from_ipv4h, tor_assert, and tor_free.

Referenced by dirvote_compute_consensuses(), dirvote_perform_vote(), and router_upload_dir_desc_to_dirservers().

4.21.3.35 static void directory_send_command (dir_connection_t * *conn*, int *purpose*, int *direct*, const char * *resource*, const char * *payload*, size_t *payload_len*, int *supports_conditional_consensus*, time_t *if_modified_since*) `[static]`

Queue an appropriate HTTP command on conn->outbuf. The other args are as in **directory_initiate_command()** (p. 576).

References dir_connection_t::_base, connection_t::address, alloc_http_authenticator(), CONN_TYPE_DIR, DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_RENDDESC_V2, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, DIR_PURPOSE_UPLOAD_DIR, DIR_PURPOSE_UPLOAD_RENDDESC, DIR_PURPOSE_UPLOAD_RENDDESC_V2, DIR_PURPOSE_UPLOAD_SIGNATURES, DIR_PURPOSE_UPLOAD_VOTE, directory_get_consensus_url(), format_rfc1123_time(), get_options(), or_options_t::HttpProxyAuthenticator, LD_BUG, LD_DIR, connection_t::port, REND_DESC_ID_V2_LEN_BASE32, dir_connection_t::requested_resource, TO_CONN, tor_assert, tor_free, tor_snprintf(), and connection_t::type.

Referenced by directory_initiate_command_rend().

4.21.3.36 time_t download_status_increment_failure (download_status_t * *dls*, int *status_code*, const char * *item*, int *server*, time_t *now*)

Called when an attempt to download **dls** has failed with HTTP status **status_code**. Increment the failure count (if the code indicates a real failure) and set **dls**->next_attempt_at to an appropriate time in the future.

References find_dl_schedule_and_len(), IMPOSSIBLE_TO_DOWNLOAD, LD_DIR, download_status_t::n_download_failures, download_status_t::next_attempt_at, and tor_assert.

Referenced by dir_routerdesc_download_failed().

4.21.3.37 void download_status_reset (download_status_t * *dls*)

Reset **dls** so that it will be considered downloadable immediately, and/or to show that we don't need it anymore.

(We find the zeroth element of the download schedule, and set next_attempt_at to be the appropriate offset from 'now'. In most cases this means setting it to 'now', so the item will be immediately downloadable; in the case of bridge descriptors, the zeroth element is an hour from now.)

References find_dl_schedule_and_len(), get_options(), download_status_t::n_download_failures, and download_status_t::next_attempt_at.

Referenced by authority_certs_fetch_missing(), learned_bridge_descriptor(), networkstatus_set_current_consensus(), router_reset_descriptor_download_failures(), and router_set_networkstatus_v2().

4.21.3.38 static void find_dl_schedule_and_len (download_status_t ∗ *dls*, int *server*, const int ∗∗ *schedule*, size_t ∗ *schedule_len*) `[static]`

Decide which download schedule we want to use, and then return a pointer to it along with a pointer to its length. Helper function for **download_status_increment_failure()** (p. 578) and **download_status_reset()** (p. 578).

References download_status_t::schedule, and tor_assert.

Referenced by download_status_increment_failure(), and download_status_reset().

4.21.3.39 static char ∗ http_get_header (const char ∗ *headers*, const char ∗ *which*) `[static]`

Return a copy of the first HTTP header in **headers** whose key is **which**. The key should be given with a terminating colon and space; this function copies everything after, up to but not including the following \r\n.

References strcasecmpstart().

Referenced by connection_dir_client_reached_eof(), directory_handle_command_get(), and http_set_address_origin().

4.21.3.40 static void http_set_address_origin (const char ∗ *headers*, connection_t ∗ *conn*) `[static]`

If **headers** indicates that a proxy was involved, then rewrite **conn**->address to describe our best guess of the address that originated this HTTP request.

References connection_t::address, escaped(), http_get_header(), is_internal_IP(), LD_DIR, tor_free, and tor_inet_aton().

Referenced by directory_handle_command().

4.21.3.41 static int load_downloaded_routers (const char ∗ *body*, smartlist_t ∗ *which*, int *descriptor_digests*, int *router_purpose*, const char ∗ *source*) `[static]`

Called when we've just fetched a bunch of router descriptors in **body**. The list **which**, if present, holds digests for descriptors we requested: descriptor digests if **descriptor_digests** is true, or identity digests otherwise. Parse the descriptors, validate them, and annotate them as having purpose **purpose** and as having been downloaded from **source**.

Return the number of routers actually added.

References control_event_bootstrap(), count_loading_descriptors_progress(), escaped(), format_iso_time(), router_load_routers_from_string(), ROUTER_PURPOSE_GENERAL, router_purpose_to_string(), SAVED_NOWHERE, tor_assert, and tor_snprintf().

Referenced by connection_dir_client_reached_eof().

4.21.3.42 int parse_http_response (const char ∗ *headers*, int ∗ *code*, time_t ∗ *date*, compress_method_t ∗ *compression*, char ∗∗ *reason*)

Parse an HTTP response string **headers** of the form

```
 * "HTTP/1.\%d \%d\%s\r\n...".
 *
```

If it's well-formed, assign the status code to ∗**code** and return 0. Otherwise, return -1.

On success: If **date** is provided, set ∗date to the Date header in the http headers, or 0 if no such header is found. If **compression** is provided, set ∗**compression** to the compression method given in the Content-Encoding header, or 0 if no such header is found, or -1 if the value of the header is not recognized. If **reason** is provided, strdup the reason string into it.

References escaped(), LD_HTTP, parse_rfc1123_time(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcmpstart(), tor_assert, tor_free, and tor_sscanf().

Referenced by connection_dir_client_reached_eof(), and connection_read_https_proxy_response().

4.21.3.43 static int parse_http_url (const char ∗ *headers*, char ∗∗ *url*) `[static]`

Parse an HTTP request string **headers** of the form

```
 * "\%s [http[s]://]\%s HTTP/1..."
 *
```

If it's well-formed, strdup the second %s into ∗**url**, and nul-terminate it. If the url doesn't start with "/tor/", rewrite it so it does. Return 0. Otherwise, return -1.

References eat_whitespace_no_nl(), find_whitespace(), LD_DIR, and strcmpstart().

Referenced by directory_handle_command_get(), and directory_handle_command_post().

4.21.3.44 static int purpose_needs_anonymity (uint8_t *dir_purpose*, uint8_t *router_purpose*) `[static]`

Return true iff the directory purpose **dir_purpose** (and if it's fetching descriptors, it's fetching them for **router_purpose**) must use an anonymous connection to a directory.

References DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, DIR_PURPOSE_UPLOAD_DIR, DIR_PURPOSE_UPLOAD_SIGNATURES, DIR_PURPOSE_UPLOAD_VOTE, get_options(), has_completed_circuit, and ROUTER_PURPOSE_BRIDGE.

Referenced by directory_get_from_dirserver(), and directory_post_to_dirservers().

4.21.3.45 int router_supports_extrainfo (const char * *identity_digest*, int *is_authority*)

Return true iff **identity_digest** is the digest of a router we believe to support extrainfo downloads. (If **is_authority** we do additional checking that's only valid for authorities.)

References routerinfo_t::caches_extra_info, routerinfo_t::platform, router_get_by_digest(), router_get_consensus_status_by_id(), tor_version_as_new_as(), and routerstatus_t::version_supports_extrainfo_upload.

Referenced by directory_post_to_dirservers(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.21.3.46 static void write_http_response_header (dir_connection_t * *conn*, ssize_t *length*, int *compressed*, long *cache_lifetime*) `[static]`

As write_http_response_header_impl, but sets encoding and content-typed based on whether the response will be **compressed** or not.

References write_http_response_header_impl().

Referenced by directory_handle_command_get().

4.21.3.47 static void write_http_response_header_impl (dir_connection_t * *conn*, ssize_t *length*, const char * *type*, const char * *encoding*, const char * *extra_headers*, long *cache_lifetime*) `[static]`

Write the header for an HTTP/1.0 response onto **conn**->outbuf, with **type** as the Content-Type.

If **length** is nonnegative, it is the Content-Length. If **encoding** is provided, it is the Content-Encoding. If **cache_lifetime** is greater than 0, the content may be cached for up to cache_lifetime seconds. Otherwise, the content may not be cached.

References dir_connection_t::_base, connection_t::addr, connection_t::address, format_rfc1123_time(), is_local_addr(), TO_CONN, tor_assert, and tor_snprintf().

Referenced by directory_handle_command_get(), directory_handle_command_post(), and write_http_response_header().

4.21.3.48 static void write_http_status_line (dir_connection_t * *conn*, int *status*, const char * *reason_phrase*) `[static]`

Create an http response for the client **conn** out of **status** and **reason_phrase**. Write it to **conn**.

References LD_BUG, TO_CONN, and tor_snprintf().

Referenced by directory_handle_command_get(), and directory_handle_command_post().

4.21.4 Variable Documentation

4.21.4.1 const int bridge_dl_schedule[] `[static]`

Initial value:

```
{
 60*60, 15*60, 15*60, 60*60
```

```
}
```

Schedule for when clients should download bridge descriptors.

4.21.4.2 const int client_consensus_dl_schedule[] [static]

Initial value:

```
 {
  0, 0, 60, 60*5, 60*10, 60*30, 60*60, 60*60, 60*60, 60*60*3, 60*60*6, 60*60*12
}
```

Schedule for when clients should download consensuses.

4.21.4.3 const int client_dl_schedule[] [static]

Initial value:

```
 {
  0, 0, 60, 60*5, 60*10, INT_MAX
}
```

Schedule for when clients should download things in general.

4.21.4.4 const int server_consensus_dl_schedule[] [static]

Initial value:

```
 {
  0, 0, 60, 60*5, 60*10, 60*30, 60*30, 60*30, 60*30, 60*30, 60*60, 60*60*2
}
```

Schedule for when servers should download consensuses.

4.21.4.5 const int server_dl_schedule[] [static]

Initial value:

```
 {
  0, 0, 0, 60, 60, 60*2, 60*5, 60*15, INT_MAX
}
```

Schedule for when servers should download things in general.

4.22 dirserv.c File Reference

Directory server core implementation. Manages directory contents and generates directories.

`#include "or.h"`

Data Structures

- struct **router_status_t**
- struct **authdir_config_t**

Defines

- #define **DIRSERV_PRIVATE**
- #define **ROUTER_ALLOW_SKEW** (60*60*12)
- #define **DIR_REGEN_SLACK_TIME** 30
- #define **MAX_UNTRUSTED_NETWORKSTATUSES** 16
- #define **MAX_V1_DIRECTORY_AGE** (30*24*60*60)
- #define **MAX_V1_RR_AGE** (7*24*60*60)
- #define **MAX_MEASUREMENT_AGE** (3*24*60*60)
- #define **FP_NAMED** 1
- #define **FP_INVALID** 2
- #define **FP_REJECT** 4
- #define **FP_BADDIR** 8
- #define **FP_BADEXIT** 16
- #define **FP_UNNAMED** 32
- #define **STUB_REGENERATE_INTERVAL** (8*60*60)
- #define **REACHABLE_TIMEOUT** (45*60)
- #define **UPTIME_TO_GUARANTEE_STABLE** (3600*24*30)
- #define **MTBF_TO_GUARANTEE_STABLE** (60*60*24*5)
- #define **BANDWIDTH_TO_GUARANTEE_FAST** ROUTER_REQUIRED_MIN_BANDWIDTH
- #define **BANDWIDTH_TO_GUARANTEE_GUARD** (250*1024)
- #define **TIME_KNOWN_TO_GUARANTEE_FAMILIAR** (8*24*60*60)
- #define **WFU_TO_GUARANTEE_GUARD** (0.995)
- #define **V_LINE_OVERHEAD** 7
- #define **DIRSERV_BUFFER_MIN** 16384

Functions

- static void **directory_remove_invalid** (void)
- static **cached_dir_t** * **dirserv_regenerate_directory** (void)
- static char * **format_versions_list** (**config_line_t** *ln)
- static int **add_fingerprint_to_dir** (const char *nickname, const char *fp, struct **authdir_config_t** *list)
- static uint32_t **dirserv_router_get_status** (const **routerinfo_t** *router, const char **msg)
- static uint32_t **dirserv_get_status_impl** (const char *fp, const char *nickname, const char *address, uint32_t addr, uint16_t or_port, const char *platform, const char *contact, const char **msg, int should_log)

- static void **clear_cached_dir** (**cached_dir_t** *d)
- static **signed_descriptor_t** * **get_signed_descriptor_by_fp** (const char *fp, int extrainfo, time_t publish_cutoff)
- static int **dirserv_add_extrainfo** (**extrainfo_t** *ei, const char **msg)
- static **authdir_config_t** * **authdir_config_new** (void)
- int **dirserv_add_own_fingerprint** (const char *nickname, **crypto_pk_env_t** *pk)
- int **dirserv_load_fingerprint_file** (void)
- int **dirserv_would_reject_router** (**routerstatus_t** *rs)
- static uint32_t **dirserv_get_name_status** (const char *id_digest, const char *nickname)
- const char * **dirserv_get_nickname_by_digest** (const char *digest)
- void **dirserv_free_fingerprint_list** (void)
- static int **dirserv_router_has_valid_address** (**routerinfo_t** *ri)
- int **authdir_wants_to_reject_router** (**routerinfo_t** *ri, const char **msg, int complain)
- static int **WRA_MORE_SEVERE** (**was_router_added_t** a, **was_router_added_t** b)
- **was_router_added_t** **dirserv_add_multiple_descriptors** (const char *desc, uint8_t purpose, const char *source, const char **msg)
- **was_router_added_t** **dirserv_add_descriptor** (**routerinfo_t** *ri, const char **msg, const char *source)
- void **directory_set_dirty** (void)
- static char * **list_single_server_status** (**routerinfo_t** *desc, int is_live)
- static INLINE int **running_long_enough_to_decide_unreachable** (void)
- void **dirserv_set_router_is_running** (**routerinfo_t** *router, time_t now)
- int **list_server_status_v1** (**smartlist_t** *routers, char **router_status_out, int for_controller)
- static int **router_is_active** (**routerinfo_t** *ri, time_t now)
- int **dirserv_dump_directory_to_string** (char **dir_out, **crypto_pk_env_t** *private_key)
- int **directory_fetches_from_authorities** (**or_options_t** *options)
- int **directory_fetches_dir_info_early** (**or_options_t** *options)
- int **directory_fetches_dir_info_later** (**or_options_t** *options)
- int **directory_caches_v2_dir_info** (**or_options_t** *options)
- int **directory_caches_dir_info** (**or_options_t** *options)
- int **directory_permits_begindir_requests** (**or_options_t** *options)
- int **directory_permits_controller_requests** (**or_options_t** *options)
- int **directory_too_idle_to_fetch_descriptors** (**or_options_t** *options, time_t now)
- static void **set_cached_dir** (**cached_dir_t** *d, char *directory, time_t when)
- void **cached_dir_decref** (**cached_dir_t** *d)
- **cached_dir_t** * **new_cached_dir** (char *s, time_t published)
- static void **_free_cached_dir** (void *_d)
- void **dirserv_set_cached_directory** (const char *directory, time_t published, int is_running_routers)
- void **dirserv_set_cached_networkstatus_v2** (const char *networkstatus, const char *identity, time_t published)
- void **dirserv_set_cached_consensus_networkstatus** (const char *networkstatus, const char *flavor_name, const **digests_t** *digests, time_t published)
- void **dirserv_clear_old_networkstatuses** (time_t cutoff)
- void **dirserv_clear_old_v1_info** (time_t now)

- static **cached_dir_t** * **dirserv_pick_cached_dir_obj** (**cached_dir_t** *cache_src, **cached_dir_t** *auth_src, time_t dirty, **cached_dir_t** *(*regenerate)(void), const char *name, **authority_type_t** auth_type)
- **cached_dir_t** * **dirserv_get_directory** (void)
- static **cached_dir_t** * **generate_runningrouters** (void)
- **cached_dir_t** * **dirserv_get_runningrouters** (void)
- **cached_dir_t** * **dirserv_get_consensus** (const char *flavor_name)
- static int **should_generate_v2_networkstatus** (void)
- static INLINE long **real_uptime** (**routerinfo_t** *router, time_t now)
- static int **dirserv_thinks_router_is_unreliable** (time_t now, **routerinfo_t** *router, int need_uptime, int need_capacity)
- static int **dirserv_thinks_router_is_hs_dir** (**routerinfo_t** *router, time_t now)
- static void **dirserv_compute_performance_thresholds** (**routerlist_t** *rl)
- static char * **version_from_platform** (const char *platform)
- int **routerstatus_format_entry** (char *buf, size_t buf_len, **routerstatus_t** *rs, const char *version, routerstatus_format_type_t format)
- static int **_compare_routerinfo_by_ip_and_bw** (const void **a, const void **b)
- static digestmap_t * **get_possible_sybil_list** (const **smartlist_t** *routers)
- void **set_routerstatus_from_routerinfo** (**routerstatus_t** *rs, **routerinfo_t** *ri, time_t now, int naming, int exits_can_be_guards, int listbadexits, int listbaddirs)
- static void **clear_status_flags_on_sybil** (**routerstatus_t** *rs)
- void **router_clear_status_flags** (**routerinfo_t** *router)
- int **measured_bw_line_parse** (measured_bw_line_t *out, const char *orig_line)
- int **measured_bw_line_apply** (measured_bw_line_t *parsed_line, **smartlist_t** *routerstatuses)
- int **dirserv_read_measured_bandwidths** (const char *from_file, **smartlist_t** *routerstatuses)
- **networkstatus_t** * **dirserv_generate_networkstatus_vote_obj** (**crypto_pk_env_t** *private_key, **authority_cert_t** *cert)
- static **cached_dir_t** * **generate_v2_networkstatus_opinion** (void)
- void **dirserv_get_networkstatus_v2_fingerprints** (**smartlist_t** *result, const char *key)
- void **dirserv_get_networkstatus_v2** (**smartlist_t** *result, const char *key)
- int **dirserv_get_routerdesc_fingerprints** (**smartlist_t** *fps_out, const char *key, const char **msg, int for_unencrypted_conn, int is_extrainfo)
- int **dirserv_get_routerdescs** (**smartlist_t** *descs_out, const char *key, const char **msg)
- void **dirserv_orconn_tls_done** (const char *address, uint16_t or_port, const char *digest_rcvd, int as_advertised)
- void **dirserv_single_reachability_test** (time_t now, **routerinfo_t** *router)
- void **dirserv_test_reachability** (time_t now)
- static **cached_dir_t** * **lookup_cached_dir_by_fp** (const char *fp)
- int **dirserv_remove_old_statuses** (**smartlist_t** *fps, time_t cutoff)
- int **dirserv_have_any_serverdesc** (**smartlist_t** *fps, int spool_src)
- int **dirserv_have_any_microdesc** (const **smartlist_t** *fps)
- size_t **dirserv_estimate_data_size** (**smartlist_t** *fps, int is_serverdescs, int compressed)
- size_t **dirserv_estimate_microdesc_size** (const **smartlist_t** *fps, int compressed)
- static int **connection_dirserv_finish_spooling** (**dir_connection_t** *conn)
- static int **connection_dirserv_add_servers_to_outbuf** (**dir_connection_t** *conn)

- static int **connection_dirserv_add_microdescs_to_outbuf** (**dir_connection_t** ∗conn)
- static int **connection_dirserv_add_dir_bytes_to_outbuf** (**dir_connection_t** ∗conn)
- static int **connection_dirserv_add_networkstatus_bytes_to_outbuf** (**dir_connection_t** ∗conn)
- int **connection_dirserv_flushed_some** (**dir_connection_t** ∗conn)
- void **dirserv_free_all** (void)

Variables

- time_t **time_of_process_start**
- static time_t **the_directory_is_dirty** = 1
- static time_t **runningrouters_is_dirty** = 1
- static time_t **the_v2_networkstatus_is_dirty** = 1
- static **cached_dir_t** ∗ **the_directory** = NULL
- static **cached_dir_t** **the_runningrouters**
- static **authdir_config_t** ∗ **fingerprint_list** = NULL
- static **cached_dir_t** ∗ **cached_directory** = NULL
- static **cached_dir_t** **cached_runningrouters**
- static digestmap_t ∗ **cached_v2_networkstatus** = NULL
- static strmap_t ∗ **cached_consensuses** = NULL
- static **cached_dir_t** ∗ **the_v2_networkstatus** = NULL
- static uint32_t **stable_uptime** = 0
- static double **stable_mtbf** = 0.0
- static int **enough_mtbf_info** = 0
- static double **guard_wfu** = 0.0
- static long **guard_tk** = 0
- static uint32_t **fast_bandwidth** = 0
- static uint32_t **guard_bandwidth_including_exits** = 0
- static uint32_t **guard_bandwidth_excluding_exits** = 0
- static uint64_t **total_bandwidth** = 0
- static uint64_t **total_exit_bandwidth** = 0

4.22.1 Detailed Description

Directory server core implementation. Manages directory contents and generates directories.

4.22.2 Define Documentation

4.22.2.1 #define BANDWIDTH_TO_GUARANTEE_FAST ROUTER_REQUIRED_MIN_BANDWIDTH

Similarly, we protect sufficiently fast nodes from being pushed out of the set of Fast nodes.

4.22.2.2 #define BANDWIDTH_TO_GUARANTEE_GUARD (250∗1024)

Similarly, every node with sufficient bandwidth can be considered for Guard status.

Referenced by set_routerstatus_from_routerinfo().

4.22.2.3 #define DIR_REGEN_SLACK_TIME 30

How many seconds do we wait before regenerating the directory?

Referenced by dirserv_pick_cached_dir_obj(), and should_generate_v2_networkstatus().

4.22.2.4 #define DIRSERV_BUFFER_MIN 16384

When we're spooling data onto our outbuf, add more whenever we dip below this threshold.

Referenced by connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), and connection_dirserv_flushed_some().

4.22.2.5 #define FP_BADDIR 8

We'll tell clients to avoid using this as a dir.

Referenced by add_fingerprint_to_dir(), authdir_wants_to_reject_router(), directory_remove_invalid(), and dirserv_get_status_impl().

4.22.2.6 #define FP_BADEXIT 16

We'll tell clients not to use this as an exit.

Referenced by add_fingerprint_to_dir(), authdir_wants_to_reject_router(), directory_remove_invalid(), and dirserv_get_status_impl().

4.22.2.7 #define FP_INVALID 2

Believed invalid.

Referenced by add_fingerprint_to_dir(), authdir_wants_to_reject_router(), directory_remove_invalid(), and dirserv_get_status_impl().

4.22.2.8 #define FP_NAMED 1

Listed in fingerprint file.

Referenced by add_fingerprint_to_dir(), authdir_wants_to_reject_router(), directory_remove_invalid(), dirserv_get_name_status(), dirserv_get_status_impl(), and set_routerstatus_from_routerinfo().

4.22.2.9 #define FP_REJECT 4

We will not publish this router.

Referenced by add_fingerprint_to_dir(), authdir_wants_to_reject_router(), directory_remove_invalid(), dirserv_get_status_impl(), dirserv_router_get_status(), and dirserv_would_reject_router().

4.22.2.10 #define FP_UNNAMED 32

Another router has this name in fingerprint file.

Referenced by dirserv_get_name_status(), dirserv_get_status_impl(), and set_routerstatus_from_routerinfo().

4.22.2.11 #define MAX_UNTRUSTED_NETWORKSTATUSES 16

If we're a cache, keep this many networkstatuses around from non-trusted directory authorities.

Referenced by dirserv_set_cached_networkstatus_v2().

4.22.2.12 #define MAX_V1_DIRECTORY_AGE (30*24*60*60)

If a v1 directory is older than this, discard it.

Referenced by dirserv_clear_old_v1_info(), and dirserv_set_cached_directory().

4.22.2.13 #define MAX_V1_RR_AGE (7*24*60*60)

If a v1 running-routers is older than this, discard it.

Referenced by dirserv_clear_old_v1_info(), and dirserv_set_cached_directory().

4.22.2.14 #define MTBF_TO_GUARANTEE_STABLE (60*60*24*5)

If a router's MTBF is at least this value, then it is always stable. See above. (Corresponds to about 7 days for current decay rates.)

Referenced by dirserv_thinks_router_is_unreliable().

4.22.2.15 #define REACHABLE_TIMEOUT (45*60)

Each server needs to have passed a reachability test no more than this number of seconds ago, or he is listed as down in the directory.

Referenced by dirserv_set_router_is_running().

4.22.2.16 #define ROUTER_ALLOW_SKEW (60*60*12)

How far in the future do we allow a router to get? (seconds)

Referenced by authdir_wants_to_reject_router().

4.22.2.17 #define TIME_KNOWN_TO_GUARANTEE_FAMILIAR (8*24*60*60)

Similarly, every node with at least this much weighted time known can be considered familiar enough to be a guard. Corresponds to about 20 days for current decay rates.

4.22.2.18 #define UPTIME_TO_GUARANTEE_STABLE (3600*24*30)

If a router's uptime is at least this value, then it is always considered stable, regardless of the rest of the network. This way we resist attacks where an attacker doubles the size of the network using allegedly high-uptime nodes, displacing all the current guards.

Referenced by dirserv_thinks_router_is_unreliable().

4.22.2.19 #define WFU_TO_GUARANTEE_GUARD (0.995)

Similarly, every node with sufficient WFU is around enough to be a guard.

4.22.3 Function Documentation

4.22.3.1 static int _compare_routerinfo_by_ip_and_bw (const void ** *a*, const void ** *b*) `[static]`

Helper for sorting: compares two routerinfos first by address, and then by descending order of "usefulness". (An authority is more useful than a non-authority; a running router is more useful than a non-running router; and a router with more bandwidth is more useful than one with less.)

References routerinfo_t::addr, routerinfo_t::cache_info, DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::is_running, and router_get_advertised_bandwidth().

Referenced by get_possible_sybil_list().

4.22.3.2 static void _free_cached_dir (void * _ *d*) `[static]`

Free all storage held by the **cached_dir_t** (p. 23) in **d**.

References cached_dir_decref().

Referenced by dirserv_free_all().

4.22.3.3 int add_fingerprint_to_dir (const char * *nickname*, const char * *fp*, authdir_config_t * *list*) `[static]`

Add the fingerprint **fp** for **nickname** to the smartlist of fingerprint_entry_t's **list**. Return 0 if it's new, or 1 if we replaced the old value.

References base16_decode(), DIGEST_LEN, digestmap_get(), digestmap_set(), escaped(), FP_BADDIR, FP_BADEXIT, authdir_config_t::fp_by_name, FP_INVALID, FP_NAMED, FP_REJECT, LD_DIRSERV, router_status_t::nickname, router_status_t::status, authdir_config_t::status_by_digest, strmap_get_lc(), strmap_set_lc(), tor_assert, tor_free, and tor_strstrip().

Referenced by dirserv_add_own_fingerprint(), and dirserv_load_fingerprint_file().

4.22.3.4 static authdir_config_t* authdir_config_new (void) `[static]`

Allocate and return a new, empty, **authdir_config_t** (p. 13).

References digestmap_new(), authdir_config_t::fp_by_name, and authdir_config_t::status_by_digest.

Referenced by dirserv_add_own_fingerprint(), dirserv_get_status_impl(), and dirserv_load_fingerprint_file().

4.22.3.5 int authdir_wants_to_reject_router (routerinfo_t * *ri*, const char ** *msg*, int *complain*)

Check whether we, as a directory server, want to accept **ri**. If so, set its is_valid,named,running fields and return 0. Otherwise, return -1.

If the router is rejected, set ***msg** to an explanation of why.

If **complain** then explain at log-level 'notice' why we refused a descriptor; else explain at log-level 'info'.

References routerinfo_t::address, routerinfo_t::cache_info, routerinfo_t::contact_info, dirserv_router_get_status(), dirserv_router_has_valid_address(), esc_router_info(), FP_BADDIR, FP_BADEXIT, FP_INVALID, FP_NAMED, FP_REJECT, routerinfo_t::is_bad_directory, routerinfo_t::is_bad_exit, routerinfo_t::is_named, routerinfo_t::is_valid, LD_DIRSERV, routerinfo_t::nickname, signed_descriptor_t::published_on, ROUTER_ALLOW_SKEW, ROUTER_MAX_AGE_TO_PUBLISH, and tor_assert.

Referenced by router_add_to_routerlist().

4.22.3.6 void cached_dir_decref (cached_dir_t * *d*)

Decrement the reference count on **d**, and free it if it no longer has any references.

References clear_cached_dir(), cached_dir_t::refcnt, and tor_free.

Referenced by _connection_free(), _free_cached_dir(), connection_dirserv_add_dir_bytes_to_outbuf(), dirserv_clear_old_networkstatuses(), dirserv_clear_old_v1_info(), dirserv_free_all(), dirserv_regenerate_directory(), dirserv_set_cached_consensus_networkstatus(), dirserv_set_cached_directory(), dirserv_set_cached_networkstatus_v2(), dirvote_add_vote(), and dirvote_clear_votes().

4.22.3.7 static void clear_cached_dir (cached_dir_t * *d*) `[static]`

Remove all storage held in **d**, but do not free **d** itself.

References cached_dir_t::dir, cached_dir_t::dir_z, and tor_free.

Referenced by cached_dir_decref(), dirserv_clear_old_v1_info(), and dirserv_free_all().

4.22.3.8 static void clear_status_flags_on_sybil (routerstatus_t * *rs*) `[static]`

Routerstatus **rs** is part of a group of routers that are on too narrow an IP-space. Clear out its flags: we don't want people using it.

References routerstatus_t::is_authority, routerstatus_t::is_bad_directory, routerstatus_t::is_bad_exit, routerstatus_t::is_exit, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerstatus_t::is_running, routerstatus_t::is_stable, routerstatus_t::is_v2_dir, and routerstatus_t::is_valid.

4.22.3.9 static int connection_dirserv_add_dir_bytes_to_outbuf (dir_connection_t * *conn*) [static]

Spooling helper: Called when we're sending a directory or networkstatus, and the outbuf has become too empty. Pulls some bytes from **conn**->cached_dir->dir_z, uncompresses them if appropriate, and puts them on the outbuf. If we run out of entries, flushes the zlib state and sets the spool source to NONE. Returns 0 on success, negative on failure.

References dir_connection_t::_base, buf_datalen(), dir_connection_t::cached_dir, cached_dir_decref(), dir_connection_t::cached_dir_offset, connection_dirserv_finish_spooling(), cached_dir_t::dir_z, cached_dir_t::dir_z_len, DIRSERV_BUFFER_MIN, connection_t::outbuf, TO_CONN, tor_assert, and dir_connection_t::zlib_state.

Referenced by connection_dirserv_add_networkstatus_bytes_to_outbuf(), and connection_dirserv_flushed_some().

4.22.3.10 static int connection_dirserv_add_microdescs_to_outbuf (dir_connection_t * *conn*) [static]

Spooling helper: called when we're sending a bunch of microdescriptors, and the outbuf has become too empty. Pulls some entries from fingerprint_stack, and writes the corresponding microdescs onto outbuf. If we run out of entries, flushes the zlib state and sets the spool source to NONE. Returns 0 on success, negative on failure.

References dir_connection_t::_base, microdesc_t::body, microdesc_t::bodylen, buf_datalen(), dir_connection_t::dir_spool_src, DIRSERV_BUFFER_MIN, dir_connection_t::fingerprint_stack, connection_t::outbuf, smartlist_free(), smartlist_pop_last(), TO_CONN, tor_free, tor_zlib_free(), and dir_connection_t::zlib_state.

Referenced by connection_dirserv_flushed_some().

4.22.3.11 static int connection_dirserv_add_networkstatus_bytes_to_outbuf (dir_connection_t * *conn*) [static]

Spooling helper: Called when we're spooling networkstatus objects on **conn**, and the outbuf has become too empty. If the current networkstatus object (in **conn**->cached_dir) has more data, pull data from there. Otherwise, pop the next fingerprint from fingerprint_stack, and start spooling the next networkstatus. (A digest of all 0 bytes is treated as a request for the current consensus.) If we run out of entries, flushes the zlib state and sets the spool source to NONE. Returns 0 on success, negative on failure.

References dir_connection_t::_base, buf_datalen(), dir_connection_t::cached_dir, dir_connection_t::cached_dir_offset, connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_finish_spooling(), dir_connection_t::dir_spool_src, DIRSERV_BUFFER_MIN, dir_connection_t::fingerprint_stack, lookup_cached_dir_by_fp(), connection_t::outbuf, cached_dir_t::refcnt, smartlist_free(), smartlist_pop_last(), tor_free, tor_zlib_new(), and dir_connection_t::zlib_state.

Referenced by connection_dirserv_flushed_some().

4.22.3.12 static int connection_dirserv_add_servers_to_outbuf (dir_connection_t * *conn*) [static]

Spooling helper: called when we're sending a bunch of server descriptors, and the outbuf has become too empty. Pulls some entries from fingerprint_stack, and writes the corresponding servers onto outbuf. If we run out of entries, flushes the zlib state and sets the spool source to NONE. Returns 0 on success, negative on failure.

References dir_connection_t::_base, buf_datalen(), connection_dir_is_encrypted(), dir_connection_t::dir_spool_src, DIRSERV_BUFFER_MIN, extrainfo_get_by_descriptor_digest(), dir_connection_t::fingerprint_stack, get_signed_descriptor_by_fp(), connection_t::outbuf, router_get_by_descriptor_digest(), ROUTER_MAX_AGE_TO_PUBLISH, signed_descriptor_t::send_unencrypted, signed_descriptor_get_body(), signed_descriptor_t::signed_descriptor_len, smartlist_free(), smartlist_pop_last(), TO_CONN, tor_free, tor_zlib_free(), and dir_connection_t::zlib_state.

Referenced by connection_dirserv_flushed_some().

4.22.3.13 static int connection_dirserv_finish_spooling (dir_connection_t * *conn*) `[static]`

Spooling helper: called when we have no more data to spool to **conn**. Flushes any remaining data to be (un)compressed, and changes the spool source to NONE. Returns 0 on success, negative on failure.

References dir_connection_t::dir_spool_src, tor_zlib_free(), and dir_connection_t::zlib_state.

Referenced by connection_dirserv_add_dir_bytes_to_outbuf(), and connection_dirserv_add_networkstatus_bytes_to_outbuf().

4.22.3.14 int connection_dirserv_flushed_some (dir_connection_t * *conn*)

Called whenever we have flushed some directory data in state SERVER_WRITING.

References dir_connection_t::_base, buf_datalen(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), DIR_CONN_STATE_SERVER_WRITING, dir_connection_t::dir_spool_src, DIRSERV_BUFFER_MIN, connection_t::outbuf, connection_t::state, and tor_assert.

Referenced by connection_flushed_some(), and directory_handle_command_get().

4.22.3.15 int directory_caches_dir_info (or_options_t * *options*)

Return 1 if we want to keep descriptors, networkstatuses, etc around and we're willing to serve them to others. Else return 0.

References advertised_server_mode(), or_options_t::BridgeRelay, or_options_t::DirPort, or_options_t::RefuseUnknownExits, and server_mode().

Referenced by authority_certs_fetch_missing(), directory_too_idle_to_fetch_descriptors(), networkstatus_set_current_consensus(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), signed_desc_digest_is_recognized(), and trusted_dirs_load_certs_from_string().

4.22.3.16 int directory_caches_v2_dir_info (or_options_t * *options*)

Return 1 if we want to cache v2 dir info (each status file).

References or_options_t::DirPort.

Referenced by add_networkstatus_to_cache(), networkstatus_v2_list_clean(), options_act_reversible(), router_reload_v2_networkstatus(), and router_set_networkstatus_v2().

4.22.3.17 int directory_fetches_dir_info_early (or_options_t ∗ *options*)

Return 1 if we should fetch new networkstatuses, descriptors, etc on the "mirror" schedule rather than the "client" schedule.

References directory_fetches_from_authorities().

Referenced by launch_router_descriptor_downloads(), options_act(), update_consensus_networkstatus_fetch_time(), update_networkstatus_downloads(), update_router_descriptor_cache_downloads_v2(), and update_router_descriptor_downloads().

4.22.3.18 int directory_fetches_dir_info_later (or_options_t ∗ *options*)

Return 1 if we should fetch new networkstatuses, descriptors, etc on a very passive schedule – waiting long enough for ordinary clients to probably have the info we want. These would include bridge users, and maybe others in the future e.g. if a Tor client uses another Tor client as a directory guard.

References or_options_t::UseBridges.

Referenced by options_act(), and update_consensus_networkstatus_fetch_time().

4.22.3.19 int directory_fetches_from_authorities (or_options_t ∗ *options*)

Return 1 if we fetch our directory material directly from the authorities, rather than from a mirror.

References advertised_server_mode(), or_options_t::BridgeRelay, routerinfo_t::dir_port, or_options_t::DirPort, or_options_t::FetchDirInfoEarly, or_options_t::RefuseUnknownExits, router_get_my_routerinfo(), router_pick_published_address(), and server_mode().

Referenced by dir_routerdesc_download_failed(), directory_command_should_use_begindir(), directory_fetches_dir_info_early(), directory_get_from_dirserver(), and directory_info_has_arrived().

4.22.3.20 int directory_permits_begindir_requests (or_options_t ∗ *options*)

Return 1 if we want to allow remote people to ask us directory requests via the "begin_dir" interface, which doesn't require having any separate port open.

References or_options_t::BridgeRelay, and or_options_t::DirPort.

Referenced by connection_exit_begin_conn().

4.22.3.21 int directory_permits_controller_requests (or_options_t ∗ *options*)

Return 1 if we want to allow controllers to ask us directory requests via the controller interface, which doesn't require having any separate port open.

References or_options_t::DirPort.

4.22.3.22 static void directory_remove_invalid (void) `[static]`

Remove all descriptors whose nicknames or fingerprints no longer are allowed by our fingerprint list. (Descriptors that used to be good can become bad when we reload the fingerprint list.)

References bool_neq, directory_set_dirty(), dirserv_router_get_status(), FP_BADDIR, FP_BADEXIT, FP_INVALID, FP_NAMED, FP_REJECT, routerinfo_t::is_bad_directory, routerinfo_t::is_bad_exit, routerinfo_t::is_named, routerinfo_t::is_valid, LD_DIRSERV, routerinfo_t::nickname, router_get_routerlist(), routerlist_assert_ok(), routerlist_remove(), and routerlist_t::routers.

Referenced by dirserv_load_fingerprint_file().

4.22.3.23 void directory_set_dirty (void)

Mark the directory as **dirty** – when we're next asked for a directory, we will rebuild it instead of reusing the most recently generated one.

References cached_dir_t::dir, cached_dir_t::published, runningrouters_is_dirty, the_directory_is_dirty, and the_v2_networkstatus_is_dirty.

Referenced by connection_tls_finish_handshake(), directory_remove_invalid(), and router_add_to_routerlist().

4.22.3.24 int directory_too_idle_to_fetch_descriptors (or_options_t ∗ *options*, time_t *now*)

Return 1 if we have no need to fetch new descriptors. This generally happens when we're not a dir cache and we haven't built any circuits lately.

References directory_caches_dir_info(), or_options_t::FetchUselessDescriptors, and rep_hist_circbuilding_dormant().

Referenced by directory_info_has_arrived(), update_consensus_router_descriptor_downloads(), and update_router_have_minimum_dir_info().

4.22.3.25 was_router_added_t dirserv_add_descriptor (routerinfo_t ∗ *ri*, const char ∗∗ *msg*, const char ∗ *source*)

Examine the parsed server descriptor in **ri** and maybe insert it into the list of server descriptors. Set ∗**msg** to a message that should be passed back to the origin of this descriptor, or NULL if there is no such message. Use **source** to produce better log messages.

Return the status of the operation

This function is only called when fresh descriptors are posted, not when we re-load the cache.

References routerinfo_t::cache_info, control_event_descriptors_changed(), control_event_or_authdir_new_descriptor(), signed_descriptor_t::identity_digest, routerinfo_t::is_valid, LD_DIR, LD_DIRSERV, MAX_DESCRIPTOR_UPLOAD_SIZE, routerinfo_t::nickname, signed_descriptor_t::published_on, router_add_to_routerlist(), router_differences_are_cosmetic(), router_get_by_digest(), router_is_me(), routerinfo_free(), signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), tor_free, WRA_WAS_ADDED(), and WRA_WAS_REJECTED().

Referenced by dirserv_add_multiple_descriptors(), and init_keys().

4.22.3.26 static was_router_added_t dirserv_add_extrainfo (extrainfo_t ∗ *ei*, const char ∗∗ *msg*) `[static]`

As dirserv_add_descriptor, but for an **extrainfo_t** (p. 99) **ei**.

References extrainfo_t::cache_info, extrainfo_free(), signed_descriptor_t::identity_digest, LD_DIR, MAX_EXTRAINFO_UPLOAD_SIZE, router_add_extrainfo_to_routerlist(), router_get_by_digest(), routerinfo_incompatible_with_extrainfo(), signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by dirserv_add_multiple_descriptors().

4.22.3.27 was_router_added_t dirserv_add_multiple_descriptors (const char ∗ *desc*, uint8_t *purpose*, const char ∗ *source*, const char ∗∗ *msg*)

As for **dirserv_add_descriptor()** (p. 594), but accepts multiple documents, and returns the most severe error that occurred for any one of them.

References dirserv_add_descriptor(), dirserv_add_extrainfo(), escaped(), format_iso_time(), ROUTER_ANNOTATION_BUF_LEN, router_parse_list_from_string(), ROUTER_PURPOSE_GENERAL, router_purpose_to_string(), SAVED_NOWHERE, smartlist_clear(), smartlist_create(), smartlist_free(), tor_assert, tor_snprintf(), WRA_MORE_SEVERE(), and WRA_WAS_ADDED().

Referenced by directory_handle_command_post().

4.22.3.28 int dirserv_add_own_fingerprint (const char ∗ *nickname*, crypto_pk_env_t ∗ *pk*)

Add the nickname and fingerprint for this OR to the global list of recognized identity key fingerprints.

References add_fingerprint_to_dir(), authdir_config_new(), crypto_pk_get_fingerprint(), FINGERPRINT_LEN, and LD_BUG.

Referenced by init_keys().

4.22.3.29 void dirserv_clear_old_networkstatuses (time_t *cutoff*)

Remove any v2 networkstatus from the directory cache that was published before **cutoff**.

References cached_dir_decref(), cached_v2_networkstatus, file_status(), LD_DIR, networkstatus_get_cache_filename(), and tor_free.

Referenced by networkstatus_v2_list_clean().

4.22.3.30 void dirserv_clear_old_v1_info (time_t *now*)

Remove any v1 info from the directory cache that was published too long ago.

References cached_dir_decref(), clear_cached_dir(), MAX_V1_DIRECTORY_AGE, MAX_V1_RR_AGE, and cached_dir_t::published.

Referenced by networkstatus_v2_list_clean().

4.22.3.31 static void dirserv_compute_performance_thresholds (routerlist_t * *rl*) `[static]`

Look through the routerlist, the Mean Time Between Failure history, and the Weighted Fractional Uptime history, and use them to set thresholds for the Stable, Fast, and Guard flags. Update the fields stable_uptime, stable_mtbf, enough_mtbf_info, guard_wfu, guard_tk, fast_bandwidth, guard_bandwidh_including_exits, guard_bandwidth_excluding_exits, total_bandwidth, and total_exit_bandwidth.

Also, set the is_exit flag of each router appropriately.

References exit_policy_is_general_exit(), real_uptime(), rep_hist_get_stability(), rep_hist_get_weighted_time_known(), router_exit_policy_rejects_all(), router_get_advertised_bandwidth(), router_is_active(), and routerlist_t::routers.

4.22.3.32 int dirserv_dump_directory_to_string (char ** *dir_out*, crypto_pk_env_t * *private_key*)

Generate a new v1 directory and write it into a newly allocated string. Point ***dir_out** to the allocated string. Sign the directory with **private_key**. Return 0 on success, -1 on failure. If **complete** is set, give us all the descriptors; otherwise leave out non-running and non-valid ones.

References crypto_pk_write_public_key_to_string(), DIGEST_LEN, format_iso_time(), format_versions_list(), get_options(), LD_BUG, note_crypto_pk_op(), router_append_dirobj_signature(), router_get_dir_hash(), tor_assert, tor_free, and tor_snprintf().

Referenced by dirserv_regenerate_directory().

4.22.3.33 size_t dirserv_estimate_data_size (smartlist_t * *fps*, int *is_serverdescs*, int *compressed*)

Return an approximate estimate of the number of bytes that will be needed to transmit the server descriptors (if is_serverdescs – they can be either d/ or fp/ queries) or networkstatus objects (if !is_serverdescs) listed in **fps**. If **compressed** is set, we guess how large the data will be after compression.

The return value is an estimate; it might be larger or smaller.

References routerinfo_t::cache_info, cached_dir_t::dir_len, cached_dir_t::dir_z_len, lookup_cached_dir_by_fp(), router_get_my_routerinfo(), signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by directory_handle_command_get().

4.22.3.34 size_t dirserv_estimate_microdesc_size (const smartlist_t * *fps*, int *compressed*)

Given a list of microdescriptor hashes, guess how many bytes will be needed to transmit them, and return the guess.

Referenced by directory_handle_command_get().

4.22.3.35 void dirserv_free_all (void)

Release all storage used by the directory server.

References _free_cached_dir(), cached_consensuses, cached_dir_decref(), cached_v2_networkstatus, clear_cached_dir(), digestmap_free(), dirserv_free_fingerprint_list(), and strmap_free().

Referenced by tor_free_all().

4.22.3.36 void dirserv_free_fingerprint_list (void)

Clear the current fingerprint list.

References _tor_free(), digestmap_free(), authdir_config_t::fp_by_name, authdir_config_t::status_by_digest, strmap_free(), and tor_free.

Referenced by dirserv_free_all(), and dirserv_load_fingerprint_file().

4.22.3.37 networkstatus_t∗ dirserv_generate_networkstatus_vote_obj (crypto_pk_env_t ∗ *private_key*, authority_cert_t ∗ *cert*)

Return a new networkstatus_t∗ containing our current opinion. (For v3 authorities)

References or_options_t::AuthDirListBadDirs, or_options_t::AuthDirListBadExits, or_options_t::ContactInfo, crypto_pk_get_digest(), DIGEST_LEN, dirserv_set_router_is_running(), format_versions_list(), get_options(), authority_cert_t::identity_key, LD_BUG, LD_NET, or_options_t::NamingAuthoritativeDir, or_options_t::RecommendedClientVersions, or_options_t::RecommendedServerVersions, resolve_my_address(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, routerlist_t::routers, tor_assert, tor_dup_ip(), tor_free, and or_options_t::VersioningAuthoritativeDir.

Referenced by dirvote_perform_vote().

4.22.3.38 cached_dir_t∗ dirserv_get_consensus (const char ∗ *flavor_name*)

Return the latest downloaded consensus networkstatus in encoded, signed, optionally compressed format, suitable for sending to clients.

References cached_consensuses, and strmap_get().

Referenced by networkstatus_set_current_consensus().

4.22.3.39 cached_dir_t∗ dirserv_get_directory (void)

Return the most recently generated encoded signed v1 directory, generating a new one as necessary. If not a v1 authoritative directory may return NULL if no directory is yet cached.

References dirserv_pick_cached_dir_obj(), dirserv_regenerate_directory(), the_directory_is_dirty, and V1_AUTHORITY.

Referenced by directory_handle_command_get(), and directory_handle_command_post().

4.22.3.40 static uint32_t dirserv_get_name_status (const char ∗ *id_digest*, const char ∗ *nickname*) `[static]`

Helper: Based only on the ID/Nickname combination, return FP_UNNAMED (unnamed), FP_NAMED (named), or 0 (neither).

References base16_encode(), DIGEST_LEN, authdir_config_t::fp_by_name, FP_NAMED, FP_UNNAMED, HEX_DIGEST_LEN, and strmap_get_lc().

Referenced by dirserv_get_status_impl(), and set_routerstatus_from_routerinfo().

4.22.3.41 void dirserv_get_networkstatus_v2 (smartlist_t ∗ *result*, const char ∗ *key*)

Look for a network status object as specified by **key**, which should be either "authority" (to find a network status generated by us), a hex identity digest (to find a network status generated by given directory), or "all" (to return all the v2 network status objects we have).

References base16_encode(), cached_v2_networkstatus, DIGEST_LEN, digestmap_get(), digestmap_new(), dirserv_get_networkstatus_v2_fingerprints(), generate_v2_networkstatus_opinion(), HEX_DIGEST_LEN, LD_DIRSERV, router_digest_is_me(), should_generate_v2_networkstatus(), smartlist_add(), smartlist_create(), smartlist_free(), tor_assert, and tor_free.

4.22.3.42 void dirserv_get_networkstatus_v2_fingerprints (smartlist_t ∗ *result*, const char ∗ *key*)

Given the portion of a networkstatus request URL after "tor/status/" in **key**, append to **result** the digests of the identity keys of the networkstatus objects that the client has requested.

References authdir_mode_v2(), routerinfo_t::cache_info, cached_v2_networkstatus, DIGEST_LEN, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_new(), digestmap_size(), dir_split_resource_into_fingerprints(), generate_v2_networkstatus_opinion(), get_options(), signed_descriptor_t::identity_digest, LD_DIRSERV, router_get_my_routerinfo(), router_get_trusted_dir_servers(), should_generate_v2_networkstatus(), smartlist_add(), smartlist_sort_digests(), strcmpstart(), tor_assert, and V2_AUTHORITY.

Referenced by directory_handle_command_get(), and dirserv_get_networkstatus_v2().

4.22.3.43 const char∗ dirserv_get_nickname_by_digest (const char ∗ *digest*)

If we are an authoritative dirserver, and the list of approved servers contains one whose identity key digest is **digest**, return that router's nickname. Otherwise return NULL.

References digestmap_get(), router_status_t::nickname, authdir_config_t::status_by_digest, and tor_assert.

Referenced by connection_or_init_conn_from_address().

4.22.3.44 int dirserv_get_routerdesc_fingerprints (smartlist_t ∗ *fps_out*, const char ∗ *key*, const char ∗∗ *msg*, int *for_unencrypted_conn*, int *is_extrainfo*)

As **dirserv_get_routerdescs()** (p. 599), but instead of getting **signed_descriptor_t** (p. 227) pointers, adds copies of digests to fps_out, and doesn't use the /tor/server/ prefix. For a /d/ request, adds descriptor digests; for other requests, adds identity digests.

References routerinfo_t::cache_info, DIGEST_LEN, dir_split_resource_into_fingerprints(), extrainfo_get_by_descriptor_digest(), get_signed_descriptor_by_fp(), signed_descriptor_t::identity_digest, router_get_by_descriptor_digest(), router_get_my_routerinfo(), router_-

get_routerlist(), routerlist_t::routers, signed_descriptor_t::send_unencrypted, smartlist_add(), strcmpstart(), and tor_free.

Referenced by directory_handle_command_get().

4.22.3.45 int dirserv_get_routerdescs (smartlist_t ∗ *descs_out*, const char ∗ *key*, const char ∗∗ *msg*)

Add a **signed_descriptor_t** (p. 227) to **descs_out** for each router matching **key**. The key should be either

- "/tor/server/authority" for our own routerinfo;
- "/tor/server/all" for all the routerinfos we have, concatenated;
- "/tor/server/fp/FP" where FP is a plus-separated sequence of hex identity digests; or
- "/tor/server/d/D" where D is a plus-separated sequence of server descriptor digests, in hex.

Return 0 if we found some matching descriptors, or -1 if we do not have any descriptors, no matching descriptors, or if we did not recognize the key (URL). If -1 is returned ∗**msg** will be set to an appropriate error message.

XXXX rename this function. It's only called from the controller. XXXX in fact, refactor this function, merging as much as possible.

References routerinfo_t::cache_info, dir_split_resource_into_fingerprints(), signed_descriptor_t::published_on, router_digest_is_me(), router_get_by_descriptor_digest(), router_get_by_digest(), router_get_my_routerinfo(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, routerlist_t::routers, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), and tor_free.

4.22.3.46 cached_dir_t∗ dirserv_get_runningrouters (void)

Set ∗**rr** to the most recently generated encoded signed running-routers list, generating a new one as necessary. Return the size of the directory on success, and 0 on failure.

References dirserv_pick_cached_dir_obj(), generate_runningrouters(), runningrouters_is_dirty, and V1_AUTHORITY.

Referenced by directory_handle_command_get().

4.22.3.47 static uint32_t dirserv_get_status_impl (const char ∗ *id_digest*, const char ∗ *nickname*, const char ∗ *address*, uint32_t *addr*, uint16_t *or_port*, const char ∗ *platform*, const char ∗ *contact*, const char ∗∗ *msg*, int *should_log*) `[static]`

Helper: As dirserv_get_router_status, but takes the router fingerprint (hex, no spaces), nickname, address (used for logging only), IP address, OR port, platform (logging only) and contact info (logging only) as arguments.

If should_log is false, do not log messages. (There's not much point in logging that we're rejecting servers we'll not download.)

References authdir_config_new(), authdir_policy_baddir_address(), authdir_policy_badexit_address(), authdir_policy_permits_address(), authdir_policy_valid_address(), or_options_t::AuthDirRejectUnlisted, digestmap_get(), digestmap_size(), dirserv_get_name_status(), esc_for_log(), escaped(), FP_BADDIR, FP_BADEXIT, authdir_config_t::fp_by_name, FP_INVALID, FP_NAMED, FP_REJECT, FP_UNNAMED, get_options(), LD_DIRSERV, router_status_t::status, authdir_config_t::status_by_digest, strmap_size(), tor_free, and tor_version_as_new_as().

Referenced by dirserv_router_get_status(), and dirserv_would_reject_router().

4.22.3.48 int dirserv_have_any_microdesc (const smartlist_t ∗ *fps*)

Return true iff any of the 256-bit elements in **fps** is the digest of a microdescriptor we have.

Referenced by directory_handle_command_get().

4.22.3.49 int dirserv_have_any_serverdesc (smartlist_t ∗ *fps*, int *spool_src*)

Return true iff we have any of the documents (extrainfo or routerdesc) specified by the fingerprints in **fps** and **spool_src**. Used to decide whether to send a 404.

References extrainfo_get_by_descriptor_digest(), get_signed_descriptor_by_fp(), router_get_by_descriptor_digest(), and ROUTER_MAX_AGE_TO_PUBLISH.

Referenced by directory_handle_command_get().

4.22.3.50 int dirserv_load_fingerprint_file (void)

Load the nickname->fingerprint mappings stored in the approved-routers file. The file format is line-based, with each non-blank holding one nickname, some space, and a fingerprint for that nickname. On success, replace the current fingerprint list with the new list and return 0. On failure, leave the current fingerprint list untouched, and return -1.

References add_fingerprint_to_dir(), authdir_config_new(), base16_decode(), config_free_lines(), config_get_lines(), DEFAULT_CLIENT_NICKNAME, DIGEST_LEN, directory_remove_invalid(), dirserv_free_fingerprint_list(), get_datadir_fname, get_options(), HEX_DIGEST_LEN, is_legal_nickname(), LD_CONFIG, LD_FS, LD_GENERAL, MAX_NICKNAME_LEN, or_options_t::NamingAuthoritativeDir, config_line_t::next, read_file_to_str(), RFTS_IGNORE_MISSING, tor_free, and tor_strstrip().

Referenced by do_hup(), and init_keys().

4.22.3.51 void dirserv_orconn_tls_done (const char ∗ *address*, uint16_t *or_port*, const char ∗ *digest_rcvd*, int *as_advertised*)

Called when a TLS handshake has completed successfully with a router listening at **address**:**or_port**, and has yielded a certificate with digest **digest_rcvd**.

Also, if as_advertised is 1, then inform the reachability checker that we could get to this guy.

References authdir_mode_bridge(), DIGEST_LEN, get_options(), LD_DIRSERV, rep_hist_note_router_reachable(), router_get_routerlist(), ROUTER_PURPOSE_BRIDGE, routerlist_t::routers, and tor_assert.

Referenced by connection_or_check_valid_tls_handshake().

4.22.3.52 static cached_dir_t* dirserv_pick_cached_dir_obj (cached_dir_t * *cache_src*, cached_dir_t * *auth_src*, time_t *dirty*, cached_dir_t *(*)(void) *regenerate*, const char * *name*, authority_type_t *auth_type*) `[static]`

Helper: If we're an authority for the right directory version (v1 or v2) (based on **auth_type**), try to regenerate auth_src as appropriate and return it, falling back to cache_src on failure. If we're a cache, simply return cache_src.

References authdir_mode_bridge(), authdir_mode_v1(), authdir_mode_v2(), DIR_REGEN_SLACK_TIME, get_options(), LD_BUG, LD_DIRSERV, V1_AUTHORITY, and V2_AUTHORITY.

Referenced by dirserv_get_directory(), and dirserv_get_runningrouters().

4.22.3.53 int dirserv_read_measured_bandwidths (const char * *from_file*, smartlist_t * *routerstatuses*)

Read the measured bandwidth file and apply it to the list of routerstatuses. Returns -1 on error, 0 otherwise.

References compare_routerstatus_entries(), escaped(), LD_CONFIG, LD_DIRSERV, measured_bw_line_apply(), measured_bw_line_parse(), smartlist_sort(), and tor_parse_ulong().

Referenced by options_validate().

4.22.3.54 static cached_dir_t * dirserv_regenerate_directory (void) `[static]`

Only called by v1 auth dirservers. Generate a fresh v1 directory; set the_directory and return a pointer to the new value.

References cached_dir_decref(), cached_dir_t::dir, cached_dir_t::dir_len, dirserv_dump_directory_to_string(), dirserv_set_cached_directory(), get_identity_key(), LD_BUG, LD_DIRSERV, new_cached_dir(), the_directory_is_dirty, and tor_free.

Referenced by dirserv_get_directory().

4.22.3.55 int dirserv_remove_old_statuses (smartlist_t * *fps*, time_t *cutoff*)

Remove from **fps** every networkstatus key where both a) we have a networkstatus document and b) it is not newer than **cutoff**.

Return 1 if any items were present at all; else return 0.

References lookup_cached_dir_by_fp(), cached_dir_t::published, and tor_free.

Referenced by directory_handle_command_get().

4.22.3.56 static uint32_t dirserv_router_get_status (const routerinfo_t * *router*, const char ** *msg*) `[static]`

Check whether **router** has a nickname/identity key combination that we recognize from the fingerprint list, or an IP we automatically act on according to our configuration. Return the appropriate router status.

If the status is 'FP_REJECT' and **msg** is provided, set ∗**msg** to an explanation of why.

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::contact_info, crypto_pk_get_digest(), DIGEST_LEN, dirserv_get_status_impl(), FP_REJECT, routerinfo_t::identity_pkey, LD_BUG, routerinfo_t::nickname, routerinfo_t::or_port, and routerinfo_t::platform.

Referenced by authdir_wants_to_reject_router(), and directory_remove_invalid().

4.22.3.57 static int dirserv_router_has_valid_address (routerinfo_t ∗ *ri*) `[static]`

Return -1 if **ri** has a private or otherwise bad address, unless we're configured to not care. Return 0 if all ok.

References routerinfo_t::address, get_options(), is_internal_IP(), LD_DIRSERV, routerinfo_t::nickname, and tor_inet_aton().

Referenced by authdir_wants_to_reject_router().

4.22.3.58 void dirserv_set_cached_consensus_networkstatus (const char ∗ *networkstatus*, const char ∗ *flavor_name*, const digests_t ∗ *digests*, time_t *published*)

Replace the v3 consensus networkstatus of type **flavor_name** that we're serving with **networkstatus**, published at **published**. No validation is performed.

References cached_consensuses, cached_dir_decref(), cached_dir_t::digests, new_cached_dir(), and strmap_set().

Referenced by networkstatus_set_current_consensus().

4.22.3.59 void dirserv_set_cached_directory (const char ∗ *directory*, time_t *published*, int *is_running_routers*)

If we have no cached v1 directory, or it is older than **published**, then replace it with **directory**, published at **published**.

If **published** is too old, do nothing.

If **is_running_routers**, this is really a v1 running_routers document rather than a v1 directory.

References cached_dir_decref(), MAX_V1_DIRECTORY_AGE, MAX_V1_RR_AGE, new_cached_dir(), and set_cached_dir().

Referenced by dirserv_regenerate_directory(), router_parse_directory(), and router_parse_runningrouters().

4.22.3.60 void dirserv_set_cached_networkstatus_v2 (const char ∗ *networkstatus*, const char ∗ *identity*, time_t *published*)

If **networkstatus** is non-NULL, we've just received a v2 network-status for an authoritative directory with identity digest **identity** published at **published** – store it so we can serve it to others.

If **networkstatus** is NULL, remove the entry with the given identity fingerprint from the v2 cache.

References cached_dir_decref(), cached_v2_networkstatus, digestmap_get(), digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_new(), digestmap_remove(), digestmap_set(), digestmap_size(), MAX_UNTRUSTED_NETWORKSTATUSES, new_cached_dir(), cached_dir_t::published, router_get_trusted_dir_servers(), and tor_assert.

Referenced by add_networkstatus_to_cache(), and networkstatus_v2_list_clean().

4.22.3.61 void dirserv_set_router_is_running (routerinfo_t ∗ *router*, time_t *now*)

Treat a router as alive if

- It's me, and I'm not hibernating. or - We've found it reachable recently.

References or_options_t::AssumeReachable, routerinfo_t::cache_info, get_options(), signed_descriptor_t::identity_digest, routerinfo_t::is_running, routerinfo_t::last_reachable, REACHABLE_TIMEOUT, rep_hist_note_router_unreachable(), router_is_me(), and we_are_hibernating().

Referenced by dirserv_generate_networkstatus_vote_obj(), generate_v2_networkstatus_opinion(), list_server_status_v1(), and networkstatus_getinfo_by_purpose().

4.22.3.62 void dirserv_single_reachability_test (time_t *now*, routerinfo_t ∗ *router*)

Helper function for **dirserv_test_reachability()** (p. 603). Start a TLS connection to **router**, and annotate it with when we started the test.

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::cache_info, connection_or_connect(), signed_descriptor_t::identity_digest, LD_OR, routerinfo_t::nickname, routerinfo_t::or_port, routerinfo_t::testing_since, and tor_addr_from_ipv4h.

Referenced by dirserv_test_reachability(), and router_add_to_routerlist().

4.22.3.63 void dirserv_test_reachability (time_t *now*)

Auth dir server only: load balance such that we only try a few connections per call.

The load balancing is such that if we get called once every ten seconds, we will cycle through all the tests in 1280 seconds (a bit over 20 minutes).

References authdir_mode_bridge(), dirserv_single_reachability_test(), get_options(), router_get_routerlist(), router_is_me(), ROUTER_PURPOSE_BRIDGE, and routerlist_t::routers.

Referenced by run_scheduled_events().

4.22.3.64 static int dirserv_thinks_router_is_hs_dir (routerinfo_t ∗ *router*, time_t *now*) `[static]`

Return true iff **router** should be assigned the "HSDir" flag. Right now this means it advertises support for it, it has a high uptime, and it's currently considered Running.

This function needs to be called after router->is_running has been set.

References get_options(), routerinfo_t::is_running, real_uptime(), and routerinfo_t::wants_to_be_hs_dir.

Referenced by set_routerstatus_from_routerinfo().

4.22.3.65 static int dirserv_thinks_router_is_unreliable (time_t *now*, routerinfo_t ∗ *router*, int *need_uptime*, int *need_capacity*) [static]

Return 1 if **router** is not suitable for these parameters, else 0. If **need_uptime** is non-zero, we require a minimum uptime. If **need_capacity** is non-zero, we require a minimum advertised bandwidth.

References routerinfo_t::cache_info, signed_descriptor_t::identity_digest, MTBF_TO_GUARANTEE_STABLE, real_uptime(), rep_hist_get_stability(), router_get_advertised_bandwidth(), and UPTIME_TO_GUARANTEE_STABLE.

Referenced by set_routerstatus_from_routerinfo().

4.22.3.66 int dirserv_would_reject_router (routerstatus_t ∗ *rs*)

Return true if there is no point in downloading the router described by **rs** because this directory would reject it.

References routerstatus_t::addr, dirserv_get_status_impl(), FP_REJECT, routerstatus_t::identity_digest, routerstatus_t::nickname, and routerstatus_t::or_port.

Referenced by update_router_descriptor_cache_downloads_v2().

4.22.3.67 static char ∗ format_versions_list (config_line_t ∗ *ln*) [static]

Given a (possibly empty) list of **config_line_t** (p. 50), each line of which contains a list of comma-separated version numbers surrounded by optional space, allocate and return a new string containing the version numbers, in order, separated by commas. Used to generate Recommended(Client|Server)?Versions

References config_line_t::next, smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_split_string(), sort_version_list(), tor_free, and config_line_t::value.

Referenced by dirserv_dump_directory_to_string(), dirserv_generate_networkstatus_vote_obj(), and generate_v2_networkstatus_opinion().

4.22.3.68 static cached_dir_t∗ generate_runningrouters (void) [static]

Only called by v1 auth dirservers. Replace the current running-routers list with a newly generated one.

References crypto_pk_write_public_key_to_string(), DIGEST_LEN, format_iso_time(), get_identity_key(), get_options(), LD_BUG, note_crypto_pk_op(), router_append_dirobj_signature(), router_get_runningrouters_hash(), runningrouters_is_dirty, set_cached_dir(), tor_free, and tor_snprintf().

Referenced by dirserv_get_runningrouters().

4.22.3.69 static cached_dir_t* generate_v2_networkstatus_opinion (void) `[static]`

For v2 authoritative directories only: Replace the contents of **the_v2_networkstatus** with a newly generated network status object.

References or_options_t::AuthDirListBadDirs, or_options_t::AuthDirListBadExits, or_options_t::ContactInfo, crypto_pk_get_fingerprint(), crypto_pk_write_public_key_to_string(), DIGEST_LEN, or_options_t::DirPort, dirserv_set_router_is_running(), FINGERPRINT_LEN, format_iso_time(), format_versions_list(), get_identity_key(), get_options(), INET_NTOA_BUF_LEN, LD_BUG, LD_NET, or_options_t::NamingAuthoritativeDir, or_options_t::RecommendedClientVersions, or_options_t::RecommendedServerVersions, resolve_my_address(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, routerlist_t::routers, RS_ENTRY_LEN, tor_inet_ntoa(), tor_snprintf(), and or_options_t::VersioningAuthoritativeDir.

Referenced by dirserv_get_networkstatus_v2(), and dirserv_get_networkstatus_v2_fingerprints().

4.22.3.70 static digestmap_t* get_possible_sybil_list (const smartlist_t * *routers*) `[static]`

Given a list of **routerinfo_t** (p. 206) in **routers**, return a new digestmap_t whose keys are the identity digests of those routers that we're going to exclude for Sybil-like appearance.

References _compare_routerinfo_by_ip_and_bw(), or_options_t::AuthDirMaxServersPerAddr, or_options_t::AuthDirMaxServersPerAuthAddr, digestmap_new(), digestmap_set(), get_options(), router_addr_is_trusted_dir(), smartlist_add_all(), smartlist_create(), and smartlist_sort().

4.22.3.71 static signed_descriptor_t * get_signed_descriptor_by_fp (const char * *fp*, int *extrainfo*, time_t *publish_cutoff*) `[static]`

Return the cache-info for identity fingerprint **fp**, or its extra-info document if **extrainfo** is true. Return NULL if not found or if the descriptor is older than **publish_cutoff**.

References routerinfo_t::cache_info, extrainfo_t::cache_info, signed_descriptor_t::extra_info_digest, extrainfo_get_by_descriptor_digest(), signed_descriptor_t::published_on, router_digest_is_me(), router_get_by_digest(), router_get_my_extrainfo(), and router_get_my_routerinfo().

Referenced by connection_dirserv_add_servers_to_outbuf(), dirserv_get_routerdesc_fingerprints(), and dirserv_have_any_serverdesc().

4.22.3.72 int list_server_status_v1 (smartlist_t * *routers*, char ** *router_status_out*, int *for_controller*)

Based on the routerinfo_ts in **routers**, allocate the contents of a v1-style router-status line, and store it in ***router_status_out**. Return 0 on success, -1 on failure.

If for_controller is true, include the routers with very old descriptors.

References authdir_mode_publishes_statuses(), dirserv_set_router_is_running(), get_options(), list_single_server_status(), MAX_VERBOSE_NICKNAME_LEN, router_get_verbose_nickname(), ROUTER_MAX_AGE_TO_PUBLISH, smartlist_add(), smartlist_-

create(), smartlist_free(), smartlist_join_strings(), tor_assert, and tor_free.

4.22.3.73 static char* list_single_server_status (routerinfo_t * *desc*, int *is_live*) [static]

Allocate and return a description of the status of the server **desc**, for use in a v1-style router-status line. The server is listed as running iff **is_live** is true.

References base16_encode(), routerinfo_t::cache_info, DIGEST_LEN, HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::is_valid, MAX_NICKNAME_LEN, routerinfo_t::nickname, and tor_assert.

Referenced by list_server_status_v1().

4.22.3.74 static cached_dir_t* lookup_cached_dir_by_fp (const char * *fp*) [static]

Given a fingerprint **fp** which is either set if we're looking for a v2 status, or zeroes if we're looking for a v3 status, or a NUL-padded flavor name if we want a flavored v3 status, return a pointer to the appropriate cached dir object, or NULL if there isn't one available.

References cached_consensuses, cached_v2_networkstatus, DIGEST_LEN, digestmap_get(), router_digest_is_me(), strmap_get(), and tor_digest_is_zero().

Referenced by connection_dirserv_add_networkstatus_bytes_to_outbuf(), dirserv_estimate_data_size(), and dirserv_remove_old_statuses().

4.22.3.75 int measured_bw_line_apply (measured_bw_line_t * *parsed_line*, smartlist_t * *routerstatuses*)

Helper function to apply a parsed measurement line to a list of bandwidth statuses. Returns true if a line is found, false otherwise.

References compare_digest_to_routerstatus_entry(), routerstatus_t::has_measured_bw, LD_DIRSERV, routerstatus_t::measured_bw, and smartlist_bsearch().

Referenced by dirserv_read_measured_bandwidths().

4.22.3.76 int measured_bw_line_parse (measured_bw_line_t * *out*, const char * *orig_line*)

Helper function to parse out a line in the measured bandwidth file into a measured_bw_line_t output structure. Returns -1 on failure or 0 on success.

References base16_decode(), DIGEST_LEN, escaped(), HEX_DIGEST_LEN, LD_DIRSERV, strcmpstart(), tor_free, and tor_parse_long().

Referenced by dirserv_read_measured_bandwidths().

4.22.3.77 cached_dir_t* new_cached_dir (char * *s*, time_t *published*)

Allocate and return a new **cached_dir_t** (p. 23) containing the string **s**, published at **published**.

References cached_dir_t::dir, cached_dir_t::dir_len, cached_dir_t::dir_z, cached_dir_t::dir_z_len, LD_BUG, cached_dir_t::published, cached_dir_t::refcnt, and tor_gzip_compress().

Referenced by dirserv_regenerate_directory(), dirserv_set_cached_consensus_networkstatus(), dirserv_set_cached_directory(), dirserv_set_cached_networkstatus_v2(), and dirvote_add_vote().

4.22.3.78 static INLINE long real_uptime (routerinfo_t * *router*, time_t *now*) [static]

Helper: estimate the uptime of a router given its stated uptime and the amount of time since it last stated its stated uptime.

References routerinfo_t::cache_info, signed_descriptor_t::published_on, and routerinfo_t::uptime.

Referenced by dirserv_compute_performance_thresholds(), dirserv_thinks_router_is_hs_dir(), and dirserv_thinks_router_is_unreliable().

4.22.3.79 void router_clear_status_flags (routerinfo_t * *router*)

Clear all the status flags in routerinfo **router**. We put this function here because it's eerily similar to **clear_status_flags_on_sybil()** (p. 590) above. One day we should merge them.

References routerinfo_t::is_bad_directory, routerinfo_t::is_bad_exit, routerinfo_t::is_exit, routerinfo_t::is_fast, routerinfo_t::is_hs_dir, routerinfo_t::is_possible_guard, routerinfo_t::is_running, routerinfo_t::is_stable, and routerinfo_t::is_valid.

Referenced by routers_update_status_from_consensus_networkstatus().

4.22.3.80 static int router_is_active (routerinfo_t * *ri*, time_t *now*) [static]

Return 1 if **ri**'s descriptor is "active" – running, valid, not hibernating, and not too old. Else return 0.

References routerinfo_t::cache_info, routerinfo_t::is_hibernating, routerinfo_t::is_running, routerinfo_t::is_valid, signed_descriptor_t::published_on, and ROUTER_MAX_AGE_TO_PUBLISH.

Referenced by dirserv_compute_performance_thresholds(), and set_routerstatus_from_routerinfo().

4.22.3.81 int routerstatus_format_entry (char * *buf*, size_t *buf_len*, routerstatus_t * *rs*, const char * *version*, routerstatus_format_type_t *format*)

Helper: write the router-status information in **rs** into **buf**, which has at least **buf_len** free characters. Do NUL-termination. Use the same format as in network-status documents. If **version** is non-NULL, add a "v" line for the platform. Return 0 on success, -1 on failure.

The format argument has three possible values: NS_V2 - Output an entry suitable for a V2 NS opinion document NS_V3_CONSENSUS - Output the first portion of a V3 NS consensus entry NS_V3_CONSENSUS_MICRODESC - Output the first portion of a V3 microdesc consensus entry. NS_V3_VOTE - Output a complete V3 NS vote NS_CONTROL_PORT - Output a NS document for the control port

References routerstatus_t::addr, routerstatus_t::bandwidth, base16_encode(), BASE64_DIGEST_LEN, routerinfo_t::cache_info, routerstatus_t::descriptor_digest, DIGEST_LEN, digest_to_base64(), routerstatus_t::dir_port, routerinfo_t::exit_policy, format_iso_time(), routerstatus_t::has_bandwidth, routerstatus_t::has_measured_bw, HEX_DIGEST_LEN, routerstatus_t::identity_digest, INET_NTOA_BUF_LEN, routerstatus_t::is_authority, routerstatus_t::is_bad_directory, routerstatus_t::is_bad_exit, routerstatus_t::is_exit, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerstatus_t::is_running, routerstatus_t::is_stable, routerstatus_t::is_unnamed, routerstatus_t::is_v2_dir, routerstatus_t::is_valid, LD_BUG, MAX_V_LINE_LEN, routerstatus_t::measured_bw, routerstatus_t::nickname, routerstatus_t::or_port, policy_summarize(), routerstatus_t::published_on, router_get_advertised_bandwidth_capped(), router_get_by_digest(), signed_descriptor_t::signed_descriptor_digest, tor_assert, tor_free, tor_inet_ntoa(), and tor_snprintf().

Referenced by format_networkstatus_vote(), and networkstatus_getinfo_helper_single().

4.22.3.82 static void set_cached_dir (cached_dir_t ∗ *d*, char ∗ *directory*, time_t *when*) [static]

Possibly replace the contents of **d** with the value of **directory** published on **when**, unless **when** is older than the last value, or too far in the future.

Does not copy **directory**; frees it if it isn't used.

References cached_dir_t::dir, cached_dir_t::dir_len, cached_dir_t::dir_z, cached_dir_t::dir_z_len, LD_BUG, LD_DIRSERV, cached_dir_t::published, ROUTER_MAX_AGE_TO_PUBLISH, tor_free, and tor_gzip_compress().

Referenced by dirserv_set_cached_directory(), and generate_runningrouters().

4.22.3.83 void set_routerstatus_from_routerinfo (routerstatus_t ∗ *rs*, routerinfo_t ∗ *ri*, time_t *now*, int *naming*, int *exits_can_be_guards*, int *listbadexits*, int *listbaddirs*)

Extract status information from **ri** and from other authority functions and store it in **rs**>. If **naming**, consider setting the named flag in **rs**. If not **exits_can_be_guards**, never mark an exit as a guard. If **listbadexits**, consider setting the badexit flag.

We assume that ri->is_running has already been set, e.g. by dirserv_set_router_is_running(ri, now);

References routerinfo_t::addr, routerstatus_t::addr, BANDWIDTH_TO_GUARANTEE_GUARD, routerinfo_t::cache_info, routerstatus_t::descriptor_digest, DIGEST_LEN, routerstatus_t::dir_port, routerinfo_t::dir_port, dirserv_get_name_status(), dirserv_thinks_router_is_hs_dir(), dirserv_thinks_router_is_unreliable(), FP_NAMED, FP_UNNAMED, routerstatus_t::identity_digest, signed_descriptor_t::identity_digest, routerstatus_t::is_authority, routerinfo_t::is_bad_directory, routerstatus_t::is_bad_directory, routerinfo_t::is_bad_exit, routerstatus_t::is_bad_exit, routerinfo_t::is_exit, routerstatus_t::is_exit, routerinfo_t::is_fast, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, routerinfo_t::is_hs_dir, routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerinfo_t::is_running, routerstatus_t::is_running, routerinfo_t::is_stable, routerstatus_t::is_stable, routerstatus_t::is_unnamed, routerstatus_t::is_v2_dir, routerinfo_t::is_valid, routerstatus_t::is_valid, routerstatus_t::nickname, routerinfo_t::nickname, routerinfo_t::or_port, routerstatus_t::or_port, routerinfo_t::platform, signed_descriptor_t::published_on, routerstatus_t::published_on, rep_hist_get_weighted_fractional_uptime(), rep_hist_get_weighted_time_known(), router_get_advertised_bandwidth(), router_is_active(), signed_descriptor_t::signed_descriptor_digest, and tor_version_as_new_as().

Referenced by networkstatus_getinfo_by_purpose().

4.22.3.84 static int should_generate_v2_networkstatus (void) `[static]`

Return true iff our opinion of the routers has been stale for long enough that we should generate a new v2 network status doc.

References authdir_mode_v2(), DIR_REGEN_SLACK_TIME, get_options(), and the_v2_networkstatus_is_dirty.

Referenced by dirserv_get_networkstatus_v2(), and dirserv_get_networkstatus_v2_fingerprints().

4.22.3.85 static char* version_from_platform (const char * *platform*) `[static]`

Given a platform string as in a **routerinfo_t** (p. 206) (possibly null), return a newly allocated version string for a networkstatus document, or NULL if the platform doesn't give a Tor version.

References find_whitespace(), and strcmpstart().

4.22.3.86 static int WRA_MORE_SEVERE (was_router_added_t *a*, was_router_added_t *b*) `[static]`

True iff **a** is more severe than **b**.

Referenced by dirserv_add_multiple_descriptors().

4.22.4 Variable Documentation

4.22.4.1 strmap_t* cached_consensuses = NULL `[static]`

Map from flavor name to the v3 consensuses that we're currently serving.

Referenced by dirserv_free_all(), dirserv_get_consensus(), dirserv_set_cached_consensus_networkstatus(), and lookup_cached_dir_by_fp().

4.22.4.2 cached_dir_t* cached_directory = NULL `[static]`

The v1 directory we'll serve (as a cache or as an authority) if requested.

4.22.4.3 cached_dir_t cached_runningrouters `[static]`

The v1 runningrouters document we'll serve (as a cache or as an authority) if requested.

4.22.4.4 digestmap_t* cached_v2_networkstatus = NULL `[static]`

Used for other dirservers' v2 network statuses. Map from hexdigest to **cached_dir_t** (p. 23).

Referenced by dirserv_clear_old_networkstatuses(), dirserv_free_all(), dirserv_get_networkstatus_v2(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), and lookup_cached_dir_by_fp().

4.22.4.5 int enough_mtbf_info = 0 [static]

If true, we have measured enough mtbf info to look at stable_mtbf rather than stable_uptime.

4.22.4.6 uint32_t fast_bandwidth = 0 [static]

Any router with a bandwidth at least this high is "Fast"

4.22.4.7 authdir_config_t* fingerprint_list = NULL [static]

Should be static; exposed for testing.

4.22.4.8 uint32_t guard_bandwidth_excluding_exits = 0 [static]

If exits can't be guards, then all guards must have a bandwidth this high.

4.22.4.9 uint32_t guard_bandwidth_including_exits = 0 [static]

If exits can be guards, then all guards must have a bandwidth this high.

4.22.4.10 long guard_tk = 0 [static]

Don't call a router a guard unless we've known about it for at least this many seconds.

4.22.4.11 double guard_wfu = 0.0 [static]

Any router with a weighted fractional uptime of at least this much might be good as a guard.

4.22.4.12 time_t runningrouters_is_dirty = 1 [static]

Do we need to regenerate the v1 runningrouters document when somebody asks for it?

Referenced by directory_set_dirty(), dirserv_get_runningrouters(), and generate_runningrouters().

4.22.4.13 double stable_mtbf = 0.0 [static]

Any router with an mtbf of at least this value is stable.

4.22.4.14 uint32_t stable_uptime = 0 [static]

Any router with an uptime of at least this value is stable.

4.22.4.15 cached_dir_t* the_directory = NULL [static]

Most recently generated encoded signed v1 directory. (v1 auth dirservers only.)

4.22.4.16 time_t the_directory_is_dirty = 1 `[static]`

Do we need to regenerate the v1 directory when someone asks for it?

Referenced by directory_set_dirty(), dirserv_get_directory(), and dirserv_regenerate_directory().

4.22.4.17 cached_dir_t the_runningrouters `[static]`

For authoritative directories: the current (v1) network status.

4.22.4.18 cached_dir_t* the_v2_networkstatus = NULL `[static]`

For authoritative directories: the current (v2) network status.

4.22.4.19 time_t the_v2_networkstatus_is_dirty = 1 `[static]`

Do we need to regenerate our v2 networkstatus document when somebody asks for it?

Referenced by directory_set_dirty(), and should_generate_v2_networkstatus().

4.22.4.20 time_t time_of_process_start

What time did this process start up?

Referenced by dumpstats(), and tor_init().

4.22.4.21 uint64_t total_bandwidth = 0 `[static]`

Total bandwidth of all the routers we're considering.

4.22.4.22 uint64_t total_exit_bandwidth = 0 `[static]`

Total bandwidth of all the exit routers we're considering.

4.23 dirvote.c File Reference

Functions to compute directory consensus, and schedule voting.

`#include "or.h"`

Data Structures

- struct **pending_consensus_t**
- struct **dir_src_ent_t**
- struct **pending_vote_t**

Defines

- #define **DIRVOTE_PRIVATE**
- #define **MAX_SUPPORTED_CONSENSUS_METHOD** 9
- #define **MIN_METHOD_FOR_FOOTER** 9
- #define **MIN_METHOD_FOR_BW_WEIGHTS** 9
- #define **MIN_METHOD_FOR_PARAMS** 7
- #define **MIN_METHOD_FOR_MICRODESC** 8
- #define **MICRODESC_LINE_LEN** 80
- #define **get_most_frequent_member**(lst) smartlist_get_most_frequent_string(lst)
- #define **RANGE_CHECK**(a, b, c, d, e, f, g, mx)
- #define **CHECK_EQ**(a, b, margin) ((a)-(b) >= 0 ? (a)-(b) <= (margin) : (b)-(a) <= (margin))

Enumerations

- enum **bw_weights_error_t** {

  BW_WEIGHTS_NO_ERROR = 0, **BW_WEIGHTS_RANGE_ERROR** = 1, **BW_WEIGHTS_SUMG_ERROR** = 2, **BW_WEIGHTS_SUME_ERROR** = 3,

  BW_WEIGHTS_SUMD_ERROR = 4, **BW_WEIGHTS_BALANCE_MID_ERROR** = 5, **BW_WEIGHTS_BALANCE_EG_ERROR** = 6 }

Functions

- static int **dirvote_add_signatures_to_all_pending_consensuses** (const char *detached_signatures_body, const char **msg_out)
- static int **dirvote_add_signatures_to_pending_consensus** (**pending_consensus_t** *pc, **ns_detached_signatures_t** *sigs, const char **msg_out)
- static char * **list_v3_auth_ids** (void)
- static void **dirvote_fetch_missing_votes** (void)
- static void **dirvote_fetch_missing_signatures** (void)
- static int **dirvote_perform_vote** (void)
- static void **dirvote_clear_votes** (int all_votes)
- static int **dirvote_compute_consensuses** (void)
- static int **dirvote_publish_consensus** (void)
- static char * **make_consensus_method_list** (int low, int high, const char *sep)

- char * **format_networkstatus_vote** (**crypto_pk_env_t** *private_signing_key, **networkstatus_t** *v3_ns)
- static **networkstatus_voter_info_t** * **get_voter** (const **networkstatus_t** *vote)
- **document_signature_t** * **voter_get_sig_by_algorithm** (const **networkstatus_voter_info_t** *voter, digest_algorithm_t alg)
- static int **_compare_votes_by_authority_id** (const void **_a, const void **_b)
- static int **_compare_dir_src_ents_by_authority_id** (const void **_a, const void **_b)
- static void **get_frequent_members** (**smartlist_t** *out, **smartlist_t** *in, int min)
- static int **compare_vote_rs** (const **vote_routerstatus_t** *a, const **vote_routerstatus_t** *b)
- static int **_compare_vote_rs** (const void **_a, const void **_b)
- static **vote_routerstatus_t** * **compute_routerstatus_consensus** (**smartlist_t** *votes, int consensus_method, char *microdesc_digest256_out)
- static void **hash_list_members** (char *digest_out, size_t len_out, **smartlist_t** *lst, digest_algorithm_t alg)
- static int **_cmp_int_strings** (const void **_a, const void **_b)
- static int **compute_consensus_method** (**smartlist_t** *votes)
- static int **consensus_method_is_supported** (int method)
- static char * **compute_consensus_versions_list** (**smartlist_t** *lst, int n_versioning)
- char * **dirvote_compute_params** (**smartlist_t** *votes)
- static bw_weights_error_t **networkstatus_check_weights** (int64_t Wgg, int64_t Wgd, int64_t Wmg, int64_t Wme, int64_t Wmd, int64_t Wee, int64_t Wed, int64_t scale, int64_t G, int64_t M, int64_t E, int64_t D, int64_t T, int64_t margin, int do_balance)
- static void **networkstatus_compute_bw_weights_v9** (**smartlist_t** *chunks, int64_t G, int64_t M, int64_t E, int64_t D, int64_t T, int64_t weight_scale)
- char * **networkstatus_compute_consensus** (**smartlist_t** *votes, int total_authorities, **crypto_pk_env_t** *identity_key, **crypto_pk_env_t** *signing_key, const char *legacy_id_key_digest, **crypto_pk_env_t** ***legacy_signing_key**, **consensus_flavor_t** flavor)
- int **networkstatus_add_detached_signatures** (**networkstatus_t** *target, **ns_detached_signatures_t** *sigs, const char **msg_out)
- static char * **networkstatus_format_signatures** (**networkstatus_t** *consensus, int for_detached_signatures)
- char * **networkstatus_get_detached_signatures** (**smartlist_t** *consensuses)
- static char * **get_detached_signatures_from_pending_consensuses** (**pending_consensus_t** *pending, int n_flavors)
- void **ns_detached_signatures_free** (**ns_detached_signatures_t** *s)
- **authority_cert_t** * **authority_cert_dup** (**authority_cert_t** *cert)
- void **dirvote_get_preferred_voting_intervals** (**vote_timing_t** *timing_out)
- time_t **dirvote_get_start_of_next_interval** (time_t now, int interval)
- void **dirvote_recalculate_timing** (**or_options_t** *options, time_t now)
- void **dirvote_act** (**or_options_t** *options, time_t now)
- static void **dirvote_clear_pending_consensuses** (void)
- **pending_vote_t** * **dirvote_add_vote** (const char *vote_body, const char **msg_out, int *status_out)
- int **dirvote_add_signatures** (const char *detached_signatures_body, const char *source, const char **msg)
- void **dirvote_free_all** (void)
- const char * **dirvote_get_pending_consensus** (**consensus_flavor_t** flav)
- const char * **dirvote_get_pending_detached_signatures** (void)

- const **cached_dir_t** * **dirvote_get_vote** (const char *fp, int flags)
- **microdesc_t** * **dirvote_create_microdescriptor** (const **routerinfo_t** *ri)
- ssize_t **dirvote_format_microdesc_vote_line** (char *out, size_t out_len, const **microdesc_t** *md)
- int **vote_routerstatus_find_microdesc_hash** (char *digest256_out, const **vote_routerstatus_t** *vrs, int method, digest_algorithm_t alg)

Variables

- struct {
  time_t **voting_starts**
  time_t **fetch_missing_votes**
  time_t **voting_ends**
  time_t **fetch_missing_signatures**
  time_t **interval_starts**
  int **have_voted**
  int **have_fetched_missing_votes**
  int **have_built_consensus**
  int **have_fetched_missing_signatures**
  int **have_published_consensus**
  } **voting_schedule**

- static **smartlist_t** * **pending_vote_list** = NULL
- static **smartlist_t** * **previous_vote_list** = NULL
- static **pending_consensus_t** **pending_consensuses** [N_CONSENSUS_FLAVORS]
- static char * **pending_consensus_signatures** = NULL
- static **smartlist_t** * **pending_consensus_signature_list** = NULL
- static char * **microdesc_consensus_methods** = NULL

4.23.1 Detailed Description

Functions to compute directory consensus, and schedule voting.

4.23.2 Define Documentation

4.23.2.1 #define get_most_frequent_member(lst) smartlist_get_most_frequent_string(lst)

Given a sorted list of strings **lst**, return the member that appears most. Break ties in favor of later-occurring members.

4.23.2.2 #define MAX_SUPPORTED_CONSENSUS_METHOD 9

The highest consensus method that we currently support.

Referenced by consensus_method_is_supported(), dirvote_format_microdesc_vote_line(), and format_networkstatus_vote().

4.23.2.3 #define MIN_METHOD_FOR_BW_WEIGHTS 9

Lowest consensus method that contains bandwidth weights

4.23.2.4 #define MIN_METHOD_FOR_FOOTER 9

Lowest consensus method that contains a 'directory-footer' marker

4.23.2.5 #define MIN_METHOD_FOR_MICRODESC 8

Lowest consensus method that generates microdescriptors

Referenced by dirvote_format_microdesc_vote_line().

4.23.2.6 #define MIN_METHOD_FOR_PARAMS 7

Lowest consensus method that contains consensus params

Referenced by networkstatus_compute_consensus().

4.23.2.7 #define RANGE_CHECK(a, b, c, d, e, f, g, mx)

Value:

```
((a) >= 0 && (a) <= (mx) && (b) >= 0 && (b) <= (mx) && \
        (c) >= 0 && (c) <= (mx) && (d) >= 0 && (d) <= (mx) && \
        (e) >= 0 && (e) <= (mx) && (f) >= 0 && (f) <= (mx) && \
        (g) >= 0 && (g) <= (mx))
```

4.23.3 Function Documentation

4.23.3.1 static int _cmp_int_strings (const void ∗∗ _*a*, const void ∗∗ _*b*) `[static]`

Sorting helper: compare two strings based on their values as base-ten positive integers. (Non-integers are treated as prior to all integers, and compared lexically.)

References tor_parse_long().

Referenced by compute_consensus_method().

4.23.3.2 static int _compare_dir_src_ents_by_authority_id (const void ∗∗ _*a*, const void ∗∗ _*b*) `[static]`

Helper: Compare the dir_src_ent_ts in ∗_**a** and ∗_**b** by their identity digests, and return -1, 0, or 1 depending on their ordering

References DIGEST_LEN, get_voter(), networkstatus_voter_info_t::identity_digest, dir_src_ent_t::is_legacy, networkstatus_voter_info_t::legacy_id_digest, and dir_src_ent_t::v.

Referenced by networkstatus_compute_consensus().

4.23.3.3 static int _compare_vote_rs (const void ∗∗ _*a*, const void ∗∗ _*b*) `[static]`

Helper for sorting routerlists based on compare_vote_rs.

References compare_vote_rs().

Referenced by compute_routerstatus_consensus().

4.23.3.4 static int _compare_votes_by_authority_id (const void ** _*a*, const void ** _*b*) [static]

Helper for sorting **networkstatus_t** (p. 122) votes (not consensuses) by the hash of their voters' identity digests.

References DIGEST_LEN, and get_voter().

Referenced by networkstatus_compute_consensus().

4.23.3.5 authority_cert_t* authority_cert_dup (authority_cert_t * *cert*)

Allocate and return a new **authority_cert_t** (p. 14) with the same contents as **cert**.

References authority_cert_t::cache_info, crypto_pk_dup_key(), authority_cert_t::identity_key, signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, authority_cert_t::signing_key, and tor_assert.

4.23.3.6 static int compare_vote_rs (const vote_routerstatus_t * *a*, const vote_routerstatus_t * *b*) [static]

Return 0 if and only if **a** and **b** are routerstatuses that come from the same routerinfo, with the same derived elements.

References routerstatus_t::addr, routerstatus_t::descriptor_digest, DIGEST_LEN, routerstatus_t::dir_port, routerstatus_t::identity_digest, routerstatus_t::nickname, routerstatus_t::or_port, routerstatus_t::published_on, and vote_routerstatus_t::status.

Referenced by _compare_vote_rs(), and compute_routerstatus_consensus().

4.23.3.7 static int compute_consensus_method (smartlist_t * *votes*) [static]

Given a list of **networkstatus_t** (p. 122) votes, determine and return the number of the highest consensus method that is supported by 2/3 of the voters.

References _cmp_int_strings(), get_frequent_members(), smartlist_add_all(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_sort(), smartlist_uniq(), tor_assert, and tor_parse_long().

Referenced by networkstatus_compute_consensus().

4.23.3.8 static char* compute_consensus_versions_list (smartlist_t * *lst*, int *n_versioning*) [static]

Helper: given **lst**, a list of version strings such that every version appears once for every versioning voter who recommends it, return a newly allocated string holding the resulting client-versions or server-versions list. May change contents of **lst**

References get_frequent_members(), smartlist_create(), smartlist_free(), smartlist_join_strings(), and sort_version_list().

Referenced by networkstatus_compute_consensus().

4.23.3.9 static vote_routerstatus_t* compute_routerstatus_consensus (smartlist_t * *votes*, int *consensus_method*, char * *microdesc_digest256_out*) [static]

Given a list of **vote_routerstatus_t** (p. 252), all for the same router identity, return whichever is most frequent, breaking ties in favor of more recently published **vote_routerstatus_t** (p. 252) and in case of ties there, in favor of smaller descriptor digest.

References _compare_vote_rs(), compare_vote_rs(), routerstatus_t::published_on, smartlist_sort(), and vote_routerstatus_t::status.

4.23.3.10 static int consensus_method_is_supported (int *method*) [static]

Return true iff **method** is a consensus method that we support.

References MAX_SUPPORTED_CONSENSUS_METHOD.

Referenced by make_consensus_method_list(), and networkstatus_compute_consensus().

4.23.3.11 void dirvote_act (or_options_t * *options*, time_t *now*)

Entry point: Take whatever voting actions are pending as of **now**.

References authdir_mode_v3(), authority_cert_t::cache_info, DIGEST_LEN, dirvote_clear_votes(), dirvote_compute_consensuses(), dirvote_fetch_missing_signatures(), dirvote_fetch_missing_votes(), dirvote_perform_vote(), dirvote_publish_consensus(), dirvote_recalculate_timing(), get_my_v3_authority_cert(), hex_str(), signed_descriptor_t::identity_digest, LD_DIR, list_v3_auth_ids(), tor_free, and voting_schedule.

Referenced by run_scheduled_events().

4.23.3.12 int dirvote_add_signatures (const char * *detached_signatures_body*, const char * *source*, const char ** *msg*)

Helper: we just got the **detached_signatures_body** sent to us as signatures on the currently pending consensus. Add them to the pending consensus (if we have one); otherwise queue them until we have a consensus. Return negative on failure, nonnegative on success.

References LD_DIR, smartlist_add(), and smartlist_create().

Referenced by connection_dir_client_reached_eof(), and directory_handle_command_post().

4.23.3.13 static int dirvote_add_signatures_to_pending_consensus (pending_consensus_t * *pc*, ns_detached_signatures_t * *sigs*, const char ** *msg_out*) [static]

Helper: we just got the **detached_signatures_body** sent to us as signatures on the currently pending consensus. Add them to **pc** as appropriate. Return the number of signatures added. (?)

References pending_consensus_t::body, pending_consensus_t::consensus, networkstatus_t::flavor, LD_DIR, networkstatus_add_detached_signatures(), networkstatus_format_signatures(), networkstatus_get_flavor_name(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), ns_detached_signatures_t::signatures, strmap_get(), and tor_assert.

4.23.3.14 pending_vote_t* dirvote_add_vote (const char * *vote_body*, const char ** *msg_out*, int * *status_out*) `[read]`

Called when we have received a networkstatus vote in **vote_body**. Parse and validate it, and on success store it as a pending vote (which we then return). Return NULL on failure. Sets ∗**msg_out** and ∗**status_out** to an HTTP response and status code. (V3 authority only)

References networkstatus_voter_info_t::address, authority_cert_get_by_digests(), authority_cert_t::cache_info, cached_dir_decref(), networkstatus_t::cert, DIGEST_LEN, format_iso_time(), get_voter(), hex_str(), signed_descriptor_t::identity_digest, networkstatus_voter_info_t::identity_digest, LD_BUG, LD_DIR, list_v3_auth_ids(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), new_cached_dir(), networkstatus_voter_info_t::nickname, networkstatus_t::published, signed_descriptor_t::signed_descriptor_body, authority_cert_t::signing_key_digest, networkstatus_voter_info_t::sigs, smartlist_create(), strcmpstart(), tor_assert, tor_free, trusted_dirs_load_certs_from_string(), trusteddirserver_get_by_v3_auth_digest(), update_consensus_router_descriptor_downloads(), networkstatus_t::valid_after, networkstatus_voter_info_t::vote_digest, networkstatus_t::voters, and voting_schedule.

Referenced by connection_dir_client_reached_eof(), directory_handle_command_post(), and dirvote_perform_vote().

4.23.3.15 static void dirvote_clear_pending_consensuses (void) `[static]`

Release all storage held by pending consensuses (those waiting for signatures).

References pending_consensus_t::body, pending_consensus_t::consensus, N_CONSENSUS_FLAVORS, networkstatus_vote_free(), and tor_free.

Referenced by dirvote_clear_votes(), dirvote_compute_consensuses(), and dirvote_free_all().

4.23.3.16 static void dirvote_clear_votes (int *all_votes*) `[static]`

Drop all currently pending votes, consensus, and detached signatures.

References cached_dir_decref(), dirvote_clear_pending_consensuses(), networkstatus_vote_free(), smartlist_add_all(), smartlist_clear(), smartlist_create(), and tor_free.

Referenced by dirvote_act(), and dirvote_free_all().

4.23.3.17 static int dirvote_compute_consensuses (void) `[static]`

Try to compute a v3 networkstatus consensus from the currently pending votes. Return 0 on success, -1 on failure. Store the consensus in pending_consensus: it won't be ready to be published until we have everybody else's signatures collected too. (V3 Authority only)

References pending_consensus_t::body, sized_chunk_t::bytes, pending_consensus_t::consensus, crypto_pk_get_digest(), DIGEST_LEN, DIR_PURPOSE_UPLOAD_SIGNATURES, directory_post_to_dirservers(), dirvote_clear_pending_consensuses(), get_datadir_fname, get_detached_signatures_from_pending_consensuses(), get_my_v3_authority_cert(), get_my_v3_authority_signing_key(), get_my_v3_legacy_cert(), get_my_v3_legacy_signing_key(), get_n_authorities(), get_options(), authority_cert_t::identity_key, LD_DIR, sized_chunk_t::len, N_CONSENSUS_FLAVORS, networkstatus_check_consensus_signature(), networkstatus_compute_consensus(), networkstatus_get_flavor_name(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), ROUTER_PURPOSE_GENERAL, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_string_isin(), tor_assert, tor_free, V3_AUTHORITY, and write_chunks_to_file().

Referenced by dirvote_act().

4.23.3.18 char∗ dirvote_compute_params (smartlist_t ∗ *votes*)

Helper: given a list of valid **networkstatus_t** (p. 122), return a new string containing the contents of the consensus network parameter set.

References smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort_strings(), tor_assert, tor_free, tor_parse_long(), and tor_snprintf().

Referenced by networkstatus_compute_consensus().

4.23.3.19 microdesc_t∗ dirvote_create_microdescriptor (const routerinfo_t ∗ *ri*)

Construct and return a new microdescriptor from a routerinfo **ri**.

XXX Right now, there is only one way to generate microdescriptors from router descriptors. This may change in future consensus methods. If so, we'll need an internal way to remember which method we used, and ask for a particular method.

References crypto_pk_write_public_key_to_string(), routerinfo_t::declared_family, routerinfo_t::exit_policy, LD_DIR, microdescs_parse_from_string(), routerinfo_t::onion_pkey, policy_summarize(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

4.23.3.20 static void dirvote_fetch_missing_signatures (void) `[static]`

Send a request to every other authority for its detached signatures, unless we have signatures from all other v3 authorities already.

References pending_consensus_t::consensus, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, directory_get_from_all_authorities(), N_CONSENSUS_FLAVORS, and networkstatus_check_consensus_signature().

Referenced by dirvote_act().

4.23.3.21 static void dirvote_fetch_missing_votes (void) `[static]`

Send an HTTP request to every other v3 authority, for the votes of every authority for which we haven't received a vote yet in this period. (V3 authority only)

References base16_encode(), DIGEST_LEN, dirvote_get_vote(), HEX_DIGEST_LEN, router_get_trusted_dir_servers(), smartlist_add(), smartlist_create(), and V3_AUTHORITY.

Referenced by dirvote_act().

4.23.3.22 ssize_t dirvote_format_microdesc_vote_line (char ∗ *out*, size_t *out_len*, const microdesc_t ∗ *md*)

Format the appropriate vote line to describe the microdescriptor **md** in a consensus vote document. Write it into the **out_len**-byte buffer in **out**. Return -1 on failure and the number of characters written on success.

References BASE64_DIGEST256_LEN, microdesc_t::digest, digest256_to_base64(), make_consensus_method_list(), MAX_SUPPORTED_CONSENSUS_METHOD, MIN_METHOD_FOR_MICRODESC, tor_assert, and tor_snprintf().

4.23.3.23 void dirvote_free_all (void)

Release all static storage held in **dirvote.c** (p. 612)

References dirvote_clear_pending_consensuses(), dirvote_clear_votes(), smartlist_free(), and tor_free.

Referenced by tor_free_all().

4.23.3.24 const char∗ dirvote_get_pending_consensus (consensus_flavor_t *flav*)

Return the body of the consensus that we're currently trying to build.

References pending_consensus_t::body, and tor_assert.

Referenced by directory_handle_command_get().

4.23.3.25 const char∗ dirvote_get_pending_detached_signatures (void)

Return the signatures that we know for the consensus that we're currently trying to build.

Referenced by directory_handle_command_get().

4.23.3.26 void dirvote_get_preferred_voting_intervals (vote_timing_t ∗ *timing_out*)

Set ∗**timing_out** to the intervals at which we would like to vote. Note that these aren't the intervals we'll use to vote; they're the ones that we'll vote to use.

References vote_timing_t::dist_delay, get_options(), vote_timing_t::n_intervals_valid, tor_assert, or_options_t::V3AuthDistDelay, or_options_t::V3AuthNIntervalsValid, or_options_t::V3AuthVoteDelay, or_options_t::V3AuthVotingInterval, vote_timing_t::vote_delay, and vote_timing_t::vote_interval.

4.23.3.27 time_t dirvote_get_start_of_next_interval (time_t *now*, int *interval*)

Return the start of the next interval of size **interval** (in seconds) after **now**. Midnight always starts a fresh interval, and if the last interval of a day would be truncated to less than half its size, it is rolled into the previous interval.

References tor_timegm().

Referenced by dirvote_recalculate_timing().

4.23.3.28 const cached_dir_t∗ dirvote_get_vote (const char ∗ *fp*, int *flags*)

Return a given vote specified by **fp**. If **by_id**, return the vote for the authority with the v3 authority identity key digest **fp**; if **by_id** is false, return the vote whose digest is **fp**. If **fp** is NULL, return our own vote. If **include_previous** is false, do not consider any votes for a consensus that's already been built. If **include_pending** is false, do not consider any votes for the consensus that's in progress. May return NULL if we have no vote for the authority in question.

References authority_cert_t::cache_info, DIGEST_LEN, get_my_v3_authority_cert(), get_voter(), networkstatus_voter_info_t::identity_digest, and signed_descriptor_t::identity_digest.

Referenced by directory_handle_command_get(), and dirvote_fetch_missing_votes().

4.23.3.29 static int dirvote_perform_vote (void) `[static]`

Generate a networkstatus vote and post it to all the v3 authorities. (V3 Authority only)

References cached_dir_t::dir, cached_dir_t::dir_len, DIR_PURPOSE_UPLOAD_VOTE, directory_post_to_dirservers(), dirserv_generate_networkstatus_vote_obj(), dirvote_add_vote(), authority_cert_t::expires, format_networkstatus_vote(), get_my_v3_authority_cert(), get_my_v3_authority_signing_key(), LD_DIR, LD_NET, networkstatus_vote_free(), ROUTER_PURPOSE_GENERAL, tor_free, V3_AUTHORITY, and pending_vote_t::vote_body.

Referenced by dirvote_act().

4.23.3.30 static int dirvote_publish_consensus (void) `[static]`

Replace the consensus that we're currently serving with the one that we've been building. (V3 Authority only)

References pending_consensus_t::body, pending_consensus_t::consensus, LD_DIR, networkstatus_check_consensus_signature(), networkstatus_get_flavor_name(), networkstatus_set_current_consensus(), and tor_assert.

Referenced by dirvote_act().

4.23.3.31 void dirvote_recalculate_timing (or_options_t * *options*, time_t *now*)

Set voting_schedule to hold the timing for the next vote we should be doing.

References authdir_mode_v3(), dirvote_get_start_of_next_interval(), networkstatus_t::dist_seconds, format_iso_time(), networkstatus_t::fresh_until, LD_DIR, networkstatus_get_live_consensus(), or_options_t::TestingV3AuthInitialDistDelay, or_options_t::TestingV3AuthInitialVoteDelay, or_options_t::TestingV3AuthInitialVotingInterval, tor_assert, networkstatus_t::valid_after, networkstatus_t::vote_seconds, and voting_schedule.

Referenced by dirvote_act(), networkstatus_set_current_consensus(), and options_act().

4.23.3.32 char* format_networkstatus_vote (crypto_pk_env_t * *private_signing_key*, networkstatus_t * *v3_ns*)

Return a new string containing the string representation of the vote in **v3_ns**, signed with our v3 signing key **private_signing_key**. For v3 authorities.

References networkstatus_voter_info_t::addr, networkstatus_voter_info_t::address, base16_encode(), authority_cert_t::cache_info, networkstatus_t::cert, networkstatus_t::client_versions, networkstatus_voter_info_t::contact, crypto_pk_get_fingerprint(), DIGEST_LEN, networkstatus_voter_info_t::dir_port, networkstatus_t::dist_seconds, FINGERPRINT_LEN, format_iso_time(), networkstatus_t::fresh_until, HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, INET_NTOA_BUF_LEN, networkstatus_t::known_flags, LD_BUG, networkstatus_voter_info_t::legacy_id_digest, make_consensus_method_list(), MAX_SUPPORTED_CONSENSUS_METHOD, networkstatus_t::net_params, networkstatus_parse_vote_from_string(), networkstatus_vote_free(), networkstatus_voter_info_t::nickname, note_crypto_pk_op(), networkstatus_voter_info_t::or_port, networkstatus_t::published, router_append_dirobj_signature(), router_get_networkstatus_v3_hash(), router_get_routerlist(), routerlist_t::routers, routerstatus_format_entry(), networkstatus_t::routerstatus_list, RS_ENTRY_LEN, networkstatus_t::server_versions, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, smartlist_join_strings(), tor_assert, tor_digest_is_zero(), tor_free, tor_inet_ntoa(), tor_snprintf(), networkstatus_t::type, networkstatus_t::valid_after, networkstatus_t::valid_until, networkstatus_t::vote_seconds, and networkstatus_t::voters.

Referenced by dirvote_perform_vote().

4.23.3.33 static char∗ get_detached_signatures_from_pending_consensuses (pending_consensus_t ∗ *pending*, int *n_flavors*) `[static]`

Return a newly allocated string holding a detached-signatures document for all of the in-progress consensuses in the **n_flavors**-element array at **pending**.

References networkstatus_get_detached_signatures(), smartlist_add(), smartlist_create(), and smartlist_free().

Referenced by dirvote_compute_consensuses().

4.23.3.34 static void get_frequent_members (smartlist_t ∗ *out*, smartlist_t ∗ *in*, int *min*) `[static]`

Given a sorted list of strings **in**, add every member to **out** that occurs more than **min** times.

References smartlist_add().

Referenced by compute_consensus_method(), and compute_consensus_versions_list().

4.23.3.35 static networkstatus_voter_info_t∗ get_voter (const networkstatus_t ∗ *vote*) `[static]`

Given a vote **vote** (not a consensus!), return its associated **networkstatus_voter_info_t** (p. 129).

References tor_assert, networkstatus_t::type, and networkstatus_t::voters.

Referenced by _compare_dir_src_ents_by_authority_id(), _compare_votes_by_authority_id(), dirvote_add_vote(), dirvote_get_vote(), and networkstatus_compute_consensus().

4.23.3.36 static void hash_list_members (char ∗ *digest_out*, size_t *len_out*, smartlist_t ∗ *lst*, digest_algorithm_t *alg*) `[static]`

Given a list of strings in **lst**, set the **len_out**-byte digest at **digest_out** to the hash of the concatenation of those strings, computed with the algorithm **alg**.

References crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), and crypto_new_digest_env().

4.23.3.37 static char * list_v3_auth_ids (void) [static]

Return a newly allocated string containing the hex-encoded v3 authority identity digest of every recognized v3 authority.

References DIGEST_LEN, hex_str(), router_get_trusted_dir_servers(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_digest_is_zero(), tor_free, and V3_AUTHORITY.

Referenced by dirvote_act(), and dirvote_add_vote().

4.23.3.38 static char * make_consensus_method_list (int *low*, int *high*, const char * *separator*) [static]

Return a newly allocated string holding the numbers between low and high (inclusive) that are supported consensus methods.

References consensus_method_is_supported(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_assert, tor_free, and tor_snprintf().

Referenced by dirvote_format_microdesc_vote_line(), and format_networkstatus_vote().

4.23.3.39 int networkstatus_add_detached_signatures (networkstatus_t * *target*, ns_detached_signatures_t * *sigs*, const char ** *msg_out*)

Given a consensus vote **target** and a set of detached signatures in **sigs** that correspond to the same consensus, check whether there are any new signatures in **src_voter_list** that should be added to **target**. (A signature should be added if we have no signature for that voter in **target** yet, or if we have no verifiable signature and the new signature is verifiable.) Return the number of signatures added or changed, or -1 if the document signed by **sigs** isn't the same document as **target**.

Make sure all the digests we know match, and at least one matches.

References authority_cert_get_by_digests(), document_signature_t::bad_signature, base16_encode(), crypto_digest_algorithm_get_name(), digests_t::d, DIGEST256_LEN, DIGEST_LEN, networkstatus_t::digests, ns_detached_signatures_t::digests, document_signature_dup(), document_signature_free(), networkstatus_t::flavor, ns_detached_signatures_t::fresh_until, networkstatus_t::fresh_until, document_signature_t::good_signature, HEX_DIGEST_LEN, LD_DIR, networkstatus_check_document_signature(), networkstatus_get_flavor_name(), networkstatus_get_voter_by_id(), ns_detached_signatures_t::signatures, networkstatus_voter_info_t::sigs, smartlist_add(), smartlist_remove(), strmap_get(), tor_assert, tor_mem_is_zero(), networkstatus_t::type, ns_detached_signatures_t::valid_after, networkstatus_t::valid_after, ns_detached_signatures_t::valid_until, networkstatus_t::valid_until, and voter_get_sig_by_algorithm().

Referenced by dirvote_add_signatures_to_pending_consensus().

4.23.3.40 static bw_weights_error_t networkstatus_check_weights (int64_t *Wgg*, int64_t *Wgd*, int64_t *Wmg*, int64_t *Wme*, int64_t *Wmd*, int64_t *Wee*, int64_t *Wed*, int64_t *scale*, int64_t *G*, int64_t *M*, int64_t *E*, int64_t *D*, int64_t *T*, int64_t *margin*, int *do_balance*) [static]

Verify that any weightings satisfy the balanced formulas.

References LD_DIR.

4.23.3.41 char* networkstatus_compute_consensus (smartlist_t * *votes*, int *total_authorities*, crypto_pk_env_t * *identity_key*, crypto_pk_env_t * *signing_key*, const char * *legacy_id_key_digest*, crypto_pk_env_t * *legacy_signing_key*, consensus_flavor_t *flavor*)

Given a list of vote **networkstatus_t** (p. 122) in **votes**, our public authority **identity_key**, our private authority **signing_key**, and the number of **total_authorities** that we believe exist in our voting quorum, generate the text of a new v3 consensus vote, and return the value in a newly allocated string.

Note: this function DOES NOT check whether the votes are from recognized authorities. (dirvote_add_vote does that.)

References _compare_dir_src_ents_by_authority_id(), _compare_votes_by_authority_id(), networkstatus_voter_info_t::addr, networkstatus_voter_info_t::address, base16_encode(), compute_consensus_method(), compute_consensus_versions_list(), consensus_method_is_supported(), networkstatus_voter_info_t::contact, dir_src_ent_t::digest, DIGEST_LEN, networkstatus_voter_info_t::dir_port, dirvote_compute_params(), format_iso_time(), get_voter(), HEX_DIGEST_LEN, networkstatus_voter_info_t::identity_digest, INET_NTOA_BUF_LEN, dir_src_ent_t::is_legacy, LD_DIR, networkstatus_voter_info_t::legacy_id_digest, MIN_DIST_SECONDS, MIN_METHOD_FOR_PARAMS, MIN_VOTE_INTERVAL, MIN_VOTE_SECONDS, networkstatus_get_flavor_name(), networkstatus_voter_info_t::nickname, networkstatus_voter_info_t::or_port, smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), smartlist_sort_strings(), smartlist_split_string(), smartlist_string_pos(), smartlist_uniq_strings(), sort_version_list(), strmap_get_lc(), strmap_set_lc(), tor_asprintf(), tor_assert, tor_digest_is_zero(), tor_free, tor_inet_ntoa(), dir_src_ent_t::v, and networkstatus_voter_info_t::vote_digest.

Referenced by dirvote_compute_consensuses().

4.23.3.42 static char* networkstatus_format_signatures (networkstatus_t * *consensus*, int *for_detached_signatures*) `[static]`

Return a newly allocated string containing all the signatures on **consensus** by all voters. If **for_detached_signatures** is true, then the signatures will be put in a detached signatures document, so prefix any non-NS-flavored signatures with "additional-signature" rather than "directory-signature".

References base16_encode(), base64_encode(), crypto_digest_algorithm_get_name(), DIGEST_LEN, networkstatus_t::flavor, HEX_DIGEST_LEN, networkstatus_get_flavor_name(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, tor_snprintf(), and networkstatus_t::voters.

Referenced by dirvote_add_signatures_to_pending_consensus(), and networkstatus_get_detached_signatures().

4.23.3.43 char* networkstatus_get_detached_signatures (smartlist_t * *consensuses*)

Return a newly allocated string holding the detached-signatures document corresponding to the signatures on **consensuses**, which must contain exactly one FLAV_NS consensus, and no more than one consensus for each other flavor.

References base16_encode(), crypto_digest_algorithm_get_name(), digests_t::d, DIGEST256_LEN, DIGEST_LEN, networkstatus_t::digests, format_iso_time(), networkstatus_t::fresh_until, HEX_DIGEST256_LEN, HEX_DIGEST_LEN, LD_BUG, LD_DIR, networkstatus_format_signatures(), networkstatus_get_flavor_name(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_assert, tor_free, tor_mem_is_zero(), tor_snprintf(), networkstatus_t::valid_after, and networkstatus_t::valid_until.

Referenced by get_detached_signatures_from_pending_consensuses().

4.23.3.44 void ns_detached_signatures_free (ns_detached_signatures_t * *s*)

Release all storage held in **s**.

References _tor_free(), ns_detached_signatures_t::digests, document_signature_free(), ns_detached_signatures_t::signatures, smartlist_free(), strmap_free(), and tor_free.

4.23.3.45 int vote_routerstatus_find_microdesc_hash (char * *digest256_out*, const vote_routerstatus_t * *vrs*, int *method*, digest_algorithm_t *alg*)

If **vrs** has a hash made for the consensus method **method** with the digest algorithm **alg**, decode it and copy it into **digest256_out** and return 0. Otherwise return -1.

References BASE64_DIGEST256_LEN, crypto_digest_algorithm_get_name(), digest256_from_base64(), vote_routerstatus_t::microdesc, and tor_snprintf().

4.23.3.46 document_signature_t* voter_get_sig_by_algorithm (const networkstatus_voter_info_t * *voter*, digest_algorithm_t *alg*)

Return the signature made by **voter** using the algorithm **alg**, or NULL if none is found.

References networkstatus_voter_info_t::sigs.

Referenced by networkstatus_add_detached_signatures(), and networkstatus_parse_vote_from_string().

4.23.4 Variable Documentation

4.23.4.1 time_t fetch_missing_signatures

When do we send an HTTP request for any signatures we're expecting to see on the consensus?

4.23.4.2 time_t fetch_missing_votes

When do we send an HTTP request for any votes that we haven't been posted yet?

4.23.4.3 time_t interval_starts

When do we publish the consensus?

4.23.4.4 char∗ microdesc_consensus_methods = NULL `[static]`

Cached space-separated string to hold

4.23.4.5 smartlist_t∗ pending_consensus_signature_list = NULL `[static]`

List of **ns_detached_signatures_t** (p. 131): hold signatures that get posted to us before we have generated the consensus on our own.

4.23.4.6 char∗ pending_consensus_signatures = NULL `[static]`

The detached signatures for the consensus that we're currently building.

4.23.4.7 smartlist_t∗ pending_vote_list = NULL `[static]`

List of **pending_vote_t** (p. 187) for the current vote. Before we've used them to build a consensus, the votes go here.

4.23.4.8 smartlist_t∗ previous_vote_list = NULL `[static]`

List of **pending_vote_t** (p. 187) for the previous vote. After we've used them to build a consensus, the votes go here for the next period.

4.23.4.9 time_t voting_ends

When do we give up on getting more votes and generate a consensus?

4.23.4.10 struct { ... } voting_schedule `[static]`

Scheduling information for a voting interval.

Referenced by dirvote_act(), dirvote_add_vote(), and dirvote_recalculate_timing().

4.23.4.11 time_t voting_starts

When do we generate and distribute our vote for this interval?

4.24 dns.c File Reference

Implements a local cache for DNS results for Tor servers. This is implemented as a wrapper around Adam Langley's eventdns.c code. (We can't just use gethostbyname() and friends because we really need to be nonblocking.).

`#include "or.h"`

`#include "ht.h"`

`#include <event.h>`

`#include "eventdns.h"`

Data Structures

- struct **pending_connection_t**
- struct **cached_resolve_t**

Defines

- #define **evdns_base_new**(x, y) tor_malloc(1)
- #define **evdns_base_clear_nameservers_and_suspend**(base) evdns_clear_nameservers_and_suspend()
- #define **evdns_base_search_clear**(base) evdns_search_clear()
- #define **evdns_base_set_default_outgoing_bind_address**(base, a, len) evdns_set_default_outgoing_bind_address((a),(len))
- #define **evdns_base_resolv_conf_parse**(base, options, fname) evdns_resolv_conf_parse((options), (fname))
- #define **evdns_base_count_nameservers**(base) evdns_count_nameservers()
- #define **evdns_base_resume**(base) evdns_resume()
- #define **evdns_base_config_windows_nameservers**(base) evdns_config_windows_nameservers()
- #define **evdns_base_set_option_**(base, opt, val) evdns_set_option((opt),(val),DNS_OPTIONS_ALL)
- #define **evdns_base_resolve_ipv4**(base, addr, options, cb, ptr) ((evdns_resolve_ipv4(addr, options, cb, ptr)<0) ? NULL : ((void∗)1))
- #define **evdns_base_resolve_reverse**(base, addr, options, cb, ptr) ((evdns_resolve_reverse(addr, options, cb, ptr)<0) ? NULL : ((void∗)1))
- #define **evdns_base_resolve_reverse_ipv6**(base, addr, options, cb, ptr) ((evdns_resolve_reverse_ipv6(addr, options, cb, ptr)<0) ? NULL : ((void∗)1))
- #define **MAX_ADDRESSLEN** 256
- #define **RESOLVE_MAX_TIMEOUT** 300
- #define **DNS_RESOLVE_FAILED_TRANSIENT** 1
- #define **DNS_RESOLVE_FAILED_PERMANENT** 2
- #define **DNS_RESOLVE_SUCCEEDED** 3
- #define **CACHED_RESOLVE_MAGIC** 0x1234F00D
- #define **CACHE_STATE_PENDING** 0
- #define **CACHE_STATE_DONE** 1
- #define **CACHE_STATE_CACHED_VALID** 2
- #define **CACHE_STATE_CACHED_FAILED** 3
- #define **assert_cache_ok**() STMT_NIL
- #define **SET**(k, v) evdns_base_set_option_(**the_evdns_base**, (k), (v))
- #define **N_WILDCARD_CHECKS** 2

Functions

- static void **purge_expired_resolves** (time_t now)
- static void **dns_found_answer** (const char *address, uint8_t is_reverse, uint32_t addr, const char *hostname, char outcome, uint32_t ttl)
- static void **send_resolved_cell** (**edge_connection_t** *conn, uint8_t answer_type)
- static int **launch_resolve** (**edge_connection_t** *exitconn)
- static void **add_wildcarded_test_address** (const char *address)
- static int **configure_nameservers** (int force)
- static int **answer_is_wildcarded** (const char *ip)
- static int **dns_resolve_impl** (**edge_connection_t** *exitconn, int is_resolve, **or_circuit_t** *oncirc, char **resolved_to_hostname)
- static void **assert_resolve_ok** (**cached_resolve_t** *resolve)
- static **HT_HEAD** (cache_map, **cached_resolve_t**)
- static INLINE unsigned int **cached_resolve_hash** (**cached_resolve_t** *a)
- **HT_PROTOTYPE** (HT_GENERATE(cache_map, HT_GENERATE(**cached_resolve_t**, HT_GENERATE(node, HT_GENERATE(cached_resolve_hash, HT_GENERATE(cached_resolves_eq)
- static void **evdns_log_cb** (int warn, const char *msg)
- static void **_dns_randfn** (char *b, size_t n)
- int **dns_init** (void)
- int **dns_reset** (void)
- int **has_dns_init_failed** (void)
- uint32_t **dns_clip_ttl** (uint32_t ttl)
- static uint32_t **dns_get_expiry_ttl** (uint32_t ttl)
- static void **_free_cached_resolve** (**cached_resolve_t** *r)
- static int **_compare_cached_resolves_by_expiry** (const void *_a, const void *_b)
- static void **set_expiry** (**cached_resolve_t** *resolve, time_t expires)
- void **dns_free_all** (void)
- static void **send_resolved_hostname_cell** (**edge_connection_t** *conn, const char *hostname)
- int **dns_resolve** (**edge_connection_t** *exitconn)
- void **assert_connection_edge_not_dns_pending** (**edge_connection_t** *conn)
- void **assert_all_pending_dns_resolves_ok** (void)
- void **connection_dns_remove** (**edge_connection_t** *conn)
- void **dns_cancel_pending_resolve** (const char *address)
- static void **add_answer_to_cache** (const char *address, uint8_t is_reverse, uint32_t addr, const char *hostname, char outcome, uint32_t ttl)
- static INLINE int **is_test_address** (const char *address)
- static int **evdns_err_is_transient** (int err)
- static void **evdns_callback** (int result, char type, int count, int ttl, void *addresses, void *arg)
- static void **wildcard_increment_answer** (const char *id)
- static void **evdns_wildcard_check_callback** (int result, char type, int count, int ttl, void *addresses, void *arg)
- static void **launch_wildcard_check** (int min_len, int max_len, const char *suffix)
- static void **launch_test_addresses** (int fd, short event, void *args)
- static void **dns_launch_wildcard_checks** (void)
- void **dns_launch_correctness_checks** (void)
- int **dns_seems_to_be_broken** (void)
- void **dns_reset_correctness_checks** (void)
- static int **dns_cache_entry_count** (void)
- void **dump_dns_mem_usage** (int severity)

Variables

- static struct evdns_base * **the_evdns_base** = NULL
- static int **nameservers_configured** = 0
- static int **nameserver_config_failed** = 0
- static char * **resolv_conf_fname** = NULL
- static time_t **resolv_conf_mtime** = 0
- static **smartlist_t** * **cached_resolve_pqueue** = NULL
- static int **n_wildcard_requests** = 0
- static strmap_t * **dns_wildcard_response_count** = NULL
- static **smartlist_t** * **dns_wildcard_list** = NULL
- static int **dns_wildcard_one_notice_given** = 0
- static int **dns_wildcard_notice_given** = 0
- static **smartlist_t** * **dns_wildcarded_test_address_list** = NULL
- static int **dns_wildcarded_test_address_notice_given** = 0
- static int **dns_is_completely_invalid** = 0

4.24.1 Detailed Description

Implements a local cache for DNS results for Tor servers. This is implemented as a wrapper around Adam Langley's eventdns.c code. (We can't just use gethostbyname() and friends because we really need to be nonblocking.).

4.24.2 Define Documentation

4.24.2.1 #define CACHE_STATE_CACHED_FAILED 3

We are caching a failure for this address. This should have no pending connections, and should appear in the hash table

Referenced by add_answer_to_cache(), dns_resolve_impl(), and purge_expired_resolves().

4.24.2.2 #define CACHE_STATE_CACHED_VALID 2

We are caching an answer for this address. This should have no pending connections, and should appear in the hash table.

Referenced by add_answer_to_cache(), dns_resolve_impl(), and purge_expired_resolves().

4.24.2.3 #define CACHE_STATE_DONE 1

This used to be a pending **cached_resolve_t** (p. 25), and we got an answer for it. Now we're waiting for this **cached_resolve_t** (p. 25) to expire. This should have no pending connections, and should not appear in the hash table.

Referenced by assert_resolve_ok(), dns_cancel_pending_resolve(), dns_found_answer(), dns_free_all(), dns_resolve_impl(), and purge_expired_resolves().

4.24.2.4 #define CACHE_STATE_PENDING 0

We are waiting for the resolver system to tell us an answer here. When we get one, or when we time out, the state of this **cached_resolve_t** (p. 25) will become "DONE" and we'll possibly add a CACHED_VALID or a CACHED_FAILED entry. This **cached_resolve_t** (p. 25) will be in the hash table so that we will know not to launch more requests for this addr, but rather to add more connections to the pending list for the addr.

Referenced by assert_resolve_ok(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), and purge_expired_resolves().

4.24.2.5 #define CACHED_RESOLVE_MAGIC 0x1234F00D

Value of 'magic' field for **cached_resolve_t** (p. 25). Used to try to catch bad pointers and memory stomping.

Referenced by add_answer_to_cache(), assert_resolve_ok(), and dns_resolve_impl().

4.24.2.6 #define DNS_RESOLVE_FAILED_TRANSIENT 1

Possible outcomes from hostname lookup: permanent failure, transient (retryable) failure, and success.

Referenced by add_answer_to_cache(), and evdns_callback().

4.24.2.7 #define MAX_ADDRESSLEN 256

Longest hostname we're willing to resolve.

Referenced by assert_resolve_ok(), and HT_HEAD().

4.24.2.8 #define RESOLVE_MAX_TIMEOUT 300

How long will we wait for an answer from the resolver before we decide that the resolver is wedged?

Referenced by dns_resolve_impl().

4.24.3 Function Documentation

4.24.3.1 static int _compare_cached_resolves_by_expiry (const void ∗ *_a*, const void ∗ *_b*) `[static]`

Compare two **cached_resolve_t** (p. 25) pointers by expiry time, and return less-than-zero, zero, or greater-than-zero as appropriate. Used for the priority queue implementation.

References cached_resolve_t::expire.

Referenced by purge_expired_resolves(), and set_expiry().

4.24.3.2 static void _dns_randfn (char ∗ *b*, size_t *n*) `[static]`

Helper: passed to eventdns.c as a callback so it can generate random numbers for transaction IDs and 0x20-hack coding.

References crypto_rand().

Referenced by dns_init().

4.24.3.3 static void _free_cached_resolve (cached_resolve_t * *r*) [static]

Helper: free storage held by an entry in the DNS cache.

References cached_resolve_t::is_reverse, pending_connection_t::next, cached_resolve_t::pending_connections, cached_resolve_t::result, and tor_free.

Referenced by dns_free_all().

4.24.3.4 static void add_answer_to_cache (const char * *address*, uint8_t *is_reverse*, uint32_t *addr*, const char * *hostname*, char *outcome*, uint32_t *ttl*) [static]

Helper: adds an entry to the DNS cache mapping **address** to the ipv4 address **addr** (if is_reverse is 0) or the hostname **hostname** (if is_reverse is 1). **ttl** is a cache ttl; **outcome** is one of DNS_RESOLVE_{FAILED_TRANSIENT|FAILED_PERMANENT|SUCCEEDED}.

References assert_resolve_ok(), CACHE_STATE_CACHED_FAILED, CACHE_STATE_CACHED_VALID, CACHED_RESOLVE_MAGIC, dns_get_expiry_ttl(), DNS_RESOLVE_FAILED_TRANSIENT, cached_resolve_t::is_reverse, cached_resolve_t::result, set_expiry(), cached_resolve_t::state, tor_assert, and cached_resolve_t::ttl.

Referenced by dns_found_answer().

4.24.3.5 static void add_wildcarded_test_address (const char * *address*) [static]

Note that a single test address (one believed to be good) seems to be getting redirected to the same IP as failures are.

References control_event_server_status(), dns_is_completely_invalid, dns_wildcarded_test_address_notice_given, get_options(), mark_my_descriptor_dirty(), or_options_t::ServerDNSTestAddresses, smartlist_add(), smartlist_create(), and smartlist_string_isin_case().

Referenced by evdns_callback().

4.24.3.6 static int answer_is_wildcarded (const char * *ip*) [static]

Return true iff we have noticed that the dotted-quad **ip** has been returned in response to requests for nonexistent hostnames.

References smartlist_string_isin().

Referenced by evdns_callback().

4.24.3.7 void assert_all_pending_dns_resolves_ok (void)

Log an error and abort if any connection waiting for a DNS resolve is corrupted.

References edge_connection_t::_base, assert_connection_ok(), pending_connection_t::conn, connection_in_array(), pending_connection_t::next, connection_t::s, TO_CONN, and tor_assert.

4.24.3.8 void assert_connection_edge_not_dns_pending (edge_connection_t ∗ *conn*)

Log an error and abort if conn is waiting for a DNS resolve.

References edge_connection_t::_base, connection_t::address, pending_connection_t::conn, pending_connection_t::next, cached_resolve_t::pending_connections, and tor_assert.

Referenced by connection_unlink().

4.24.3.9 static void assert_resolve_ok (cached_resolve_t ∗ *resolve*) `[static]`

Exit with an assertion if **resolve** is corrupt.

References CACHE_STATE_DONE, CACHE_STATE_PENDING, CACHED_RESOLVE_MAGIC, cached_resolve_t::is_reverse, MAX_ADDRESSLEN, cached_resolve_t::pending_connections, cached_resolve_t::result, cached_resolve_t::state, tor_assert, tor_strisnonupper(), and cached_resolve_t::ttl.

Referenced by add_answer_to_cache(), dns_found_answer(), and HT_HEAD().

4.24.3.10 static INLINE unsigned int cached_resolve_hash (cached_resolve_t ∗ *a*) `[static]`

Hash function for cached_resolve objects

4.24.3.11 static int configure_nameservers (int *force*) `[static]`

Configure eventdns nameservers if force is true, or if the configuration has changed since the last time we called this function, or if we failed on our last attempt. On Unix, this reads from /etc/resolv.conf or options->ServerDNSResolvConfFile; on Windows, this reads from options->ServerDNSResolvConfFile or the registry. Return 0 on success or -1 on failure.

References dns_servers_relaunch_checks(), evdns_log_cb(), get_options(), LD_BUG, LD_CONFIG, mark_my_descriptor_dirty(), nameserver_config_failed, nameservers_configured, or_options_t::OutboundBindAddress, resolv_conf_fname, resolv_conf_mtime, or_options_t::ServerDNSRandomizeCase, or_options_t::ServerDNSResolvConfFile, the_evdns_base, tor_addr_from_str(), tor_addr_to_sockaddr(), tor_free, and tor_libevent_get_base().

Referenced by dns_init(), dns_reset(), and launch_resolve().

4.24.3.12 void connection_dns_remove (edge_connection_t ∗ *conn*)

Remove **conn** from the list of connections waiting for conn->address.

References edge_connection_t::_base, connection_t::address, assert_connection_ok(), pending_connection_t::conn, CONN_TYPE_EXIT, escaped_safe_str(), EXIT_CONN_STATE_RESOLVING, LD_BUG, pending_connection_t::next, cached_resolve_t::pending_connections, connection_t::s, connection_t::state, TO_CONN, tor_assert, tor_free, and connection_t::type.

Referenced by connection_about_to_close_connection().

4.24.3.13 static int dns_cache_entry_count (void) [static]

Return the number of DNS cache entries as an int

Referenced by dump_dns_mem_usage().

4.24.3.14 void dns_cancel_pending_resolve (const char * *address*)

Mark all connections waiting for **address** for close. Then cancel the resolve for **address** itself, and remove any cached results for **address** from the cache.

References edge_connection_t::_base, assert_connection_ok(), CACHE_STATE_DONE, CACHE_STATE_PENDING, circuit_detach_stream(), circuit_get_by_edge_conn(), pending_connection_t::conn, connection_edge_end(), connection_free(), escaped_safe_str(), EXIT_CONN_STATE_RESOLVEFAILED, LD_BUG, connection_t::marked_for_close, pending_connection_t::next, cached_resolve_t::pending_connections, connection_t::s, connection_t::state, cached_resolve_t::state, TO_CONN, tor_assert, tor_fragile_assert, and tor_free.

Referenced by dns_resolve().

4.24.3.15 uint32_t dns_clip_ttl (uint32_t *ttl*)

Helper: Given a TTL from a DNS response, determine what TTL to give the OP that asked us to resolve it.

References MAX_DNS_TTL, and MIN_DNS_TTL.

Referenced by client_dns_set_addressmap_impl(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_connect(), send_resolved_cell(), and send_resolved_hostname_cell().

4.24.3.16 static void dns_found_answer (const char * *address*, uint8_t *is_reverse*, uint32_t *addr*, const char * *hostname*, char *outcome*, uint32_t *ttl*) [static]

Called on the OR side when a DNS worker or the eventdns library tells us the outcome of a DNS resolve: tell all pending connections about the result of the lookup, and cache the value. (**address** is a NUL-terminated string containing the address to look up; **addr** is an IPv4 address in host order; **outcome** is one of DNS_RESOLVE_{FAILED_TRANSIENT|FAILED_PERMANENT|SUCCEEDED}.

References edge_connection_t::_base, add_answer_to_cache(), connection_t::addr, edge_connection_t::address_ttl, assert_connection_ok(), assert_resolve_ok(), CACHE_STATE_DONE, CACHE_STATE_PENDING, circuit_detach_stream(), circuit_get_by_edge_conn(), CIRCUIT_IS_ORIGIN, pending_connection_t::conn, connection_edge_end(), connection_exit_connect(), connection_free(), escaped_safe_str(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_RESOLVEFAILED, EXIT_PURPOSE_CONNECT, is_test_address(), LD_BUG, or_circuit_t::n_streams, pending_connection_t::next, edge_connection_t::next_stream, edge_connection_t::on_circuit, cached_resolve_t::pending_connections, connection_t::purpose, send_resolved_cell(), send_resolved_hostname_cell(), connection_t::state, cached_resolve_t::state, TO_CONN, TO_OR_CIRCUIT(), tor_addr_from_ipv4h, tor_assert, and tor_free.

Referenced by evdns_callback().

4.24.3.17 void dns_free_all (void)

Free all storage held in the DNS cache and related structures.

References _free_cached_resolve(), CACHE_STATE_DONE, resolv_conf_fname, smartlist_free(), and tor_free.

Referenced by tor_free_all().

4.24.3.18 static uint32_t dns_get_expiry_ttl (uint32_t *ttl*) `[static]`

Helper: Given a TTL from a DNS response, determine how long to hold it in our cache.

References MAX_DNS_ENTRY_AGE, and MIN_DNS_TTL.

Referenced by add_answer_to_cache().

4.24.3.19 int dns_init (void)

Initialize the DNS subsystem; called by the OR process.

References _dns_randfn(), configure_nameservers(), get_options(), and server_mode().

Referenced by do_main_loop(), and run_scheduled_events().

4.24.3.20 void dns_launch_correctness_checks (void)

If appropriate, start testing whether our DNS servers tend to lie to us.

References dns_launch_wildcard_checks(), get_options(), launch_test_addresses(), LD_BUG, tor_evtimer_new(), and tor_libevent_get_base().

Referenced by run_scheduled_events().

4.24.3.21 static void dns_launch_wildcard_checks (void) `[static]`

Launch DNS requests for a few nonexistent hostnames and a few well-known hostnames, and see if we can catch our nameserver trying to hijack them and map them to a stupid "I couldn't find ggoogle.com but maybe you'd like to buy these lovely encyclopedias" page.

References launch_wildcard_check().

Referenced by dns_launch_correctness_checks().

4.24.3.22 int dns_reset (void)

Called when DNS-related options change (or may have changed). Returns -1 on failure, 0 on success.

References configure_nameservers(), get_options(), LD_BUG, nameservers_configured, resolv_conf_fname, resolv_conf_mtime, server_mode(), the_evdns_base, tor_free, and tor_libevent_get_base().

Referenced by do_hup(), and options_act().

4.24.3.23 void dns_reset_correctness_checks (void)

Forget what we've previously learned about our DNS servers' correctness.

References _tor_free(), dns_is_completely_invalid, dns_wildcard_notice_given, dns_wildcard_one_notice_given, dns_wildcard_response_count, dns_wildcarded_test_address_notice_given, n_wildcard_requests, smartlist_clear(), strmap_free(), and tor_free.

Referenced by dns_servers_relaunch_checks().

4.24.3.24 int dns_resolve (edge_connection_t * *exitconn*)

See if we have a cache entry for **exitconn**->address. if so, if resolve valid, put it into **exitconn**->addr and return 1. If resolve failed, free exitconn and return -1.

(For EXIT_PURPOSE_RESOLVE connections, send back a RESOLVED error cell on returning -1. For EXIT_PURPOSE_CONNECT connections, there's no need to send back an END cell, since connection_exit_begin_conn will do that for us.)

If we have a cached answer, send the answer back along **exitconn**'s circuit.

Else, if seen before and pending, add conn to the pending list, and return 0.

Else, if not seen before, add conn to pending list, hand to dns farm, and return 0.

Exitconn's on_circuit field must be set, but exitconn should not yet be linked onto the n_streams/resolving_streams list of that circuit. On success, link the connection to n_streams if it's an exit connection. On "pending", link the connection to resolving streams. Otherwise, clear its on_circuit field.

References edge_connection_t::_base, connection_t::address, connection_free(), dns_cancel_pending_resolve(), dns_resolve_impl(), EXIT_CONN_STATE_RESOLVING, EXIT_PURPOSE_RESOLVE, connection_t::marked_for_close, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, connection_t::purpose, or_circuit_t::resolving_streams, send_resolved_cell(), send_resolved_hostname_cell(), connection_t::state, TO_CONN, TO_OR_CIRCUIT(), tor_assert, and tor_free.

Referenced by connection_exit_begin_conn(), and connection_exit_begin_resolve().

4.24.3.25 static int dns_resolve_impl (edge_connection_t * *exitconn*, int *is_resolve*, or_circuit_t * *oncirc*, char ** *hostname_out*) `[static]`

Helper function for dns_resolve: same functionality, but does not handle:

- marking connections on error and clearing their on_circuit
- linking connections to n_streams/resolving_streams,
- sending resolved cells if we have an answer/error right away,

Return -2 on a transient error. If it's a reverse resolve and it's successful, sets *__hostname_out__ to a newly allocated string holding the cached reverse DNS value.

References edge_connection_t::_base, connection_t::addr, connection_t::address, address_is_invalid_destination(), edge_connection_t::address_ttl, assert_connection_ok(), CACHE_STATE_CACHED_FAILED, CACHE_STATE_CACHED_VALID, CACHE_STATE_DONE, CACHE_STATE_PENDING, CACHED_RESOLVE_MAGIC, pending_connection_t::conn, DEFAULT_DNS_TTL, escaped_safe_str(), routerinfo_t::exit_policy, cached_resolve_t::expire, cached_resolve_t::is_reverse, launch_resolve(), LD_BUG, cached_resolve_t::minheap_idx, pending_connection_t::next, cached_resolve_t::pending_connections, policy_is_reject_star(), purge_expired_resolves(), RESOLVE_MAX_TIMEOUT, cached_resolve_t::result, router_get_my_routerinfo(), connection_t::s, set_expiry(), cached_resolve_t::state, TO_CONN, tor_addr_copy(), tor_addr_family(), tor_addr_from_ipv4h, tor_addr_from_str(), tor_addr_is_internal(), tor_addr_parse_reverse_lookup_name(), tor_assert, tor_fragile_assert, tor_strlower(), and cached_resolve_t::ttl.

Referenced by dns_resolve().

4.24.3.26 int dns_seems_to_be_broken (void)

Return true iff our DNS servers lie to us too much to be trusted.

References dns_is_completely_invalid.

Referenced by router_dump_router_to_string().

4.24.3.27 void dump_dns_mem_usage (int *severity*)

Log memory information about our internal DNS cache at level 'severity'.

References dns_cache_entry_count(), and LD_MM.

Referenced by dumpmemusage().

4.24.3.28 static void evdns_callback (int *result*, char *type*, int *count*, int *ttl*, void * *addresses*, void * *arg*) [static]

For eventdns: Called when we get an answer for a request we launched. See **eventdns.h** (p. ??) for arguments; 'arg' holds the address we tried to resolve.

References add_wildcarded_test_address(), answer_is_wildcarded(), dns_found_answer(), DNS_RESOLVE_FAILED_TRANSIENT, esc_for_log(), escaped_safe_str(), evdns_err_is_transient(), INET_NTOA_BUF_LEN, is_test_address(), LD_BUG, safe_str(), tor_free, and tor_inet_ntoa().

Referenced by launch_resolve(), and launch_test_addresses().

4.24.3.29 static int evdns_err_is_transient (int *err*) [static]

Eventdns helper: return true iff the eventdns result **err** is a transient failure.

Referenced by evdns_callback().

4.24.3.30 static void evdns_log_cb (int *warn*, const char * *msg*) [static]

Helper: called by eventdns when eventdns wants to log something.

References control_event_server_status(), escaped(), get_options(), strcmpstart(), tor_assert, and tor_free.

Referenced by configure_nameservers().

4.24.3.31 static void evdns_wildcard_check_callback (int *result*, char *type*, int *count*, int *ttl*, void ∗ *addresses*, void ∗ *arg*) `[static]`

Callback function when we get an answer (possibly failing) for a request for a (hopefully) nonexistent domain.

References dns_wildcard_one_notice_given, dns_wildcard_response_count, INET_NTOA_BUF_LEN, n_wildcard_requests, strmap_size(), tor_free, tor_inet_ntoa(), and wildcard_increment_answer().

Referenced by launch_wildcard_check().

4.24.3.32 int has_dns_init_failed (void)

Return true iff the most recent attempt to initialize the DNS subsystem failed.

References nameserver_config_failed.

Referenced by router_dump_router_to_string(), and run_scheduled_events().

4.24.3.33 static HT_HEAD (cache_map, cached_resolve_t) `[static]`

Hash table of cached_resolve objects.

References assert_resolve_ok(), and MAX_ADDRESSLEN.

4.24.3.34 HT_PROTOTYPE (HT_GENERATE(*cache_map*, HT_GENERATE(*cached_resolve_t*, HT_GENERATE(*node*, HT_GENERATE(*cached_resolve_hash*, HT_GENERATE(*cached_resolves_eq*)

Initialize the DNS cache.

4.24.3.35 static INLINE int is_test_address (const char ∗ *address*) `[static]`

Return true iff **address** is one of the addresses we use to verify that well-known sites aren't being hijacked by our DNS servers.

References get_options(), or_options_t::ServerDNSTestAddresses, and smartlist_string_isin_case().

Referenced by dns_found_answer(), and evdns_callback().

4.24.3.36 static int launch_resolve (edge_connection_t ∗ *exitconn*) `[static]`

For eventdns: start resolving as necessary to find the target for **exitconn**. Returns -1 on error, -2 on transient error, 0 on "resolve launched."

References edge_connection_t::_base, connection_t::address, configure_nameservers(), escaped_safe_str(), evdns_callback(), get_options(), LD_BUG, nameservers_configured, or_options_t::ServerDNSSearchDomains, the_evdns_base, tor_addr_family(), tor_addr_parse_reverse_lookup_name(), tor_addr_to_in(), tor_addr_to_in6(), tor_assert, and tor_free.

Referenced by dns_resolve_impl().

4.24.3.37 static void launch_test_addresses (int *fd*, short *event*, void ∗ *args*) `[static]`

Launch attempts to resolve a bunch of known-good addresses (configured in ServerDNSTestAddresses). [Callback for a libevent timer]

References escaped_safe_str(), evdns_callback(), get_options(), or_options_t::ServerDNSTestAddresses, the_evdns_base, tor_assert, and tor_free.

Referenced by dns_launch_correctness_checks().

4.24.3.38 static void launch_wildcard_check (int *min_len*, int *max_len*, const char ∗ *suffix*) `[static]`

Launch a single request for a nonexistent hostname consisting of between **min_len** and **max_len** random (plausible) characters followed by **suffix**

References crypto_random_hostname(), evdns_wildcard_check_callback(), the_evdns_base, tor_assert, and tor_free.

Referenced by dns_launch_wildcard_checks().

4.24.3.39 static void purge_expired_resolves (time_t *now*) `[static]`

Remove every cached_resolve whose **expire** time is before or equal to **now** from the cache.

References edge_connection_t::_base, _compare_cached_resolves_by_expiry(), CACHE_STATE_CACHED_FAILED, CACHE_STATE_CACHED_VALID, CACHE_STATE_DONE, CACHE_STATE_PENDING, circuit_detach_stream(), circuit_get_by_edge_conn(), pending_connection_t::conn, connection_edge_end(), connection_free(), escaped_safe_str(), cached_resolve_t::expire, cached_resolve_t::is_reverse, LD_BUG, pending_connection_t::next, cached_resolve_t::pending_connections, cached_resolve_t::result, connection_t::s, smartlist_pqueue_pop(), cached_resolve_t::state, STRUCT_OFFSET, TO_CONN, tor_assert, tor_fragile_assert, and tor_free.

Referenced by dns_resolve_impl().

4.24.3.40 static void send_resolved_cell (edge_connection_t ∗ *conn*, uint8_t *answer_type*) `[static]`

Send a response to the RESOLVE request of a connection. **answer_type** must be one of RESOLVED_TYPE_(IPV4|ERROR|ERROR_TRANSIENT).

If **circ** is provided, and we have a cached answer, send the answer back along circ; otherwise, send the answer back along **conn**'s attached circuit.

References edge_connection_t::_base, connection_t::addr, edge_connection_t::address_ttl, connection_edge_send_command(), dns_clip_ttl(), RELAY_PAYLOAD_SIZE, set_uint32(), tor_addr_to_ipv4n(), and tor_assert.

Referenced by dns_found_answer(), and dns_resolve().

4.24.3.41 static void send_resolved_hostname_cell (edge_connection_t * *conn*, const char * *hostname*) [static]

Send a response to the RESOLVE request of a connection for an in-addr.arpa address on connection **conn** which yielded the result **hostname**. The answer type will be RESOLVED_HOSTNAME.

If **circ** is provided, and we have a cached answer, send the answer back along circ; otherwise, send the answer back along **conn**'s attached circuit.

References edge_connection_t::address_ttl, connection_edge_send_command(), dns_clip_ttl(), RELAY_PAYLOAD_SIZE, set_uint32(), and tor_assert.

Referenced by dns_found_answer(), and dns_resolve().

4.24.3.42 static void set_expiry (cached_resolve_t * *resolve*, time_t *expires*) [static]

Set an expiry time for a **cached_resolve_t** (p. 25), and add it to the expiry priority queue

References _compare_cached_resolves_by_expiry(), cached_resolve_t::expire, smartlist_create(), smartlist_pqueue_add(), STRUCT_OFFSET, and tor_assert.

Referenced by add_answer_to_cache(), and dns_resolve_impl().

4.24.3.43 static void wildcard_increment_answer (const char * *id*) [static]

Called when we see **id** (a dotted quad) in response to a request for a hopefully bogus address.

References control_event_server_status(), dns_wildcard_notice_given, dns_wildcard_response_count, n_wildcard_requests, smartlist_add(), smartlist_create(), smartlist_string_isin(), strmap_get(), and strmap_set().

Referenced by evdns_wildcard_check_callback().

4.24.4 Variable Documentation

4.24.4.1 smartlist_t* cached_resolve_pqueue = NULL [static]

Priority queue of **cached_resolve_t** (p. 25) objects to let us know when they will expire.

4.24.4.2 int dns_is_completely_invalid = 0 [static]

True iff all addresses seem to be getting wildcarded.

Referenced by add_wildcarded_test_address(), dns_reset_correctness_checks(), and dns_seems_to_be_broken().

4.24.4.3 smartlist_t* dns_wildcard_list = NULL [static]

If present, a list of dotted-quad IP addresses that we are pretty sure our nameserver wants to return in response to requests for nonexistent domains.

4.24.4.4 int dns_wildcard_notice_given = 0 [static]

True iff we've warned that our DNS server is wildcarding too many failures.

Referenced by dns_reset_correctness_checks(), and wildcard_increment_answer().

4.24.4.5 int dns_wildcard_one_notice_given = 0 [static]

True iff we've logged about a single address getting wildcarded. Subsequent warnings will be less severe.

Referenced by dns_reset_correctness_checks(), and evdns_wildcard_check_callback().

4.24.4.6 strmap_t* dns_wildcard_response_count = NULL [static]

Map from dotted-quad IP address in response to an int holding how many times we've seen it for a randomly generated (hopefully bogus) address. It would be easier to use definitely-invalid addresses (as specified by RFC2606), but see comment in **dns_launch_wildcard_checks()** (p. 634).

Referenced by dns_reset_correctness_checks(), evdns_wildcard_check_callback(), and wildcard_increment_answer().

4.24.4.7 smartlist_t* dns_wildcarded_test_address_list = NULL [static]

List of supposedly good addresses that are getting wildcarded to the same addresses as nonexistent addresses.

4.24.4.8 int dns_wildcarded_test_address_notice_given = 0 [static]

True iff we've warned about a test address getting wildcarded

Referenced by add_wildcarded_test_address(), and dns_reset_correctness_checks().

4.24.4.9 int n_wildcard_requests = 0 [static]

How many requests for bogus addresses have we launched so far?

Referenced by dns_reset_correctness_checks(), evdns_wildcard_check_callback(), and wildcard_increment_answer().

4.24.4.10 int nameserver_config_failed = 0 [static]

Did our most recent attempt to configure nameservers with eventdns fail?

Referenced by configure_nameservers(), and has_dns_init_failed().

4.24.4.11 int nameservers_configured = 0 [static]

Have we currently configured nameservers with eventdns?

Referenced by configure_nameservers(), dns_reset(), and launch_resolve().

4.24.4.12 char* resolv_conf_fname = NULL [static]

What was the resolv_conf fname we last used when configuring the nameservers? Used to check whether we need to reconfigure.

Referenced by configure_nameservers(), dns_free_all(), and dns_reset().

4.24.4.13 time_t resolv_conf_mtime = 0 [static]

What was the mtime on the resolv.conf file we last used when configuring the nameservers? Used to check whether we need to reconfigure.

Referenced by configure_nameservers(), and dns_reset().

4.24.4.14 struct evdns_base* the_evdns_base = NULL [static]

Our evdns_base; this structure handles all our name lookups.

Referenced by configure_nameservers(), dns_reset(), launch_resolve(), launch_test_addresses(), and launch_wildcard_check().

4.25 dnsserv.c File Reference

Implements client-side DNS proxy server code. Note: this is the DNS Server code, not the Server DNS code. Confused? This code runs on client-side, and acts as a DNS server. The code in **dns.c** (p. 627), on the other hand, runs on Tor servers, and acts as a DNS client.

`#include "or.h"`

`#include "eventdns.h"`

Functions

- static void **evdns_server_callback** (struct evdns_server_request *req, void *_data)
- int **dnsserv_launch_request** (const char *name, int reverse)
- void **dnsserv_reject_request** (**edge_connection_t** *conn)
- static const char * **evdns_get_orig_address** (const struct evdns_server_request *req, int rtype, const char *addr)
- void **dnsserv_resolved** (**edge_connection_t** *conn, int answer_type, size_t answer_len, const char *answer, int ttl)
- void **dnsserv_configure_listener** (**connection_t** *conn)
- void **dnsserv_close_listener** (**connection_t** *conn)

4.25.1 Detailed Description

Implements client-side DNS proxy server code. Note: this is the DNS Server code, not the Server DNS code. Confused? This code runs on client-side, and acts as a DNS server. The code in **dns.c** (p. 627), on the other hand, runs on Tor servers, and acts as a DNS client.

4.25.2 Function Documentation

4.25.2.1 void dnsserv_close_listener (connection_t * *conn*)

Free the evdns server port for **conn**, which must be an AP_DNS_LISTENER.

References CONN_TYPE_AP_DNS_LISTENER, connection_t::dns_server_port, tor_assert, and connection_t::type.

Referenced by connection_unregister_events().

4.25.2.2 void dnsserv_configure_listener (connection_t * *conn*)

Set up the evdns server port for the UDP socket on **conn**, which must be an AP_DNS_LISTENER

References CONN_TYPE_AP_DNS_LISTENER, connection_t::dns_server_port, evdns_server_callback(), connection_t::s, tor_assert, and connection_t::type.

Referenced by connection_create_listener().

4.25.2.3 int dnsserv_launch_request (const char * *name*, int *reverse*)

Helper function: called whenever the client sends a resolve request to our controller. We need to eventually answer the request **req**. Returns 0 if the controller will be getting (or has gotten) an event in response; -1 if we couldn't launch the request.

References edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_RESOLVE_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_add(), connection_ap_handshake_rewrite_and_attach(), connection_free(), edge_connection_new(), escaped_safe_str_client(), edge_connection_t::is_dns_request, LD_APP, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, edge_connection_t::socks_request, connection_t::state, TO_CONN, and tor_free.

Referenced by handle_control_resolve().

4.25.2.4 void dnsserv_reject_request (edge_connection_t * *conn*)

If there is a pending request on **conn** that's waiting for an answer, send back an error and free the request.

References edge_connection_t::dns_server_request.

Referenced by connection_about_to_close_connection().

4.25.2.5 void dnsserv_resolved (edge_connection_t * *conn*, int *answer_type*, size_t *answer_len*, const char * *answer*, int *ttl*)

Tell the dns request waiting for an answer on **conn** that we have an answer of type **answer_type** (RESOLVE_TYPE_IPV4/IPV6/ERR), of length **answer_len**, in **answer**, with TTL **ttl**. Doesn't do any caching; that's handled elsewhere.

References socks_request_t::address, socks_request_t::command, edge_connection_t::dns_server_request, evdns_get_orig_address(), LD_APP, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, edge_connection_t::socks_request, and tor_free.

Referenced by connection_ap_handshake_socks_resolved().

4.25.2.6 static const char* evdns_get_orig_address (const struct evdns_server_request * *req*, int *rtype*, const char * *addr*) `[static]`

Look up the original name that corresponds to 'addr' in req. We use this to preserve case in order to facilitate people using 0x20-hacks to avoid DNS poisoning.

References tor_fragile_assert.

Referenced by dnsserv_resolved().

4.25.2.7 static void evdns_server_callback (struct evdns_server_request * *req*, void * *_data*) `[static]`

Helper function: called by evdns whenever the client sends a request to our DNSPort. We need to eventually answer the request **req**.

References edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_RESOLVE_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_add(), connection_ap_handshake_rewrite_and_attach(), connection_free(), control_event_stream_status(), edge_connection_t::dns_server_request, edge_connection_new(), escaped_safe_str_client(), edge_connection_t::is_dns_request, LD_APP, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, socks_policy_permits_address(), edge_connection_t::socks_request, connection_t::state, TO_CONN, tor_addr_copy(), tor_addr_from_sockaddr(), tor_assert, tor_dup_addr(), and tor_free.

Referenced by dnsserv_configure_listener().

4.26 geoip.c File Reference

Functions related to maintaining an IP-to-country database and to summarizing client connections by country.

`#include "or.h"`

`#include "ht.h"`

Data Structures

- struct **geoip_entry_t**
- struct **geoip_country_t**
- struct **clientmap_entry_t**
- struct **c_hist_t**
- struct **dirreq_map_entry_t**

Defines

- #define **GEOIP_PRIVATE**
- #define **REQUEST_HIST_LEN** 1
- #define **REQUEST_HIST_PERIOD** (24*60*60)
- #define **ACTION_MASK** 3
- #define **REQUEST_SHARE_INTERVAL** (15 * 60)
- #define **MIN_IPS_TO_NOTE_COUNTRY** 1
- #define **MIN_IPS_TO_NOTE_ANYTHING** 1
- #define **IP_GRANULARITY** 8
- #define **DIRREQ_TIMEOUT** (10*60)
- #define **DIR_REQ_GRANULARITY** 4
- #define **MIN_DIR_REQ_RESPONSES** 16
- #define **GEOIP_MIN_OBSERVATION_TIME** (12*60*60)
- #define **RESPONSE_GRANULARITY** 8

Functions

- static void **clear_geoip_db** (void)
- **country_t geoip_get_country** (const char *country)
- static void **geoip_add_entry** (uint32_t low, uint32_t high, const char *country)
- int **geoip_parse_entry** (const char *line)
- static int **_geoip_compare_entries** (const void **_a, const void **_b)
- static int **_geoip_compare_key_to_entry** (const void *_key, const void **_member)
- int **should_record_bridge_info** (**or_options_t** *options)
- int **geoip_load_file** (const char *filename, **or_options_t** *options)
- int **geoip_get_country_by_ip** (uint32_t ipaddr)
- int **geoip_get_n_countries** (void)
- const char * **geoip_get_country_name** (**country_t** num)
- int **geoip_is_loaded** (void)
- static **HT_HEAD** (clientmap, **clientmap_entry_t**)
- static INLINE int **clientmap_entries_eq** (const **clientmap_entry_t** *a, const **clientmap_entry_t** *b)

- **HT_PROTOTYPE** (clientmap, **clientmap_entry_t**, node, clientmap_entry_hash, clientmap_entries_eq)
- **HT_GENERATE** (clientmap, **clientmap_entry_t**, node, clientmap_entry_hash, clientmap_entries_eq, 0.6, malloc, realloc, free)
- static void **geoip_determine_shares** (time_t now)
- static int **geoip_get_mean_shares** (time_t now, double ∗v2_share_out, double ∗v3_share_out)
- static void **rotate_request_period** (void)
- void **geoip_note_client_seen** (**geoip_client_action_t** action, uint32_t addr, time_t now)
- static int **_remove_old_client_helper** (struct **clientmap_entry_t** ∗ent, void ∗_cutoff)
- void **geoip_remove_old_clients** (time_t cutoff)
- void **geoip_note_ns_response** (**geoip_client_action_t** action, **geoip_ns_response_t** response)
- time_t **geoip_get_history_start** (void)
- static int **_c_hist_compare** (const void ∗∗_a, const void ∗∗_b)
- static **HT_HEAD** (dirreqmap, **dirreq_map_entry_t**)
- static unsigned **dirreq_map_ent_hash** (const **dirreq_map_entry_t** ∗entry)
- **HT_PROTOTYPE** (dirreqmap, **dirreq_map_entry_t**, node, dirreq_map_ent_hash, dirreq_map_ent_eq)
- **HT_GENERATE** (dirreqmap, **dirreq_map_entry_t**, node, dirreq_map_ent_hash, dirreq_map_ent_eq, 0.6, malloc, realloc, free)
- static void **_dirreq_map_put** (**dirreq_map_entry_t** ∗entry, **dirreq_type_t** type, uint64_t dirreq_id)
- static **dirreq_map_entry_t** ∗ **_dirreq_map_get** (**dirreq_type_t** type, uint64_t dirreq_id)
- void **geoip_start_dirreq** (uint64_t dirreq_id, size_t response_size, **geoip_client_action_t** action, **dirreq_type_t** type)
- void **geoip_change_dirreq_state** (uint64_t dirreq_id, **dirreq_type_t** type, **dirreq_state_t** new_state)
- static char ∗ **geoip_get_dirreq_history** (**geoip_client_action_t** action, **dirreq_type_t** type)
- static char ∗ **geoip_get_client_history** (time_t now, **geoip_client_action_t** action, int min_observation_time, unsigned granularity)
- char ∗ **geoip_get_client_history_dirreq** (time_t now, **geoip_client_action_t** action)
- char ∗ **geoip_get_client_history_bridge** (time_t now, **geoip_client_action_t** action)
- char ∗ **geoip_get_request_history** (time_t now, **geoip_client_action_t** action)
- void **geoip_dirreq_stats_init** (time_t now)
- void **geoip_dirreq_stats_write** (time_t now)
- void **geoip_bridge_stats_init** (time_t now)
- static char ∗ **parse_bridge_stats_controller** (const char ∗stats_str, time_t now)
- time_t **geoip_bridge_stats_write** (time_t now)
- static void **load_bridge_stats** (time_t now)
- const char ∗ **geoip_get_bridge_stats_extrainfo** (time_t now)
- const char ∗ **geoip_get_bridge_stats_controller** (time_t now)
- void **geoip_entry_stats_init** (time_t now)
- void **geoip_entry_stats_write** (time_t now)
- int **getinfo_helper_geoip** (**control_connection_t** ∗control_conn, const char ∗question, char ∗∗answer)
- void **geoip_free_all** (void)

Variables

- static **smartlist_t** ∗ **geoip_countries** = NULL
- static strmap_t ∗ **country_idxplus1_by_lc_code** = NULL
- static **smartlist_t** ∗ **geoip_entries** = NULL
- static time_t **last_time_determined_shares** = 0
- static double **v2_share_times_seconds**
- static double **v3_share_times_seconds**
- static int **share_seconds**
- static uint32_t **ns_v2_responses** [GEOIP_NS_RESPONSE_NUM]
- static uint32_t **ns_v3_responses** [GEOIP_NS_RESPONSE_NUM]
- static time_t **start_of_dirreq_stats_interval**
- static time_t **start_of_bridge_stats_interval**
- static char ∗ **bridge_stats_extrainfo** = NULL
- static char ∗ **bridge_stats_controller** = NULL
- static time_t **start_of_entry_stats_interval**

4.26.1 Detailed Description

Functions related to maintaining an IP-to-country database and to summarizing client connections by country.

4.26.2 Define Documentation

4.26.2.1 #define DIRREQ_TIMEOUT (10∗60)

When there are incomplete directory requests at the end of a 24-hour period, consider those requests running for longer than this timeout as failed, the others as still running.

Referenced by geoip_get_dirreq_history().

4.26.2.2 #define GEOIP_MIN_OBSERVATION_TIME (12∗60∗60)

How long do we have to have observed per-country request history before we are willing to talk about it?

Referenced by geoip_get_client_history_bridge(), and geoip_get_request_history().

4.26.2.3 #define IP_GRANULARITY 8

When reporting geoip data about countries, round up to the nearest multiple of this value.

Referenced by geoip_get_client_history_bridge(), and geoip_get_request_history().

4.26.2.4 #define MIN_IPS_TO_NOTE_ANYTHING 1

Do not report any geoip data at all if we have fewer than this number of IPs to report about.

Referenced by geoip_get_client_history().

4.26.2.5 #define MIN_IPS_TO_NOTE_COUNTRY 1

Do not mention any country from which fewer than this number of IPs have connected. This conceivably avoids reporting information that could deanonymize users, though analysis is lacking.

Referenced by geoip_get_client_history().

4.26.2.6 #define REQUEST_HIST_LEN 1

For how many periods should we remember per-country request history?

Referenced by geoip_get_request_history(), and geoip_note_client_seen().

4.26.2.7 #define REQUEST_HIST_PERIOD (24*60*60)

How long are the periods for which we should remember request history?

Referenced by geoip_note_client_seen().

4.26.2.8 #define REQUEST_SHARE_INTERVAL (15 * 60)

How often do we update our estimate which share of v2 and v3 directory requests is sent to us? We could as well trigger updates of shares from network status updates, but that means adding a lot of calls into code that is independent from geoip stats (and keeping them up-to-date). We are perfectly fine with an approximation of 15-minute granularity.

Referenced by geoip_note_client_seen().

4.26.3 Function Documentation

4.26.3.1 static int _c_hist_compare (const void ** _*a*, const void ** _*b*) `[static]`

Sorting helper: return -1, 1, or 0 based on comparison of two **geoip_entry_t** (p. 102). Sort in descending order of total, and then by country code.

References c_hist_t::country, and c_hist_t::total.

Referenced by geoip_get_client_history(), and geoip_get_request_history().

4.26.3.2 static dirreq_map_entry_t* _dirreq_map_get (dirreq_type_t *type*, uint64_t *dirreq_id*) `[static]`

Helper: Look up and return an entry in the map of directory requests using **tunneled** and **dirreq_id** as key parts. If there is no such entry, return NULL.

Referenced by geoip_change_dirreq_state().

4.26.3.3 static void _dirreq_map_put (dirreq_map_entry_t * *entry*, dirreq_type_t *type*, uint64_t *dirreq_id*) `[static]`

Helper: Put **entry** into map of directory requests using **tunneled** and **dirreq_id** as key parts. If there is already an entry for that key, print out a BUG warning and return.

References LD_BUG, and tor_assert.

Referenced by geoip_start_dirreq().

4.26.3.4 static int _geoip_compare_entries (const void ** *_a*, const void ** *_b*) [static]

Sorting helper: return -1, 1, or 0 based on comparison of two **geoip_entry_t** (p. 102)

References geoip_entry_t::ip_low.

Referenced by geoip_load_file().

4.26.3.5 static int _geoip_compare_key_to_entry (const void * *_key*, const void ** *_member*) [static]

bsearch helper: return -1, 1, or 0 based on comparison of an IP (a pointer to a uint32_t in host order) to a **geoip_entry_t** (p. 102)

References geoip_entry_t::ip_high.

Referenced by geoip_get_country_by_ip().

4.26.3.6 static int _remove_old_client_helper (struct clientmap_entry_t * *ent*, void * *_cutoff*) [static]

HT_FOREACH helper: remove a **clientmap_entry_t** (p. 46) from the hashtable if it's older than a certain time.

References tor_free.

Referenced by geoip_remove_old_clients().

4.26.3.7 static void clear_geoip_db (void) [static]

Release all storage held by the GeoIP database.

References country_idxplus1_by_lc_code, smartlist_free(), strmap_free(), and tor_free.

Referenced by geoip_free_all(), and geoip_load_file().

4.26.3.8 static INLINE int clientmap_entries_eq (const clientmap_entry_t * *a*, const clientmap_entry_t * *b*) [static]

Hashtable helper: compare two **clientmap_entry_t** (p. 46) values for equality.

4.26.3.9 static void geoip_add_entry (uint32_t *low*, uint32_t *high*, const char * *country*) [static]

Add an entry to the GeoIP table, mapping all IPs between **low** and **high**, inclusive, to the 2-letter country code **country**.

References geoip_entry_t::country, country_idxplus1_by_lc_code, geoip_country_t::countrycode, geoip_entry_t::ip_high, geoip_entry_t::ip_low, smartlist_add(), strmap_get_lc(), strmap_set_lc(), tor_assert, and tor_strlower().

Referenced by geoip_parse_entry().

4.26.3.10 void geoip_bridge_stats_init (time_t *now*)

Initialize bridge stats.

Referenced by run_scheduled_events().

4.26.3.11 time_t geoip_bridge_stats_write (time_t *now*)

Write bridge statistics to $DATADIR/stats/bridge-stats and return when we should next try to write statistics.

References check_private_dir(), control_event_clients_seen(), DIR_ENTRY_RECORD_USAGE_RETAIN_IPS, format_iso_time(), GEOIP_CLIENT_CONNECT, geoip_get_client_history_bridge(), geoip_remove_old_clients(), get_datadir_fname, get_datadir_fname2, parse_bridge_stats_controller(), tor_free, tor_snprintf(), and write_str_to_file().

Referenced by run_scheduled_events().

4.26.3.12 void geoip_change_dirreq_state (uint64_t *dirreq_id*, dirreq_type_t *type*, dirreq_state_t *new_state*)

Change the state of the either direct or tunneled (see **type**) directory request with **dirreq_id** to **new_state** and possibly mark it as completed. If no entry can be found for the given key parts (e.g., if this is a directory request that we are not measuring, or one that was started in the previous measurement period), or if the state cannot be advanced to **new_state**, do nothing.

References _dirreq_map_get(), DIRREQ_FLUSHING_DIR_CONN_FINISHED, DIRREQ_IS_FOR_NETWORK_STATUS, DIRREQ_OR_CONN_BUFFER_FLUSHED, get_options(), and tor_gettimeofday().

Referenced by connection_dir_finished_flushing(), connection_handle_write_impl(), connection_or_flush_from_first_active_circuit(), and relay_send_command_from_edge().

4.26.3.13 static void geoip_determine_shares (time_t *now*) `[static]`

Try to determine which fraction of v2 and v3 directory requests aimed at caches will be sent to us at time **now** and store that value in order to take a mean value later on.

References last_time_determined_shares, router_get_my_share_of_directory_requests(), share_seconds, v2_share_times_seconds, and v3_share_times_seconds.

Referenced by geoip_get_mean_shares(), and geoip_note_client_seen().

4.26.3.14 void geoip_dirreq_stats_init (time_t *now*)

Initialize directory request stats.

Referenced by run_scheduled_events().

4.26.3.15 void geoip_dirreq_stats_write (time_t *now*)

Write dirreq statistics to $DATADIR/stats/dirreq-stats.

References abort_writing_to_file(), check_private_dir(), finish_writing_to_file(), format_iso_time(), GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, geoip_get_client_history_dirreq(), geoip_get_dirreq_history(), geoip_get_mean_shares(), geoip_get_request_history(), GEOIP_REJECT_BUSY, GEOIP_REJECT_NOT_ENOUGH_SIGS, GEOIP_REJECT_NOT_FOUND, GEOIP_REJECT_NOT_MODIFIED, GEOIP_REJECT_UNAVAILABLE, geoip_remove_old_clients(), GEOIP_SUCCESS, get_datadir_fname, get_datadir_fname2, get_options(), round_uint32_to_next_multiple_of(), start_writing_to_stdio_file(), and tor_free.

Referenced by run_scheduled_events().

4.26.3.16 void geoip_entry_stats_init (time_t *now*)

Initialize entry stats.

Referenced by run_scheduled_events().

4.26.3.17 void geoip_entry_stats_write (time_t *now*)

Write entry statistics to $DATADIR/stats/entry-stats.

References abort_writing_to_file(), check_private_dir(), finish_writing_to_file(), format_iso_time(), GEOIP_CLIENT_CONNECT, geoip_get_client_history_dirreq(), geoip_remove_old_clients(), get_datadir_fname, get_datadir_fname2, get_options(), start_writing_to_stdio_file(), and tor_free.

Referenced by run_scheduled_events().

4.26.3.18 void geoip_free_all (void)

Release all storage held in this file.

References clear_geoip_db(), and tor_free.

Referenced by tor_free_all().

4.26.3.19 const char∗ geoip_get_bridge_stats_controller (time_t *now*)

Return most recent bridge statistics to be returned to controller clients, or NULL if we don't have recent bridge statistics.

References load_bridge_stats().

Referenced by getinfo_helper_events().

4.26.3.20 const char∗ geoip_get_bridge_stats_extrainfo (time_t *now*)

Return most recent bridge statistics for inclusion in extra-info descriptors, or NULL if we don't have recent bridge statistics.

References load_bridge_stats().

Referenced by extrainfo_dump_to_string().

4.26.3.21 static char∗ geoip_get_client_history (time_t *now*, geoip_client_action_t *action*, int *min_observation_time*, unsigned *granularity*) `[static]`

Helper for **geoip_get_client_history_dirreq()** (p. 652) and **geoip_get_client_history_bridge()** (p. 652).

unresolved requests are stored at index 0.

References _c_hist_compare(), c_hist_t::country, geoip_get_country_by_ip(), geoip_get_country_name(), geoip_get_n_countries(), geoip_is_loaded(), MIN_IPS_TO_NOTE_ANYTHING, MIN_IPS_TO_NOTE_COUNTRY, round_to_next_multiple_of(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), tor_asprintf(), tor_assert, tor_free, and c_hist_t::total.

Referenced by geoip_get_client_history_bridge(), and geoip_get_client_history_dirreq().

4.26.3.22 char∗ geoip_get_client_history_bridge (time_t *now*, geoip_client_action_t *action*)

Return a newly allocated comma-separated string containing entries for all the countries from which we've seen enough clients connect as a bridge. The entry format is cc=num where num is the number of IPs we've seen connecting from that country, and cc is a lowercased country code. Returns NULL if we don't want to export geoip data yet.

References geoip_get_client_history(), GEOIP_MIN_OBSERVATION_TIME, and IP_GRANULARITY.

Referenced by geoip_bridge_stats_write().

4.26.3.23 char∗ geoip_get_client_history_dirreq (time_t *now*, geoip_client_action_t *action*)

Return a newly allocated comma-separated string containing entries for all the countries from which we've seen enough clients connect as a directory. The entry format is cc=num where num is the number of IPs we've seen connecting from that country, and cc is a lowercased country code. Returns NULL if we don't want to export geoip data yet.

References DIR_RECORD_USAGE_GRANULARITY, DIR_RECORD_USAGE_MIN_OBSERVATION_TIME, and geoip_get_client_history().

Referenced by geoip_dirreq_stats_write(), and geoip_entry_stats_write().

4.26.3.24 country_t geoip_get_country (const char ∗ *country*)

Return the index of the **country**'s entry in the GeoIP DB if it is a valid 2-letter country code, otherwise return -1.

References country_idxplus1_by_lc_code, and strmap_get_lc().

Referenced by routerset_refresh_countries().

4.26.3.25 int geoip_get_country_by_ip (uint32_t *ipaddr*)

Given an IP address in host order, return a number representing the country to which that address belongs, or -1 for unknown. The return value will always be less than **geoip_get_n_countries()**

(p. 653). To decode it, call **geoip_get_country_name()** (p. 653).

References _geoip_compare_key_to_entry(), geoip_entry_t::country, and smartlist_bsearch().

Referenced by geoip_get_client_history(), geoip_note_client_seen(), getinfo_helper_geoip(), routerinfo_set_country(), and routerset_contains().

4.26.3.26 const char∗ geoip_get_country_name (country_t *num*)

Return the two-letter country code associated with the number **num**, or "??" for an unknown value.

References geoip_country_t::countrycode.

Referenced by geoip_get_client_history(), and getinfo_helper_geoip().

4.26.3.27 static char∗ geoip_get_dirreq_history (geoip_client_action_t *action*, dirreq_type_t *type*) `[static]`

Return a newly allocated comma-separated string containing statistics on network status downloads. The string contains the number of completed requests, timeouts, and still running requests as well as the download times by deciles and quartiles. Return NULL if we have not observed requests for long enough.

References DIRREQ_TIMEOUT, GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, round_uint32_to_next_multiple_of(), smartlist_add(), smartlist_create(), smartlist_free(), tor_free, tor_gettimeofday(), tor_snprintf(), and tv_mdiff().

Referenced by geoip_dirreq_stats_write().

4.26.3.28 time_t geoip_get_history_start (void)

Return the time at which we started recording geoip data.

4.26.3.29 static int geoip_get_mean_shares (time_t *now*, double ∗ *v2_share_out*, double ∗ *v3_share_out*) `[static]`

Calculate which fraction of v2 and v3 directory requests aimed at caches have been sent to us since the last call of this function up to time **now**. Set ∗**v2_share_out** and ∗**v3_share_out** to the fractions of v2 and v3 protocol shares we expect to have seen. Reset counters afterwards. Return 0 on success, -1 on failure (e.g. when zero seconds have passed since the last call).

References geoip_determine_shares(), share_seconds, v2_share_times_seconds, and v3_share_times_seconds.

Referenced by geoip_dirreq_stats_write().

4.26.3.30 int geoip_get_n_countries (void)

Return the number of countries recognized by the GeoIP database.

Referenced by geoip_get_client_history(), and routerset_refresh_countries().

4.26.3.31 char* geoip_get_request_history (time_t *now*, geoip_client_action_t *action*)

Return a newly allocated string holding the per-country request history for **action** in a format suitable for an extra-info document, or NULL on failure.

References _c_hist_compare(), GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, GEOIP_MIN_OBSERVATION_TIME, IP_GRANULARITY, REQUEST_HIST_LEN, round_to_next_multiple_of(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), tor_asprintf(), and tor_free.

Referenced by geoip_dirreq_stats_write().

4.26.3.32 int geoip_is_loaded (void)

Return true iff we have loaded a GeoIP database.

Referenced by geoip_get_client_history(), getinfo_helper_geoip(), options_act(), and routerset_refresh_countries().

4.26.3.33 int geoip_load_file (const char * *filename*, or_options_t * *options*)

Clear the GeoIP database and reload it from the file **filename**. Return 0 on success, -1 on failure.

Recognized line formats are: INTIPLOW,INTIPHIGH,CC and "INTIPLOW","INTIPHIGH","CC","CC3","COUNTRY NAME" where INTIPLOW and INTIPHIGH are IPv4 addresses encoded as 4-byte unsigned integers, and CC is a country code.

It also recognizes, and skips over, blank lines and lines that start with '#' (comments).

References _geoip_compare_entries(), clear_geoip_db(), country_idxplus1_by_lc_code, geoip_country_t::countrycode, geoip_parse_entry(), LD_GENERAL, options_need_geoip_info(), refresh_all_country_info(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_sort(), and tor_free.

Referenced by options_act().

4.26.3.34 void geoip_note_client_seen (geoip_client_action_t *action*, uint32_t *addr*, time_t *now*)

Note that we've seen a client connect from the IP **addr** (host order) at time **now**. Ignored by all but bridges and directories if configured accordingly.

unresolved requests are stored at index 0.

References or_options_t::BridgeAuthoritativeDir, or_options_t::BridgeRecordUsageByCountry, or_options_t::BridgeRelay, or_options_t::DirReqStatistics, or_options_t::EntryStatistics, GEOIP_CLIENT_CONNECT, GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, geoip_determine_shares(), geoip_get_country_by_ip(), geoip_remove_old_clients(), get_options(), last_time_determined_shares, geoip_country_t::n_v2_ns_requests, geoip_country_t::n_v3_ns_requests, REQUEST_HIST_LEN, REQUEST_HIST_PERIOD, REQUEST_SHARE_INTERVAL, and smartlist_create().

Referenced by connection_or_set_state_open(), and directory_handle_command_get().

4.26.3.35 void geoip_note_ns_response (geoip_client_action_t *action*, geoip_ns_response_t *response*)

Note that we've rejected a client's request for a v2 or v3 network status, encoded in **action** for reason **reason** at time **now**.

References GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, get_options(), and tor_assert.

Referenced by directory_handle_command_get().

4.26.3.36 int geoip_parse_entry (const char * *line*)

Add an entry to the GeoIP table, parsing it from **line**. The format is as for **geoip_load_file()** (p. 654).

References country_idxplus1_by_lc_code, escaped(), geoip_add_entry(), LD_GENERAL, and smartlist_create().

Referenced by geoip_load_file().

4.26.3.37 void geoip_remove_old_clients (time_t *cutoff*)

Forget about all clients that haven't connected since **cutoff**. If **cutoff** is in the future, clients won't be added to the history until this time is reached. This is useful to prevent relays that switch to bridges from reporting unbelievable numbers of clients.

References _remove_old_client_helper().

Referenced by geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), geoip_note_client_seen(), and options_act().

4.26.3.38 void geoip_start_dirreq (uint64_t *dirreq_id*, size_t *response_size*, geoip_client_action_t *action*, dirreq_type_t *type*)

Note that an either direct or tunneled (see **type**) directory request for a network status with unique ID **dirreq_id** of size **response_size** and action **action** (either v2 or v3) has started.

References _dirreq_map_put(), get_options(), and tor_gettimeofday().

Referenced by directory_handle_command_get().

4.26.3.39 int getinfo_helper_geoip (control_connection_t * *control_conn*, const char * *question*, char ** *answer*)

Helper used to implement GETINFO ip-to-country/... controller command.

References geoip_get_country_by_ip(), geoip_get_country_name(), geoip_is_loaded(), strcmpstart(), and tor_inet_aton().

4.26.3.40 static HT_HEAD (dirreqmap, dirreq_map_entry_t) `[static]`

Map of all directory requests asking for v2 or v3 network statuses in the current geoip-stats interval. Values are of type ***dirreq_map_entry_t** (p. **85**).

4.26.3.41 static HT_HEAD (clientmap, clientmap_entry_t) [static]

Map from client IP address to last time seen. Hashtable helper: compute a hash of a **clientmap_entry_t** (p. 46).

4.26.3.42 static void load_bridge_stats (time_t *now*) [static]

Try to load the most recent bridge statistics from disk, unless we have finished a measurement interval lately.

References check_private_dir(), get_datadir_fname, get_datadir_fname2, parse_bridge_stats_controller(), read_file_to_str(), RFTS_IGNORE_MISSING, and tor_free.

Referenced by geoip_get_bridge_stats_controller(), and geoip_get_bridge_stats_extrainfo().

4.26.3.43 static char∗ parse_bridge_stats_controller (const char ∗ *stats_str*, time_t *now*) [static]

Parse the bridge statistics as they are written to extra-info descriptors for being returned to controller clients. Return the controller string if successful, or NULL otherwise.

References eat_whitespace_no_nl(), find_str_at_start_of_line(), format_iso_time(), parse_iso_time(), tor_asprintf(), tor_assert, and tor_free.

Referenced by geoip_bridge_stats_write(), and load_bridge_stats().

4.26.3.44 int should_record_bridge_info (or_options_t ∗ *options*)

Return 1 if we should collect geoip stats on bridge users, and include them in our extrainfo descriptor. Else return 0.

References or_options_t::BridgeRecordUsageByCountry, and or_options_t::BridgeRelay.

Referenced by extrainfo_dump_to_string(), options_act(), and run_scheduled_events().

4.26.4 Variable Documentation

4.26.4.1 char∗ bridge_stats_controller = NULL [static]

Most recent bridge statistics formatted to be returned to controller clients.

4.26.4.2 char∗ bridge_stats_extrainfo = NULL [static]

Most recent bridge statistics formatted to be written to extra-info descriptors.

4.26.4.3 strmap_t∗ country_idxplus1_by_lc_code = NULL [static]

A map from lowercased country codes to their position in geoip_countries. The index is encoded in the pointer, and 1 is added so that NULL can mean not found.

Referenced by clear_geoip_db(), geoip_add_entry(), geoip_get_country(), geoip_load_file(), and geoip_parse_entry().

4.26.4.4 smartlist_t* geoip_countries = NULL `[static]`

A list of **geoip_country_t** (p. 101)

4.26.4.5 smartlist_t* geoip_entries = NULL `[static]`

A list of all known **geoip_entry_t** (p. 102), sorted by ip_low.

4.26.4.6 time_t last_time_determined_shares = 0 `[static]`

When did we last determine which share of v2 and v3 directory requests is sent to us?

Referenced by geoip_determine_shares(), and geoip_note_client_seen().

4.26.4.7 uint32_t ns_v2_responses[GEOIP_NS_RESPONSE_NUM] `[static]`

How many responses are we giving to clients requesting v2 network statuses?

4.26.4.8 uint32_t ns_v3_responses[GEOIP_NS_RESPONSE_NUM] `[static]`

How many responses are we giving to clients requesting v3 network statuses?

4.26.4.9 int share_seconds `[static]`

Number of seconds we are determining v2 and v3 shares.

Referenced by geoip_determine_shares(), and geoip_get_mean_shares().

4.26.4.10 time_t start_of_bridge_stats_interval `[static]`

Start time of bridge stats.

4.26.4.11 time_t start_of_dirreq_stats_interval `[static]`

Start time of directory request stats.

4.26.4.12 time_t start_of_entry_stats_interval `[static]`

Start time of entry stats.

4.26.4.13 double v2_share_times_seconds `[static]`

Sum of products of v2 shares times the number of seconds for which we consider these shares as valid.

Referenced by geoip_determine_shares(), and geoip_get_mean_shares().

4.26.4.14 double v3_share_times_seconds `[static]`

Sum of products of v3 shares times the number of seconds for which we consider these shares as valid.

Referenced by geoip_determine_shares(), and geoip_get_mean_shares().

4.27 hibernate.c File Reference

Functions to close listeners, stop allowing new circuits, etc in preparation for closing down or going dormant; and to track bandwidth and time intervals to know when to hibernate and when to stop hibernating.

`#include "or.h"`

Defines

- #define **NOTE_INTERVAL** (600)
- #define **NOTE_BYTES** 20∗(1024∗1024)
- #define **GUESS_TIME_TO_USE_BANDWIDTH** (24∗60∗60)
- #define **ROUND_UP**(x) (((x) + 0x3ff) & ∼0x3ff)

Enumerations

- enum **hibernate_state_t** { **HIBERNATE_STATE_LIVE** = 1, **HIBERNATE_STATE_EXITING** = 2, **HIBERNATE_STATE_LOWBANDWIDTH** = 3, **HIBERNATE_STATE_DORMANT** = 4 }
- enum **time_unit_t** { **UNIT_MONTH** = 1, **UNIT_WEEK** = 2, **UNIT_DAY** = 3 }

Functions

- static void **reset_accounting** (time_t now)
- static int **read_bandwidth_usage** (void)
- static time_t **start_of_accounting_period_after** (time_t now)
- static time_t **start_of_accounting_period_containing** (time_t now)
- static void **accounting_set_wakeup_time** (void)
- int **accounting_parse_options** (**or_options_t** ∗options, int validate_only)
- int **accounting_is_enabled** (**or_options_t** ∗options)
- void **accounting_add_bytes** (size_t n_read, size_t n_written, int seconds)
- static time_t **edge_of_accounting_period_containing** (time_t now, int get_end)
- void **configure_accounting** (time_t now)
- static void **update_expected_bandwidth** (void)
- static INLINE int **time_to_record_bandwidth_usage** (time_t now)
- void **accounting_run_housekeeping** (time_t now)
- int **accounting_record_bandwidth_usage** (time_t now, **or_state_t** ∗state)
- static int **hibernate_hard_limit_reached** (void)
- static int **hibernate_soft_limit_reached** (void)
- static void **hibernate_begin** (**hibernate_state_t** new_state, time_t now)
- static void **hibernate_end** (**hibernate_state_t** new_state)
- void **hibernate_begin_shutdown** (void)
- int **we_are_hibernating** (void)
- static void **hibernate_go_dormant** (time_t now)
- static void **hibernate_end_time_elapsed** (time_t now)
- void **consider_hibernation** (time_t now)
- int **getinfo_helper_accounting** (**control_connection_t** ∗conn, const char ∗question, char ∗∗answer)

Variables

- long **stats_n_seconds_working**
- static **hibernate_state_t hibernate_state** = HIBERNATE_STATE_LIVE
- static time_t **hibernate_end_time** = 0
- static time_t **shutdown_time** = 0
- static uint64_t **n_bytes_read_in_interval** = 0
- static uint64_t **n_bytes_written_in_interval** = 0
- static uint32_t **n_seconds_active_in_interval** = 0
- static time_t **interval_start_time** = 0
- static time_t **interval_end_time** = 0
- static time_t **interval_wakeup_time** = 0
- static uint64_t **expected_bandwidth_usage** = 0
- static **time_unit_t cfg_unit** = UNIT_MONTH
- static int **cfg_start_day** = 0
- static int **cfg_start_hour** = 0
- static int **cfg_start_min** = 0

4.27.1 Detailed Description

Functions to close listeners, stop allowing new circuits, etc in preparation for closing down or going dormant; and to track bandwidth and time intervals to know when to hibernate and when to stop hibernating.

4.27.2 Define Documentation

4.27.2.1 #define GUESS_TIME_TO_USE_BANDWIDTH (24*60*60)

When we have no idea how fast we are, how long do we assume it will take us to exhaust our bandwidth?

Referenced by accounting_set_wakeup_time().

4.27.3 Enumeration Type Documentation

4.27.3.1 enum hibernate_state_t

Possible values of hibernate_state

Enumerator:

HIBERNATE_STATE_LIVE We are running normally.

HIBERNATE_STATE_EXITING We're trying to shut down cleanly, and we'll kill all active connections at shutdown_time.

HIBERNATE_STATE_LOWBANDWIDTH We're running low on allocated bandwidth for this period, so we won't accept any new connections.

HIBERNATE_STATE_DORMANT We are hibernating, and we won't wake up till there's more bandwidth to use.

4.27.3.2 enum time_unit_t

Possible accounting periods.

4.27.4 Function Documentation

4.27.4.1 void accounting_add_bytes (size_t *n_read*, size_t *n_written*, int *seconds*)

Called from **main.c** (p. 690) to tell us that **seconds** seconds have passed, **n_read** bytes have been read, and **n_written** bytes have been written.

References n_bytes_read_in_interval, n_bytes_written_in_interval, and n_seconds_active_in_interval.

Referenced by second_elapsed_callback().

4.27.4.2 int accounting_is_enabled (or_options_t * *options*)

If we want to manage the accounting system and potentially hibernate, return 1, else return 0.

References or_options_t::AccountingMax.

Referenced by decide_to_advertise_dirport(), do_hup(), getinfo_helper_accounting(), options_act(), or_state_save(), run_scheduled_events(), second_elapsed_callback(), and tor_cleanup().

4.27.4.3 int accounting_parse_options (or_options_t * *options*, int *validate_only*)

Configure accounting start/end time settings based on options->AccountingStart. Return 0 on success, -1 on failure. If **validate_only** is true, do not change the current settings.

References or_options_t::AccountingStart, cfg_start_day, cfg_unit, LD_CONFIG, smartlist_create(), smartlist_free(), smartlist_split_string(), tor_assert, tor_free, and tor_parse_long().

Referenced by options_act().

4.27.4.4 int accounting_record_bandwidth_usage (time_t *now*, or_state_t * *state*)

Save all our bandwidth tracking information to disk. Return 0 on success, -1 on failure.

References or_state_t::AccountingBytesReadInInterval, or_state_t::AccountingBytesWrittenInInterval, or_state_t::AccountingExpectedUsage, or_state_t::AccountingIntervalStart, or_state_t::AccountingSecondsActive, expected_bandwidth_usage, get_options(), interval_start_time, n_bytes_read_in_interval, n_bytes_written_in_interval, n_seconds_active_in_interval, and or_state_mark_dirty().

Referenced by accounting_run_housekeeping(), do_hup(), hibernate_begin(), hibernate_go_dormant(), and tor_cleanup().

4.27.4.5 void accounting_run_housekeeping (time_t *now*)

Invoked once per second. Checks whether it is time to hibernate, record bandwidth used, etc.

References accounting_record_bandwidth_usage(), configure_accounting(), get_or_state(), interval_end_time, LD_FS, and time_to_record_bandwidth_usage().

Referenced by hibernate_end_time_elapsed(), or_state_save(), and run_scheduled_events().

4.27.4.6 static void accounting_set_wakeup_time (void) `[static]`

Based on our interval and our estimated bandwidth, choose a deterministic (but random-ish) time to wake up.

References or_options_t::AccountingMax, crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), crypto_pk_get_digest(), DIGEST_LEN, expected_bandwidth_usage, format_iso_time(), format_local_iso_time(), get_identity_key(), get_options(), get_uint32(), GUESS_TIME_TO_USE_BANDWIDTH, identity_key_is_set(), init_keys(), interval_end_time, interval_start_time, interval_wakeup_time, LD_ACCT, LD_BUG, and tor_assert.

Referenced by configure_accounting().

4.27.4.7 void configure_accounting (time_t *now*)

Initialize the accounting subsystem.

References accounting_set_wakeup_time(), interval_end_time, interval_start_time, LD_ACCT, read_bandwidth_usage(), reset_accounting(), start_of_accounting_period_after(), and start_of_accounting_period_containing().

Referenced by accounting_run_housekeeping(), and options_act().

4.27.4.8 void consider_hibernation (time_t *now*)

Consider our environment and decide if it's time to start/stop hibernating.

References or_options_t::AccountingMax, format_local_iso_time(), get_options(), hibernate_begin(), hibernate_end_time, hibernate_end_time_elapsed(), hibernate_go_dormant(), hibernate_hard_limit_reached(), hibernate_soft_limit_reached(), hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_EXITING, HIBERNATE_STATE_LIVE, HIBERNATE_STATE_LOWBANDWIDTH, interval_wakeup_time, LD_ACCT, LD_GENERAL, shutdown_time, tor_assert, and tor_cleanup().

Referenced by run_scheduled_events().

4.27.4.9 static time_t edge_of_accounting_period_containing (time_t *now*, int *get_end*) `[static]`

If get_end, return the end of the accounting period that contains the time **now**. Else, return the start of the accounting period that contains the time **now**

References cfg_start_day, cfg_unit, and tor_assert.

Referenced by start_of_accounting_period_after(), and start_of_accounting_period_containing().

4.27.4.10 int getinfo_helper_accounting (control_connection_t * *conn*, const char * *question*, char ** *answer*)

Helper function: called when we get a GETINFO request for an accounting-related key on the control connection **conn**. If we can answer the request for **question**, then set ***answer** to a newly allocated string holding the result. Otherwise, set ***answer** to NULL.

References accounting_is_enabled(), or_options_t::AccountingMax, format_iso_time(), get_options(), hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_LOWBANDWIDTH, interval_end_time, interval_start_time, interval_wakeup_time, n_bytes_read_in_interval, n_bytes_written_in_interval, and tor_snprintf().

4.27.4.11 static void hibernate_begin (hibernate_state_t *new_state*, time_t *now*) `[static]`

Called when we get a SIGINT, or when bandwidth soft limit is reached. Puts us into "loose hibernation": we don't accept new connections, but we continue handling old ones.

References accounting_record_bandwidth_usage(), CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_DIR_LISTENER, CONN_TYPE_OR_LISTENER, connection_get_by_type(), get_options(), get_or_state(), hibernate_end_time, hibernate_state, HIBERNATE_STATE_EXITING, HIBERNATE_STATE_LIVE, interval_end_time, LD_GENERAL, LD_NET, or_state_mark_dirty(), shutdown_time, or_options_t::ShutdownWaitLength, tor_cleanup(), and connection_t::type.

Referenced by consider_hibernation(), hibernate_begin_shutdown(), and hibernate_go_dormant().

4.27.4.12 void hibernate_begin_shutdown (void)

A wrapper around hibernate_begin, for when we get SIGINT.

References hibernate_begin(), and HIBERNATE_STATE_EXITING.

Referenced by signal_callback().

4.27.4.13 static void hibernate_end (hibernate_state_t *new_state*) `[static]`

Called when we've been hibernating and our timeout is reached.

References hibernate_end_time, hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_LOWBANDWIDTH, LD_ACCT, stats_n_seconds_working, and tor_assert.

Referenced by hibernate_end_time_elapsed().

4.27.4.14 static void hibernate_end_time_elapsed (time_t *now*) `[static]`

Called when hibernate_end_time has arrived.

References accounting_run_housekeeping(), format_iso_time(), hibernate_end(), hibernate_end_time, hibernate_go_dormant(), hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_LIVE, interval_wakeup_time, and LD_ACCT.

Referenced by consider_hibernation().

4.27.4.15 static void hibernate_go_dormant (time_t *now*) `[static]`

If we aren't currently dormant, close all connections and become dormant.

References accounting_record_bandwidth_usage(), CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_edge_end(), connection_get_by_type(), get_options(), get_or_state(), hibernate_begin(), hibernate_end_time, hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_LOWBANDWIDTH, interval_end_time, interval_wakeup_time, LD_ACCT, LD_NET, or_state_mark_dirty(), TO_EDGE_CONN(), and connection_t::type.

Referenced by consider_hibernation(), and hibernate_end_time_elapsed().

4.27.4.16 static int hibernate_hard_limit_reached (void) `[static]`

Return true iff we have sent/received all the bytes we are willing to send/receive this interval.

References or_options_t::AccountingMax, get_options(), n_bytes_read_in_interval, and n_bytes_written_in_interval.

Referenced by consider_hibernation().

4.27.4.17 static int hibernate_soft_limit_reached (void) `[static]`

Return true iff we have sent/received almost all the bytes we are willing to send/receive this interval.

References get_options(), n_bytes_read_in_interval, and n_bytes_written_in_interval.

Referenced by consider_hibernation().

4.27.4.18 static int read_bandwidth_usage (void) `[static]`

Read stored accounting information from disk. Return 0 on success; return -1 and change nothing on failure.

References or_state_t::AccountingBytesReadInInterval, or_state_t::AccountingBytesWrittenInInterval, or_state_t::AccountingExpectedUsage, or_state_t::AccountingIntervalStart, or_state_t::AccountingSecondsActive, expected_bandwidth_usage, format_iso_time(), get_datadir_fname, get_or_state(), interval_start_time, or_state_t::LastWritten, LD_ACCT, n_bytes_read_in_interval, n_bytes_written_in_interval, n_seconds_active_in_interval, and tor_free.

Referenced by configure_accounting().

4.27.4.19 static void reset_accounting (time_t *now*) `[static]`

Called at the start of a new accounting interval: reset our expected bandwidth usage based on what happened last time, set up the start and end of the interval, and clear byte/time totals.

References interval_end_time, interval_start_time, LD_ACCT, n_bytes_read_in_interval, n_bytes_written_in_interval, n_seconds_active_in_interval, start_of_accounting_period_after(), start_of_accounting_period_containing(), and update_expected_bandwidth().

Referenced by configure_accounting().

4.27.4.20 static time_t start_of_accounting_period_after (time_t *now*) [static]

Return the start of the accounting period that comes after the one containing the time **now**.

References edge_of_accounting_period_containing().

Referenced by configure_accounting(), and reset_accounting().

4.27.4.21 static time_t start_of_accounting_period_containing (time_t *now*) [static]

Return the start of the accounting period containing the time **now**.

References edge_of_accounting_period_containing().

Referenced by configure_accounting(), and reset_accounting().

4.27.4.22 static INLINE int time_to_record_bandwidth_usage (time_t *now*) [static]

Return true iff we should save our bandwidth usage to disk.

References interval_end_time, n_bytes_read_in_interval, and n_bytes_written_in_interval.

Referenced by accounting_run_housekeeping().

4.27.4.23 static void update_expected_bandwidth (void) [static]

Set expected_bandwidth_usage based on how much we sent/received per minute last interval (if we were up for at least 30 minutes), or based on our declared bandwidth otherwise.

References or_options_t::BandwidthRate, expected_bandwidth_usage, get_options(), n_bytes_read_in_interval, n_bytes_written_in_interval, and n_seconds_active_in_interval.

Referenced by reset_accounting().

4.27.4.24 int we_are_hibernating (void)

Return true iff we are currently hibernating.

References hibernate_state, and HIBERNATE_STATE_LIVE.

Referenced by check_descriptor_bandwidth_changed(), check_whether_dirport_reachable(), command_process_create_cell(), connection_exit_begin_conn(), decide_to_advertise_dirport(), directory_info_has_arrived(), dirserv_set_router_is_running(), router_dump_router_to_string(), router_rebuild_descriptor(), router_set_status(), run_connection_housekeeping(), run_scheduled_events(), and second_elapsed_callback().

4.27.5 Variable Documentation

4.27.5.1 int cfg_start_day = 0 [static]

How many days,hours,minutes into each unit does our accounting interval start?

Referenced by accounting_parse_options(), and edge_of_accounting_period_containing().

4.27.5.2 time_unit_t cfg_unit = UNIT_MONTH [static]

What unit are we using for our accounting?

Referenced by accounting_parse_options(), and edge_of_accounting_period_containing().

4.27.5.3 uint64_t expected_bandwidth_usage = 0 [static]

How much bandwidth do we 'expect' to use per minute? (0 if we have no info from the last period.)

Referenced by accounting_record_bandwidth_usage(), accounting_set_wakeup_time(), read_bandwidth_usage(), and update_expected_bandwidth().

4.27.5.4 time_t hibernate_end_time = 0 [static]

If are hibernating, when do we plan to wake up? Set to 0 if we aren't hibernating.

Referenced by consider_hibernation(), hibernate_begin(), hibernate_end(), hibernate_end_time_elapsed(), and hibernate_go_dormant().

4.27.5.5 hibernate_state_t hibernate_state = HIBERNATE_STATE_LIVE [static]

Are we currently awake, asleep, running out of bandwidth, or shutting down?

Referenced by consider_hibernation(), getinfo_helper_accounting(), hibernate_begin(), hibernate_end(), hibernate_end_time_elapsed(), hibernate_go_dormant(), and we_are_hibernating().

4.27.5.6 time_t interval_end_time = 0 [static]

When will this accounting interval end?

Referenced by accounting_run_housekeeping(), accounting_set_wakeup_time(), configure_accounting(), getinfo_helper_accounting(), hibernate_begin(), hibernate_go_dormant(), reset_accounting(), and time_to_record_bandwidth_usage().

4.27.5.7 time_t interval_start_time = 0 [static]

When did this accounting interval start?

Referenced by accounting_record_bandwidth_usage(), accounting_set_wakeup_time(), configure_accounting(), getinfo_helper_accounting(), read_bandwidth_usage(), and reset_accounting().

4.27.5.8 time_t interval_wakeup_time = 0 [static]

How far into the accounting interval should we hibernate?

Referenced by accounting_set_wakeup_time(), consider_hibernation(), getinfo_helper_accounting(), hibernate_end_time_elapsed(), and hibernate_go_dormant().

4.27.5.9 uint64_t n_bytes_read_in_interval = 0 [static]

How many bytes have we read in this accounting interval?

Referenced by accounting_add_bytes(), accounting_record_bandwidth_usage(), getinfo_helper_accounting(), hibernate_hard_limit_reached(), hibernate_soft_limit_reached(), read_bandwidth_usage(), reset_accounting(), time_to_record_bandwidth_usage(), and update_expected_bandwidth().

4.27.5.10 uint64_t n_bytes_written_in_interval = 0 [static]

How many bytes have we written in this accounting interval?

Referenced by accounting_add_bytes(), accounting_record_bandwidth_usage(), getinfo_helper_accounting(), hibernate_hard_limit_reached(), hibernate_soft_limit_reached(), read_bandwidth_usage(), reset_accounting(), time_to_record_bandwidth_usage(), and update_expected_bandwidth().

4.27.5.11 uint32_t n_seconds_active_in_interval = 0 [static]

How many seconds have we been running this interval?

Referenced by accounting_add_bytes(), accounting_record_bandwidth_usage(), read_bandwidth_usage(), reset_accounting(), and update_expected_bandwidth().

4.27.5.12 time_t shutdown_time = 0 [static]

If we are shutting down, when do we plan finally exit? Set to 0 if we aren't shutting down.

Referenced by consider_hibernation(), and hibernate_begin().

4.27.5.13 long stats_n_seconds_working

How many seconds have we been running?

Referenced by directory_all_unreachable(), hibernate_end(), ip_address_changed(), router_dump_router_to_string(), run_scheduled_events(), and second_elapsed_callback().

4.28 log.c File Reference

Functions to send messages to log files or the console.

`#include "orconfig.h"`

`#include <stdarg.h>`

`#include <assert.h>`

`#include <stdlib.h>`

`#include <string.h>`

`#include <sys/time.h>`

`#include <time.h>`

`#include <unistd.h>`

`#include <sys/types.h>`

`#include <fcntl.h>`

`#include "compat.h"`

`#include "util.h"`

`#include "log.h"`

`#include "container.h"`

Data Structures

- struct **logfile_t**

Defines

- #define **LOG_PRIVATE**
- #define **TRUNCATED_STR** "[...truncated]"
- #define **TRUNCATED_STR_LEN** 14
- #define **LOCK_LOGS**()
- #define **UNLOCK_LOGS**() STMT_BEGIN tor_mutex_release(&**log_mutex**); STMT_END

Functions

- static void **log_free** (**logfile_t** ∗victim)
- static INLINE const char ∗ **sev_to_string** (int severity)
- static INLINE int **should_log_function_name** (log_domain_mask_t domain, int severity)
- static void **delete_log** (**logfile_t** ∗victim)
- static void **close_log** (**logfile_t** ∗victim)
- void **log_set_application_name** (const char ∗name)
- static INLINE size_t **_log_prefix** (char ∗buf, size_t buf_len, int severity)
- static int **log_tor_version** (**logfile_t** ∗lf, int reset)

- static INLINE char ∗ **format_msg** (char ∗buf, size_t buf_len, log_domain_mask_t domain, int severity, const char ∗funcname, const char ∗format, va_list ap, size_t ∗msg_len_out)
- static void **logv** (int severity, log_domain_mask_t domain, const char ∗funcname, const char ∗format, va_list ap)
- void **tor_log** (int severity, log_domain_mask_t domain, const char ∗format,...)
- void **_log_fn** (int severity, log_domain_mask_t domain, const char ∗format,...)
- void **_log_debug** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_info** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_notice** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_warn** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_err** (log_domain_mask_t domain, const char ∗format,...)
- void **logs_free_all** (void)
- void **set_log_severity_config** (int loglevelMin, int loglevelMax, **log_severity_list_t** ∗severity_out)
- static void **add_stream_log_impl** (const **log_severity_list_t** ∗severity, const char ∗name, int fd)
- void **add_stream_log** (const **log_severity_list_t** ∗severity, const char ∗name, int fd)
- void **init_logging** (void)
- void **add_temp_log** (int min_severity)
- int **add_callback_log** (const **log_severity_list_t** ∗severity, **log_callback** cb)
- void **change_callback_log_severity** (int loglevelMin, int loglevelMax, **log_callback** cb)
- void **close_temp_logs** (void)
- void **rollback_log_changes** (void)
- void **mark_logs_temp** (void)
- int **add_file_log** (const **log_severity_list_t** ∗severity, const char ∗filename)
- int **add_syslog_log** (const **log_severity_list_t** ∗severity)
- int **parse_log_level** (const char ∗level)
- const char ∗ **log_level_to_string** (int level)
- static log_domain_mask_t **parse_log_domain** (const char ∗domain)
- int **parse_log_severity_config** (const char ∗∗cfg_ptr, **log_severity_list_t** ∗severity_out)
- int **get_min_log_level** (void)
- void **switch_logs_debug** (void)

Variables

- static **tor_mutex_t log_mutex**
- static int **log_mutex_initialized** = 0
- static **logfile_t** ∗ **logfiles** = NULL
- static int **syslog_count** = 0
- int **_log_global_min_severity** = LOG_NOTICE
- static char ∗ **appname** = NULL
- const char ∗ **_log_fn_function_name** = NULL
- static const char ∗ **domain_list** []

4.28.1 Detailed Description

Functions to send messages to log files or the console.

4.28.2 Define Documentation

4.28.2.1 #define LOCK_LOGS()

Value:

```
STMT_BEGIN                                          \
  tor_mutex_acquire(&log_mutex);                                        \
  STMT_END
```

4.28.3 Function Documentation

4.28.3.1 static INLINE size_t _log_prefix (char * *buf*, size_t *buf_len*, int *severity*) [static]

Helper: Write the standard prefix for log lines to a **buf_len** character buffer in **buf**.

References sev_to_string(), tor_gettimeofday(), and tor_snprintf().

Referenced by format_msg(), and log_tor_version().

4.28.3.2 int add_callback_log (const log_severity_list_t * *severity*, log_callback *cb*)

Add a log handler to send messages in **severity** to the function **cb**.

References _log_global_min_severity, logfile_t::callback, logfile_t::fd, logfile_t::filename, get_min_log_level(), logfile_t::next, and logfile_t::severities.

Referenced by options_act_reversible().

4.28.3.3 int add_file_log (const log_severity_list_t * *severity*, const char * *filename*)

Add a log handler to send messages to **filename**. If opening the logfile fails, -1 is returned and errno is set appropriately (by open(2)).

References _log_global_min_severity, add_stream_log_impl(), delete_log(), get_min_log_level(), log_tor_version(), logfile_t::needs_close, and tor_fd_seekend().

Referenced by options_init_logs().

4.28.3.4 void add_stream_log (const log_severity_list_t * *severity*, const char * *name*, int *fd*)

Add a log handler named **name** to send all messages in **severity** to **fd**. Steals a reference to **severity**; the caller must not use it after calling this function.

References add_stream_log_impl().

Referenced by options_init_logs().

4.28.3.5 static void add_stream_log_impl (const log_severity_list_t ∗ *severity*, const char ∗ *name*, int *fd*) `[static]`

Add a log handler named **name** to send all messages in **severity** to **fd**. Copies **severity**. Helper: does no locking.

References _log_global_min_severity, logfile_t::fd, logfile_t::filename, get_min_log_level(), logfile_t::next, and logfile_t::severities.

Referenced by add_file_log(), add_stream_log(), and add_temp_log().

4.28.3.6 int add_syslog_log (const log_severity_list_t ∗ *severity*)

Add a log handler to send messages to they system log facility.

References _log_global_min_severity, logfile_t::fd, logfile_t::filename, get_min_log_level(), logfile_t::is_syslog, logfile_t::next, logfile_t::severities, and syslog_count.

Referenced by options_init_logs().

4.28.3.7 void add_temp_log (int *min_severity*)

Add a log handler to receive messages during startup (before the real logs are initialized).

References add_stream_log_impl(), logfile_t::is_temporary, set_log_severity_config(), and tor_free.

Referenced by tor_init().

4.28.3.8 void change_callback_log_severity (int *loglevelMin*, int *loglevelMax*, log_callback *cb*)

Adjust the configured severity of any logs whose callback function is **cb**.

References _log_global_min_severity, logfile_t::callback, get_min_log_level(), logfile_t::next, set_log_severity_config(), and logfile_t::severities.

Referenced by control_adjust_event_log_severity().

4.28.3.9 static void close_log (logfile_t ∗ *victim*) `[static]`

Helper: release system resources (but not memory) held by a single **logfile_t** (p. 108).

References logfile_t::fd, logfile_t::is_syslog, logfile_t::needs_close, and syslog_count.

Referenced by close_temp_logs(), and logs_free_all().

4.28.3.10 void close_temp_logs (void)

Close any log handlers added by **add_temp_log()** (p. 671) or marked by **mark_logs_temp()** (p. 673).

References _log_global_min_severity, close_log(), get_min_log_level(), log_free(), and logfile_t::next.

Referenced by options_act_reversible(), and rollback_log_changes().

4.28.3.11 static void delete_log (logfile_t * *victim*) [static]

Remove and free the log entry **victim** from the linked-list logfiles (it is probably present, but it might not be due to thread racing issues). After this function is called, the caller shouldn't refer to **victim** anymore.

Long-term, we need to do something about races in the log subsystem in general. See bug 222 for more details.

References log_free(), and logfile_t::next.

Referenced by add_file_log().

4.28.3.12 static INLINE char* format_msg (char * *buf*, size_t *buf_len*, log_domain_mask_t *domain*, int *severity*, const char * *funcname*, const char * *format*, va_list *ap*, size_t * *msg_len_out*) [static]

Helper: Format a log message into a fixed-sized buffer. (This is factored out of **logv** so that we never format a message more than once.) Return a pointer to the first character of the message portion of the formatted string.

References _log_prefix(), LD_BUG, should_log_function_name(), tor_snprintf(), and tor_vsnprintf().

Referenced by logv().

4.28.3.13 int get_min_log_level (void)

Return the least severe log level that any current log is interested in.

References log_severity_list_t::masks, logfile_t::next, and logfile_t::severities.

Referenced by add_callback_log(), add_file_log(), add_stream_log_impl(), add_syslog_log(), change_callback_log_severity(), close_temp_logs(), signal_callback(), and switch_logs_debug().

4.28.3.14 void init_logging (void)

Initialize the global logging facility

References tor_mutex_init().

Referenced by tor_main().

4.28.3.15 static void log_free (logfile_t * *victim*) [static]

Free all storage held by **victim**.

References logfile_t::filename, logfile_t::severities, and tor_free.

Referenced by close_temp_logs(), delete_log(), and logs_free_all().

4.28.3.16 const char* log_level_to_string (int *level*)

Return the string equivalent of a given log level.

References sev_to_string().

4.28.3.17 void log_set_application_name (const char ∗ *name*)

Set the "application name" for the logs to **name**: we'll use this name in the message we write when starting up, and at the start of each new log.

Tor uses this string to write the version number to the log file.

References appname, and tor_free.

Referenced by tor_init().

4.28.3.18 static int log_tor_version (logfile_t ∗ *lf*, int *reset*) [static]

If lf refers to an actual file that we have just opened, and the file contains no data, log an "opening new logfile" message at the top.

Return -1 if the log is broken and needs to be deleted, else return 0.

References _log_prefix(), appname, logfile_t::fd, logfile_t::is_temporary, logfile_t::needs_close, tor_fd_getpos(), tor_snprintf(), and write_all().

Referenced by add_file_log().

4.28.3.19 void logs_free_all (void)

Close all open log files, and free other static memory.

References appname, close_log(), log_free(), logfile_t::next, and tor_free.

Referenced by tor_free_all().

4.28.3.20 static void logv (int *severity*, log_domain_mask_t *domain*, const char ∗ *funcname*, const char ∗ *format*, va_list *ap*) [static]

Helper: sends a message to the appropriate logfiles, at loglevel **severity**. If provided, **funcname** is prepended to the message. The actual message is derived as from tor_snprintf(format,ap).

References logfile_t::callback, logfile_t::fd, format_msg(), logfile_t::is_syslog, log_severity_list_t::masks, logfile_t::next, logfile_t::seems_dead, logfile_t::severities, tor_free, and write_all().

Referenced by tor_log().

4.28.3.21 void mark_logs_temp (void)

Configure all log handles to be closed by **close_temp_logs()** (p. 671).

References logfile_t::is_temporary, and logfile_t::next.

Referenced by options_act_reversible().

4.28.3.22 static log_domain_mask_t parse_log_domain (const char ∗ *domain*) [static]

Return a bitmask for the log domain for which **domain** is the name, or 0 if there is no such name.

Referenced by parse_log_severity_config().

4.28.3.23 int parse_log_level (const char * *level*)

If **level** is a valid log severity, return the corresponding numeric value. Otherwise, return -1.

Referenced by parse_log_severity_config().

4.28.3.24 int parse_log_severity_config (const char ** *cfg_ptr*, log_severity_list_t * *severity_out*)

Parse a log severity pattern in ***cfg_ptr**. Advance cfg_ptr after the end of the severityPattern. Set the value of **severity_out** to the parsed pattern. Return 0 on success, -1 on failure.

The syntax for a SeverityPattern is:

```
SeverityPattern = *(DomainSeverity SP)* DomainSeverity
DomainSeverity = (DomainList SP)? SeverityRange
SeverityRange = MinSeverity ("-" MaxSeverity )?
DomainList = "[" (SP? DomainSpec SP? ",") SP? DomainSpec "]"
DomainSpec = "*" | Domain | "~" Domain
```

A missing MaxSeverity defaults to ERR. Severities and domains are case-insensitive. "∼" indicates negation for a domain; negation happens last inside a DomainList. Only one SeverityRange without a DomainList is allowed per line.

References eat_whitespace(), LD_CONFIG, log_severity_list_t::masks, parse_log_domain(), parse_log_level(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcasecmpstart(), and tor_free.

Referenced by options_init_logs().

4.28.3.25 void rollback_log_changes (void)

Make all currently temporary logs (set to be closed by close_temp_logs) live again, and close all non-temporary logs.

References close_temp_logs(), logfile_t::is_temporary, and logfile_t::next.

Referenced by options_act_reversible().

4.28.3.26 void set_log_severity_config (int *loglevelMin*, int *loglevelMax*, log_severity_list_t * *severity_out*)

Adjust a log severity configuration in **severity_out** to contain every domain between **loglevelMin** and **loglevelMax**, inclusive.

References log_severity_list_t::masks, and tor_assert.

Referenced by add_temp_log(), and change_callback_log_severity().

4.28.3.27 static INLINE const char* sev_to_string (int *severity*) `[static]`

Helper: map a log severity to descriptive string.

Referenced by _log_prefix(), and log_level_to_string().

4.28.3.28 static INLINE int should_log_function_name (log_domain_mask_t *domain*, int *severity*) [static]

Helper: decide whether to include the function name in the log message.

References LD_BUG.

Referenced by format_msg().

4.28.3.29 void switch_logs_debug (void)

Switch all logs to output at most verbose level.

References _log_global_min_severity, get_min_log_level(), log_severity_list_t::masks, logfile_t::next, and logfile_t::severities.

Referenced by signal_callback().

4.28.3.30 void tor_log (int *severity*, log_domain_mask_t *domain*, const char * *format*, *...*)

Output a message to the log.

References _log_global_min_severity, and logv().

Referenced by circuit_log_path(), circuit_note_clock_jumped(), and tor_log_mallinfo().

4.28.4 Variable Documentation

4.28.4.1 const char* _log_fn_function_name = NULL

Output a message to the log, prefixed with a function name **fn**.

4.28.4.2 int _log_global_min_severity = LOG_NOTICE

What's the lowest log level anybody cares about? Checking this lets us bail out early from log_-debug if we aren't debugging.

Referenced by add_callback_log(), add_file_log(), add_stream_log_impl(), add_syslog_log(), change_callback_log_severity(), close_temp_logs(), switch_logs_debug(), and tor_log().

4.28.4.3 char* appname = NULL [static]

Name of the application: used to generate the message we write at the start of each new log.

Referenced by log_set_application_name(), log_tor_version(), and logs_free_all().

4.28.4.4 const char* domain_list[] [static]

Initial value:

```
{
  "GENERAL", "CRYPTO", "NET", "CONFIG", "FS", "PROTOCOL", "MM",
  "HTTP", "APP", "CONTROL", "CIRC", "REND", "BUG", "DIR", "DIRSERV",
```

```
  "OR", "EDGE", "ACCT", "HIST", "HANDSHAKE", NULL
}
```

NULL-terminated array of names for log domains such that domain_list[dom] is a description of **dom**.

4.28.4.5 tor_mutex_t log_mutex `[static]`

A mutex to guard changes to logfiles and logging.

4.28.4.6 logfile_t* logfiles = NULL `[static]`

Linked list of **logfile_t** (p. 108).

4.28.4.7 int syslog_count = 0 `[static]`

The number of open syslog log handlers that we have. When this reaches 0, we can close our connection to the syslog facility.

Referenced by add_syslog_log(), and close_log().

4.29 log.h File Reference

Headers for **log.c** (p. 668).

`#include "compat.h"`

`#include <syslog.h>`

Data Structures

- struct **log_severity_list_t**

Defines

- #define **LOG_WARN** LOG_WARNING
- #define **LD_GENERAL** (1u<<0)
- #define **LD_CRYPTO** (1u<<1)
- #define **LD_NET** (1u<<2)
- #define **LD_CONFIG** (1u<<3)
- #define **LD_FS** (1u<<4)
- #define **LD_PROTOCOL** (1u<<5)
- #define **LD_MM** (1u<<6)
- #define **LD_HTTP** (1u<<7)
- #define **LD_APP** (1u<<8)
- #define **LD_CONTROL** (1u<<9)
- #define **LD_CIRC** (1u<<10)
- #define **LD_REND** (1u<<11)
- #define **LD_BUG** (1u<<12)
- #define **LD_DIR** (1u<<13)
- #define **LD_DIRSERV** (1u<<14)
- #define **LD_OR** (1u<<15)
- #define **LD_EDGE** (1u<<16)
- #define **LD_EXIT** LD_EDGE
- #define **LD_ACCT** (1u<<17)
- #define **LD_HIST** (1u<<18)
- #define **LD_HANDSHAKE** (1u<<19)
- #define **N_LOGGING_DOMAINS** 20
- #define **log** tor_log
- #define **log_fn** (**_log_fn_function_name**=__func__),_log_fn
- #define **log_debug** (**_log_fn_function_name**=__func__),_log_debug
- #define **log_info** (**_log_fn_function_name**=__func__),_log_info
- #define **log_notice** (**_log_fn_function_name**=__func__),_log_notice
- #define **log_warn** (**_log_fn_function_name**=__func__),_log_warn
- #define **log_err** (**_log_fn_function_name**=__func__),_log_err
- #define **_TOR_LOG_H**

Typedefs

- typedef uint32_t **log_domain_mask_t**
- typedef void(∗ **log_callback**)(int severity, uint32_t domain, const char ∗msg)

Functions

- void **init_logging** (void)
- int **parse_log_level** (const char ∗level)
- const char ∗ **log_level_to_string** (int level)
- int **parse_log_severity_config** (const char ∗∗cfg, **log_severity_list_t** ∗severity_out)
- void **set_log_severity_config** (int minSeverity, int maxSeverity, **log_severity_list_t** ∗severity_out)
- void **add_stream_log** (const **log_severity_list_t** ∗severity, const char ∗name, int fd)
- int **add_file_log** (const **log_severity_list_t** ∗severity, const char ∗filename)
- int **add_syslog_log** (const **log_severity_list_t** ∗severity)
- int **add_callback_log** (const **log_severity_list_t** ∗severity, **log_callback** cb)
- int **get_min_log_level** (void)
- void **switch_logs_debug** (void)
- void **logs_free_all** (void)
- void **add_temp_log** (int min_severity)
- void **close_temp_logs** (void)
- void **rollback_log_changes** (void)
- void **mark_logs_temp** (void)
- void **change_callback_log_severity** (int loglevelMin, int loglevelMax, **log_callback** cb)
- void **log_set_application_name** (const char ∗name)
- void **tor_log** (int severity, log_domain_mask_t domain, const char ∗format,...) CHECK_PRINTF(3
- void **_log_fn** (int severity, log_domain_mask_t domain, const char ∗format,...)
- void **_log_debug** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_info** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_notice** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_warn** (log_domain_mask_t domain, const char ∗format,...)
- void **_log_err** (log_domain_mask_t domain, const char ∗format,...)

Variables

- const char ∗ **_log_fn_function_name**

4.29.1 Detailed Description

Headers for **log.c** (p. 668).

4.29.2 Define Documentation

4.29.2.1 #define LD_ACCT (1u<<17)

Bandwidth accounting.

Referenced by accounting_set_wakeup_time(), configure_accounting(), consider_hibernation(), hibernate_end(), hibernate_end_time_elapsed(), hibernate_go_dormant(), read_bandwidth_usage(), and reset_accounting().

4.29.2.2 #define LD_APP (1u<<8)

Application (socks) requests.

Referenced by addressmap_register(), addressmap_register_virtual_address(), addressmap_rewrite(), addressmap_rewrite_reverse(), circuit_consider_stop_edge_reading(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_resume_edge_reading(), circuit_truncated(), client_dns_incr_failures(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_consider_sending_sendme(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_send_command(), connection_handle_listener_read(), consider_plaintext_ports(), dnsserv_launch_request(), dnsserv_resolved(), evdns_server_callback(), fetch_from_buf_socks(), get_unique_stream_id_by_circ(), link_apconn_to_circ(), parse_extended_hostname(), relay_lookup_conn(), and rend_process_relay_cell().

4.29.2.3 #define LD_BUG (1u<<12)

Internal errors in this Tor process.

Referenced by _circuit_mark_for_close(), _connection_free(), _connection_mark_for_close(), _connection_mark_unattached_ap(), _connection_write_to_buf_impl(), _dirreq_map_put(), _find_by_keyword(), accounting_set_wakeup_time(), add_default_trusted_dir_authorities(), addr_policy_permits_tor_addr(), addressmap_get_virtual_address(), addressmap_register_virtual_address(), append_exit_policy_string(), assert_cpath_layer_ok(), base32_decode(), buf_shrink_freelists(), check_sockaddr_family_match(), choose_good_exit_server(), circuit_detach_stream(), circuit_end_reason_to_control_string(), circuit_get_by_circid_orconn_impl(), circuit_init_cpath_crypto(), circuit_launch_by_extend_info(), circuit_package_relay_cell(), circuit_receive_relay_cell(), circuit_send_next_onion_skin(), circuit_state_to_string(), command_process_var_cell(), config_dump(), configure_nameservers(), conn_read_callback(), conn_state_to_string(), conn_type_to_string(), conn_write_callback(), connection_about_to_close_connection(), connection_ap_get_original_destination(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_reply(), connection_buckets_decrement(), connection_close_immediate(), connection_create_listener(), connection_dir_client_reached_eof(), connection_dir_finished_flushing(), connection_dns_remove(), connection_edge_end(), connection_edge_finished_flushing(), connection_edge_package_raw_inbuf(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), connection_edge_send_command(), connection_exit_begin_conn(), connection_exit_connect(), connection_finished_connecting(), connection_finished_flushing(), connection_free(), connection_handle_write_impl(), connection_or_finished_flushing(), connection_or_flush_from_first_active_circuit(), connection_or_remove_from_identity_map(), connection_printf_to_buf(), connection_process_inbuf(), connection_proxy_connect(), connection_reached_eof(), connection_read_proxy_handshake(), connection_read_to_buf(), connection_tls_start_handshake(), connection_unregister_events(), control_event_circuit_status(), control_event_logmsg(), control_event_or_conn_status(), control_event_status(), control_event_stream_status(), control_signal_act(), cpuworker_main(), crypto_pk_public_checksig_digest(), cull_wedged_cpuworkers(), dir_conn_purpose_to_string(), dir_networkstatus_download_failed(), dir_routerdesc_download_failed(), directory_get_from_dirserver(), directory_get_from_hs_dir(), directory_send_command(), dirserv_add_own_fingerprint(), dirserv_dump_directory_to_string(), dirserv_generate_networkstatus_vote_obj(), dirserv_pick_cached_dir_obj(), dirserv_regenerate_directory(), dirserv_router_get_status(), dirvote_add_vote(), dns_cancel_pending_resolve(), dns_found_answer(), dns_launch_correctness_checks(), dns_reset(), dns_resolve_impl(), do_list_fingerprint(), do_main_loop(), entry_guards_parse_state(), evdns_callback(), extrainfo_dump_to_string(), format_msg(), format_networkstatus_vote(), generate_runningrouters(), generate_v2_networkstatus_opinion(), get_assigned_option(), get_unique_circ_id_by_conn(), getinfo_helper_events(), handle_signals(), init_keys(), launch_resolve(), networkstatus_get_detached_signatures(), networkstatus_set_current_consensus(), networkstatus_v2_parse_from_string(), networkstatus_verify_bw_weights(), new_cached_dir(), note_crypto_pk_op(), onionskin_answer(), option_reset(), options_act(), options_dup(), or_state_load(), orconn_end_reason_to_control_string(), purge_expired_resolves(), relay_crypt_one_payload(), relay_send_command_from_edge(), rend_cache_store(), rend_check_authorization(), rend_client_remove_intro_point(), rend_client_send_establish_rendezvous(), rend_config_services(), rend_encode_v2_descriptors(), rend_mid_establish_intro(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_load_keys(), rotate_onion_key(), router_append_dirobj_signature(), router_dirport_found_reachable(), router_dump_router_to_string(), router_get_hash_impl(), router_get_hashes_impl(), router_orport_found_reachable(), router_rebuild_descriptor(), router_rebuild_store(), routerlist_replace(), routerstatus_format_entry(), run_scheduled_events(), send_control_event_impl(), set_cached_dir(), set_options(), should_log_function_name(), smartlist_choose_by_bandwidth(), smartlist_choose_by_bandwidth_weights(), tor_addr_is_internal(), tor_addr_is_null(), tor_asprintf(), tor_close_socket(), tor_gzip_compress(), tor_gzip_uncompress(), tor_init(), tor_main(), tor_timegm(), tor_tls_free(), tor_tls_get_n_raw_bytes(), tor_tls_handshake(), tor_tls_server_info_callback(), tor_version_as_new_as(), tor_version_is_obsolete(), tor_zlib_new(), update_router_descriptor_cache_downloads_v2(), upload_service_descriptor(), warn_if_last_router_excluded(), write_configuration_file(), write_http_status_line(), and write_str_to_file().

4.29.2.4 #define LD_CIRC (1u<<10)

Building, using, and managing circuits.

Referenced by add_an_entry_guard(), choose_good_exit_server_general(), choose_good_middle_server(), choose_random_entry(), circuit_build_times_add_time(), circuit_build_times_add_timeout(), circuit_build_times_add_timeout_worker(), circuit_build_times_get_initial_timeout(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), circuit_build_times_new_consensus_params(), circuit_build_times_parse_state(), circuit_build_times_rewind_history(), circuit_build_times_set_timeout(), circuit_build_times_shuffle_and_store_array(), circuit_build_times_update_alpha(), circuit_consider_sending_sendme(), circuit_count_pending_on_or_conn(), circuit_deliver_create_cell(), circuit_dump_details(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_expire_old_circuits_serverside(), circuit_extend(), circuit_extend_to_new_exit(), circuit_find_to_cannibalize(), circuit_finish_handshake(), circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_handle_first_hop(), circuit_increment_failure_count(), circuit_is_acceptable(), circuit_launch_by_extend_info(), circuit_n_conn_done(), circuit_predict_and_launch_new(), circuit_remove_handled_ports(), circuit_send_next_onion_skin(), circuit_truncated(), connection_ap_handshake_attach_circuit(), consider_testing_reachability(), entry_guard_set_status(), entry_guards_compute_status(), entry_guards_prepend_from_config(), entry_nodes_should_be_added(), get_unique_circ_id_by_conn(), link_apconn_to_circ(), log_entry_guards(), new_route_len(), onion_extend_cpath(), onion_pending_add(), onion_pick_cpath_exit(), onion_populate_cpath(), onionskin_answer(), options_act(), remove_dead_entry_guards(), remove_obsolete_entry_guards(), rend_service_intro_has_opened(), rep_hist_get_predicted_ports(), smartlist_choose_-

by_bandwidth(), smartlist_choose_by_bandwidth_weights(), and warn_if_last_router_excluded().

4.29.2.5 #define LD_CONFIG (1u<<3)

Parsing and acting on our configuration.

Referenced by accounting_parse_options(), add_nickname_list_to_smartlist(), add_trusted_dir_server(), addressmap_get_virtual_address(), addressmap_register(), addressmap_rewrite(), check_descriptor_ipaddress_changed(), config_assign_line(), config_assign_value(), config_find_option(), config_get_commandlines(), config_parse_interval(), config_parse_units(), config_register_addressmaps(), configure_nameservers(), connection_connect(), create_inet_sockaddr(), dirserv_load_fingerprint_file(), dirserv_read_measured_bandwidths(), do_hup(), exit_policy_remove_redundancies(), expand_abbrev(), expand_filename(), find_torrc_filename(), get_assigned_option(), get_user_homedir(), load_torrc_from_disk(), normalize_data_directory(), options_act(), options_init_from_torrc(), options_init_logs(), options_validate(), parse_addr_policy(), parse_bridge_line(), parse_dir_server_line(), parse_log_severity_config(), parse_port_config(), parse_reachable_addresses(), rend_add_service(), rend_config_services(), rend_parse_client_keys(), rend_parse_service_authorization(), rend_service_load_keys(), resolve_my_address(), router_pick_published_address(), router_rebuild_descriptor(), routerset_parse(), routerset_refresh_countries(), run_scheduled_events(), second_elapsed_callback(), set_max_file_descriptors(), signal_callback(), switch_id(), tor_init(), tor_version_is_obsolete(), validate_data_directory(), validate_dir_authorities(), and write_configuration_file().

4.29.2.6 #define LD_CONTROL (1u<<9)

Communication via the controller protocol.

Referenced by connection_control_reached_eof(), control_connection_new(), control_event_bootstrap(), control_event_bootstrap_problem(), control_setconf_helper(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_extendcircuit(), handle_control_loadconf(), handle_control_mapaddress(), handle_control_resolve(), run_scheduled_events(), and signal_callback().

4.29.2.7 #define LD_CRYPTO (1u<<1)

The cryptography subsystem.

Referenced by _check_no_tls_errors(), crypto_create_init_cipher(), crypto_dh_compute_secret(), crypto_dh_generate_public(), crypto_dh_get_public(), crypto_global_init(), crypto_log_errors(), crypto_pk_private_hybrid_decrypt(), crypto_pk_public_checksig_digest(), crypto_pk_read_private_key_from_filename(), crypto_seed_rng(), log_engine(), tor_check_dh_key(), and tor_tls_context_new().

4.29.2.8 #define LD_DIR (1u<<13)

Learning and using information about Tor servers.

Referenced by _free_duplicate_routerstatus_entry(), any_pending_bridge_descriptor_fetches(), authority_cert_parse_from_string(), authority_certs_fetch_missing(), bridges_retry_helper(), check_signature_token(), circuit_get_open_circ_or_launch(), client_likes_consensus(), connection_dir_client_reached_eof(), connection_dir_finished_flushing(), connection_dir_request_failed(), decide_to_advertise_dirport(), dir_signing_key_is_trusted(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_get_from_dirserver(), directory_handle_command_post(), directory_info_has_arrived(), directory_initiate_command_rend(), directory_initiate_command_routerstatus_rend(), directory_post_to_dirservers(), directory_send_command(), dirserv_add_descriptor(), dirserv_add_extrainfo(), dirserv_clear_old_networkstatuses(), dirvote_act(), dirvote_add_signatures(), dirvote_add_signatures_to_pending_consensus(), dirvote_add_vote(), dirvote_compute_consensuses(), dirvote_create_microdescriptor(), dirvote_perform_vote(), dirvote_publish_consensus(), dirvote_recalculate_timing(), do_main_loop(), download_status_increment_failure(), dump_desc(), dump_routerlist_mem_usage(), extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), fetch_bridge_descriptors(), find_dir_signing_key(), http_set_address_origin(), init_keys(), launch_router_descriptor_downloads(), learned_bridge_descriptor(), load_authority_keyset(), microdesc_cache_rebuild(), microdesc_cache_reload(), microdescs_parse_from_string(), networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), networkstatus_check_document_signature(), networkstatus_check_weights(), networkstatus_compute_consensus(), networkstatus_get_detached_signatures(), networkstatus_getinfo_by_purpose(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_set_current_consensus(), networkstatus_v2_list_clean(), networkstatus_v2_parse_from_string(), networkstatus_verify_bw_weights(), parse_bridge_line(), parse_dir_server_line(), parse_http_url(), rend_parse_v2_service_descriptor(), router_add_exit_policy(), router_add_to_routerlist(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_load_single_router(), router_new_address_suggestion(), router_parse_addr_policy(), router_parse_addr_policy_item_from_string(), router_parse_directory(), router_parse_entry_from_string(), router_parse_list_from_string(), router_parse_runningrouters(), router_pick_directory_server(), router_pick_trusteddirserver(), router_rebuild_store(), router_reload_router_list_impl(), router_reload_v2_networkstatus(), router_set_networkstatus_v2(), router_set_status(), routerlist_remove_old_routers(), routerstatus_parse_entry_from_string(), run_connection_housekeeping(), should_delay_dir_fetches(), signed_descriptor_get_body_impl(), tokenize_string(), tor_version_as_new_as(), trusted_dirs_load_certs_from_string(), update_consensus_networkstatus_downloads(), update_consensus_networkstatus_fetch_time(), update_consensus_router_descriptor_downloads(), update_extrainfo_downloads(), update_router_have_minimum_dir_info(), and v3_authority_check_key_expiry().

4.29.2.9 #define LD_DIRSERV (1u<<14)

Learning and using information about Tor servers.

Referenced by add_fingerprint_to_dir(), authdir_wants_to_reject_router(), connection_dir_finished_flushing(), connection_handle_listener_read(), directory_handle_command(), directory_handle_command_get(), directory_handle_command_post(), directory_remove_invalid(), dirserv_add_descriptor(), dirserv_get_networkstatus_v2(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_status_impl(), dirserv_orconn_tls_done(), dirserv_pick_cached_dir_obj(), dirserv_read_measured_bandwidths(), dirserv_regenerate_directory(), dirserv_router_has_valid_address(), measured_bw_line_apply(), measured_bw_line_parse(), router_dirport_found_reachable(), and set_cached_dir().

4.29.2.10 #define LD_EDGE (1u<<16)

Generic edge-connection functionality.

Referenced by connection_edge_destroy(), connection_edge_end(), connection_edge_process_ inbuf(), and connection_edge_reached_eof().

4.29.2.11 #define LD_FS (1u<<4)

Reading and writing from the filesystem.

Referenced by accounting_run_housekeeping(), add_networkstatus_to_cache(), check_private_dir(), dirserv_load_fingerprint_file(), fdopen_file(), finish_writing_to_file_impl(), init_cookie_authentication(), init_key_from_file(), init_keys(), or_state_load(), or_state_save(), read_file_to_str(), rend_service_load_keys(), rotate_onion_key(), router_rebuild_store(), router_reload_consensus_networkstatus(), router_reload_v2_networkstatus(), signed_desc_append_to_journal(), start_writing_to_file(), tor_listdir(), tor_lockfile_lock(), tor_lockfile_unlock(), tor_mmap_file(), write_chunks_to_file_impl(), write_configuration_file(), and write_pidfile().

4.29.2.12 #define LD_GENERAL (1u<<0)

Catch-all for miscellaneous events and fatal errors.

Referenced by _connection_free(), append_cell_to_circuit_queue(), check_descriptor_bandwidth_changed(), check_private_dir(), circuit_note_clock_jumped(), circuit_send_next_onion_skin(), circuit_testing_failed(), command_process_create_cell(), command_process_netinfo_cell(), conn_close_if_marked(), connection_cpu_reached_eof(), connection_dir_client_reached_eof(), connection_handle_write_impl(), connection_or_flush_from_first_active_circuit(), connection_read_to_buf(), consider_hibernation(), dirserv_load_fingerprint_file(), do_hup(), do_list_fingerprint(), do_main_loop(), dump_distinct_digest_count(), dumpmemusage(), dumpstats(), extrainfo_dump_to_string(), finish_daemon(), geoip_load_file(), geoip_parse_entry(), getinfo_helper_events(), hibernate_begin(), init_key_from_file(), init_keys(), libevent_logging_callback(), log_addr_has_changed(), log_cert_lifetime(), log_credential_status(), networkstatus_set_current_consensus(), onion_pending_add(), onion_pending_remove(), onion_skin_server_handshake(), options_act(), or_state_load(), or_state_save(), or_state_set(), parse_addr_and_port_range(), parse_addr_port(), parse_iso_time(), parse_port_range(), parse_rfc1123_time(), remove_file_if_very_old(), rend_client_receive_rendezvous(), rend_client_send_establish_rendezvous(), rend_mid_establish_intro(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_dump_stats(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), rotate_onion_key(), router_get_consensus_status_by_nickname(), router_get_my_descriptor(), router_load_routers_from_string(), router_set_networkstatus_v2(), router_upload_dir_desc_to_dirservers(), routers_update_all_from_networkstatus(), run_scheduled_events(), signal_callback(), spawn_enough_cpuworkers(), start_daemon(), start_writing_to_file(), switch_id(), tor_addr_parse_mask_ports(), tor_check_libevent_header_compatibility(), tor_check_libevent_version(), tor_gettimeofday(), tor_gzip_compress(), tor_gzip_uncompress(), tor_init(), tor_libevent_initialize(), tor_mlockall(), tor_mutex_acquire(), tor_mutex_init(), tor_mutex_release(), tor_mutex_uninit(), tor_tls_init(), tor_zlib_process(), try_locking(), tv_mdiff(), and tv_udiff().

4.29.2.13 #define LD_HANDSHAKE (1u<<19)

OR handshaking

Referenced by connection_or_check_valid_tls_handshake(), connection_or_finished_connecting(), connection_tls_finish_handshake(), connection_tls_start_handshake(), tor_tls_handshake(), tor_tls_peer_has_cert(), tor_tls_renegotiate(), and tor_tls_verify().

4.29.2.14 #define LD_HIST (1u<<18)

Router history

Referenced by dump_pk_ops(), rep_hist_buffer_stats_write(), rep_hist_downrate_old_runs(), rep_hist_dump_stats(), rep_hist_exit_stats_write(), rep_hist_load_mtbf_data(), rep_hist_load_state(), rep_hist_note_exit_bytes_read(), rep_hist_note_exit_bytes_written(), rep_hist_note_exit_stream_opened(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), and rep_hist_record_mtbf_data().

4.29.2.15 #define LD_HTTP (1u<<7)

HTTP implementation.

Referenced by connection_dir_client_reached_eof(), connection_dir_finished_connecting(), connection_dir_process_inbuf(), connection_dir_reached_eof(), fetch_from_buf_http(), and parse_http_response().

4.29.2.16 #define LD_MM (1u<<6)

Memory management.

Referenced by _tor_malloc(), _tor_realloc(), _tor_strdup(), buf_dump_freelist_sizes(), buf_shrink_freelists(), dump_cell_pool_usage(), dump_dns_mem_usage(), mp_pool_log_status(), mp_pool_new(), policies_free_all(), tor_log_mallinfo(), and tor_tls_free_all().

4.29.2.17 #define LD_NET (1u<<2)

Networking.

Referenced by _check_no_tls_errors(), _connection_free(), _connection_write_to_buf_impl(), check_sockaddr(), client_check_address_changed(), conn_close_if_marked(), conn_read_callback(), conn_write_callback(), connection_add(), connection_ap_get_original_destination(), connection_bucket_refill(), connection_bucket_refill_helper(), connection_close_immediate(), connection_connect(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_create_listener(), connection_expire_held_open(), connection_handle_listener_read(), connection_handle_write_impl(), connection_proxy_connect(), connection_read_https_proxy_response(), connection_read_proxy_handshake(), connection_read_to_buf(), connection_remove(), connection_start_reading(), connection_start_writing(), connection_stop_reading(), connection_stop_writing(), directory_all_unreachable(), directory_initiate_command_rend(), dirserv_generate_networkstatus_vote_obj(), dirvote_perform_vote(), do_main_loop(), dumpstats(), fetch_from_buf_socks_client(), flush_chunk(), flush_chunk_tls(), generate_v2_networkstatus_opinion(), get_interface_address6(), got_libevent_error(), hibernate_begin(), hibernate_go_dormant(), libevent_logging_callback(), log_cert_lifetime(), network_init(), options_act_reversible(), parse_addr_port(), read_to_chunk(), rectify_client_ciphers(), resolve_my_address(), router_set_status(), second_elapsed_callback(), set_constrained_socket_buffers(), set_max_file_descriptors(), spawn_cpuworker(), tor_close_socket(), tor_tls_check_lifetime(), tor_tls_client_is_using_v2_ciphers(), tor_tls_context_new(), tor_tls_create_certificate(), tor_tls_get_error(), tor_tls_new(), tor_tls_read(), tor_tls_shutdown(), tor_tls_write(), and warn_too_many_conns().

4.29.2.18 #define LD_OR (1u<<15)

Onion routing protocol.

Referenced by assign_onionskin_to_cpuworker(), cell_ewma_set_scale_factor(), circuit_build_failed(), circuit_extend(), circuit_package_relay_cell(), circuit_receive_relay_cell(), command_process_cell(), command_process_create_cell(), command_process_created_cell(), command_process_destroy_cell(), command_process_netinfo_cell(), command_process_relay_cell(), command_process_var_cell(), command_process_versions_cell(), connection_cpu_process_inbuf(), connection_or_group_set_badness(), connection_or_process_cells_from_inbuf(), connection_or_reached_eof(), connection_or_send_destroy(), connection_or_set_state_open(), connection_proxy_connect(), connection_tls_continue_handshake(), cpuworker_main(), dirserv_single_reachability_test(), errno_to_orconn_end_reason(), inform_testing_reachability(), process_pending_task(), relay_crypt(), relay_send_command_from_edge(), router_orport_found_reachable(), router_perform_bandwidth_test(), run_connection_housekeeping(), and spawn_cpuworker().

4.29.2.19 #define LD_PROTOCOL (1u<<5)

Other servers' (non)compliance with the Tor protocol.

Referenced by circuit_extend(), circuit_finish_handshake(), circuit_receive_relay_cell(), command_process_cell(), command_process_create_cell(), command_process_created_cell(), command_process_relay_cell(), connection_dir_client_reached_eof(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_or_connect(), connection_read_https_proxy_response(), directory_handle_command(), fast_client_handshake(), fetch_from_buf_http(), fetch_from_buf_socks(), onion_skin_client_handshake(), onion_skin_server_handshake(), relay_crypt(), rend_cache_store(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_rendezvous_acked(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_parse_service_descriptor(), rend_process_relay_cell(), rend_service_intro_established(), rend_service_introduce(), run_connection_housekeeping(), stream_end_reason_to_socks5_response(), stream_end_reason_to_string(), tor_gzip_uncompress(), tor_tls_verify(), and try_to_extract_certs_from_tls().

4.29.2.20 #define LD_REND (1u<<11)

Hidden services.

Referenced by _circuit_mark_for_close(), circuit_build_failed(), circuit_expire_building(), circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_receive_relay_cell(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_dir_client_reached_eof(), connection_exit_begin_conn(), directory_get_from_hs_dir(), directory_handle_command_get(), directory_handle_command_post(), directory_post_to_hs_dir(), hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), rend_add_service(), rend_cache_clean_v2_descs_as_dir(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_introcirc_has_opened(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_refetch_v2_renddesc(), rend_client_remove_intro_point(), rend_client_rendcirc_has_opened(), rend_client_rendezvous_acked(), rend_client_send_establish_rendezvous(), rend_client_send_introduction(), rend_compute_v2_desc_id(), rend_config_services(), rend_decrypt_introduction_points(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_encrypt_v2_intro_points_basic(), rend_encrypt_v2_intro_points_stealth(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_load_keys(), rend_service_relaunch_rendezvous(), rend_service_rendezvous_has_opened(), rend_service_set_connection_addr_port(), rend_services_introduce(), and upload_service_descriptor().

4.29.2.21 #define N_LOGGING_DOMAINS 20

Number of logging domains in the code.

4.29.3 Typedef Documentation

4.29.3.1 typedef void(* log_callback)(int severity, uint32_t domain, const char *msg)

Callback type used for add_callback_log.

4.29.4 Function Documentation

4.29.4.1 int add_callback_log (const log_severity_list_t * *severity*, log_callback *cb*)

Add a log handler to send messages in **severity** to the function **cb**.

References _log_global_min_severity, logfile_t::callback, logfile_t::fd, logfile_t::filename, get_min_log_level(), logfile_t::next, and logfile_t::severities.

Referenced by options_act_reversible().

4.29.4.2 int add_file_log (const log_severity_list_t * *severity*, const char * *filename*)

Add a log handler to send messages to **filename**. If opening the logfile fails, -1 is returned and errno is set appropriately (by open(2)).

References _log_global_min_severity, add_stream_log_impl(), delete_log(), get_min_log_level(), log_tor_version(), logfile_t::needs_close, and tor_fd_seekend().

Referenced by options_init_logs().

4.29.4.3 void add_stream_log (const log_severity_list_t * *severity*, const char * *name*, int *fd*)

Add a log handler named **name** to send all messages in **severity** to **fd**. Steals a reference to **severity**; the caller must not use it after calling this function.

References add_stream_log_impl().

Referenced by options_init_logs().

4.29.4.4 int add_syslog_log (const log_severity_list_t * *severity*)

Add a log handler to send messages to they system log facility.

References _log_global_min_severity, logfile_t::fd, logfile_t::filename, get_min_log_level(), logfile_t::is_syslog, logfile_t::next, logfile_t::severities, and syslog_count.

Referenced by options_init_logs().

4.29.4.5 void add_temp_log (int *min_severity*)

Add a log handler to receive messages during startup (before the real logs are initialized).

References add_stream_log_impl(), logfile_t::is_temporary, set_log_severity_config(), and tor_free.

Referenced by tor_init().

4.29.4.6 void change_callback_log_severity (int *loglevelMin*, int *loglevelMax*, log_callback *cb*)

Adjust the configured severity of any logs whose callback function is **cb**.

References _log_global_min_severity, logfile_t::callback, get_min_log_level(), logfile_t::next, set_log_severity_config(), and logfile_t::severities.

Referenced by control_adjust_event_log_severity().

4.29.4.7 void close_temp_logs (void)

Close any log handlers added by **add_temp_log()** (p. 671) or marked by **mark_logs_temp()** (p. 673).

References _log_global_min_severity, close_log(), get_min_log_level(), log_free(), and logfile_t::next.

Referenced by options_act_reversible(), and rollback_log_changes().

4.29.4.8 int get_min_log_level (void)

Return the least severe log level that any current log is interested in.

References log_severity_list_t::masks, logfile_t::next, and logfile_t::severities.

Referenced by add_callback_log(), add_file_log(), add_stream_log_impl(), add_syslog_log(), change_callback_log_severity(), close_temp_logs(), signal_callback(), and switch_logs_debug().

4.29.4.9 void init_logging (void)

Initialize the global logging facility

References tor_mutex_init().

Referenced by tor_main().

4.29.4.10 const char* log_level_to_string (int *level*)

Return the string equivalent of a given log level.

References sev_to_string().

4.29.4.11 void log_set_application_name (const char * *name*)

Set the "application name" for the logs to **name**: we'll use this name in the message we write when starting up, and at the start of each new log.

Tor uses this string to write the version number to the log file.

References appname, and tor_free.

Referenced by tor_init().

4.29.4.12 void logs_free_all (void)

Close all open log files, and free other static memory.

References appname, close_log(), log_free(), logfile_t::next, and tor_free.

Referenced by tor_free_all().

4.29.4.13 void mark_logs_temp (void)

Configure all log handles to be closed by **close_temp_logs()** (p. 671).

References logfile_t::is_temporary, and logfile_t::next.

Referenced by options_act_reversible().

4.29.4.14 int parse_log_level (const char * *level*)

If **level** is a valid log severity, return the corresponding numeric value. Otherwise, return -1.

Referenced by parse_log_severity_config().

4.29.4.15 int parse_log_severity_config (const char ** *cfg_ptr*, log_severity_list_t * *severity_out*)

Parse a log severity pattern in ***cfg_ptr**. Advance cfg_ptr after the end of the severityPattern. Set the value of **severity_out** to the parsed pattern. Return 0 on success, -1 on failure.

The syntax for a SeverityPattern is:

```
SeverityPattern = *(DomainSeverity SP)* DomainSeverity
DomainSeverity = (DomainList SP)? SeverityRange
SeverityRange = MinSeverity ("-" MaxSeverity )?
DomainList = "[" (SP? DomainSpec SP? ",") SP? DomainSpec "]"
DomainSpec = "*" | Domain | "~" Domain
```

A missing MaxSeverity defaults to ERR. Severities and domains are case-insensitive. "∼" indicates negation for a domain; negation happens last inside a DomainList. Only one SeverityRange without a DomainList is allowed per line.

References eat_whitespace(), LD_CONFIG, log_severity_list_t::masks, parse_log_domain(), parse_log_level(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcasecmpstart(), and tor_free.

Referenced by options_init_logs().

4.29.4.16 void rollback_log_changes (void)

Make all currently temporary logs (set to be closed by close_temp_logs) live again, and close all non-temporary logs.

References close_temp_logs(), logfile_t::is_temporary, and logfile_t::next.

Referenced by options_act_reversible().

4.29.4.17 void set_log_severity_config (int *loglevelMin*, int *loglevelMax*, log_severity_list_t ∗ *severity_out*)

Adjust a log severity configuration in **severity_out** to contain every domain between **loglevelMin** and **loglevelMax**, inclusive.

References log_severity_list_t::masks, and tor_assert.

Referenced by add_temp_log(), and change_callback_log_severity().

4.29.4.18 void switch_logs_debug (void)

Switch all logs to output at most verbose level.

References _log_global_min_severity, get_min_log_level(), log_severity_list_t::masks, logfile_t::next, and logfile_t::severities.

Referenced by signal_callback().

4.29.5 Variable Documentation

4.29.5.1 const char∗ _log_fn_function_name

Output a message to the log, prefixed with a function name **fn**.

4.30 main.c File Reference

Toplevel module. Handles signals, multiplexes between connections, implements main loop, and drives scheduled events.

`#include "or.h"`

`#include "memarea.h"`

`#include <event.h>`

Defines

- #define **MAIN_PRIVATE**
- #define **MAX_SIGNEWNYM_RATE** 10
- #define **GREEDY_DESCRIPTOR_RETRY_INTERVAL** (10)
- #define **LAZY_DESCRIPTOR_RETRY_INTERVAL** (60)
- #define **DESCRIPTOR_FAILURE_RESET_INTERVAL** (60*60)
- #define **DIR_CONN_MAX_STALL** (5*60)
- #define **TLS_HANDSHAKE_TIMEOUT** (60)
- #define **IDLE_OR_CONN_TIMEOUT** 180
- #define **ENTROPY_INTERVAL** (60*60)
- #define **SAVE_STABILITY_INTERVAL** (30*60)
- #define **CHECK_V3_CERTIFICATE_INTERVAL** (5*60)
- #define **NS_EXPIRY_SLOP** (24*60*60)
- #define **CHECK_EXPIRED_NS_INTERVAL** (2*60)
- #define **WRITE_STATS_INTERVAL** (24*60*60)
- #define **CLEAN_CACHES_INTERVAL** (30*60)
- #define **RETRY_DNS_INTERVAL** (10*60)
- #define **CHECK_DESCRIPTOR_INTERVAL** (60)
- #define **CHECK_IPADDRESS_INTERVAL** (15*60)
- #define **FORCE_REGENERATE_DESCRIPTOR_INTERVAL** (18*60*60)
- #define **BANDWIDTH_RECHECK_INTERVAL** (12*60*60)
- #define **MEM_SHRINK_INTERVAL** (60)
- #define **BRIDGE_STATUSFILE_INTERVAL** (30*60)
- #define **NUM_JUMPED_SECONDS_BEFORE_WARN** 100
- #define **UPTIME_CUTOFF_FOR_NEW_BANDWIDTH_TEST** (6*60*60)

Functions

- void **evdns_shutdown** (int)
- static void **dumpmemusage** (int severity)
- static void **dumpstats** (int severity)
- static void **conn_read_callback** (int fd, short event, void *_conn)
- static void **conn_write_callback** (int fd, short event, void *_conn)
- static void **signal_callback** (int fd, short events, void *arg)
- static void **second_elapsed_callback** (int fd, short event, void *args)
- static int **conn_close_if_marked** (int i)
- static void **connection_start_reading_from_linked_conn** (**connection_t** *conn)
- static int **connection_should_read_from_linked_conn** (**connection_t** *conn)
- int **connection_add** (**connection_t** *conn)

- void **connection_unregister_events** (**connection_t** ∗conn)
- int **connection_remove** (**connection_t** ∗conn)
- static void **connection_unlink** (**connection_t** ∗conn)
- void **add_connection_to_closeable_list** (**connection_t** ∗conn)
- int **connection_is_on_closeable_list** (**connection_t** ∗conn)
- int **connection_in_array** (**connection_t** ∗conn)
- **smartlist_t** ∗ **get_connection_array** (void)
- void **connection_watch_events** (**connection_t** ∗conn, watchable_events_t events)
- int **connection_is_reading** (**connection_t** ∗conn)
- void **connection_stop_reading** (**connection_t** ∗conn)
- void **connection_start_reading** (**connection_t** ∗conn)
- int **connection_is_writing** (**connection_t** ∗conn)
- void **connection_stop_writing** (**connection_t** ∗conn)
- void **connection_start_writing** (**connection_t** ∗conn)
- void **connection_stop_reading_from_linked_conn** (**connection_t** ∗conn)
- static void **close_closeable_connections** (void)
- void **directory_all_unreachable** (time_t now)
- void **directory_info_has_arrived** (time_t now, int from_cache)
- static void **run_connection_housekeeping** (int i, time_t now)
- static void **signewnym_impl** (time_t now)
- static void **run_scheduled_events** (time_t now)
- static int **got_libevent_error** (void)
- void **ip_address_changed** (int at_interface)
- void **dns_servers_relaunch_checks** (void)
- static int **do_hup** (void)
- int **do_main_loop** (void)
- void **control_signal_act** (int the_signal)
- static void **exit_function** (void)
- void **handle_signals** (int is_parent)
- int **tor_init** (int argc, char ∗argv[ ])
- int **try_locking** (**or_options_t** ∗options, int err_if_locked)
- int **have_lockfile** (void)
- void **release_lockfile** (void)
- void **tor_free_all** (int postfork)
- void **tor_cleanup** (void)
- int **do_list_fingerprint** (void)
- void **do_hash_password** (void)
- int **tor_main** (int argc, char ∗argv[ ])

Variables

- int **global_read_bucket**
- int **global_write_bucket**
- int **global_relayed_read_bucket**
- int **global_relayed_write_bucket**
- static int **stats_prev_global_read_bucket**
- static int **stats_prev_global_write_bucket**
- static uint64_t **stats_n_bytes_read** = 0
- static uint64_t **stats_n_bytes_written** = 0
- time_t **time_of_process_start** = 0

- long **stats_n_seconds_working** = 0
- static time_t **time_to_check_for_correct_dns** = 0
- static time_t **time_of_last_signewnym** = 0
- static int **signewnym_is_pending** = 0
- static **smartlist_t** ∗ **connection_array** = NULL
- static **smartlist_t** ∗ **closeable_connection_lst** = NULL
- static **smartlist_t** ∗ **active_linked_connection_lst** = NULL
- static int **called_loop_once** = 0
- int **has_completed_circuit** = 0
- static struct event ∗ **timeout_event** = NULL
- static int **n_libevent_errors** = 0
- uint64_t **rephist_total_alloc**
- uint32_t **rephist_total_num**
- static **tor_lockfile_t** ∗ **lockfile** = NULL

4.30.1 Detailed Description

Toplevel module. Handles signals, multiplexes between connections, implements main loop, and drives scheduled events.

4.30.2 Define Documentation

4.30.2.1 #define DESCRIPTOR_FAILURE_RESET_INTERVAL (60∗60)

How often do we 'forgive' undownloadable router descriptors and attempt to download them again?

Referenced by run_scheduled_events().

4.30.2.2 #define DIR_CONN_MAX_STALL (5∗60)

How long do we let a directory connection stall before expiring it?

Referenced by run_connection_housekeeping().

4.30.2.3 #define GREEDY_DESCRIPTOR_RETRY_INTERVAL (10)

How often do we check for router descriptors that we should download when we have too little directory info?

Referenced by run_scheduled_events().

4.30.2.4 #define IDLE_OR_CONN_TIMEOUT 180

How long do we wait before killing OR connections with no circuits? In Tor versions up to 0.2.1.25 and 0.2.2.12-alpha, we waited 15 minutes before cancelling these connections, which caused fast relays to accrue many many idle connections. Hopefully 3 minutes is low enough that it kills most idle connections, without being so low that we cause clients to bounce on and off.

Referenced by run_connection_housekeeping().

4.30.2.5 #define LAZY_DESCRIPTOR_RETRY_INTERVAL (60)

How often do we check for router descriptors that we should download when we have enough directory info?

Referenced by run_scheduled_events().

4.30.2.6 #define MAX_SIGNEWNYM_RATE 10

How often will we honor SIGNEWNYM requests?

Referenced by run_scheduled_events(), and signal_callback().

4.30.2.7 #define TLS_HANDSHAKE_TIMEOUT (60)

How long do we let OR connections handshake before we decide that they are obsolete?

4.30.3 Function Documentation

4.30.3.1 void add_connection_to_closeable_list (connection_t * *conn*)

Schedule **conn** to be closed.

References assert_connection_ok(), connection_t::marked_for_close, smartlist_add(), smartlist_isin(), and tor_assert.

Referenced by _connection_mark_for_close().

4.30.3.2 static void close_closeable_connections (void) `[static]`

Close all connections that have been scheduled to get closed.

References connection_t::conn_array_index, conn_close_if_marked(), and connection_unlink().

Referenced by conn_read_callback(), conn_write_callback(), and run_scheduled_events().

4.30.3.3 static int conn_close_if_marked (int *i*) `[static]`

If the connection at connection_array[i] is marked for close, then:

- If it has data that it wants to flush, try to flush it.
- If it _still_ has data to flush, and conn->hold_open_until_flushed is true, then leave the connection open and return.
- Otherwise, remove the connection from connection_array and from all other lists, close it, and free it. Returns 1 if the connection was closed, 0 otherwise.

References connection_t::address, assert_connection_ok(), buf_datalen(), CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, conn_type_to_string(), connection_bucket_write_limit(), connection_start_reading_from_linked_conn(), connection_unlink(), connection_wants_to_flush(), DIR_PURPOSE_SERVER, escaped_safe_str_client(), flush_buf(), flush_buf_tls(), get_options(), connection_t::hold_open_until_flushed, connection_t::inbuf, LD_GENERAL, LD_NET, connection_t::linked_conn, LOG_FN_CONN, connection_t::marked_for_close, connection_t::marked_for_close_file, move_buf_to_buf(), OR_CONN_STATE_OPEN, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::purpose, connection_t::s, server_mode(), connection_t::state, connection_t::timestamp_lastwritten, TO_OR_CONN(), and connection_t::type.

Referenced by close_closeable_connections().

4.30.3.4 static void conn_read_callback (int *fd*, short *event*, void * _ *conn*) `[static]`

Libevent callback: this gets invoked when (connection_t*)**conn** has some data to read.

References assert_connection_ok(), close_closeable_connections(), CONN_IS_EDGE, conn_type_to_string(), connection_edge_end_errno(), LD_BUG, LD_NET, connection_t::marked_for_close, connection_t::s, TO_EDGE_CONN(), tor_fragile_assert, and connection_t::type.

Referenced by connection_add().

4.30.3.5 static void conn_write_callback (int *fd*, short *events*, void * _ *conn*) `[static]`

Libevent callback: this gets invoked when (connection_t*)**conn** has some data to write.

References assert_connection_ok(), close_closeable_connections(), CONN_IS_EDGE, conn_type_to_string(), connection_close_immediate(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, LD_BUG, LD_NET, LOG_FN_CONN, connection_t::marked_for_close, connection_t::s, TO_EDGE_CONN(), tor_fragile_assert, and connection_t::type.

Referenced by connection_add().

4.30.3.6 int connection_add (connection_t * *conn*)

Add **conn** to the array of connections that we can poll on. The connection's socket must be set; the connection starts out non-reading and non-writing.

References connection_t::address, connection_t::conn_array_index, conn_read_callback(), CONN_TYPE_AP, conn_type_to_string(), conn_write_callback(), LD_NET, connection_t::linked, connection_t::read_event, connection_t::s, smartlist_add(), TO_EDGE_CONN(), tor_assert, tor_event_new(), tor_libevent_get_base(), connection_t::type, and connection_t::write_event.

Referenced by connection_ap_make_link(), connection_connect(), connection_create_listener(), connection_exit_connect_dir(), connection_handle_listener_read(), directory_initiate_command_rend(), dnsserv_launch_request(), evdns_server_callback(), and spawn_cpuworker().

4.30.3.7 int connection_in_array (connection_t * *conn*)

Return true iff conn is in the current poll array.

References smartlist_isin().

Referenced by assert_all_pending_dns_resolves_ok(), and connection_free().

4.30.3.8 int connection_is_on_closeable_list (connection_t ∗ *conn*)

Return 1 if conn is on the closeable list, else return 0.

References smartlist_isin().

Referenced by connection_free().

4.30.3.9 int connection_is_reading (connection_t ∗ *conn*)

Return true iff **conn** is listening for read events.

References connection_t::read_event, connection_t::reading_from_linked_conn, and tor_assert.

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), and set_streams_blocked_on_circ().

4.30.3.10 int connection_is_writing (connection_t ∗ *conn*)

Return true iff **conn** is listening for write events.

References tor_assert, connection_t::write_event, and connection_t::writing_to_linked_conn.

Referenced by assert_connection_ok(), and connection_read_to_buf().

4.30.3.11 int connection_remove (connection_t ∗ *conn*)

Remove the connection from the global list, and remove the corresponding poll entry. Calling this function will shift the last connection (if any) into the position occupied by conn.

References connection_t::conn_array_index, conn_type_to_string(), connection_unregister_events(), LD_NET, connection_t::s, smartlist_del(), tor_assert, and connection_t::type.

Referenced by connection_unlink().

4.30.3.12 static int connection_should_read_from_linked_conn (connection_t ∗ *conn*) `[static]`

Return true iff **conn** is linked conn, and reading from the conn linked to it would be good and feasible. (Reading is "feasible" if the other conn exists and has data in its outbuf, and is "good" if we have our reading_from_linked_conn flag set and the other conn has its writing_to_linked_conn flag set.)

References buf_datalen(), connection_t::linked, connection_t::linked_conn, connection_t::outbuf, connection_t::reading_from_linked_conn, and connection_t::writing_to_linked_conn.

Referenced by connection_start_reading(), and connection_start_writing().

4.30.3.13 void connection_start_reading (connection_t ∗ *conn*)

Tell the main loop to start notifying **conn** of any read events.

References connection_should_read_from_linked_conn(), connection_start_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::read_event, connection_t::reading_from_linked_conn, connection_t::s, and tor_assert.

Referenced by circuit_resume_edge_reading_helper(), connection_bucket_refill(), connection_create_listener(), connection_edge_process_relay_cell(), connection_exit_connect_dir(), connection_free(), connection_init_accepted_conn(), connection_or_finished_connecting(), connection_or_set_state_open(), connection_tls_continue_handshake(), connection_tls_start_handshake(), connection_unlink(), connection_watch_events(), directory_initiate_command_rend(), set_streams_blocked_on_circ(), and spawn_cpuworker().

4.30.3.14 static void connection_start_reading_from_linked_conn (connection_t * *conn*) `[static]`

Helper: Tell the main loop to begin reading bytes into **conn** from its linked connection, if it is not doing so already. Called by connection_start_reading and connection_start_writing as appropriate.

References connection_t::active_on_link, called_loop_once, connection_t::linked, smartlist_add(), smartlist_isin(), tor_assert, and tor_libevent_get_base().

Referenced by conn_close_if_marked(), connection_start_reading(), and connection_start_writing().

4.30.3.15 void connection_start_writing (connection_t * *conn*)

Tell the main loop to start notifying **conn** of any write events.

References connection_should_read_from_linked_conn(), connection_start_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::linked_conn, connection_t::s, tor_assert, connection_t::write_event, and connection_t::writing_to_linked_conn.

Referenced by _connection_write_to_buf_impl(), connection_bucket_refill(), connection_edge_finished_connecting(), connection_read_to_buf(), connection_tls_continue_handshake(), and connection_watch_events().

4.30.3.16 void connection_stop_reading (connection_t * *conn*)

Tell the main loop to stop notifying **conn** of any read events.

References connection_stop_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::read_event, connection_t::reading_from_linked_conn, connection_t::s, and tor_assert.

Referenced by circuit_consider_stop_edge_reading(), connection_consider_empty_read_buckets(), connection_control_process_inbuf(), connection_edge_package_raw_inbuf(), connection_watch_events(), and set_streams_blocked_on_circ().

4.30.3.17 void connection_stop_reading_from_linked_conn (connection_t * *conn*)

Tell the main loop to stop reading bytes into **conn** from its linked connection, if is currently doing so. Called by connection_stop_reading, connection_stop_writing, and connection_read.

References connection_t::active_on_link, connection_t::linked, smartlist_isin(), smartlist_remove(), and tor_assert.

Referenced by connection_handle_read_impl(), connection_stop_reading(), and connection_stop_writing().

4.30.3.18 void connection_stop_writing (connection_t * *conn*)

Tell the main loop to stop notifying **conn** of any write events.

References connection_stop_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::linked_conn, connection_t::s, tor_assert, connection_t::write_event, and connection_t::writing_to_linked_conn.

Referenced by connection_consider_empty_write_buckets(), connection_control_finished_flushing(), connection_cpu_finished_flushing(), connection_dir_finished_flushing(), connection_edge_finished_flushing(), connection_handle_write_impl(), connection_or_finished_flushing(), connection_tls_continue_handshake(), and connection_watch_events().

4.30.3.19 static void connection_unlink (connection_t * *conn*) `[static]`

If **conn** is an edge conn, remove it from the list of conn's on this circuit. If it's not on an edge, flush and send destroys for all circuits on this conn.

Remove it from connection_array (if applicable) and from closeable_connection_list.

Then free it.

References assert_connection_edge_not_dns_pending(), connection_t::conn_array_index, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_about_to_close_connection(), connection_free(), connection_or_remove_from_identity_map(), connection_remove(), connection_start_reading(), connection_t::linked_conn, connection_t::marked_for_close, connection_t::reading_from_linked_conn, smartlist_remove(), TO_EDGE_CONN(), TO_OR_CONN(), tor_digest_is_zero(), and connection_t::type.

Referenced by close_closeable_connections(), and conn_close_if_marked().

4.30.3.20 void connection_unregister_events (connection_t * *conn*)

Tell libevent that we don't care about **conn** any more.

References connection_t::dns_server_port, dnsserv_close_listener(), LD_BUG, connection_t::read_event, connection_t::s, tor_free, and connection_t::write_event.

Referenced by connection_close_immediate(), connection_free(), and connection_remove().

4.30.3.21 void connection_watch_events (connection_t * *conn*, watchable_events_t *events*)

Set the event mask on **conn** to **events**. (The event mask is a bitmask whose bits are READ_EVENT and WRITE_EVENT)

References connection_start_reading(), connection_start_writing(), connection_stop_reading(), and connection_stop_writing().

Referenced by connection_edge_finished_connecting(), connection_exit_connect(), connection_or_connect(), and directory_initiate_command_rend().

4.30.3.22 void control_signal_act (int *the_signal*)

Used to implement the SIGNAL control command: if we accept **the_signal** as a remote pseudo-signal, act on it.

References LD_BUG, and signal_callback().

Referenced by handle_control_signal().

4.30.3.23 void directory_all_unreachable (time_t *now*)

We've just tried every dirserver we know about, and none of them were reachable. Assume the network is down. Change state so next time an application connection arrives we'll delay it and try another directory fetch. Kill off all the circuit_wait streams that are waiting now, since they will all timeout anyway.

References socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, CONN_TYPE_AP, connection_get_by_type_state(), control_event_general_status(), END_STREAM_REASON_NET_UNREACHABLE, LD_NET, socks_request_t::port, safe_str_client(), edge_connection_t::socks_request, stats_n_seconds_working, and TO_EDGE_CONN().

Referenced by directory_get_from_dirserver().

4.30.3.24 void directory_info_has_arrived (time_t *now*, int *from_cache*)

This function is called whenever we successfully pull down some new network statuses or server descriptors.

References any_predicted_circuits(), consider_testing_reachability(), directory_fetches_from_authorities(), directory_too_idle_to_fetch_descriptors(), or_options_t::DownloadExtraInfo, entry_guards_compute_status(), get_dir_info_status_string(), get_options(), has_completed_circuit, LD_DIR, router_have_minimum_dir_info(), server_mode(), update_extrainfo_downloads(), update_router_descriptor_downloads(), and we_are_hibernating().

Referenced by connection_dir_client_reached_eof(), and do_main_loop().

4.30.3.25 void dns_servers_relaunch_checks (void)

Forget what we've learned about the correctness of our DNS servers, and start learning again.

References dns_reset_correctness_checks(), get_options(), server_mode(), and time_to_check_for_correct_dns.

Referenced by configure_nameservers(), and ip_address_changed().

4.30.3.26 void do_hash_password (void)

Entry point for password hashing: take the desired password from the command line, and print its salted hash to stdout.

References base16_encode(), crypto_rand(), DIGEST_LEN, get_options(), S2K_SPECIFIER_LEN, and secret_to_key().

Referenced by tor_main().

4.30.3.27 static int do_hup (void) [static]

Called when we get a SIGHUP: reload configuration files and keys, retry all connections, and so on.

References accounting_is_enabled(), accounting_record_bandwidth_usage(), addressmap_clear_transient(), authdir_mode_handles_descs(), circuit_expire_all_dirty_circs(), cpuworkers_rotate(), dirserv_load_fingerprint_file(), dns_reset(), get_options(), get_or_state(), LD_CONFIG, LD_GENERAL, options_init_from_torrc(), or_options_t::ReloadTorrcOnSIGHUP, router_reset_descriptor_download_failures(), router_reset_status_download_failures(), router_reset_warnings(), routerlist_reset_warnings(), server_mode(), and update_networkstatus_downloads().

Referenced by signal_callback().

4.30.3.28 int do_list_fingerprint (void)

Read/create keys as needed, and echo our fingerprint to stdout.

References crypto_pk_get_fingerprint(), FINGERPRINT_LEN, get_identity_key(), get_options(), init_keys(), LD_BUG, LD_GENERAL, or_options_t::Nickname, server_mode(), and tor_assert.

Referenced by tor_main().

4.30.3.29 int do_main_loop (void)

Tor main loop.

References called_loop_once, connection_bucket_init(), control_event_bootstrap(), cpu_init(), directory_info_has_arrived(), dns_init(), get_options(), global_read_bucket, global_write_bucket, got_libevent_error(), handle_signals(), identity_key_is_set(), init_cell_pool(), init_keys(), LD_BUG, LD_DIR, LD_GENERAL, LD_NET, router_reload_consensus_networkstatus(), router_reload_router_list(), router_reload_v2_networkstatus(), second_elapsed_callback(), server_mode(), stats_prev_global_read_bucket, stats_prev_global_write_bucket, tor_libevent_get_base(), tor_libevent_get_method(), trusted_dirs_reload_certs(), and update_approx_time().

Referenced by tor_main().

4.30.3.30 static void dumpmemusage (int *severity*) [static]

Write current memory usage information to the log.

References buf_dump_freelist_sizes(), connection_dump_buffer_mem_stats(), dump_cell_pool_usage(), dump_dns_mem_usage(), dump_routerlist_mem_usage(), LD_GENERAL, rephist_total_alloc, rephist_total_num, and tor_log_mallinfo().

Referenced by dumpstats().

4.30.3.31 static void dumpstats (int *severity*) `[static]`

Write all statistics to the log, with log level **severity**. Called in response to a SIGUSR1.

References buf_allocation(), buf_datalen(), circuit_dump_by_conn(), conn_state_to_string(), CONN_TYPE_OR, conn_type_to_string(), connection_is_listener(), dump_distinct_digest_count(), dump_pk_ops(), dumpmemusage(), LD_GENERAL, LD_NET, RELAY_PAYLOAD_SIZE, rend_service_dump_stats(), rep_hist_dump_stats(), safe_str_client(), stats_n_bytes_read, stats_n_bytes_written, stats_n_create_cells_processed, stats_n_created_cells_processed, stats_n_data_bytes_packaged, stats_n_data_bytes_received, stats_n_data_cells_packaged, stats_n_data_cells_received, stats_n_destroy_cells_processed, stats_n_padding_cells_processed, stats_n_relay_cells_delivered, stats_n_relay_cells_processed, stats_n_relay_cells_relayed, time_of_process_start, or_connection_t::tls, TO_OR_CONN(), and tor_tls_get_buffer_sizes().

Referenced by signal_callback().

4.30.3.32 static void exit_function (void) `[static]`

Called by exit() as we shut down the process.

Referenced by tor_init().

4.30.3.33 smartlist_t* get_connection_array (void)

Set ∗**array** to an array of all connections, and ∗**n** to the length of the array. ∗**array** and ∗**n** must not be modified.

References smartlist_create().

Referenced by any_pending_bridge_descriptor_fetches(), choose_good_exit_server_general(), circuit_discard_optional_exit_enclaves(), connection_ap_attach_pending(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_bucket_refill(), connection_dump_buffer_mem_stats(), connection_expire_held_open(), connection_free_all(), connection_get_by_global_id(), connection_get_by_type(), connection_get_by_type_addr_port_purpose(), connection_get_by_type_purpose(), connection_get_by_type_state(), connection_get_by_type_state_rendquery(), connection_or_clear_identity_map(), control_event_stream_bandwidth_used(), control_update_global_event_mask(), cull_wedged_cpuworkers(), getinfo_helper_events(), list_pending_downloads(), rend_client_desc_trynow(), retry_listeners(), and send_control_event_string().

4.30.3.34 static int got_libevent_error (void) `[static]`

Called when a possibly ignorable libevent error occurs; ensures that we don't get into an infinite loop by ignoring too many errors from libevent.

References LD_NET, and n_libevent_errors.

Referenced by do_main_loop().

4.30.3.35 void handle_signals (int *is_parent*)

Set up the signal handlers for either parent or child.

References LD_BUG, signal_callback(), tor_evsignal_new(), and tor_libevent_get_base().

Referenced by cpuworker_main(), and do_main_loop().

4.30.3.36 int have_lockfile (void)

Return true iff we've successfully acquired the lock file.

Referenced by init_key_from_file(), and options_act().

4.30.3.37 void ip_address_changed (int *at_interface*)

Called when our IP address seems to have changed. **at_interface** should be true if we detected a change in our interface, and false if we detected a change in our published address.

References dns_servers_relaunch_checks(), get_options(), init_keys(), mark_my_descriptor_dirty(), reset_bandwidth_test(), router_reset_reachability(), server_mode(), and stats_n_seconds_working.

Referenced by check_descriptor_ipaddress_changed(), client_check_address_changed(), options_act(), resolve_my_address(), and router_new_address_suggestion().

4.30.3.38 void release_lockfile (void)

If we have successfully acquired the lock file, release it.

References tor_lockfile_unlock().

Referenced by tor_free_all().

4.30.3.39 static void run_connection_housekeeping (int *i*, time_t *now*) `[static]`

Perform regular maintenance tasks for a single connection. This function gets run once per second per connection by run_scheduled_events.

References connection_t::address, buf_datalen(), cell_t::command, CONN_TYPE_DIR, CONN_TYPE_OR, connection_dir_reached_eof(), connection_or_connect_failed(), connection_or_write_cell_to_buf(), connection_state_is_open(), DIR_CONN_IS_SERVER, DIR_CONN_MAX_STALL, DIR_PURPOSE_FETCH_SERVERDESC, get_options(), connection_t::hold_open_until_flushed, IDLE_OR_CONN_TIMEOUT, connection_t::inbuf, or_connection_t::is_bad_for_new_circs, or_options_t::KeepalivePeriod, LD_DIR, LD_OR, LD_PROTOCOL, connection_t::marked_for_close, or_connection_t::n_circuits, OR_CONN_STATE_CONNECTING, connection_t::outbuf, connection_t::port, connection_t::purpose, connection_t::s, connection_t::state, or_connection_t::timestamp_last_added_nonpadding, or_connection_t::timestamp_lastempty, connection_t::timestamp_lastread, connection_t::timestamp_lastwritten, TO_DIR_CONN(), TO_OR_CONN(), tor_assert, connection_t::type, and we_are_hibernating().

Referenced by run_scheduled_events().

4.30.3.40 static void run_scheduled_events (time_t *now*) `[static]`

Perform regular maintenance tasks. This function gets run once per second by **second_elapsed_callback()** (p. 703).

0. See if we've been asked to shut down and our timeout has expired; or if our bandwidth limits are exhausted and we should hibernate; or if it's time to wake up from hibernation.

1a. Every MIN_ONION_KEY_LIFETIME seconds, rotate the onion keys, shut down and restart all cpuworkers, and update the directory if necessary.

1b. Every MAX_SSL_KEY_LIFETIME seconds, we change our TLS context.

How often do we add more entropy to OpenSSL's RNG pool?

1c. If we have to change the accounting interval or record bandwidth used in this accounting interval, do so.

1d. Periodically, we discount older stability information so that new stability info counts more, and save the stability information to disk as appropriate.

2. Periodically, we consider force-uploading our descriptor (if we've passed our internal checks).

How often do we check whether part of our router info has changed in a way that would require an upload?

How often do we (as a router) check whether our IP address has changed?

If our router descriptor ever goes this long without being regenerated because something changed, we force an immediate regenerate-and-upload.

2c. Let directory voting happen.

3a. Every second, we examine pending circuits and prune the ones which have been pending for more than a few seconds. We do this before step 4, so it can try building more if it's not comfortable with the number of available circuits.

3b. Also look at pending streams and prune the ones that 'began' a long time ago but haven't gotten a 'connected' yet. Do this before step 4, so we can put them back into pending state to be picked up by the new circuit.

3c. And expire connections that we've held open for too long.

3d. And every 60 seconds, we relaunch listeners if any died.

4. Every second, we try a new circuit if there are no valid circuits. Every NewCircuitPeriod seconds, we expire circuits that became dirty more than MaxCircuitDirtiness seconds ago, and we make a new circ if there are no clean circuits.

5. We do housekeeping for each connection...

How often do we check buffers and pools for empty space that can be deallocated?

6. And remove any marked circuits...

7. And upload service descriptors if necessary.

8. and blow away any connections that need to die. have to do this now, because if we marked a conn for close and left its socket -1, then we'll pass it to poll/select and bad things will happen.

8b. And if anything in our state is ready to get flushed to disk, we flush it.

9. and if we're a server, check whether our DNS is telling stories to us.

10b. write bridge networkstatus file to disk

References accounting_is_enabled(), accounting_run_housekeeping(), advertised_server_mode(), any_predicted_circuits(), authdir_mode_tests_reachability(), authdir_mode_v3(), routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, or_options_t::BridgeAuthoritativeDir, buf_shrink(), buf_shrink_freelists(), or_options_t::CellStatistics, check_descriptor_bandwidth_changed(), check_descriptor_ipaddress_changed(), circuit_build_needed_circs(), circuit_close_all_marked(), circuit_expire_building(), circuit_expire_old_circuits_serverside(), clean_cell_pool(), close_closeable_connections(), connection_ap_expire_beginning(), connection_expire_held_open(), connection_or_set_bad_connections(), consider_hibernation(), consider_publishable_server(), consider_testing_reachability(), cpuworkers_rotate(), crypto_rand_int(), crypto_seed_rng(), DESCRIPTOR_FAILURE_RESET_INTERVAL, or_options_t::DirReqStatistics, dirserv_test_reachability(), dirvote_act(), dns_init(), dns_launch_correctness_checks(), or_options_t::EntryStatistics, or_options_t::ExitPortStatistics, fetch_bridge_descriptors(), geoip_bridge_stats_init(), geoip_bridge_stats_write(), geoip_dirreq_stats_init(), geoip_dirreq_stats_write(), geoip_entry_stats_init(), geoip_entry_stats_write(), get_identity_key(), get_onion_key_set_at(), get_options(), GREEDY_DESCRIPTOR_RETRY_INTERVAL, has_completed_circuit, has_dns_init_failed(), LAZY_DESCRIPTOR_RETRY_INTERVAL, LD_BUG, LD_CONFIG, LD_CONTROL, LD_GENERAL, mark_my_descriptor_dirty_if_older_than(), MAX_SIGNEWNYM_RATE, MAX_SSL_KEY_LIFETIME, MIN_ONION_KEY_LIFETIME, networkstatus_dump_bridge_status_to_file(), networkstatus_get_latest_consensus(), networkstatus_v2_list_clean(), or_state_save(), rend_cache_clean(), rend_cache_clean_v2_descs_as_dir(), rend_consider_descriptor_republication(), rend_consider_services_upload(), rep_hist_buffer_stats_init(), rep_hist_buffer_stats_write(), rep_hist_downrate_old_runs(), rep_hist_exit_stats_init(), rep_hist_exit_stats_write(), rep_hist_record_mtbf_data(), rep_history_clean(), or_options_t::RephistTrackTime, reset_bandwidth_test(), retry_all_listeners(), rotate_onion_key(), router_dir_info_changed(), router_get_my_routerinfo(), router_have_minimum_dir_info(), router_rebuild_descriptor(), router_reset_descriptor_download_failures(), router_upload_dir_desc_to_dirservers(), routerlist_remove_old_routers(), run_connection_housekeeping(), server_mode(), should_record_bridge_info(), signewnym_impl(), signewnym_is_pending, stats_n_seconds_working, time_of_last_signewnym, time_to_check_for_correct_dns, TIMEOUT_UNTIL_UNREACHABILITY_COMPLAINT, tor_tls_context_new(), update_extrainfo_downloads(), update_networkstatus_downloads(), update_router_descriptor_downloads(), or_options_t::UseBridges, v3_authority_check_key_expiry(), networkstatus_t::valid_until, and we_are_hibernating().

Referenced by second_elapsed_callback().

4.30.3.41 static void second_elapsed_callback (int *fd*, short *event*, void * *args*) `[static]`

Libevent callback: invoked once every second.

If more than this many seconds have elapsed, probably the clock jumped: doesn't count.

References accounting_add_bytes(), accounting_is_enabled(), routerinfo_t::address, check_whether_dirport_reachable(), check_whether_orport_reachable(), circuit_note_clock_jumped(), connection_bucket_refill(), control_event_bandwidth_used(), control_event_server_status(), control_event_stream_bandwidth_used(), routerinfo_t::dir_port, get_options(), global_read_bucket, global_write_bucket, has_completed_circuit, LD_CONFIG, LD_NET, n_libevent_errors, routerinfo_t::or_port, router_get_my_routerinfo(), run_scheduled_events(), server_mode(), stats_n_bytes_read, stats_n_bytes_written, stats_n_seconds_working, stats_prev_global_read_bucket, stats_prev_global_write_bucket, timeout_event, TIMEOUT_UNTIL_UNREACHABILITY_COMPLAINT, tor_evtimer_new(), tor_libevent_get_base(), update_approx_time(), and we_are_hibernating().

Referenced by do_main_loop().

4.30.3.42 static void signal_callback (int *fd*, short *events*, void * *arg*) `[static]`

Libevent callback: invoked when we get a signal.

References addressmap_clear_transient(), do_hup(), dumpstats(), get_min_log_level(), get_-

options(), hibernate_begin_shutdown(), LD_CONFIG, LD_CONTROL, LD_GENERAL, MAX_SIGNEWNYM_RATE, server_mode(), signewnym_impl(), signewnym_is_pending, switch_logs_debug(), time_of_last_signewnym, and tor_cleanup().

Referenced by control_signal_act(), and handle_signals().

4.30.3.43 static void signewnym_impl (time_t *now*) [static]

Honor a NEWNYM request: make future requests unlinkable to past requests.

References addressmap_clear_transient(), circuit_expire_all_dirty_circs(), signewnym_is_pending, and time_of_last_signewnym.

Referenced by run_scheduled_events(), and signal_callback().

4.30.3.44 void tor_cleanup (void)

Do whatever cleanup is necessary before shutting Tor down.

References accounting_is_enabled(), accounting_record_bandwidth_usage(), authdir_mode_tests_reachability(), or_options_t::command, crypto_global_cleanup(), get_options(), get_or_state(), or_state_mark_dirty(), or_state_save(), or_options_t::PidFile, rep_hist_record_mtbf_data(), and tor_free_all().

Referenced by consider_hibernation(), hibernate_begin(), signal_callback(), and tor_main().

4.30.3.45 void tor_free_all (int *postfork*)

Free all memory that we might have allocated somewhere. If **postfork**, we are a worker process and we want to free only the parts of memory that we won't touch. If !**postfork**, Tor is shutting down and we should free everything.

Helps us find the real leaks with dmalloc and the like. Also valgrind should then report 0 reachable in its leak report (in an ideal world – in practice libevent, SSL, libc etc never quite free everything).

References addressmap_free_all(), buf_shrink_freelists(), circuit_free_all(), clear_pending_onions(), config_free_all(), connection_free_all(), dirserv_free_all(), dirvote_free_all(), dns_free_all(), entry_guards_free_all(), esc_router_info(), escaped(), free_cell_pool(), geoip_free_all(), logs_free_all(), memarea_clear_freelist(), networkstatus_free_all(), policies_free_all(), release_lockfile(), rend_cache_free_all(), rend_service_authorization_free_all(), rend_service_free_all(), rep_hist_free_all(), router_free_all(), routerlist_free_all(), smartlist_free(), timeout_event, tor_free, and tor_tls_free_all().

Referenced by cpuworker_main(), and tor_cleanup().

4.30.3.46 int tor_init (int *argc*, char ∗ *argv*[ ])

Main entry point for the Tor command-line client.

References add_temp_log(), addressmap_init(), crypto_global_init(), exit_function(), get_options(), get_uname(), get_version(), LD_BUG, LD_CONFIG, LD_GENERAL, log_set_application_name(), network_init(), options_init_from_torrc(), rend_cache_init(), rep_hist_init(), smartlist_create(), time_of_process_start, and tor_snprintf().

Referenced by tor_main().

4.30.3.47 int tor_main (int *argc*, char ∗ *argv*[ ])

Main entry point for the Tor process. Called from **main()** (p. 1210).

References _tor_free(), _tor_malloc(), _tor_realloc(), do_hash_password(), do_list_fingerprint(), do_main_loop(), get_options(), init_logging(), LD_BUG, tor_assert, tor_cleanup(), tor_init(), tor_threads_init(), and update_approx_time().

Referenced by main().

4.30.3.48 int try_locking (or_options_t ∗ *options*, int *err_if_locked*)

Try to grab the lock file described in **options**, if we do not already have it. If **err_if_locked** is true, warn if somebody else is holding the lock, and exit if we can't get it after waiting. Otherwise, return -1 if we can't get the lockfile. Return 0 on success.

References LD_GENERAL, options_get_datadir_fname2_suffix(), tor_free, tor_lockfile_lock(), and try_locking().

Referenced by init_key_from_file(), options_act(), and try_locking().

4.30.4 Variable Documentation

4.30.4.1 smartlist_t∗ active_linked_connection_lst = NULL `[static]`

List of linked connections that are currently reading data into their inbuf from their partner's outbuf.

4.30.4.2 int called_loop_once = 0 `[static]`

Flag: Set to true iff we entered the current libevent main loop via **loop_once**. If so, there's no need to trigger a loopexit in order to handle linked connections.

Referenced by connection_start_reading_from_linked_conn(), and do_main_loop().

4.30.4.3 smartlist_t∗ closeable_connection_lst = NULL `[static]`

List of connections that have been marked for close and need to be freed and removed from connection_array.

4.30.4.4 smartlist_t∗ connection_array = NULL `[static]`

Smartlist of all open connections.

4.30.4.5 int global_read_bucket

Max number of bytes I can read this second.

Referenced by do_main_loop(), and second_elapsed_callback().

4.30.4.6 int global_relayed_read_bucket

Max number of relayed (bandwidth class 1) bytes I can read this second.

4.30.4.7 int global_relayed_write_bucket

Max number of relayed (bandwidth class 1) bytes I can write this second.

4.30.4.8 int global_write_bucket

Max number of bytes I can write this second.

Referenced by do_main_loop(), and second_elapsed_callback().

4.30.4.9 int has_completed_circuit = 0

We set this to 1 when we've opened a circuit, so we can print a log entry to inform the user that Tor is working.

Referenced by circuit_note_clock_jumped(), circuit_send_next_onion_skin(), directory_info_has_arrived(), getinfo_helper_events(), options_act(), purpose_needs_anonymity(), run_scheduled_events(), and second_elapsed_callback().

4.30.4.10 tor_lockfile_t* lockfile = NULL `[static]`

A lockfile structure, used to prevent two Tors from messing with the data directory at once. If this variable is non-NULL, we're holding the lockfile.

4.30.4.11 int n_libevent_errors = 0 `[static]`

Number of libevent errors in the last second: we die if we get too many.

Referenced by got_libevent_error(), and second_elapsed_callback().

4.30.4.12 uint64_t rephist_total_alloc

Total number of bytes currently allocated in fields used by **rephist.c** (p. 1095).

Referenced by _free_link_history(), add_predicted_port(), bw_array_new(), dumpmemusage(), free_or_history(), get_link_history(), get_or_history(), predicted_ports_free(), rep_hist_get_predicted_ports(), and rep_history_clean().

4.30.4.13 uint32_t rephist_total_num

Number of **or_history_t** (p. 145) objects currently allocated.

Referenced by dumpmemusage(), free_or_history(), and get_or_history().

4.30.4.14 int signewnym_is_pending = 0 `[static]`

Is there a signewnym request we're currently waiting to handle?

Referenced by run_scheduled_events(), signal_callback(), and signewnym_impl().

4.30.4.15 uint64_t stats_n_bytes_read = 0 [static]

How many bytes have we read since we started the process?

Referenced by dumpstats(), and second_elapsed_callback().

4.30.4.16 uint64_t stats_n_bytes_written = 0 [static]

How many bytes have we written since we started the process?

Referenced by dumpstats(), and second_elapsed_callback().

4.30.4.17 long stats_n_seconds_working = 0

How many seconds have we been running?

Referenced by directory_all_unreachable(), hibernate_end(), ip_address_changed(), router_dump_router_to_string(), run_scheduled_events(), and second_elapsed_callback().

4.30.4.18 int stats_prev_global_read_bucket [static]

What was the read bucket before the last **second_elapsed_callback()** (p. 703) call? (used to determine how many bytes we've read).

Referenced by do_main_loop(), and second_elapsed_callback().

4.30.4.19 int stats_prev_global_write_bucket [static]

What was the write bucket before the last **second_elapsed_callback()** (p. 703) call? (used to determine how many bytes we've written).

Referenced by do_main_loop(), and second_elapsed_callback().

4.30.4.20 time_t time_of_last_signewnym = 0 [static]

When did we last process a SIGNEWNYM request?

Referenced by run_scheduled_events(), signal_callback(), and signewnym_impl().

4.30.4.21 time_t time_of_process_start = 0

What time did this process start up?

Referenced by dumpstats(), and tor_init().

4.30.4.22 time_t time_to_check_for_correct_dns = 0 [static]

When do we next launch DNS wildcarding checks?

Referenced by dns_servers_relaunch_checks(), and run_scheduled_events().

4.30.4.23 struct event∗ timeout_event = NULL `[static]`

Libevent timer: used to invoke **second_elapsed_callback()** (p. 703) once per second.

Referenced by second_elapsed_callback(), and tor_free_all().

4.31 memarea.c File Reference

Implementation for **memarea_t** (p. 112), an allocator for allocating lots of small objects that will be freed all at once.

`#include "orconfig.h"`

`#include <stdlib.h>`

`#include "memarea.h"`

`#include "util.h"`

`#include "compat.h"`

`#include "log.h"`

Data Structures

- struct **memarea_chunk_t**
- struct **memarea_t**

Defines

- #define **USE_SENTINELS**
- #define **MEMAREA_ALIGN** SIZEOF_VOID_P
- #define **MEMAREA_ALIGN_MASK** 7lu
- #define **SENTINEL_VAL** 0x90806622u
- #define **SENTINEL_LEN** sizeof(uint32_t)
- #define **SET_SENTINEL**(chunk)
- #define **CHECK_SENTINEL**(chunk)
- #define **CHUNK_HEADER_SIZE** STRUCT_OFFSET(**memarea_chunk_t**, u)
- #define **CHUNK_SIZE** 4096
- #define **MAX_FREELIST_LEN** 4

Functions

- static INLINE void ∗ **realign_pointer** (void ∗ptr)
- static **memarea_chunk_t** ∗ **alloc_chunk** (size_t sz, int freelist_ok)
- static void **chunk_free_unchecked** (**memarea_chunk_t** ∗chunk)
- **memarea_t** ∗ **memarea_new** (void)
- void **memarea_drop_all** (**memarea_t** ∗area)
- void **memarea_clear** (**memarea_t** ∗area)
- void **memarea_clear_freelist** (void)
- int **memarea_owns_ptr** (const **memarea_t** ∗area, const void ∗p)
- void ∗ **memarea_alloc** (**memarea_t** ∗area, size_t sz)
- void ∗ **memarea_alloc_zero** (**memarea_t** ∗area, size_t sz)
- void ∗ **memarea_memdup** (**memarea_t** ∗area, const void ∗s, size_t n)
- char ∗ **memarea_strdup** (**memarea_t** ∗area, const char ∗s)
- char ∗ **memarea_strndup** (**memarea_t** ∗area, const char ∗s, size_t n)
- void **memarea_get_stats** (**memarea_t** ∗area, size_t ∗allocated_out, size_t ∗used_out)
- void **memarea_assert_ok** (**memarea_t** ∗area)

Variables

- static int **freelist_len** = 0
- static **memarea_chunk_t** ∗ **freelist** = NULL

4.31.1 Detailed Description

Implementation for **memarea_t** (p. 112), an allocator for allocating lots of small objects that will be freed all at once.

4.31.2 Define Documentation

4.31.2.1 #define CHECK_SENTINEL(chunk)

Value:

```
STMT_BEGIN                                                              \
  uint32_t sent_val = get_uint32(&(chunk)->u.mem[chunk->mem_size]);     \
  tor_assert(sent_val == SENTINEL_VAL);                                 \
  STMT_END
```

4.31.2.2 #define MAX_FREELIST_LEN 4

How many chunks will we put into the freelist before freeing them?

Referenced by chunk_free_unchecked().

4.31.2.3 #define MEMAREA_ALIGN SIZEOF_VOID_P

All returned pointers should be aligned to the nearest multiple of this value.

4.31.2.4 #define SET_SENTINEL(chunk)

Value:

```
STMT_BEGIN                                                  \
  set_uint32( &(chunk)->u.mem[chunk->mem_size], SENTINEL_VAL ); \
  STMT_END
```

4.31.2.5 #define USE_SENTINELS

If true, we try to detect any attempts to write beyond the length of a memarea.

4.31.3 Function Documentation

4.31.3.1 static memarea_chunk_t∗ alloc_chunk (size_t *sz*, int *freelist_ok*) [static]

Helper: allocate a new memarea chunk of around **chunk_size** bytes.

References freelist_len, memarea_chunk_t::mem, memarea_chunk_t::mem_size, memarea_chunk_t::next_chunk, memarea_chunk_t::next_mem, realign_pointer(), tor_assert, and memarea_chunk_t::u.

Referenced by memarea_alloc(), and memarea_new().

4.31.3.2 static void chunk_free_unchecked (memarea_chunk_t * *chunk*) `[static]`

Release **chunk** from a memarea, either by adding it to the freelist or by freeing it if the freelist is already too big.

References freelist_len, MAX_FREELIST_LEN, memarea_chunk_t::mem, memarea_chunk_t::next_chunk, memarea_chunk_t::next_mem, tor_free, and memarea_chunk_t::u.

Referenced by buf_clear(), buf_pullup(), buf_remove_from_front(), memarea_clear(), and memarea_drop_all().

4.31.3.3 void* memarea_alloc (memarea_t * *area*, size_t *sz*)

Return a pointer to a chunk of memory in **area** of at least **sz** bytes. **sz** should be significantly smaller than the area's chunk size, though we can deal if it isn't.

References alloc_chunk(), memarea_t::first, memarea_chunk_t::mem, memarea_chunk_t::mem_size, memarea_chunk_t::next_chunk, memarea_chunk_t::next_mem, realign_pointer(), tor_assert, and memarea_chunk_t::u.

Referenced by memarea_alloc_zero(), memarea_memdup(), and memarea_strndup().

4.31.3.4 void* memarea_alloc_zero (memarea_t * *area*, size_t *sz*)

As **memarea_alloc()** (p. 711), but clears the memory it returns.

References memarea_alloc().

4.31.3.5 void memarea_assert_ok (memarea_t * *area*)

Assert that **area** is okay.

References memarea_t::first, memarea_chunk_t::mem, memarea_chunk_t::mem_size, memarea_chunk_t::next_chunk, memarea_chunk_t::next_mem, realign_pointer(), tor_assert, and memarea_chunk_t::u.

4.31.3.6 void memarea_clear (memarea_t * *area*)

Forget about having allocated anything in **area**, and free some of the backing storage associated with it, as appropriate. Invalidates all pointers returned from **memarea_alloc()** (p. 711) for this area.

References chunk_free_unchecked(), memarea_t::first, memarea_chunk_t::mem, memarea_chunk_t::next_chunk, memarea_chunk_t::next_mem, and memarea_chunk_t::u.

Referenced by microdescs_parse_from_string(), networkstatus_v2_parse_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), router_parse_directory(), and routerstatus_parse_entry_from_string().

4.31.3.7 void memarea_clear_freelist (void)

Remove all unused memarea chunks from the internal freelist.

References freelist_len, memarea_chunk_t::next_chunk, and tor_free.

Referenced by tor_free_all().

4.31.3.8 void memarea_drop_all (memarea_t * *area*)

Free **area**, invalidating all pointers returned from **memarea_alloc()** (p. 711) and friends for this area

References chunk_free_unchecked(), memarea_t::first, memarea_chunk_t::next_chunk, and tor_free.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), find_dir_signing_key(), microdescs_parse_from_string(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), router_parse_addr_policy_item_from_string(), router_parse_directory(), router_parse_entry_from_string(), and router_parse_runningrouters().

4.31.3.9 void memarea_get_stats (memarea_t * *area*, size_t * *allocated_out*, size_t * *used_out*)

Set **allocated_out** to the number of bytes allocated in **area**, and **used_out** to the number of bytes currently used.

References memarea_t::first, memarea_chunk_t::mem, memarea_chunk_t::mem_size, memarea_chunk_t::next_chunk, memarea_chunk_t::next_mem, tor_assert, and memarea_chunk_t::u.

4.31.3.10 void* memarea_memdup (memarea_t * *area*, const void * *s*, size_t *n*)

As memdup, but returns the memory from **area**.

References memarea_alloc().

Referenced by get_token_arguments(), and memarea_strdup().

4.31.3.11 memarea_t* memarea_new (void)

Allocate and return new memarea.

References alloc_chunk(), and memarea_t::first.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), find_dir_signing_key(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), router_parse_addr_policy_item_from_string(), router_parse_directory(), router_parse_entry_from_string(), and router_parse_runningrouters().

4.31.3.12 int memarea_owns_ptr (const memarea_t * *area*, const void * *p*)

Return true iff **p** is in a range that has been returned by an allocation from **area**.

References memarea_t::first, memarea_chunk_t::mem, memarea_chunk_t::next_chunk, and memarea_chunk_t::u.

4.31.3.13 char* memarea_strdup (memarea_t * *area*, const char * *s*)

As strdup, but returns the memory from **area**.

References memarea_memdup().

4.31.3.14 char* memarea_strndup (memarea_t * *area*, const char * *s*, size_t *n*)

As strndup, but returns the memory from **area**.

References memarea_alloc().

Referenced by get_token_arguments().

4.31.3.15 static INLINE void* realign_pointer (void * *ptr*) `[static]`

Increment **ptr** until it is aligned to MEMAREA_ALIGN.

References tor_assert.

Referenced by alloc_chunk(), memarea_alloc(), and memarea_assert_ok().

4.31.4 Variable Documentation

4.31.4.1 memarea_chunk_t* freelist = NULL `[static]`

A linked list of unused memory area chunks. Used to prevent us from spinning in malloc/free loops.

Referenced by chunk_free_unchecked(), and chunk_new_with_alloc_size().

4.31.4.2 int freelist_len = 0 `[static]`

The number of memarea chunks currently in our freelist.

Referenced by alloc_chunk(), chunk_free_unchecked(), and memarea_clear_freelist().

4.32 mempool.h File Reference

Headers for mempool.c.

Typedefs

- typedef struct **mp_pool_t** **mp_pool_t**

Functions

- void ∗ **mp_pool_get** (**mp_pool_t** ∗pool)
- void **mp_pool_release** (void ∗item)
- **mp_pool_t** ∗ **mp_pool_new** (size_t item_size, size_t chunk_capacity)
- void **mp_pool_clean** (**mp_pool_t** ∗pool, int n_to_keep, int keep_recently_used)
- void **mp_pool_destroy** (**mp_pool_t** ∗pool)
- void **mp_pool_assert_ok** (**mp_pool_t** ∗pool)
- void **mp_pool_log_status** (**mp_pool_t** ∗pool, int severity)

4.32.1 Detailed Description

Headers for mempool.c.

4.32.2 Typedef Documentation

4.32.2.1 typedef struct mp_pool_t mp_pool_t

A memory pool is a context in which a large number of fixed-sized objects can be allocated efficiently. See mempool.c for implementation details.

4.32.3 Function Documentation

4.32.3.1 void mp_pool_assert_ok (mp_pool_t ∗ *pool*)

Fail with an assertion if **pool** is not internally consistent.

4.32.3.2 void mp_pool_clean (mp_pool_t ∗ *pool*, int *n_to_keep*, int *keep_recently_used*)

If there are more than **n** empty chunks in **pool**, free the excess ones that have been empty for the longest. If **keep_recently_used** is true, do not free chunks unless they have been empty since the last call to this function.

References mp_chunk_t::magic, and mp_chunk_t::next.

4.32.3.3 void mp_pool_destroy (mp_pool_t ∗ *pool*)

Free all space held in **pool** This makes all pointers returned from mp_pool_get(**pool**) invalid.

4.32.3.4 void* mp_pool_get (mp_pool_t * *pool*)

Return a newly allocated item from **pool**.

References mp_chunk_t::capacity, mp_chunk_t::first_free, mp_allocated_t::in_chunk, mp_chunk_t::mem, mp_chunk_t::mem_size, mp_chunk_t::n_allocated, mp_chunk_t::next, mp_allocated_t::next_free, mp_chunk_t::next_mem, mp_chunk_t::prev, and mp_allocated_t::u.

4.32.3.5 void mp_pool_log_status (mp_pool_t * *pool*, int *severity*)

Dump information about **pool**'s memory usage to the Tor log at level **severity**.

References LD_MM, mp_chunk_t::mem_size, mp_chunk_t::n_allocated, and mp_chunk_t::next.

4.32.3.6 mp_pool_t* mp_pool_new (size_t *item_size*, size_t *chunk_capacity*)

Allocate a new memory pool to hold items of size **item_size**. We'll try to fit about **chunk_capacity** bytes in each chunk.

References LD_MM, round_to_power_of_2(), STRUCT_OFFSET, and tor_assert.

4.32.3.7 void mp_pool_release (void * *item*)

Return an allocated memory item to its memory pool.

References mp_chunk_t::capacity, mp_chunk_t::first_free, mp_allocated_t::in_chunk, mp_chunk_t::magic, mp_chunk_t::mem, mp_chunk_t::n_allocated, mp_chunk_t::next, mp_allocated_t::next_free, mp_chunk_t::next_mem, mp_chunk_t::pool, mp_chunk_t::prev, and mp_allocated_t::u.

4.33 networkstatus.c File Reference

Functions and structures for handling network status documents as a client or cache.

`#include "or.h"`

Data Structures

- struct **consensus_waiting_for_certs_t**

Defines

- #define **NETWORKSTATUS_ALLOW_SKEW** (24*60*60)
- #define **V2_NETWORKSTATUS_ROUTER_LIFETIME** (3*60*60)
- #define **AUTHORITY_NS_CACHE_INTERVAL** (10*60)
- #define **NONAUTHORITY_NS_CACHE_INTERVAL** (60*60)
- #define **CONSENSUS_NETWORKSTATUS_MAX_DL_TRIES** 8
- #define **DELAY_WHILE_FETCHING_CERTS** (20*60)
- #define **CONSENSUS_MIN_SECONDS_BEFORE_CACHING** 120
- #define **REASONABLY_LIVE_TIME** (24*60*60)

Functions

- static void **download_status_map_update_from_v2_networkstatus** (void)
- static void **routerstatus_list_update_named_server_map** (void)
- void **networkstatus_reset_warnings** (void)
- void **networkstatus_reset_download_failures** (void)
- int **router_reload_v2_networkstatus** (void)
- int **router_reload_consensus_networkstatus** (void)
- static void **vote_routerstatus_free** (**vote_routerstatus_t** *rs)
- void **routerstatus_free** (**routerstatus_t** *rs)
- void **networkstatus_v2_free** (**networkstatus_v2_t** *ns)
- void **document_signature_free** (**document_signature_t** *sig)
- **document_signature_t** * **document_signature_dup** (const **document_signature_t** *sig)
- void **networkstatus_vote_free** (**networkstatus_t** *ns)
- **networkstatus_voter_info_t** * **networkstatus_get_voter_by_id** (**networkstatus_t** *vote, const char *identity)
- int **networkstatus_check_document_signature** (const **networkstatus_t** *consensus, **document_signature_t** *sig, const **authority_cert_t** *cert)
- int **networkstatus_check_consensus_signature** (**networkstatus_t** *consensus, int warn)
- char * **networkstatus_get_cache_filename** (const char *identity_digest)
- static int **_compare_networkstatus_v2_published_on** (const void **_a, const void **_b)
- static int **add_networkstatus_to_cache** (const char *s, **v2_networkstatus_source_t** source, **networkstatus_v2_t** *ns)
- int **router_set_networkstatus_v2** (const char *s, time_t arrived_at, **v2_networkstatus_source_t** source, **smartlist_t** *requested_fingerprints)
- void **networkstatus_v2_list_clean** (time_t now)

- int **compare_digest_to_routerstatus_entry** (const void *_key, const void **_member)
- **routerstatus_t** * **networkstatus_v2_find_entry** (**networkstatus_v2_t** *ns, const char *digest)
- **routerstatus_t** * **networkstatus_vote_find_entry** (**networkstatus_t** *ns, const char *digest)
- int **networkstatus_vote_find_entry_idx** (**networkstatus_t** *ns, const char *digest, int *found_out)
- const **smartlist_t** * **networkstatus_get_v2_list** (void)
- **routerstatus_t** * **router_get_consensus_status_by_descriptor_digest** (const char *digest)
- **download_status_t** * **router_get_dl_status_by_descriptor_digest** (const char *d)
- **routerstatus_t** * **router_get_consensus_status_by_id** (const char *digest)
- **routerstatus_t** * **router_get_consensus_status_by_nickname** (const char *nickname, int warn_if_unnamed)
- const char * **networkstatus_get_router_digest_by_nickname** (const char *nickname)
- int **networkstatus_nickname_is_unnamed** (const char *nickname)
- static void **update_v2_networkstatus_cache_downloads** (time_t now)
- static void **update_consensus_networkstatus_downloads** (time_t now)
- void **networkstatus_consensus_download_failed** (int status_code)
- void **update_consensus_networkstatus_fetch_time** (time_t now)
- int **should_delay_dir_fetches** (**or_options_t** *options)
- void **update_networkstatus_downloads** (time_t now)
- void **update_certificate_downloads** (time_t now)
- int **consensus_is_waiting_for_certs** (void)
- **networkstatus_v2_t** * **networkstatus_v2_get_by_digest** (const char *digest)
- **networkstatus_t** * **networkstatus_get_latest_consensus** (void)
- **networkstatus_t** * **networkstatus_get_live_consensus** (time_t now)
- **networkstatus_t** * **networkstatus_get_reasonably_live_consensus** (time_t now)
- static int **routerstatus_has_changed** (const **routerstatus_t** *a, const **routerstatus_t** *b)
- static void **notify_control_networkstatus_changed** (const **networkstatus_t** *old_c, const **networkstatus_t** *new_c)
- static void **networkstatus_copy_old_consensus_info** (**networkstatus_t** *new_c, const **networkstatus_t** *old_c)
- int **networkstatus_set_current_consensus** (const char *consensus, const char *flavor, unsigned flags)
- void **networkstatus_note_certs_arrived** (void)
- void **routers_update_all_from_networkstatus** (time_t now, int dir_version)
- void **routers_update_status_from_consensus_networkstatus** (**smartlist_t** *routers, int reset_failures)
- void **signed_descs_update_status_from_consensus_networkstatus** (**smartlist_t** *descs)
- char * **networkstatus_getinfo_helper_single** (**routerstatus_t** *rs)
- char * **networkstatus_getinfo_by_purpose** (const char *purpose_string, time_t now)
- void **networkstatus_dump_bridge_status_to_file** (time_t now)
- int32_t **get_net_param_from_list** (**smartlist_t** *net_params, const char *param_name, int default_val)

- int32_t **networkstatus_get_param** (**networkstatus_t** *ns, const char *param_name, int32_t default_val)
- int32_t **networkstatus_get_bw_weight** (**networkstatus_t** *ns, const char *weight_name, int32_t default_val)
- const char * **networkstatus_get_flavor_name** (**consensus_flavor_t** flav)
- int **networkstatus_parse_flavor_name** (const char *flavname)
- int **getinfo_helper_networkstatus** (**control_connection_t** *conn, const char *question, char **answer)
- void **networkstatus_free_all** (void)

Variables

- static digestmap_t * **v2_download_status_map** = NULL
- static **smartlist_t** * **networkstatus_v2_list** = NULL
- static int **networkstatus_v2_list_has_changed** = 0
- static strmap_t * **named_server_map** = NULL
- static strmap_t * **unnamed_server_map** = NULL
- static **networkstatus_t** * **current_consensus** = NULL
- static **consensus_waiting_for_certs_t** **consensus_waiting_for_certs** [N_CONSENSUS_FLAVORS]
- static time_t **last_networkstatus_download_attempted** = 0
- static time_t **time_to_download_next_consensus** = 0
- static **download_status_t** **consensus_dl_status** [N_CONSENSUS_FLAVORS]
- static int **have_warned_about_old_version** = 0
- static int **have_warned_about_new_version** = 0

4.33.1 Detailed Description

Functions and structures for handling network status documents as a client or cache.

4.33.2 Define Documentation

4.33.2.1 #define AUTHORITY_NS_CACHE_INTERVAL (10*60)

How frequently do directory authorities re-download fresh networkstatus documents?

Referenced by update_v2_networkstatus_cache_downloads().

4.33.2.2 #define CONSENSUS_MIN_SECONDS_BEFORE_CACHING 120

How long do we (as a cache) wait after a consensus becomes non-fresh before trying to fetch another?

Referenced by update_consensus_networkstatus_fetch_time().

4.33.2.3 #define CONSENSUS_NETWORKSTATUS_MAX_DL_TRIES 8

How many times will we try to fetch a consensus before we give up?

Referenced by update_consensus_networkstatus_downloads().

4.33.2.4 #define DELAY_WHILE_FETCHING_CERTS (20*60)

How long will we hang onto a possibly live consensus for which we're fetching certs before we check whether there is a better one?

Referenced by update_consensus_networkstatus_downloads().

4.33.2.5 #define NETWORKSTATUS_ALLOW_SKEW (24*60*60)

How far in the future do we allow a network-status to get before removing it? (seconds)

Referenced by router_set_networkstatus_v2().

4.33.2.6 #define NONAUTHORITY_NS_CACHE_INTERVAL (60*60)

How frequently do non-authority directory caches re-download fresh networkstatus documents?

Referenced by update_v2_networkstatus_cache_downloads().

4.33.3 Function Documentation

4.33.3.1 static int _compare_networkstatus_v2_published_on (const void ** _*a*, const void ** _*b*) `[static]`

Helper for smartlist_sort: Compare two networkstatus objects by publication date.

References networkstatus_v2_t::published_on.

4.33.3.2 static int add_networkstatus_to_cache (const char * *s*, v2_networkstatus_source_t *source*, networkstatus_v2_t * *ns*) `[static]`

Add the parsed v2 networkstatus in **ns** (with original document in **s**) to the disk cache (and the in-memory directory server cache) as appropriate.

References directory_caches_v2_dir_info(), dirserv_set_cached_networkstatus_v2(), get_options(), networkstatus_v2_t::identity_digest, LD_FS, networkstatus_get_cache_filename(), networkstatus_v2_t::published_on, tor_free, and write_str_to_file().

Referenced by router_set_networkstatus_v2().

4.33.3.3 int compare_digest_to_routerstatus_entry (const void * _*key*, const void ** _*member*)

Helper for bsearching a list of **routerstatus_t** (p. 218) pointers: compare a digest in the key to the identity digest of a **routerstatus_t** (p. 218).

References DIGEST_LEN, and routerstatus_t::identity_digest.

Referenced by measured_bw_line_apply(), networkstatus_v2_find_entry(), networkstatus_vote_find_entry(), networkstatus_vote_find_entry_idx(), and router_get_consensus_status_by_id().

4.33.3.4 int consensus_is_waiting_for_certs (void)

Return 1 if we have a consensus but we don't have enough certificates to start using it yet.

References consensus_waiting_for_certs_t::consensus, and USABLE_CONSENSUS_FLAVOR.

Referenced by connection_edge_process_relay_cell_not_open().

4.33.3.5 document_signature_t* document_signature_dup (const document_signature_t * *sig*)

Return a newly allocated copy of **sig**

References document_signature_t::signature, and document_signature_t::signature_len.

Referenced by networkstatus_add_detached_signatures().

4.33.3.6 void document_signature_free (document_signature_t * *sig*)

Free all storage held in **sig**

References document_signature_t::signature, and tor_free.

Referenced by networkstatus_add_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), and ns_detached_signatures_free().

4.33.3.7 static void download_status_map_update_from_v2_networkstatus (void) `[static]`

Update v2_download_status_map to contain an entry for every router descriptor listed in the v2 networkstatuses.

References _tor_free(), digestmap_free(), digestmap_get(), digestmap_new(), digestmap_remove(), digestmap_set(), networkstatus_v2_list_has_changed, and v2_download_status_map.

Referenced by routers_update_all_from_networkstatus().

4.33.3.8 int getinfo_helper_networkstatus (control_connection_t * *conn*, const char * *question*, char ** *answer*)

If **question** is a string beginning with "ns/" in a format the control interface expects for a GETINFO question, set *__answer__* to a newly-allocated string containing networkstatus lines for the appropriate ORs. Return 0 on success, -1 on unrecognized question format.

References networkstatus_getinfo_helper_single(), networkstatus_t::routerstatus_list, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), and tor_free.

4.33.3.9 int networkstatus_check_consensus_signature (networkstatus_t * *consensus*, int *warn*)

Given a v3 networkstatus consensus in **consensus**, check every as-yet-unchecked signature on **consensus**. Return 1 if there is a signature from every recognized authority on it, 0 if there are enough good signatures from recognized authorities on it, -1 if we might get enough good signatures by fetching missing certificates, and -2 otherwise. Log messages at INFO or WARN: if **warn** is over 1, warn about every problem; if warn is at least 1, warn only if we can't get enough signatures; if warn is negative, log nothing at all.

References authority_cert_get_by_digests(), DIGEST_LEN, authority_cert_t::expires, get_n_authorities(), hex_str(), LD_DIR, networkstatus_check_document_signature(), networkstatus_get_voter_by_id(), router_get_trusted_dir_servers(), smartlist_add(), smartlist_create(), smartlist_free(), tor_assert, trusteddirserver_get_by_v3_auth_digest(), networkstatus_t::type, V3_AUTHORITY, and networkstatus_t::voters.

Referenced by dirvote_compute_consensuses(), dirvote_fetch_missing_signatures(), dirvote_publish_consensus(), networkstatus_note_certs_arrived(), and networkstatus_set_current_consensus().

4.33.3.10 int networkstatus_check_document_signature (const networkstatus_t * *consensus*, document_signature_t * *sig*, const authority_cert_t * *cert*)

Check whether the signature **sig** is correctly signed with the signing key in **cert**. Return -1 if **cert** doesn't match the signing key; otherwise set the good_signature or bad_signature flag on **voter**, and return 0.

References document_signature_t::alg, document_signature_t::bad_signature, authority_cert_t::cache_info, crypto_pk_get_digest(), crypto_pk_keysize(), crypto_pk_public_checksig(), digests_t::d, DIGEST256_LEN, DIGEST_LEN, networkstatus_t::digests, document_signature_t::good_signature, signed_descriptor_t::identity_digest, document_signature_t::identity_digest, LD_DIR, document_signature_t::signature, document_signature_t::signature_len, authority_cert_t::signing_key, document_signature_t::signing_key_digest, and tor_free.

Referenced by networkstatus_add_detached_signatures(), and networkstatus_check_consensus_signature().

4.33.3.11 void networkstatus_consensus_download_failed (int *status_code*)

Called when an attempt to download a consensus fails: note that the failure occurred, and possibly retry.

References download_status_failed, and update_consensus_networkstatus_downloads().

Referenced by connection_dir_client_reached_eof(), and connection_dir_request_failed().

4.33.3.12 static void networkstatus_copy_old_consensus_info (networkstatus_t * *new_c*, const networkstatus_t * *old_c*) `[static]`

Copy all the ancillary information (like router download status and so on) from **old_c** to **new_c**.

References DIGEST_LEN, and networkstatus_t::routerstatus_list.

Referenced by networkstatus_set_current_consensus().

4.33.3.13 void networkstatus_dump_bridge_status_to_file (time_t *now*)

Write out router status entries for all our bridge descriptors.

References or_options_t::DataDirectory, get_options(), networkstatus_getinfo_by_purpose(), tor_free, tor_snprintf(), and write_str_to_file().

Referenced by run_scheduled_events().

4.33.3.14 void networkstatus_free_all (void)

Free all storage held locally in this module.

References _tor_free(), consensus_waiting_for_certs_t::body, consensus_waiting_for_certs_t::consensus, digestmap_free(), named_server_map, networkstatus_v2_free(), networkstatus_vote_free(), smartlist_free(), strmap_free(), tor_free, unnamed_server_map, and v2_download_status_map.

Referenced by tor_free_all().

4.33.3.15 int32_t networkstatus_get_bw_weight (networkstatus_t * *ns*, const char * *weight_name*, int32_t *default_val*)

Return the value of a integer bw weight parameter from the networkstatus **ns** whose name is **weight_name**. If **ns** is NULL, try loading the latest consensus ourselves. Return **default_val** if no latest consensus, or if it has no parameter called **param_name**.

References networkstatus_get_latest_consensus(), and networkstatus_t::weight_params.

Referenced by networkstatus_verify_bw_weights(), and smartlist_choose_by_bandwidth_weights().

4.33.3.16 char* networkstatus_get_cache_filename (const char * *identity_digest*)

Helper: return a newly allocated string containing the name of the filename where we plan to cache the network status with the given identity digest.

References base16_encode(), DIGEST_LEN, get_datadir_fname2, and HEX_DIGEST_LEN.

Referenced by add_networkstatus_to_cache(), dirserv_clear_old_networkstatuses(), networkstatus_v2_list_clean(), and router_set_networkstatus_v2().

4.33.3.17 const char* networkstatus_get_flavor_name (consensus_flavor_t *flav*)

Return the name of the consensus flavor **flav** as used to identify the flavor in directory documents.

References tor_fragile_assert.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_compute_consensuses(), dirvote_publish_consensus(), networkstatus_add_detached_signatures(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), networkstatus_note_certs_arrived(), networkstatus_set_current_consensus(), and router_reload_consensus_networkstatus().

4.33.3.18 networkstatus_t* networkstatus_get_latest_consensus (void)

Return the most recent consensus that we have downloaded, or NULL if we don't have one.

Referenced by directory_get_from_dirserver(), directory_handle_command_get(), getinfo_helper_events(), hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), networkstatus_get_bw_weight(), networkstatus_get_param(), options_act(), router_add_to_routerlist(), router_pick_directory_server_impl(), routerlist_remove_old_routers(), run_-

scheduled_events(), signed_desc_digest_is_recognized(), update_router_have_minimum_dir_info(), and upload_service_descriptor().

4.33.3.19 networkstatus_t* networkstatus_get_live_consensus (time_t *now*)

Return the most recent consensus that we have downloaded, or NULL if it is no longer live.

References networkstatus_t::valid_after.

Referenced by dirvote_recalculate_timing(), routers_update_all_from_networkstatus(), update_consensus_networkstatus_downloads(), and update_consensus_networkstatus_fetch_time().

4.33.3.20 int32_t networkstatus_get_param (networkstatus_t * *ns*, const char * *param_name*, int32_t *default_val*)

Return the value of a integer parameter from the networkstatus **ns** whose name is **param_name**. If **ns** is NULL, try loading the latest consensus ourselves. Return **default_val** if no latest consensus, or if it has no parameter called **param_name**.

References networkstatus_t::net_params, and networkstatus_get_latest_consensus().

Referenced by cell_ewma_set_scale_factor(), circuit_build_times_new_consensus_params(), circuit_initial_package_window(), connection_or_init_conn_from_address(), networkstatus_verify_bw_weights(), and smartlist_choose_by_bandwidth_weights().

4.33.3.21 networkstatus_t* networkstatus_get_reasonably_live_consensus (time_t *now*)

As **networkstatus_get_live_consensus()** (p. 723), but is way more tolerant of expired consensuses.

References networkstatus_t::valid_after.

Referenced by count_loading_descriptors_progress(), update_router_descriptor_downloads(), and update_router_have_minimum_dir_info().

4.33.3.22 const char* networkstatus_get_router_digest_by_nickname (const char * *nickname*)

Return the identity digest that's mapped to officially by **nickname**.

References named_server_map, and strmap_get_lc().

Referenced by router_get_by_nickname().

4.33.3.23 const smartlist_t* networkstatus_get_v2_list (void)

Return a list of the v2 networkstatus documents.

References smartlist_create().

Referenced by networkstatus_reset_download_failures(), router_add_to_routerlist(), routerlist_remove_old_routers(), signed_desc_digest_is_recognized(), and update_router_descriptor_cache_downloads_v2().

4.33.3.24 networkstatus_voter_info_t* networkstatus_get_voter_by_id (networkstatus_t * *vote*, const char * *identity*)

Return the voter info from **vote** for the voter whose identity digest is **identity**, or NULL if no such voter is associated with **vote**.

References DIGEST_LEN, and networkstatus_t::voters.

Referenced by networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), and networkstatus_parse_vote_from_string().

4.33.3.25 char* networkstatus_getinfo_by_purpose (const char * *purpose_string*, time_t *now*)

Alloc and return a string describing routerstatuses for the most recent info of each router we know about that is of purpose **purpose_string**. Return NULL if unrecognized purpose.

Right now this function is oriented toward listing bridges (you shouldn't use this for general-purpose routers, since those should be listed from the consensus, not from the routers list).

References authdir_mode_bridge(), dirserv_set_router_is_running(), get_options(), LD_DIR, networkstatus_getinfo_helper_single(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, ROUTER_PURPOSE_BRIDGE, router_purpose_from_string(), ROUTER_PURPOSE_UNKNOWN, routerlist_t::routers, set_routerstatus_from_routerinfo(), smartlist_add(), and smartlist_create().

Referenced by directory_handle_command_get(), and networkstatus_dump_bridge_status_to_file().

4.33.3.26 char* networkstatus_getinfo_helper_single (routerstatus_t * *rs*)

Generate networkstatus lines for a single **routerstatus_t** (p. 218) object, and return the result in a newly allocated string. Used only by controller interface (for now.)

References routerstatus_format_entry(), and RS_ENTRY_LEN.

Referenced by control_event_networkstatus_changed_helper(), getinfo_helper_networkstatus(), and networkstatus_getinfo_by_purpose().

4.33.3.27 int networkstatus_nickname_is_unnamed (const char * *nickname*)

Return true iff **nickname** is disallowed from being the nickname of any server.

References strmap_get_lc(), and unnamed_server_map.

Referenced by router_get_by_nickname().

4.33.3.28 void networkstatus_note_certs_arrived (void)

Called when we have gotten more certificates: see whether we can now verify a pending consensus.

References consensus_waiting_for_certs_t::body, consensus_waiting_for_certs_t::consensus, networkstatus_check_consensus_signature(), networkstatus_get_flavor_name(), networkstatus_set_current_consensus(), and tor_free.

Referenced by trusted_dirs_load_certs_from_string().

4.33.3.29 int networkstatus_parse_flavor_name (const char * *flavname*)

Return the consensus_flavor_t value for the flavor called **flavname**, or -1 if the flavor is not recognized.

Referenced by networkstatus_parse_vote_from_string(), and networkstatus_set_current_consensus().

4.33.3.30 void networkstatus_reset_download_failures (void)

Reset the descriptor download failure count on all networkstatus docs, so that we can retry any long-failed documents immediately.

References networkstatus_get_v2_list(), and router_get_by_descriptor_digest().

Referenced by router_reset_descriptor_download_failures().

4.33.3.31 void networkstatus_reset_warnings (void)

Forget that we've warned about anything networkstatus-related, so we will give fresh warnings if the same behavior happens again.

References have_warned_about_new_version, have_warned_about_old_version, and networkstatus_t::routerstatus_list.

Referenced by routerlist_reset_warnings().

4.33.3.32 int networkstatus_set_current_consensus (const char * *consensus*, const char * *flavor*, unsigned *flags*)

Try to replace the current cached v3 networkstatus with the one in **consensus**. If we don't have enough certificates to validate it, store it in consensus_waiting_for_certs and launch a certificate fetch.

If flags & NSSET_FROM_CACHE, this networkstatus has come from the disk cache. If flags & NSSET_WAS_WAITING_FOR_CERTS, this networkstatus was already received, but we were waiting for certificates on it. If flags & NSSET_DONT_DOWNLOAD_CERTS, do not launch certificate downloads as needed. If flags & NSSET_ACCEPT_OBSOLETE, then we should be willing to take this consensus, even if it comes from many days in the past.

Return 0 on success, <0 on failure. On failure, caller should increment the failure count as appropriate.

We return -1 for mild failures that don't need to be reported to the user, and -2 for more serious problems.

References authority_certs_fetch_missing(), consensus_waiting_for_certs_t::body, cell_ewma_set_scale_factor(), circ_times, circuit_build_times_new_consensus_params(), consensus_waiting_for_certs_t::consensus, control_event_client_status(), control_event_general_status(), cached_dir_t::digests, networkstatus_t::digests, directory_caches_dir_info(), dirserv_get_consensus(), dirserv_set_cached_consensus_networkstatus(), dirvote_recalculate_timing(), consensus_waiting_for_certs_t::dl_failed, download_status_failed, download_status_reset(), networkstatus_t::flavor, format_iso_time(), format_time_interval(), ftime_definitely_before(), get_datadir_fname, get_options(), LD_BUG, LD_DIR, LD_GENERAL, networkstatus_check_consensus_signature(), networkstatus_copy_old_consensus_info(), networkstatus_get_flavor_name(), networkstatus_parse_flavor_-

name(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), notify_control_networkstatus_changed(), OLD_ROUTER_DESC_MAX_AGE, cached_dir_t::published, router_dir_info_changed(), routerstatus_list_update_named_server_map(), consensus_waiting_for_certs_t::set_at, tor_free, tor_snprintf(), update_consensus_networkstatus_fetch_time(), USABLE_CONSENSUS_FLAVOR, networkstatus_t::valid_after, networkstatus_t::valid_until, and write_str_to_file().

Referenced by connection_dir_client_reached_eof(), dirvote_publish_consensus(), networkstatus_note_certs_arrived(), and router_reload_consensus_networkstatus().

4.33.3.33 routerstatus_t∗ networkstatus_v2_find_entry (networkstatus_v2_t ∗ *ns*, const char ∗ *digest*)

Return the entry in **ns** for the identity digest **digest**, or NULL if none was found.

References compare_digest_to_routerstatus_entry(), networkstatus_v2_t::entries, and smartlist_bsearch().

Referenced by router_add_to_routerlist(), and signed_desc_digest_is_recognized().

4.33.3.34 void networkstatus_v2_free (networkstatus_v2_t ∗ *ns*)

Free all storage held by the networkstatus object **ns**.

References networkstatus_v2_t::client_versions, networkstatus_v2_t::contact, crypto_free_pk_env(), networkstatus_v2_t::entries, routerstatus_free(), networkstatus_v2_t::server_versions, networkstatus_v2_t::signing_key, smartlist_free(), networkstatus_v2_t::source_address, and tor_free.

Referenced by networkstatus_free_all(), networkstatus_v2_list_clean(), networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

4.33.3.35 networkstatus_v2_t∗ networkstatus_v2_get_by_digest (const char ∗ *digest*)

Return the network status with a given identity digest.

References DIGEST_LEN.

4.33.3.36 void networkstatus_v2_list_clean (time_t *now*)

Remove all very-old network_status_t objects from memory and from the disk cache.

References directory_caches_v2_dir_info(), dirserv_clear_old_networkstatuses(), dirserv_clear_old_v1_info(), dirserv_set_cached_networkstatus_v2(), file_status(), get_options(), networkstatus_v2_t::identity_digest, LD_DIR, MAX_NETWORKSTATUS_AGE, networkstatus_get_cache_filename(), networkstatus_v2_free(), networkstatus_v2_t::published_on, smartlist_del(), and tor_free.

Referenced by router_reload_v2_networkstatus(), and run_scheduled_events().

4.33.3.37 routerstatus_t∗ networkstatus_vote_find_entry (networkstatus_t ∗ *ns*, const char ∗ *digest*)

Return the entry in **ns** for the identity digest **digest**, or NULL if none was found.

References compare_digest_to_routerstatus_entry(), networkstatus_t::routerstatus_list, and smartlist_bsearch().

Referenced by hid_serv_acting_as_directory(), router_add_to_routerlist(), router_get_consensus_status_by_nickname(), and signed_desc_digest_is_recognized().

4.33.3.38 int networkstatus_vote_find_entry_idx (networkstatus_t * *ns*, const char * *digest*, int * *found_out*)

Search the routerstatuses in **ns** for one whose identity digest is **digest**. Return value and set ***found_out** as for **smartlist_bsearch_idx()** (p. 471).

References compare_digest_to_routerstatus_entry(), networkstatus_t::routerstatus_list, and smartlist_bsearch_idx().

Referenced by hid_serv_get_responsible_directories().

4.33.3.39 void networkstatus_vote_free (networkstatus_t * *ns*)

Free all storage held in **ns**.

References authority_cert_free(), networkstatus_t::cert, networkstatus_t::client_versions, networkstatus_t::desc_digest_map, digestmap_free(), document_signature_free(), networkstatus_t::known_flags, networkstatus_t::net_params, routerstatus_free(), networkstatus_t::routerstatus_list, networkstatus_t::server_versions, smartlist_free(), networkstatus_t::supported_methods, tor_free, networkstatus_t::type, vote_routerstatus_free(), and networkstatus_t::voters.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_add_vote(), dirvote_clear_pending_consensuses(), dirvote_clear_votes(), dirvote_compute_consensuses(), dirvote_perform_vote(), format_networkstatus_vote(), networkstatus_free_all(), networkstatus_parse_vote_from_string(), and networkstatus_set_current_consensus().

4.33.3.40 static void notify_control_networkstatus_changed (const networkstatus_t * *old_c*, const networkstatus_t * *new_c*) `[static]`

Notify controllers of any router status entries that changed between **old_c** and **new_c**.

References control_event_is_interesting(), control_event_networkstatus_changed(), control_event_newconsensus(), DIGEST_LEN, routerstatus_has_changed(), networkstatus_t::routerstatus_list, smartlist_add(), smartlist_create(), and smartlist_free().

Referenced by networkstatus_set_current_consensus().

4.33.3.41 routerstatus_t* router_get_consensus_status_by_descriptor_digest (const char * *digest*)

Return the consensus view of the status of the router whose current *descriptor* digest is **digest**, or NULL if no such router is known.

References networkstatus_t::desc_digest_map, digestmap_new(), digestmap_set(), and networkstatus_t::routerstatus_list.

Referenced by router_get_dl_status_by_descriptor_digest(), and signed_descs_update_status_from_consensus_networkstatus().

4.33.3.42 routerstatus_t∗ router_get_consensus_status_by_id (const char ∗ *digest*)

Return the consensus view of the status of the router whose identity digest is **digest**, or NULL if we don't know about any such router.

References compare_digest_to_routerstatus_entry(), networkstatus_t::routerstatus_list, and smartlist_bsearch().

Referenced by circuit_list_path_impl(), connection_dir_client_reached_eof(), connection_or_digest_is_known_relay(), router_get_my_share_of_directory_requests(), router_set_status(), router_supports_extrainfo(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.33.3.43 routerstatus_t∗ router_get_consensus_status_by_nickname (const char ∗ *nickname*, int *warn_if_unnamed*)

Given a nickname (possibly verbose, possibly a hexadecimal digest), return the corresponding **routerstatus_t** (p. 218), or NULL if none exists. Warn the user if **warn_if_unnamed** is set, and they have specified a router by nickname, but the Named flag isn't set for that router.

References base16_decode(), DIGEST_LEN, escaped(), HEX_DIGEST_LEN, LD_GENERAL, named_server_map, networkstatus_vote_find_entry(), networkstatus_t::routerstatus_list, smartlist_create(), smartlist_free(), strmap_get_lc(), tor_fragile_assert, and unnamed_server_map.

Referenced by add_nickname_list_to_smartlist().

4.33.3.44 download_status_t∗ router_get_dl_status_by_descriptor_digest (const char ∗ *d*)

Given the digest of a router descriptor, return its current download status, or NULL if the digest is unrecognized.

References digestmap_get(), routerstatus_t::dl_status, router_get_consensus_status_by_descriptor_digest(), and v2_download_status_map.

Referenced by dir_routerdesc_download_failed(), and router_load_routers_from_string().

4.33.3.45 int router_reload_consensus_networkstatus (void)

Read every cached v3 consensus networkstatus from the disk.

References get_datadir_fname, get_options(), LD_FS, named_server_map, networkstatus_get_flavor_name(), networkstatus_set_current_consensus(), read_file_to_str(), RFTS_IGNORE_MISSING, routers_update_all_from_networkstatus(), tor_free, tor_snprintf(), unnamed_server_map, update_certificate_downloads(), and networkstatus_t::valid_after.

Referenced by do_main_loop().

4.33.3.46 int router_reload_v2_networkstatus (void)

Repopulate our list of network_status_t objects from the list cached on disk. Return 0 on success, -1 on failure.

References base16_decode(), DIGEST_LEN, directory_caches_v2_dir_info(), get_datadir_fname, get_datadir_fname2, get_options(), HEX_DIGEST_LEN, LD_DIR, LD_FS, networkstatus_v2_list_clean(), read_file_to_str(), remove_file_if_very_old(), router_set_networkstatus_v2(), routers_update_all_from_networkstatus(), smartlist_create(), smartlist_free(), tor_free, and tor_listdir().

Referenced by do_main_loop().

4.33.3.47 int router_set_networkstatus_v2 (const char ∗ *s*, time_t *arrived_at*, v2_networkstatus_source_t *source*, smartlist_t ∗ *requested_fingerprints*)

Given a string **s** containing a network status that we received at **arrived_at** from **source**, try to parse it, see if we want to store it, and put it into our cache as necessary.

If **source** is NS_FROM_DIR or NS_FROM_CACHE, do not replace our own **networkstatus_t** (p. 122) (if we're an authoritative directory server).

If **source** is NS_FROM_CACHE, do not write our **networkstatus_t** (p. 122) to the cache.

If **requested_fingerprints** is provided, it must contain a list of uppercased identity fingerprints. Do not update any networkstatus whose fingerprint is not on the list; after updating a networkstatus, remove its fingerprint from the list.

Return 0 on success, -1 on failure.

Callers should make sure that **routers_update_all_from_networkstatus()** (p. 729) is invoked after this function succeeds.

Retain any routerinfo mentioned in a V2 networkstatus for at least this long.

References add_networkstatus_to_cache(), base16_encode(), control_event_general_status(), trusted_dir_server_t::description, DIGEST_LEN, directory_caches_v2_dir_info(), download_status_failed, download_status_reset(), networkstatus_v2_t::entries, format_iso_time(), format_time_interval(), get_options(), HEX_DIGEST_LEN, networkstatus_v2_t::identity_digest, signed_descriptor_t::last_listed_as_valid_until, LD_DIR, LD_GENERAL, NETWORKSTATUS_ALLOW_SKEW, networkstatus_v2_t::networkstatus_digest, networkstatus_get_cache_filename(), networkstatus_v2_free(), networkstatus_v2_parse_from_string(), networkstatus_v2_t::published_on, networkstatus_v2_t::received_on, router_digest_is_me(), router_get_by_descriptor_digest(), router_get_trusteddirserver_by_digest(), smartlist_add(), smartlist_create(), smartlist_join_strings(), smartlist_string_isin(), smartlist_string_remove(), networkstatus_v2_t::source_address, networkstatus_v2_t::source_dirport, tor_assert, tor_free, touch_file(), trusted_dir_server_t::type, V2_AUTHORITY, and trusted_dir_server_t::v2_ns_dl_status.

Referenced by connection_dir_client_reached_eof(), and router_reload_v2_networkstatus().

4.33.3.48 void routers_update_all_from_networkstatus (time_t *now*, int *dir_version*)

If the network-status list has changed since the last time we called this function, update the status of every routerinfo from the network-status list. If **dir_version** is 2, it's a v2 networkstatus that changed. If **dir_version** is 3, it's a v3 consensus that changed.

References networkstatus_t::client_versions, control_event_general_status(), download_status_map_update_from_v2_networkstatus(), get_options(), have_warned_about_new_version, have_warned_about_old_version, LD_GENERAL, networkstatus_get_live_consensus(), networkstatus_v2_list_has_changed, routerlist_t::old_routers, router_get_-

routerlist(), routerlist_t::routers, routers_update_status_from_consensus_networkstatus(), server_mode(), networkstatus_t::server_versions, signed_descs_update_status_from_consensus_networkstatus(), tor_version_is_obsolete(), VS_EMPTY, VS_NEW, VS_NEW_IN_SERIES, VS_OLD, and VS_RECOMMENDED.

Referenced by connection_dir_client_reached_eof(), router_reload_consensus_networkstatus(), and router_reload_v2_networkstatus().

4.33.3.49 void routers_update_status_from_consensus_networkstatus (smartlist_t * *routers*, int *reset_failures*)

Given a list **routers** of **routerinfo_t** (p. 206) *, update each status field according to our current consensus networkstatus. May re-order **routers**.

References authdir_mode_v2(), authdir_mode_v3(), DIGEST_LEN, get_options(), or_options_t::NamingAuthoritativeDir, router_clear_status_flags(), ROUTER_PURPOSE_GENERAL, routers_sort_by_identity(), networkstatus_t::routerstatus_list, and smartlist_create().

Referenced by router_load_routers_from_string(), router_load_single_router(), and routers_update_all_from_networkstatus().

4.33.3.50 void routerstatus_free (routerstatus_t * *rs*)

Free all storage held by the routerstatus object **rs**.

References routerstatus_t::exitsummary, and tor_free.

Referenced by _free_duplicate_routerstatus_entry(), networkstatus_v2_free(), networkstatus_vote_free(), and routerstatus_parse_entry_from_string().

4.33.3.51 static int routerstatus_has_changed (const routerstatus_t * *a*, const routerstatus_t * *b*) `[static]`

Given two router status entries for the same router identity, return 1 if if the contents have changed between them. Otherwise, return 0.

References routerstatus_t::addr, routerstatus_t::descriptor_digest, DIGEST_LEN, routerstatus_t::dir_port, routerstatus_t::identity_digest, routerstatus_t::is_authority, routerstatus_t::is_bad_directory, routerstatus_t::is_bad_exit, routerstatus_t::is_exit, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerstatus_t::is_running, routerstatus_t::is_stable, routerstatus_t::is_unnamed, routerstatus_t::is_v2_dir, routerstatus_t::is_valid, routerstatus_t::nickname, routerstatus_t::or_port, tor_assert, routerstatus_t::version_known, routerstatus_t::version_supports_begindir, routerstatus_t::version_supports_conditional_consensus, routerstatus_t::version_supports_extrainfo_upload, and routerstatus_t::version_supports_v3_dir.

Referenced by notify_control_networkstatus_changed().

4.33.3.52 static void routerstatus_list_update_named_server_map (void) `[static]`

Update our view of the list of named servers from the most recently retrieved networkstatus consensus.

References _tor_free(), DIGEST_LEN, named_server_map, networkstatus_t::routerstatus_list, strmap_free(), strmap_set_lc(), and unnamed_server_map.

Referenced by networkstatus_set_current_consensus().

4.33.3.53 int should_delay_dir_fetches (or_options_t ∗ *options*)

Return 1 if there's a reason we shouldn't try any directory fetches yet (e.g. we demand bridges and none are yet known). Else return 0.

References any_bridge_descriptors_known(), LD_DIR, and or_options_t::UseBridges.

Referenced by authority_certs_fetch_missing(), update_extrainfo_downloads(), update_networkstatus_downloads(), update_router_descriptor_downloads(), and update_router_have_minimum_dir_info().

4.33.3.54 void signed_descs_update_status_from_consensus_networkstatus (smartlist_t ∗ *descs*)

Given a list of **signed_descriptor_t** (p. 227), update their fields (mainly, when they were last listed) from the most recent consensus.

References networkstatus_t::desc_digest_map, DIGEST_LEN, digestmap_get(), router_get_consensus_status_by_descriptor_digest(), and networkstatus_t::valid_until.

Referenced by routers_update_all_from_networkstatus().

4.33.3.55 void update_certificate_downloads (time_t *now*)

Launch requests as appropriate for missing directory authority certificates.

References authority_certs_fetch_missing().

Referenced by connection_dir_download_cert_failed(), router_reload_consensus_networkstatus(), and update_networkstatus_downloads().

4.33.3.56 static void update_consensus_networkstatus_downloads (time_t *now*) `[static]`

If we want to download a fresh consensus, launch a new download as appropriate.

References authdir_mode_v3(), CONN_TYPE_DIR, connection_get_by_type_purpose(), consensus_waiting_for_certs_t::consensus, CONSENSUS_NETWORKSTATUS_MAX_DL_TRIES, DELAY_WHILE_FETCHING_CERTS, DIR_PURPOSE_FETCH_CONSENSUS, directory_get_from_dirserver(), consensus_waiting_for_certs_t::dl_failed, download_status_failed, download_status_is_ready(), get_options(), LD_DIR, networkstatus_get_live_consensus(), PDS_RETRY_IF_NO_SERVERS, ROUTER_PURPOSE_GENERAL, consensus_waiting_for_certs_t::set_at, and time_to_download_next_consensus.

Referenced by networkstatus_consensus_download_failed(), and update_networkstatus_downloads().

4.33.3.57 void update_consensus_networkstatus_fetch_time (time_t *now*)

Update the time at which we'll consider replacing the current consensus.

References CONSENSUS_MIN_SECONDS_BEFORE_CACHING, crypto_rand_int(), directory_fetches_dir_info_early(), directory_fetches_dir_info_later(), or_options_t::FetchDirInfoExtraEarly, format_local_iso_time(), networkstatus_t::fresh_until, get_options(), LD_DIR, networkstatus_get_live_consensus(), time_to_download_next_consensus, tor_assert, networkstatus_t::valid_after, and networkstatus_t::valid_until.

Referenced by networkstatus_set_current_consensus(), and options_act().

4.33.3.58 void update_networkstatus_downloads (time_t *now*)

Launch requests for networkstatus documents and authority certificates as appropriate.

References directory_fetches_dir_info_early(), get_options(), should_delay_dir_fetches(), update_certificate_downloads(), update_consensus_networkstatus_downloads(), and update_v2_networkstatus_cache_downloads().

Referenced by do_hup(), routerlist_retry_directory_downloads(), and run_scheduled_events().

4.33.3.59 static void update_v2_networkstatus_cache_downloads (time_t *now*) `[static]`

We are a directory server, and so cache network_status documents. Initiate downloads as needed to update them. For v2 authorities, this means asking each trusted directory for its network-status. For caches, this means asking a random v2 authority for all network-statuses.

References tor_addr_t::addr, authdir_mode_v2(), AUTHORITY_NS_CACHE_INTERVAL, base16_encode(), CONN_TYPE_DIR, connection_get_by_type_addr_port_purpose(), connection_get_by_type_purpose(), DIGEST_LEN, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, directory_get_from_dirserver(), directory_initiate_command_routerstatus(), get_options(), HEX_DIGEST_LEN, last_networkstatus_download_attempted, NONAUTHORITY_NS_CACHE_INTERVAL, PDS_RETRY_IF_NO_SERVERS, router_digest_is_me(), router_get_trusted_dir_servers(), ROUTER_PURPOSE_GENERAL, tor_addr_from_ipv4h, trusted_dir_servers, and V2_AUTHORITY.

Referenced by update_networkstatus_downloads().

4.33.3.60 static void vote_routerstatus_free (vote_routerstatus_t * *rs*) `[static]`

Free all storage held by the vote_routerstatus object **rs**.

References routerstatus_t::exitsummary, vote_routerstatus_t::microdesc, vote_routerstatus_t::status, tor_free, and vote_routerstatus_t::version.

Referenced by networkstatus_vote_free().

4.33.4 Variable Documentation

4.33.4.1 download_status_t consensus_dl_status[N_CONSENSUS_FLAVORS] `[static]`

Download status for the current consensus networkstatus.

4.33.4.2 networkstatus_t* current_consensus = NULL [static]

Most recently received and validated v3 consensus network status.

4.33.4.3 int have_warned_about_new_version = 0 [static]

True iff we have logged a warning about this OR's version being newer than listed by the authorities.

Referenced by networkstatus_reset_warnings(), and routers_update_all_from_networkstatus().

4.33.4.4 int have_warned_about_old_version = 0 [static]

True iff we have logged a warning about this OR's version being older than listed by the authorities.

Referenced by networkstatus_reset_warnings(), and routers_update_all_from_networkstatus().

4.33.4.5 time_t last_networkstatus_download_attempted = 0 [static]

The last time we tried to download a networkstatus, or 0 for "never". We use this to rate-limit download attempts for directory caches (including mirrors). Clients don't use this now.

Referenced by update_v2_networkstatus_cache_downloads().

4.33.4.6 strmap_t* named_server_map = NULL [static]

Map from lowercase nickname to identity digest of named server, if any.

Referenced by networkstatus_free_all(), networkstatus_get_router_digest_by_nickname(), router_get_consensus_status_by_nickname(), router_reload_consensus_networkstatus(), and routerstatus_list_update_named_server_map().

4.33.4.7 smartlist_t* networkstatus_v2_list = NULL [static]

Global list of all of the current v2 network_status documents that we know about. This list is kept sorted by published_on.

Referenced by router_add_to_routerlist(), routerlist_remove_old_routers(), signed_desc_digest_is_recognized(), and update_router_descriptor_cache_downloads_v2().

4.33.4.8 int networkstatus_v2_list_has_changed = 0 [static]

True iff any member of networkstatus_v2_list has changed since the last time we called **download_status_map_update_from_v2_networkstatus()** (p. 720)

Referenced by download_status_map_update_from_v2_networkstatus(), and routers_update_all_from_networkstatus().

4.33.4.9 time_t time_to_download_next_consensus = 0 [static]

A time before which we shouldn't try to replace the current consensus: this will be at some point after the next consensus becomes valid, but before the current consensus becomes invalid.

Referenced by update_consensus_networkstatus_downloads(), and update_consensus_networkstatus_fetch_time().

4.33.4.10 strmap_t* unnamed_server_map = NULL [static]

Map from lowercase nickname to (void*)1 for all names that are listed as unnamed for some server in the consensus.

Referenced by networkstatus_free_all(), networkstatus_nickname_is_unnamed(), router_get_consensus_status_by_nickname(), router_reload_consensus_networkstatus(), and routerstatus_list_update_named_server_map().

4.33.4.11 digestmap_t* v2_download_status_map = NULL [static]

Map from descriptor digest of routers listed in the v2 networkstatus documents to download_status_t*

Referenced by download_status_map_update_from_v2_networkstatus(), networkstatus_free_all(), and router_get_dl_status_by_descriptor_digest().

4.34 onion.c File Reference

Functions to queue create cells, and handle onionskin parsing and creation.

`#include "or.h"`

Data Structures

- struct **onion_queue_t**

Defines

- #define **ONIONQUEUE_WAIT_CUTOFF** 5
- #define **WARN_TOO_MANY_CIRC_CREATIONS_INTERVAL** (60)

Functions

- int **onion_pending_add** (**or_circuit_t** ∗circ, char ∗onionskin)
- **or_circuit_t** ∗ **onion_next_task** (char ∗∗onionskin_out)
- void **onion_pending_remove** (**or_circuit_t** ∗circ)
- int **onion_skin_create** (**crypto_pk_env_t** ∗dest_router_key, **crypto_dh_env_t** ∗∗handshake_state_out, char ∗onion_skin_out)
- int **onion_skin_server_handshake** (const char ∗onion_skin, **crypto_pk_env_t** ∗private_key, **crypto_pk_env_t** ∗prev_private_key, char ∗handshake_reply_out, char ∗key_out, size_t key_out_len)
- int **onion_skin_client_handshake** (**crypto_dh_env_t** ∗handshake_state, const char ∗handshake_reply, char ∗key_out, size_t key_out_len)
- int **fast_server_handshake** (const char ∗key_in, char ∗handshake_reply_out, char ∗key_out, size_t key_out_len)
- int **fast_client_handshake** (const char ∗handshake_state, const char ∗handshake_reply_out, char ∗key_out, size_t key_out_len)
- void **clear_pending_onions** (void)

Variables

- static **onion_queue_t** ∗ **ol_list** = NULL
- static **onion_queue_t** ∗ **ol_tail** = NULL
- static int **ol_length** = 0

4.34.1 Detailed Description

Functions to queue create cells, and handle onionskin parsing and creation.

4.34.2 Define Documentation

4.34.2.1 #define ONIONQUEUE_WAIT_CUTOFF 5

5 seconds on the onion queue til we just send back a destroy

Referenced by onion_pending_add().

4.34.3 Function Documentation

4.34.3.1 void clear_pending_onions (void)

Remove all circuits from the pending list. Called from tor_free_all.

References onion_queue_t::next, ol_length, onion_queue_t::onionskin, and tor_free.

Referenced by tor_free_all().

4.34.3.2 int fast_client_handshake (const char ∗ *handshake_state*, const char ∗ *handshake_reply_out*, char ∗ *key_out*, size_t *key_out_len*)

Implement the second half of the client side of the CREATE_FAST handshake. We sent the server **handshake_state** ("x") already, and the server told us **handshake_reply_out** (y|H(x|y)). Make sure that the hash is correct, and generate key material in **key_out**. Return 0 on success, true on failure.

NOTE: The "CREATE_FAST" handshake path is distinguishable from regular "onionskin" handshakes, and is not secure if an adversary can see or modify the messages. Therefore, it should only be used by clients, and only as the first hop of a circuit (since the first hop is already authenticated and protected by TLS).

References crypto_expand_key_material(), DIGEST_LEN, LD_PROTOCOL, and tor_free.

Referenced by circuit_finish_handshake().

4.34.3.3 int fast_server_handshake (const char ∗ *key_in*, char ∗ *handshake_reply_out*, char ∗ *key_out*, size_t *key_out_len*)

Implement the server side of the CREATE_FAST abbreviated handshake. The client has provided DIGEST_LEN key bytes in **key_in** ("x"). We generate a reply of DIGEST_LEN∗2 bytes in **key_out**, consisting of a new random "y", followed by H(x|y) to check for correctness. We set **key_out_len** bytes of key material in **key_out**. Return 0 on success, <0 on failure.

References crypto_expand_key_material(), crypto_rand(), DIGEST_LEN, and tor_free.

Referenced by command_process_create_cell().

4.34.3.4 or_circuit_t∗ onion_next_task (char ∗∗ *onionskin_out*)

Remove the first item from ol_list and return it, or return NULL if the list is empty.

References onion_queue_t::circ, ol_length, onion_pending_remove(), onion_queue_t::onionskin, or_circuit_t::p_conn, and tor_assert.

Referenced by process_pending_task().

4.34.3.5 int onion_pending_add (or_circuit_t ∗ *circ*, char ∗ *onionskin*)

Add **circ** to the end of ol_list and return 0, except if ol_list is too long, in which case do nothing and return -1.

References onion_queue_t::circ, get_options(), LD_CIRC, LD_GENERAL, onion_queue_t::next, ol_length, onion_pending_remove(), ONIONQUEUE_WAIT_CUTOFF, onion_queue_t::onionskin, TO_CIRCUIT, tor_assert, tor_free, and onion_queue_t::when_added.

Referenced by assign_onionskin_to_cpuworker().

4.34.3.6 void onion_pending_remove (or_circuit_t * *circ*)

Go through ol_list, find the **onion_queue_t** (p. 132) element which points to circ, remove and free that element. Leave circ itself alone.

References onion_queue_t::circ, LD_GENERAL, onion_queue_t::next, ol_length, onion_queue_t::onionskin, or_circuit_t::p_circ_id, and tor_free.

Referenced by _circuit_mark_for_close(), onion_next_task(), and onion_pending_add().

4.34.3.7 int onion_skin_client_handshake (crypto_dh_env_t * *handshake_state*, const char * *handshake_reply*, char * *key_out*, size_t *key_out_len*)

Finish the client side of the DH handshake. Given the 128 byte DH reply + 20 byte hash as generated by onion_skin_server_handshake and the handshake state generated by onion_skin_create, verify H(K) with the first 20 bytes of shared key material, then generate key_out_len more bytes of shared key material and store them in key_out.

After the invocation, call crypto_dh_free on handshake_state.

References crypto_dh_compute_secret(), crypto_dh_get_bytes(), DIGEST_LEN, LD_PROTOCOL, tor_assert, and tor_free.

Referenced by circuit_finish_handshake().

4.34.3.8 int onion_skin_create (crypto_pk_env_t * *dest_router_key*, crypto_dh_env_t ** *handshake_state_out*, char * *onion_skin_out*)

Given a router's 128 byte public key, stores the following in onion_skin_out:

42 bytes	OAEP padding
16 bytes	Symmetric key for encrypting blob past RSA
70 bytes	g^x part 1 (inside the RSA)
58 bytes	g^x part 2 (symmetrically encrypted)

Stores the DH private key into handshake_state_out for later completion of the handshake.

The meeting point/cookies and auth are zeroed out for now.

References crypto_dh_free(), crypto_dh_get_bytes(), crypto_dh_get_public(), crypto_dh_new(), crypto_pk_keysize(), crypto_pk_public_hybrid_encrypt(), note_crypto_pk_op(), PK_PKCS1_OAEP_PADDING, and tor_assert.

Referenced by circuit_send_next_onion_skin().

4.34.3.9 int onion_skin_server_handshake (const char * *onion_skin*, crypto_pk_env_t * *private_key*, crypto_pk_env_t * *prev_private_key*, char * *handshake_reply_out*, char * *key_out*, size_t *key_out_len*)

Given an encrypted DH public key as generated by onion_skin_create, and the private key for this onion router, generate the reply (128-byte DH plus the first 20 bytes of shared key material), and store the next key_out_len bytes of key material in key_out.

References crypto_dh_compute_secret(), crypto_dh_free(), crypto_dh_get_public(), crypto_dh_new(), crypto_pk_private_hybrid_decrypt(), DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, note_crypto_pk_op(), PK_PKCS1_OAEP_PADDING, and tor_free.

Referenced by cpuworker_main().

4.34.4 Variable Documentation

4.34.4.1 int ol_length = 0 `[static]`

Length of ol_list

Referenced by clear_pending_onions(), onion_next_task(), onion_pending_add(), and onion_pending_remove().

4.34.4.2 onion_queue_t* ol_list = NULL `[static]`

First and last elements in the linked list of circuits waiting for CPU workers, or NULL if the list is empty.

4.35 or.h File Reference

Master header file for Tor-specific functionality.

```
#include "orconfig.h"
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <sys/param.h>
#include "torint.h"
#include <sys/wait.h>
#include <sys/fcntl.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/un.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <errno.h>
#include <assert.h>
#include <time.h>
#include "crypto.h"
#include "tortls.h"
#include "../common/log.h"
#include "compat.h"
#include "container.h"
#include "util.h"
#include "torgzip.h"
#include "address.h"
#include "compat_libevent.h"
#include "ht.h"
```

Data Structures

- struct **rend_service_authorization_t**
- struct **rend_data_t**
- struct **cell_t**
- struct **var_cell_t**
- struct **packed_cell_t**
- struct **insertion_time_elem_t**
- struct **insertion_time_queue_t**
- struct **cell_queue_t**

- struct **relay_header_t**
- struct **connection_t**
- struct **or_handshake_state_t**
- struct **or_connection_t**
- struct **edge_connection_t**
- struct **dir_connection_t**
- struct **control_connection_t**
- struct **addr_policy_t**
- struct **cached_dir_t**
- struct **download_status_t**
- struct **signed_descriptor_t**
- struct **routerinfo_t**
- struct **extrainfo_t**
- struct **routerstatus_t**
- struct **microdesc_t**
- struct **networkstatus_v2_t**
- struct **vote_microdesc_hash_t**
- struct **vote_routerstatus_t**
- struct **document_signature_t**
- struct **networkstatus_voter_info_t**
- struct **networkstatus_t**
- struct **ns_detached_signatures_t**
- struct **desc_store_t**
- struct **routerlist_t**
- struct **extend_info_t**
- struct **authority_cert_t**
- struct **crypt_path_t**
- struct **cpath_build_state_t**
- struct **cell_ewma_t**
- struct **circuit_t**
- struct **origin_circuit_t**
- struct **or_circuit_t**
- struct **config_line_t**
- struct **or_options_t**
- struct **or_state_t**
- struct **socks_request_t**
- struct **network_liveness_t**
- struct **circuit_build_times_t**
- struct **fp_pair_t**
- struct **vote_timing_t**
- struct **rend_authorized_client_t**
- struct **rend_encoded_v2_service_descriptor_t**
- struct **rend_intro_point_t**
- struct **rend_service_descriptor_t**
- struct **rend_cache_entry_t**
- struct **trusted_dir_server_t**
- struct **tor_version_t**

Defines

- #define **SIGHUP** 1
- #define **SIGINT** 2
- #define **SIGUSR1** 10
- #define **SIGUSR2** 12
- #define **SIGTERM** 15
- #define **SIGNEWNYM** 129
- #define **SIGCLEARDNSCACHE** 130
- #define **MAX_NICKNAME_LEN** 19
- #define **MAX_HEX_NICKNAME_LEN** (HEX_DIGEST_LEN+1)
- #define **MAX_VERBOSE_NICKNAME_LEN** (1+HEX_DIGEST_LEN+1+MAX_NICKNAME_LEN)
- #define **MAX_BUF_SIZE** ((1<<24)-1)
- #define **MAX_DIR_DL_SIZE** MAX_BUF_SIZE
- #define **MAX_HEADERS_SIZE** 50000
- #define **MAX_DIR_UL_SIZE** MAX_BUF_SIZE
- #define **MAX_DESCRIPTOR_UPLOAD_SIZE** 20000
- #define **MAX_EXTRAINFO_UPLOAD_SIZE** 50000
- #define **MAX_DNS_ENTRY_AGE** (30*60)
- #define **DEFAULT_DNS_TTL** (30*60)
- #define **MAX_DNS_TTL** (3*60*60)
- #define **MIN_DNS_TTL** (60)
- #define **MIN_ONION_KEY_LIFETIME** (7*24*60*60)
- #define **MAX_SSL_KEY_LIFETIME** (2*60*60)
- #define **ROUTER_MAX_AGE** (60*60*48)
- #define **ROUTER_MAX_AGE_TO_PUBLISH** (60*60*20)
- #define **OLD_ROUTER_DESC_MAX_AGE** (60*60*24*5)
- #define **_CONN_TYPE_MIN** 3
- #define **CONN_TYPE_OR_LISTENER** 3
- #define **CONN_TYPE_OR** 4
- #define **CONN_TYPE_EXIT** 5
- #define **CONN_TYPE_AP_LISTENER** 6
- #define **CONN_TYPE_AP** 7
- #define **CONN_TYPE_DIR_LISTENER** 8
- #define **CONN_TYPE_DIR** 9
- #define **CONN_TYPE_CPUWORKER** 10
- #define **CONN_TYPE_CONTROL_LISTENER** 11
- #define **CONN_TYPE_CONTROL** 12
- #define **CONN_TYPE_AP_TRANS_LISTENER** 13
- #define **CONN_TYPE_AP_NATD_LISTENER** 14
- #define **CONN_TYPE_AP_DNS_LISTENER** 15
- #define **_CONN_TYPE_MAX** 15
- #define **PROXY_NONE** 0
- #define **PROXY_CONNECT** 1
- #define **PROXY_SOCKS4** 2
- #define **PROXY_SOCKS5** 3
- #define **PROXY_HTTPS_WANT_CONNECT_OK** 1
- #define **PROXY_SOCKS4_WANT_CONNECT_OK** 2
- #define **PROXY_SOCKS5_WANT_AUTH_METHOD_NONE** 3

- #define **PROXY_SOCKS5_WANT_AUTH_METHOD_RFC1929** 4
- #define **PROXY_SOCKS5_WANT_AUTH_RFC1929_OK** 5
- #define **PROXY_SOCKS5_WANT_CONNECT_OK** 6
- #define **PROXY_CONNECTED** 7
- #define **CONN_IS_EDGE**(x) ((x) → type == CONN_TYPE_EXIT || (x) → type == CONN_TYPE_AP)
- #define **LISTENER_STATE_READY** 0
- #define **_CPUWORKER_STATE_MIN** 1
- #define **CPUWORKER_STATE_IDLE** 1
- #define **CPUWORKER_STATE_BUSY_ONION** 2
- #define **_CPUWORKER_STATE_MAX** 2
- #define **CPUWORKER_TASK_ONION** CPUWORKER_STATE_BUSY_ONION
- #define **_OR_CONN_STATE_MIN** 1
- #define **OR_CONN_STATE_CONNECTING** 1
- #define **OR_CONN_STATE_PROXY_HANDSHAKING** 2
- #define **OR_CONN_STATE_TLS_HANDSHAKING** 3
- #define **OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING** 4
- #define **OR_CONN_STATE_TLS_SERVER_RENEGOTIATING** 5
- #define **OR_CONN_STATE_OR_HANDSHAKING** 6
- #define **OR_CONN_STATE_OPEN** 7
- #define **_OR_CONN_STATE_MAX** 7
- #define **_EXIT_CONN_STATE_MIN** 1
- #define **EXIT_CONN_STATE_RESOLVING** 1
- #define **EXIT_CONN_STATE_CONNECTING** 2
- #define **EXIT_CONN_STATE_OPEN** 3
- #define **EXIT_CONN_STATE_RESOLVEFAILED** 4
- #define **_EXIT_CONN_STATE_MAX** 4
- #define **_AP_CONN_STATE_MIN** 5
- #define **AP_CONN_STATE_SOCKS_WAIT** 5
- #define **AP_CONN_STATE_RENDDESC_WAIT** 6
- #define **AP_CONN_STATE_CONTROLLER_WAIT** 7
- #define **AP_CONN_STATE_CIRCUIT_WAIT** 8
- #define **AP_CONN_STATE_CONNECT_WAIT** 9
- #define **AP_CONN_STATE_RESOLVE_WAIT** 10
- #define **AP_CONN_STATE_OPEN** 11
- #define **AP_CONN_STATE_NATD_WAIT** 12
- #define **_AP_CONN_STATE_MAX** 12
- #define **AP_CONN_STATE_IS_UNATTACHED**(s) ((s) <= AP_CONN_STATE_CIRCUIT_WAIT || (s) == AP_CONN_STATE_NATD_WAIT)
- #define **_DIR_CONN_STATE_MIN** 1
- #define **DIR_CONN_STATE_CONNECTING** 1
- #define **DIR_CONN_STATE_CLIENT_SENDING** 2
- #define **DIR_CONN_STATE_CLIENT_READING** 3
- #define **DIR_CONN_STATE_CLIENT_FINISHED** 4
- #define **DIR_CONN_STATE_SERVER_COMMAND_WAIT** 5
- #define **DIR_CONN_STATE_SERVER_WRITING** 6
- #define **_DIR_CONN_STATE_MAX** 6
- #define **DIR_CONN_IS_SERVER**(conn) ((conn) → purpose == DIR_PURPOSE_SERVER)
- #define **_CONTROL_CONN_STATE_MIN** 1

- #define **CONTROL_CONN_STATE_OPEN** 1
- #define **CONTROL_CONN_STATE_NEEDAUTH** 2
- #define **_CONTROL_CONN_STATE_MAX** 2
- #define **_DIR_PURPOSE_MIN** 3
- #define **DIR_PURPOSE_FETCH_RENDDESC** 3
- #define **DIR_PURPOSE_HAS_FETCHED_RENDDESC** 4
- #define **DIR_PURPOSE_FETCH_V2_NETWORKSTATUS** 5
- #define **DIR_PURPOSE_FETCH_SERVERDESC** 6
- #define **DIR_PURPOSE_FETCH_EXTRAINFO** 7
- #define **DIR_PURPOSE_UPLOAD_DIR** 8
- #define **DIR_PURPOSE_UPLOAD_RENDDESC** 9
- #define **DIR_PURPOSE_UPLOAD_VOTE** 10
- #define **DIR_PURPOSE_UPLOAD_SIGNATURES** 11
- #define **DIR_PURPOSE_FETCH_STATUS_VOTE** 12
- #define **DIR_PURPOSE_FETCH_DETACHED_SIGNATURES** 13
- #define **DIR_PURPOSE_FETCH_CONSENSUS** 14
- #define **DIR_PURPOSE_FETCH_CERTIFICATE** 15
- #define **DIR_PURPOSE_SERVER** 16
- #define **DIR_PURPOSE_UPLOAD_RENDDESC_V2** 17
- #define **DIR_PURPOSE_FETCH_RENDDESC_V2** 18
- #define **_DIR_PURPOSE_MAX** 18
- #define **DIR_PURPOSE_IS_UPLOAD**(p)
- #define **_EXIT_PURPOSE_MIN** 1
- #define **EXIT_PURPOSE_CONNECT** 1
- #define **EXIT_PURPOSE_RESOLVE** 2
- #define **_EXIT_PURPOSE_MAX** 2
- #define **CIRCUIT_STATE_BUILDING** 0
- #define **CIRCUIT_STATE_ONIONSKIN_PENDING** 1
- #define **CIRCUIT_STATE_OR_WAIT** 2
- #define **CIRCUIT_STATE_OPEN** 3
- #define **_CIRCUIT_PURPOSE_MIN** 1
- #define **_CIRCUIT_PURPOSE_OR_MIN** 1
- #define **CIRCUIT_PURPOSE_OR** 1
- #define **CIRCUIT_PURPOSE_INTRO_POINT** 2
- #define **CIRCUIT_PURPOSE_REND_POINT_WAITING** 3
- #define **CIRCUIT_PURPOSE_REND_ESTABLISHED** 4
- #define **_CIRCUIT_PURPOSE_OR_MAX** 4
- #define **CIRCUIT_PURPOSE_C_GENERAL** 5
- #define **CIRCUIT_PURPOSE_C_INTRODUCING** 6
- #define **CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT** 7
- #define **CIRCUIT_PURPOSE_C_INTRODUCE_ACKED** 8
- #define **CIRCUIT_PURPOSE_C_ESTABLISH_REND** 9
- #define **CIRCUIT_PURPOSE_C_REND_READY** 10
- #define **CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED** 11
- #define **CIRCUIT_PURPOSE_C_REND_JOINED** 12
- #define **_CIRCUIT_PURPOSE_C_MAX** 12
- #define **CIRCUIT_PURPOSE_S_ESTABLISH_INTRO** 13
- #define **CIRCUIT_PURPOSE_S_INTRO** 14
- #define **CIRCUIT_PURPOSE_S_CONNECT_REND** 15
- #define **CIRCUIT_PURPOSE_S_REND_JOINED** 16

- #define **CIRCUIT_PURPOSE_TESTING** 17
- #define **CIRCUIT_PURPOSE_CONTROLLER** 18
- #define **_CIRCUIT_PURPOSE_MAX** 18
- #define **CIRCUIT_PURPOSE_UNKNOWN** 255
- #define **CIRCUIT_PURPOSE_IS_ORIGIN**(p) ((p)>_CIRCUIT_PURPOSE_OR_MAX)
- #define **CIRCUIT_PURPOSE_IS_CLIENT**(p)
- #define **CIRCUIT_IS_ORIGIN**(c) (CIRCUIT_PURPOSE_IS_ORIGIN((c) → purpose))
- #define **CIRCUIT_PURPOSE_IS_ESTABLISHED_REND**(p)
- #define **MIN_CIRCUITS_HANDLING_STREAM** 2
- #define **RELAY_COMMAND_BEGIN** 1
- #define **RELAY_COMMAND_DATA** 2
- #define **RELAY_COMMAND_END** 3
- #define **RELAY_COMMAND_CONNECTED** 4
- #define **RELAY_COMMAND_SENDME** 5
- #define **RELAY_COMMAND_EXTEND** 6
- #define **RELAY_COMMAND_EXTENDED** 7
- #define **RELAY_COMMAND_TRUNCATE** 8
- #define **RELAY_COMMAND_TRUNCATED** 9
- #define **RELAY_COMMAND_DROP** 10
- #define **RELAY_COMMAND_RESOLVE** 11
- #define **RELAY_COMMAND_RESOLVED** 12
- #define **RELAY_COMMAND_BEGIN_DIR** 13
- #define **RELAY_COMMAND_ESTABLISH_INTRO** 32
- #define **RELAY_COMMAND_ESTABLISH_RENDEZVOUS** 33
- #define **RELAY_COMMAND_INTRODUCE1** 34
- #define **RELAY_COMMAND_INTRODUCE2** 35
- #define **RELAY_COMMAND_RENDEZVOUS1** 36
- #define **RELAY_COMMAND_RENDEZVOUS2** 37
- #define **RELAY_COMMAND_INTRO_ESTABLISHED** 38
- #define **RELAY_COMMAND_RENDEZVOUS_ESTABLISHED** 39
- #define **RELAY_COMMAND_INTRODUCE_ACK** 40
- #define **END_OR_CONN_REASON_DONE** 1
- #define **END_OR_CONN_REASON_REFUSED** 2
- #define **END_OR_CONN_REASON_OR_IDENTITY** 3
- #define **END_OR_CONN_REASON_CONNRESET** 4
- #define **END_OR_CONN_REASON_TIMEOUT** 5
- #define **END_OR_CONN_REASON_NO_ROUTE** 6
- #define **END_OR_CONN_REASON_IO_ERROR** 7
- #define **END_OR_CONN_REASON_RESOURCE_LIMIT** 8
- #define **END_OR_CONN_REASON_MISC** 9
- #define **END_STREAM_REASON_MISC** 1
- #define **END_STREAM_REASON_RESOLVEFAILED** 2
- #define **END_STREAM_REASON_CONNECTREFUSED** 3
- #define **END_STREAM_REASON_EXITPOLICY** 4
- #define **END_STREAM_REASON_DESTROY** 5
- #define **END_STREAM_REASON_DONE** 6
- #define **END_STREAM_REASON_TIMEOUT** 7
- #define **END_STREAM_REASON_HIBERNATING** 9

- #define **END_STREAM_REASON_INTERNAL** 10
- #define **END_STREAM_REASON_RESOURCELIMIT** 11
- #define **END_STREAM_REASON_CONNRESET** 12
- #define **END_STREAM_REASON_TORPROTOCOL** 13
- #define **END_STREAM_REASON_NOTDIRECTORY** 14
- #define **END_STREAM_REASON_ENTRYPOLICY** 15
- #define **END_STREAM_REASON_CANT_ATTACH** 257
- #define **END_STREAM_REASON_NET_UNREACHABLE** 258
- #define **END_STREAM_REASON_SOCKSPROTOCOL** 259
- #define **END_STREAM_REASON_CANT_FETCH_ORIG_DEST** 260
- #define **END_STREAM_REASON_INVALID_NATD_DEST** 261
- #define **END_STREAM_REASON_MASK** 511
- #define **END_STREAM_REASON_FLAG_REMOTE** 512
- #define **END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED** 1024
- #define **END_STREAM_REASON_FLAG_ALREADY_SOCKS_REPLIED** 2048
- #define **REMAP_STREAM_SOURCE_CACHE** 1
- #define **REMAP_STREAM_SOURCE_EXIT** 2
- #define **RESOLVED_TYPE_HOSTNAME** 0
- #define **RESOLVED_TYPE_IPV4** 4
- #define **RESOLVED_TYPE_IPV6** 6
- #define **RESOLVED_TYPE_ERROR_TRANSIENT** 0xF0
- #define **RESOLVED_TYPE_ERROR** 0xF1
- #define **END_CIRC_REASON_NOPATH** -2
- #define **END_CIRC_AT_ORIGIN** -1
- #define **_END_CIRC_REASON_MIN** 0
- #define **END_CIRC_REASON_NONE** 0
- #define **END_CIRC_REASON_TORPROTOCOL** 1
- #define **END_CIRC_REASON_INTERNAL** 2
- #define **END_CIRC_REASON_REQUESTED** 3
- #define **END_CIRC_REASON_HIBERNATING** 4
- #define **END_CIRC_REASON_RESOURCELIMIT** 5
- #define **END_CIRC_REASON_CONNECTFAILED** 6
- #define **END_CIRC_REASON_OR_IDENTITY** 7
- #define **END_CIRC_REASON_OR_CONN_CLOSED** 8
- #define **END_CIRC_REASON_FINISHED** 9
- #define **END_CIRC_REASON_TIMEOUT** 10
- #define **END_CIRC_REASON_DESTROYED** 11
- #define **END_CIRC_REASON_NOSUCHSERVICE** 12
- #define **_END_CIRC_REASON_MAX** 12
- #define **END_CIRC_REASON_FLAG_REMOTE** 512
- #define **REND_SERVICE_ID_LEN_BASE32** 16
- #define **REND_SERVICE_ADDRESS_LEN** (16+1+5)
- #define **REND_SERVICE_ID_LEN** 10
- #define **REND_TIME_PERIOD_V2_DESC_VALIDITY** (24*60*60)
- #define **REND_TIME_PERIOD_OVERLAPPING_V2_DESCS** (60*60)
- #define **REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS** 2
- #define **REND_NUMBER_OF_CONSECUTIVE_REPLICAS** 3
- #define **REND_DESC_ID_V2_LEN_BASE32** 32

- #define **REND_SECRET_ID_PART_LEN_BASE32** 32
- #define **REND_INTRO_POINT_ID_LEN_BASE32** 32
- #define **REND_DESC_COOKIE_LEN** 16
- #define **REND_DESC_COOKIE_LEN_BASE64** 22
- #define **REND_BASIC_AUTH_CLIENT_ID_LEN** 4
- #define **REND_BASIC_AUTH_CLIENT_MULTIPLE** 16
- #define **REND_BASIC_AUTH_CLIENT_ENTRY_LEN**
- #define **REND_DESC_MAX_SIZE** (20 ∗ 1024)
- #define **REND_LEGAL_CLIENTNAME_CHARACTERS** "abcdefghijklmnopqrstuvwxyzABCDEFGHIJ
_"
- #define **REND_CLIENTNAME_MAX_LEN** 16
- #define **REND_COOKIE_LEN** DIGEST_LEN
- #define **REND_REPLAY_TIME_INTERVAL** (60 ∗ 60)
- #define **CIRCWINDOW_START** 1000
- #define **CIRCWINDOW_INCREMENT** 100
- #define **STREAMWINDOW_START** 500
- #define **STREAMWINDOW_INCREMENT** 50
- #define **CELL_PADDING** 0
- #define **CELL_CREATE** 1
- #define **CELL_CREATED** 2
- #define **CELL_RELAY** 3
- #define **CELL_DESTROY** 4
- #define **CELL_CREATE_FAST** 5
- #define **CELL_CREATED_FAST** 6
- #define **CELL_VERSIONS** 7
- #define **CELL_NETINFO** 8
- #define **CELL_RELAY_EARLY** 9
- #define **CELL_COMMAND_IS_VAR_LENGTH**(x) ((x) == CELL_VERSIONS)
- #define **TIMEOUT_UNTIL_UNREACHABILITY_COMPLAINT** (20∗60)
- #define **LEGAL_NICKNAME_CHARACTERS** "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
- #define **DEFAULT_CLIENT_NICKNAME** "client"
- #define **SOCKS4_NETWORK_LEN** 8
- #define **CELL_PAYLOAD_SIZE** 509
- #define **CELL_NETWORK_SIZE** 512
- #define **VAR_CELL_HEADER_SIZE** 5
- #define **RELAY_HEADER_SIZE** (1+2+2+4+2)
- #define **RELAY_PAYLOAD_SIZE** (CELL_PAYLOAD_SIZE-RELAY_HEADER_SIZE)
- #define **BASE_CONNECTION_MAGIC** 0x7C3C304Eu
- #define **OR_CONNECTION_MAGIC** 0x7D31FF03u
- #define **EDGE_CONNECTION_MAGIC** 0xF0374013u
- #define **DIR_CONNECTION_MAGIC** 0x9988ffeeu
- #define **CONTROL_CONNECTION_MAGIC** 0x8abc765du
- #define **TO_CONN**(c) (&(((c) → _base)))
- #define **DOWNCAST**(to, ptr) ((to∗)SUBTYPE_P(ptr, to, _base))
- #define **IMPOSSIBLE_TO_DOWNLOAD** 255
- #define **ROUTER_ANNOTATION_BUF_LEN** 256
- #define **ROUTER_PURPOSE_GENERAL** 0
- #define **ROUTER_PURPOSE_CONTROLLER** 1

- #define **ROUTER_PURPOSE_BRIDGE** 2
- #define **ROUTER_PURPOSE_UNKNOWN** 255
- #define **MAX_ROUTERDESC_DOWNLOAD_FAILURES** 8
- #define **USABLE_CONSENSUS_FLAVOR** FLAV_NS
- #define **N_CONSENSUS_FLAVORS** ((int)(FLAV_MICRODESC)+1)
- #define **CRYPT_PATH_MAGIC** 0x70127012u
- #define **CPATH_STATE_CLOSED** 0
- #define **CPATH_STATE_AWAITING_KEYS** 1
- #define **CPATH_STATE_OPEN** 2
- #define **CPATH_KEY_MATERIAL_LEN** (20∗2+16∗2)
- #define **DH_KEY_LEN** DH_BYTES
- #define **ONIONSKIN_CHALLENGE_LEN**
- #define **ONIONSKIN_REPLY_LEN** (DH_KEY_LEN+DIGEST_LEN)
- #define **ORIGIN_CIRCUIT_MAGIC** 0x35315243u
- #define **OR_CIRCUIT_MAGIC** 0x98ABC04Fu
- #define **MAX_RELAY_EARLY_CELLS_PER_CIRCUIT** 8
- #define **REND_TOKEN_LEN** REND_COOKIE_LEN
- #define **TO_CIRCUIT**(x) (&((x) → _base))
- #define **MIN_CONSTRAINED_TCP_BUFFER** 2048
- #define **MAX_CONSTRAINED_TCP_BUFFER** 262144
- #define **LOG_PROTOCOL_WARN**
- #define **MAX_SOCKS_REPLY_LEN** 1024
- #define **MAX_SOCKS_ADDR_LEN** 256
- #define **SOCKS_COMMAND_CONNECT** 0x01
- #define **SOCKS_COMMAND_RESOLVE** 0xF0
- #define **SOCKS_COMMAND_RESOLVE_PTR** 0xF1
- #define **SOCKS_COMMAND_IS_CONNECT**(c) ((c)==SOCKS_COMMAND_CONNECT)
- #define **SOCKS_COMMAND_IS_RESOLVE**(c)
- #define **CBT_NCIRCUITS_TO_OBSERVE** 1000
- #define **CBT_MAX_SYNTHETIC_QUANTILE** 0.985
- #define **CBT_BIN_WIDTH** ((**build_time_t**)50)
- #define **CBT_BUILD_TIME_MAX** ((**build_time_t**)(INT32_MAX))
- #define **CBT_SAVE_STATE_EVERY** 10
- #define **CBT_NETWORK_NONLIVE_TIMEOUT_COUNT** (3)
- #define **CBT_NETWORK_NONLIVE_DISCARD_COUNT** (CBT_NETWORK_NONLIVE_TIMEOUT_COUNT∗2)
- #define **CBT_DEFAULT_RECENT_CIRCUITS** 20
- #define **CBT_DEFAULT_MAX_RECENT_TIMEOUT_COUNT** (CBT_DEFAULT_RECENT_CIRCUITS∗9/10)
- #define **CBT_DEFAULT_MIN_CIRCUITS_TO_OBSERVE** 100
- #define **CBT_DEFAULT_QUANTILE_CUTOFF** 80
- #define **CBT_DEFAULT_TEST_FREQUENCY** 60
- #define **CBT_DEFAULT_TIMEOUT_MIN_VALUE** (2∗1000)
- #define **CBT_DEFAULT_TIMEOUT_INITIAL_VALUE** (60∗1000)
- #define **circuit_mark_for_close**(c, reason) _circuit_mark_for_close((c), (reason), __LINE__, _SHORT_FILE_)
- #define **CIRCLAUNCH_ONEHOP_TUNNEL** (1<<0)
- #define **CIRCLAUNCH_NEED_UPTIME** (1<<1)
- #define **CIRCLAUNCH_NEED_CAPACITY** (1<<2)

- #define **CIRCLAUNCH_IS_INTERNAL** (1<<3)
- #define **get_datadir_fname2_suffix**(sub1, sub2, suffix) options_get_datadir_fname2_suffix(get_options(), (sub1), (sub2), (suffix))
- #define **get_datadir_fname**(sub1) get_datadir_fname2_suffix((sub1), NULL, NULL)
- #define **get_datadir_fname2**(sub1, sub2) get_datadir_fname2_suffix((sub1), (sub2), NULL)
- #define **get_datadir_fname_suffix**(sub1, suffix) get_datadir_fname2_suffix((sub1), NULL, (suffix))
- #define **connection_mark_for_close**(c) _connection_mark_for_close((c), __LINE__, _SHORT_FILE_)
- #define **connection_speaks_cells**(conn) ((conn) → type == CONN_TYPE_OR)
- #define **connection_mark_unattached_ap**(conn, endreason) _connection_mark_unattached_ap((conn), (endreason), __LINE__, _SHORT_FILE_)
- #define **CONN_LOG_PROTECT**(conn, stmt)
- #define **LOG_FN_CONN**(conn, args) CONN_LOG_PROTECT(conn, log_fn args)
- #define **EVENT_AUTHDIR_NEWDESCS** 0x000D
- #define **EVENT_NS** 0x000F
- #define **DSR_HEX** (1<<0)
- #define **DSR_BASE64** (1<<1)
- #define **DSR_DIGEST256** (1<<2)
- #define **DSR_SORT_UNIQ** (1<<3)
- #define **download_status_failed**(dls, sc)
- #define **MAX_EXITPOLICY_SUMMARY_LEN** (1000)
- #define **MAX_V_LINE_LEN** 128
- #define **MAX_FLAG_LINE_LEN** 96
- #define **MAX_WEIGHT_LINE_LEN** (12+10+10+10+1)
- #define **MAX_POLICY_LINE_LEN** (3+MAX_EXITPOLICY_SUMMARY_LEN)
- #define **RS_ENTRY_LEN**
- #define **UNNAMED_ROUTER_NICKNAME** "Unnamed"
- #define **directory_caches_v1_dir_info**(o) directory_caches_v2_dir_info(o)
- #define **MIN_VOTE_SECONDS** 20
- #define **MIN_DIST_SECONDS** 20
- #define **MIN_VOTE_INTERVAL** 300
- #define **BW_WEIGHT_SCALE** 10000
- #define **DGV_BY_ID** 1
- #define **DGV_INCLUDE_PENDING** 2
- #define **DGV_INCLUDE_PREVIOUS** 4
- #define **DIR_RECORD_USAGE_GRANULARITY** 8
- #define **DIR_ENTRY_RECORD_USAGE_RETAIN_IPS** (24∗60∗60)
- #define **DIR_RECORD_USAGE_MIN_OBSERVATION_TIME** (12∗60∗60)
- #define **GEOIP_NS_RESPONSE_NUM** 6
- #define **MAX_NETWORKSTATUS_AGE** (10∗24∗60∗60)
- #define **NSSET_FROM_CACHE** 1
- #define **NSSET_WAS_WAITING_FOR_CERTS** 2
- #define **NSSET_DONT_DOWNLOAD_CERTS** 4
- #define **NSSET_ACCEPT_OBSOLETE** 8
- #define **NSSET_REQUIRE_FLAVOR** 16
- #define **nt_service_is_stopping**() (0)
- #define **POLICY_BUF_LEN** 52
- #define **ROUTER_REQUIRED_MIN_BANDWIDTH** (20∗1024)

- #define **ROUTER_MAX_DECLARED_BANDWIDTH** INT32_MAX
- #define **PDS_ALLOW_SELF** (1<<0)
- #define **PDS_RETRY_IF_NO_SERVERS** (1<<1)
- #define **PDS_IGNORE_FASCISTFIREWALL** (1<<2)
- #define **PDS_NO_EXISTING_SERVERDESC_FETCH** (1<<3)
- #define **_PDS_PREFER_TUNNELED_DIR_CONNS** (1<<16)
- #define **router_digest_is_trusted_dir**(d) router_digest_is_trusted_dir_type((d), NO_AUTHORITY)
- #define **MAX_STATUS_TAG_LEN** 32

Typedefs

- typedef uint16_t **circid_t**
- typedef uint16_t **streamid_t**
- typedef struct **buf_t** **buf_t**
- typedef struct **socks_request_t** **socks_request_t**
- typedef int16_t **country_t**
- typedef struct **routerset_t** **routerset_t**
- typedef uint32_t **build_time_t**
- typedef enum watchable_events **watchable_events_t**
- typedef struct **microdesc_cache_t** **microdesc_cache_t**

Enumerations

- enum **circ_id_type_t** { **CIRC_ID_TYPE_LOWER** = 0, **CIRC_ID_TYPE_HIGHER** = 1, **CIRC_ID_TYPE_NEITHER** = 2 }
- enum **rend_auth_type_t** { **REND_NO_AUTH** = 0, **REND_BASIC_AUTH** = 1, **REND_STEALTH_AUTH** = 2 }
- enum **cell_direction_t** { **CELL_DIRECTION_IN** = 1, **CELL_DIRECTION_OUT** = 2 }
- enum **addr_policy_action_t** { **ADDR_POLICY_ACCEPT** = 1, **ADDR_POLICY_REJECT** = 2 }
- enum **saved_location_t** { **SAVED_NOWHERE** = 0, **SAVED_IN_CACHE**, **SAVED_IN_JOURNAL** }
- enum **download_schedule_t** { **DL_SCHED_GENERIC** = 0, **DL_SCHED_CONSENSUS** = 1, **DL_SCHED_BRIDGE** = 2 }
- enum **networkstatus_type_t** { **NS_TYPE_VOTE**, **NS_TYPE_CONSENSUS**, **NS_TYPE_OPINION** }
- enum **consensus_flavor_t** { **FLAV_NS** = 0, **FLAV_MICRODESC** = 1 }
- enum **store_type_t** { **ROUTER_STORE** = 0, **EXTRAINFO_STORE** = 1 }
- enum **authority_type_t** {

  NO_AUTHORITY = 0, **V1_AUTHORITY** = 1 << 0, **V2_AUTHORITY** = 1 << 1, **V3_AUTHORITY** = 1 << 2,

  HIDSERV_AUTHORITY = 1 << 3, **BRIDGE_AUTHORITY** = 1 << 4, **EXTRAINFO_CACHE** = 1 << 5 }
- enum **invalid_router_usage_t** {

  ALLOW_INVALID_ENTRY = 1, **ALLOW_INVALID_EXIT** = 2, **ALLOW_INVALID_MIDDLE** = 4, **ALLOW_INVALID_RENDEZVOUS** = 8,

  ALLOW_INVALID_INTRODUCTION = 16 }

- enum **setopt_err_t** {

  SETOPT_OK = 0, **SETOPT_ERR_MISC** = -1, **SETOPT_ERR_PARSE** = -2, **SETOPT_ERR_TRANSITION** = -3,

  SETOPT_ERR_SETTING = -4 }
- enum **addressmap_entry_source_t** { **ADDRMAPSRC_CONTROLLER**, **ADDRMAPSRC_TORRC**, **ADDRMAPSRC_TRACKEXIT**, **ADDRMAPSRC_DNS** }
- enum **hostname_type_t** { **NORMAL_HOSTNAME**, **ONION_HOSTNAME**, **EXIT_HOSTNAME**, **BAD_HOSTNAME** }
- enum **circuit_status_event_t** {

  CIRC_EVENT_LAUNCHED = 0, **CIRC_EVENT_BUILT** = 1, **CIRC_EVENT_EXTENDED** = 2, **CIRC_EVENT_FAILED** = 3,

  CIRC_EVENT_CLOSED = 4 }
- enum **stream_status_event_t** {

  STREAM_EVENT_SENT_CONNECT = 0, **STREAM_EVENT_SENT_RESOLVE** = 1, **STREAM_EVENT_SUCCEEDED** = 2, **STREAM_EVENT_FAILED** = 3,

  STREAM_EVENT_CLOSED = 4, **STREAM_EVENT_NEW** = 5, **STREAM_EVENT_NEW_RESOLVE** = 6, **STREAM_EVENT_FAILED_RETRIABLE** = 7,

  STREAM_EVENT_REMAP = 8 }
- enum **or_conn_status_event_t** {

  OR_CONN_EVENT_LAUNCHED = 0, **OR_CONN_EVENT_CONNECTED** = 1, **OR_CONN_EVENT_FAILED** = 2, **OR_CONN_EVENT_CLOSED** = 3,

  OR_CONN_EVENT_NEW = 4 }
- enum **buildtimeout_set_event_t** {

  BUILDTIMEOUT_SET_EVENT_COMPUTED = 0, **BUILDTIMEOUT_SET_EVENT_RESET** = 1, **BUILDTIMEOUT_SET_EVENT_SUSPENDED** = 2, **BUILDTIMEOUT_SET_EVENT_DISCARD** = 3,

  BUILDTIMEOUT_SET_EVENT_RESUME = 4 }
- enum **bootstrap_status_t** {

  BOOTSTRAP_STATUS_UNDEF = -1, **BOOTSTRAP_STATUS_STARTING** = 0, **BOOTSTRAP_STATUS_CONN_DIR** = 5, **BOOTSTRAP_STATUS_HANDSHAKE** = -2,

  BOOTSTRAP_STATUS_HANDSHAKE_DIR = 10, **BOOTSTRAP_STATUS_ONEHOP_CREATE** = 15, **BOOTSTRAP_STATUS_REQUESTING_STATUS** = 20, **BOOTSTRAP_STATUS_LOADING_STATUS** = 25,

  BOOTSTRAP_STATUS_LOADING_KEYS = 40, **BOOTSTRAP_STATUS_REQUESTING_DESCRIPTORS** = 45, **BOOTSTRAP_STATUS_LOADING_DESCRIPTORS** = 50, **BOOTSTRAP_STATUS_CONN_OR** = 80,

  BOOTSTRAP_STATUS_HANDSHAKE_OR = 85, **BOOTSTRAP_STATUS_CIRCUIT_CREATE** = 90, **BOOTSTRAP_STATUS_DONE** = 100 }
- enum **routerstatus_format_type_t** {

  NS_V2, **NS_V3_CONSENSUS**, **NS_V3_VOTE**, **NS_CONTROL_PORT**,

  NS_V3_CONSENSUS_MICRODESC }

- enum **geoip_client_action_t** { **GEOIP_CLIENT_CONNECT** = 0, **GEOIP_CLIENT_NETWORKSTATUS** = 1, **GEOIP_CLIENT_NETWORKSTATUS_V2** = 2 }
- enum **geoip_ns_response_t** {

  GEOIP_SUCCESS = 0, **GEOIP_REJECT_NOT_ENOUGH_SIGS** = 1, **GEOIP_REJECT_UNAVAILABLE** = 2, **GEOIP_REJECT_NOT_FOUND** = 3,

  GEOIP_REJECT_NOT_MODIFIED = 4, **GEOIP_REJECT_BUSY** = 5 }
- enum **dirreq_type_t** { **DIRREQ_DIRECT** = 0, **DIRREQ_TUNNELED** = 1 }
- enum **dirreq_state_t** {

  DIRREQ_IS_FOR_NETWORK_STATUS = 0, **DIRREQ_FLUSHING_DIR_CONN_FINISHED** = 1, **DIRREQ_END_CELL_SENT** = 2, **DIRREQ_CIRC_QUEUE_FLUSHED** = 3,

  DIRREQ_OR_CONN_BUFFER_FLUSHED = 4 }
- enum **watchable_events** { **READ_EVENT** = 0x02, **WRITE_EVENT** = 0x04 }
- enum **v2_networkstatus_source_t** { **NS_FROM_CACHE**, **NS_FROM_DIR_BY_FP**, **NS_FROM_DIR_ALL**, **NS_GENERATED** }
- enum **version_status_t** {

  VS_RECOMMENDED = 0, **VS_OLD** = 1, **VS_NEW** = 2, **VS_NEW_IN_SERIES** = 3,

  VS_UNRECOMMENDED = 4, **VS_EMPTY** = 5, **VS_UNKNOWN** }
- enum **addr_policy_result_t** { **ADDR_POLICY_ACCEPTED** = 0, **ADDR_POLICY_REJECTED** = -1, **ADDR_POLICY_PROBABLY_ACCEPTED** = 1, **ADDR_POLICY_PROBABLY_REJECTED** = 2 }
- enum **pk_op_t** {

  SIGN_DIR, **SIGN_RTR**, **VERIFY_DIR**, **VERIFY_RTR**,

  ENC_ONIONSKIN, **DEC_ONIONSKIN**, **TLS_HANDSHAKE_C**, **TLS_HANDSHAKE_S**,

  REND_CLIENT, **REND_MID**, **REND_SERVER** }
- enum **bandwidth_weight_rule_t** {

  NO_WEIGHTING, **WEIGHT_FOR_EXIT**, **WEIGHT_FOR_MID**, **WEIGHT_FOR_GUARD**,

  WEIGHT_FOR_DIR }
- enum **router_crn_flags_t** {

  CRN_NEED_UPTIME = 1<<0, **CRN_NEED_CAPACITY** = 1<<1, **CRN_NEED_GUARD** = 1<<2, **CRN_ALLOW_INVALID** = 1<<3,

  CRN_WEIGHT_AS_EXIT = 1<<5 }
- enum **was_router_added_t** {

  ROUTER_ADDED_SUCCESSFULLY = 1, **ROUTER_ADDED_NOTIFY_GENERATOR** = 0, **ROUTER_BAD_EI** = -1, **ROUTER_WAS_NOT_NEW** = -2,

  ROUTER_NOT_IN_CONSENSUS = -3, **ROUTER_NOT_IN_CONSENSUS_OR_NETWORKSTATUS** = -4, **ROUTER_AUTHDIR_REJECTS** = -5 }

Functions

- static **or_connection_t** ∗ **TO_OR_CONN** (**connection_t** ∗)
- static **dir_connection_t** ∗ **TO_DIR_CONN** (**connection_t** ∗)
- static **edge_connection_t** ∗ **TO_EDGE_CONN** (**connection_t** ∗)
- static **control_connection_t** ∗ **TO_CONTROL_CONN** (**connection_t** ∗)
- static **or_circuit_t** ∗ **TO_OR_CIRCUIT** (**circuit_t** ∗)
- static **origin_circuit_t** ∗ **TO_ORIGIN_CIRCUIT** (**circuit_t** ∗)
- static INLINE void **or_state_mark_dirty** (**or_state_t** ∗state, time_t when)
- **buf_t** ∗ **buf_new** (void)
- **buf_t** ∗ **buf_new_with_capacity** (size_t size)
- void **buf_free** (**buf_t** ∗buf)
- void **buf_clear** (**buf_t** ∗buf)
- void **buf_shrink** (**buf_t** ∗buf)
- void **buf_shrink_freelists** (int free_all)
- void **buf_dump_freelist_sizes** (int severity)
- size_t **buf_datalen** (const **buf_t** ∗buf)
- size_t **buf_allocation** (const **buf_t** ∗buf)
- size_t **buf_slack** (const **buf_t** ∗buf)
- const char ∗ **_buf_peek_raw_buffer** (const **buf_t** ∗buf)
- int **read_to_buf** (int s, size_t at_most, **buf_t** ∗buf, int ∗reached_eof, int ∗socket_error)
- int **read_to_buf_tls** (**tor_tls_t** ∗tls, size_t at_most, **buf_t** ∗buf)
- int **flush_buf** (int s, **buf_t** ∗buf, size_t sz, size_t ∗buf_flushlen)
- int **flush_buf_tls** (**tor_tls_t** ∗tls, **buf_t** ∗buf, size_t sz, size_t ∗buf_flushlen)
- int **write_to_buf** (const char ∗string, size_t string_len, **buf_t** ∗buf)
- int **write_to_buf_zlib** (**buf_t** ∗buf, **tor_zlib_state_t** ∗state, const char ∗data, size_t data_len, int done)
- int **move_buf_to_buf** (**buf_t** ∗buf_out, **buf_t** ∗buf_in, size_t ∗buf_flushlen)
- int **fetch_from_buf** (char ∗string, size_t string_len, **buf_t** ∗buf)
- int **fetch_var_cell_from_buf** (**buf_t** ∗buf, **var_cell_t** ∗∗out, int linkproto)
- int **fetch_from_buf_http** (**buf_t** ∗buf, char ∗∗headers_out, size_t max_headerlen, char ∗∗body_out, size_t ∗body_used, size_t max_bodylen, int force_complete)
- int **fetch_from_buf_socks** (**buf_t** ∗buf, **socks_request_t** ∗req, int log_sockstype, int safe_socks)
- int **fetch_from_buf_socks_client** (**buf_t** ∗buf, int state, char ∗∗reason)
- int **fetch_from_buf_line** (**buf_t** ∗buf, char ∗data_out, size_t ∗data_len)
- int **peek_buf_has_control0_command** (**buf_t** ∗buf)
- void **assert_buf_ok** (**buf_t** ∗buf)
- char ∗ **circuit_list_path** (**origin_circuit_t** ∗circ, int verbose)
- char ∗ **circuit_list_path_for_controller** (**origin_circuit_t** ∗circ)
- void **circuit_log_path** (int severity, unsigned int domain, **origin_circuit_t** ∗circ)
- void **circuit_rep_hist_note_result** (**origin_circuit_t** ∗circ)
- **origin_circuit_t** ∗ **origin_circuit_init** (uint8_t purpose, int flags)
- **origin_circuit_t** ∗ **circuit_establish_circuit** (uint8_t purpose, **extend_info_t** ∗exit, int flags)
- int **circuit_handle_first_hop** (**origin_circuit_t** ∗circ)
- void **circuit_n_conn_done** (**or_connection_t** ∗or_conn, int status)
- int **inform_testing_reachability** (void)
- int **circuit_send_next_onion_skin** (**origin_circuit_t** ∗circ)
- void **circuit_note_clock_jumped** (int seconds_elapsed)
- int **circuit_extend** (**cell_t** ∗cell, **circuit_t** ∗circ)

- int **circuit_init_cpath_crypto** (**crypt_path_t** *cpath, const char *key_data, int reverse)
- int **circuit_finish_handshake** (**origin_circuit_t** *circ, uint8_t cell_type, const char *reply)
- int **circuit_truncated** (**origin_circuit_t** *circ, **crypt_path_t** *layer)
- int **onionskin_answer** (**or_circuit_t** *circ, uint8_t cell_type, const char *payload, const char *keys)
- int **circuit_all_predicted_ports_handled** (time_t now, int *need_uptime, int *need_capacity)
- int **circuit_append_new_exit** (**origin_circuit_t** *circ, **extend_info_t** *info)
- int **circuit_extend_to_new_exit** (**origin_circuit_t** *circ, **extend_info_t** *info)
- void **onion_append_to_cpath** (**crypt_path_t** **head_ptr, **crypt_path_t** *new_hop)
- **extend_info_t** * **extend_info_alloc** (const char *nickname, const char *digest, **crypto_pk_env_t** *onion_key, const **tor_addr_t** *addr, uint16_t port)
- **extend_info_t** * **extend_info_from_router** (**routerinfo_t** *r)
- **extend_info_t** * **extend_info_dup** (**extend_info_t** *info)
- void **extend_info_free** (**extend_info_t** *info)
- **routerinfo_t** * **build_state_get_exit_router** (**cpath_build_state_t** *state)
- const char * **build_state_get_exit_nickname** (**cpath_build_state_t** *state)
- void **entry_guards_compute_status** (void)
- int **entry_guard_register_connect_status** (const char *digest, int succeeded, int mark_relay_status, time_t now)
- void **entry_nodes_should_be_added** (void)
- int **entry_list_is_constrained** (**or_options_t** *options)
- **routerinfo_t** * **choose_random_entry** (**cpath_build_state_t** *state)
- int **entry_guards_parse_state** (**or_state_t** *state, int set, char **msg)
- void **entry_guards_update_state** (**or_state_t** *state)
- int **getinfo_helper_entry_guards** (**control_connection_t** *conn, const char *question, char **answer)
- void **clear_bridge_list** (void)
- int **routerinfo_is_a_configured_bridge** (**routerinfo_t** *ri)
- void **bridge_add_from_config** (const **tor_addr_t** *addr, uint16_t port, char *digest)
- void **retry_bridge_descriptor_fetch_directly** (const char *digest)
- void **fetch_bridge_descriptors** (time_t now)
- void **learned_bridge_descriptor** (**routerinfo_t** *ri, int from_cache)
- int **any_bridge_descriptors_known** (void)
- int **any_pending_bridge_descriptor_fetches** (void)
- int **bridges_known_but_down** (void)
- void **bridges_retry_all** (void)
- void **entry_guards_free_all** (void)
- double **circuit_build_times_quantile_cutoff** (void)
- int32_t **circuit_build_times_initial_timeout** (void)
- void **circuit_build_times_update_state** (**circuit_build_times_t** *cbt, **or_state_t** *state)
- int **circuit_build_times_parse_state** (**circuit_build_times_t** *cbt, **or_state_t** *state, char **msg)
- int **circuit_build_times_add_timeout** (**circuit_build_times_t** *cbt, int did_onehop, time_t start_time)
- void **circuit_build_times_set_timeout** (**circuit_build_times_t** *cbt)

- int **circuit_build_times_add_time** (**circuit_build_times_t** *cbt, **build_time_t** time)
- int **circuit_build_times_needs_circuits** (**circuit_build_times_t** *cbt)
- int **circuit_build_times_needs_circuits_now** (**circuit_build_times_t** *cbt)
- void **circuit_build_times_init** (**circuit_build_times_t** *cbt)
- void **circuit_build_times_new_consensus_params** (**circuit_build_times_t** *cbt, **networkstatus_t** *ns)
- void **circuit_build_times_network_is_live** (**circuit_build_times_t** *cbt)
- int **circuit_build_times_network_check_live** (**circuit_build_times_t** *cbt)
- void **circuit_build_times_network_circ_success** (**circuit_build_times_t** *cbt)
- **circuit_t** * **_circuit_get_global_list** (void)
- const char * **circuit_state_to_string** (int state)
- const char * **circuit_purpose_to_controller_string** (uint8_t purpose)
- void **circuit_dump_by_conn** (**connection_t** *conn, int severity)
- void **circuit_set_p_circid_orconn** (**or_circuit_t** *circ, **circid_t** id, **or_connection_t** *conn)
- void **circuit_set_n_circid_orconn** (**circuit_t** *circ, **circid_t** id, **or_connection_t** *conn)
- void **circuit_set_state** (**circuit_t** *circ, uint8_t state)
- void **circuit_close_all_marked** (void)
- int32_t **circuit_initial_package_window** (void)
- **origin_circuit_t** * **origin_circuit_new** (void)
- **or_circuit_t** * **or_circuit_new** (**circid_t** p_circ_id, **or_connection_t** *p_conn)
- **circuit_t** * **circuit_get_by_circid_orconn** (**circid_t** circ_id, **or_connection_t** *conn)
- int **circuit_id_in_use_on_orconn** (**circid_t** circ_id, **or_connection_t** *conn)
- **circuit_t** * **circuit_get_by_edge_conn** (**edge_connection_t** *conn)
- void **circuit_unlink_all_from_or_conn** (**or_connection_t** *conn, int reason)
- **origin_circuit_t** * **circuit_get_by_global_id** (uint32_t id)
- **origin_circuit_t** * **circuit_get_by_rend_query_and_purpose** (const char *rend_query, uint8_t purpose)
- **origin_circuit_t** * **circuit_get_next_by_pk_and_purpose** (**origin_circuit_t** *start, const char *digest, uint8_t purpose)
- **or_circuit_t** * **circuit_get_rendezvous** (const char *cookie)
- **or_circuit_t** * **circuit_get_intro_point** (const char *digest)
- **origin_circuit_t** * **circuit_find_to_cannibalize** (uint8_t purpose, **extend_info_t** *info, int flags)
- void **circuit_mark_all_unused_circs** (void)
- void **circuit_expire_all_dirty_circs** (void)
- void **_circuit_mark_for_close** (**circuit_t** *circ, int reason, int line, const char *file)
- int **circuit_get_cpath_len** (**origin_circuit_t** *circ)
- **crypt_path_t** * **circuit_get_cpath_hop** (**origin_circuit_t** *circ, int hopnum)
- void **circuit_get_all_pending_on_or_conn** (**smartlist_t** *out, **or_connection_t** *or_conn)
- int **circuit_count_pending_on_or_conn** (**or_connection_t** *or_conn)
- void **assert_cpath_layer_ok** (const **crypt_path_t** *cp)
- void **assert_circuit_ok** (const **circuit_t** *c)
- void **circuit_free_all** (void)
- void **circuit_expire_building** (time_t now)
- void **circuit_remove_handled_ports** (**smartlist_t** *needed_ports)

- int **circuit_stream_is_being_handled** (**edge_connection_t** ∗conn, uint16_t port, int min)
- int **circuit_conforms_to_options** (const **origin_circuit_t** ∗circ, const **or_options_t** ∗options)
- void **circuit_build_needed_circs** (time_t now)
- void **circuit_detach_stream** (**circuit_t** ∗circ, **edge_connection_t** ∗conn)
- void **circuit_expire_old_circuits_serverside** (time_t now)
- void **reset_bandwidth_test** (void)
- int **circuit_enough_testing_circs** (void)
- void **circuit_has_opened** (**origin_circuit_t** ∗circ)
- void **circuit_build_failed** (**origin_circuit_t** ∗circ)
- **origin_circuit_t** ∗ **circuit_launch_by_extend_info** (uint8_t purpose, **extend_info_t** ∗info, int flags)
- **origin_circuit_t** ∗ **circuit_launch_by_router** (uint8_t purpose, **routerinfo_t** ∗exit, int flags)
- void **circuit_reset_failure_count** (int timeout)
- int **connection_ap_handshake_attach_chosen_circuit** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ, **crypt_path_t** ∗cpath)
- int **connection_ap_handshake_attach_circuit** (**edge_connection_t** ∗conn)
- void **command_process_cell** (**cell_t** ∗cell, **or_connection_t** ∗conn)
- void **command_process_var_cell** (**var_cell_t** ∗cell, **or_connection_t** ∗conn)
- const char ∗ **get_dirportfrontpage** (void)
- **or_options_t** ∗ **get_options** (void)
- int **set_options** (**or_options_t** ∗new_val, char ∗∗msg)
- void **config_free_all** (void)
- const char ∗ **safe_str_client** (const char ∗address)
- const char ∗ **safe_str** (const char ∗address)
- const char ∗ **escaped_safe_str_client** (const char ∗address)
- const char ∗ **escaped_safe_str** (const char ∗address)
- const char ∗ **get_version** (void)
- int **config_get_lines** (const char ∗string, **config_line_t** ∗∗result)
- void **config_free_lines** (**config_line_t** ∗front)
- **setopt_err_t** **options_trial_assign** (**config_line_t** ∗list, int use_defaults, int clear_first, char ∗∗msg)
- int **resolve_my_address** (int warn_severity, **or_options_t** ∗options, uint32_t ∗addr, char ∗∗hostname_out)
- int **is_local_addr** (const **tor_addr_t** ∗addr) ATTR_PURE
- void **options_init** (**or_options_t** ∗options)
- char ∗ **options_dump** (**or_options_t** ∗options, int minimal)
- int **options_init_from_torrc** (int argc, char ∗∗argv)
- **setopt_err_t** **options_init_from_string** (const char ∗cf, int command, const char ∗command_arg, char ∗∗msg)
- int **option_is_recognized** (const char ∗key)
- const char ∗ **option_get_canonical_name** (const char ∗key)
- **config_line_t** ∗ **option_get_assignment** (**or_options_t** ∗options, const char ∗key)
- int **options_save_current** (void)
- const char ∗ **get_torrc_fname** (void)
- char ∗ **options_get_datadir_fname2_suffix** (**or_options_t** ∗options, const char ∗sub1, const char ∗sub2, const char ∗suffix)
- **or_state_t** ∗ **get_or_state** (void)
- int **or_state_save** (time_t now)

- int **options_need_geoip_info** (**or_options_t** ∗options, const char ∗∗reason_out)
- int **getinfo_helper_config** (**control_connection_t** ∗conn, const char ∗question, char ∗∗answer)
- const char ∗ **tor_get_digests** (void)
- uint32_t **get_effective_bwrate** (**or_options_t** ∗options)
- uint32_t **get_effective_bwburst** (**or_options_t** ∗options)
- const char ∗ **conn_type_to_string** (int type)
- const char ∗ **conn_state_to_string** (int type, int state)
- **dir_connection_t** ∗ **dir_connection_new** (int socket_family)
- **or_connection_t** ∗ **or_connection_new** (int socket_family)
- **edge_connection_t** ∗ **edge_connection_new** (int type, int socket_family)
- **control_connection_t** ∗ **control_connection_new** (int socket_family)
- **connection_t** ∗ **connection_new** (int type, int socket_family)
- void **connection_link_connections** (**connection_t** ∗conn_a, **connection_t** ∗conn_b)
- void **connection_free** (**connection_t** ∗conn)
- void **connection_free_all** (void)
- void **connection_about_to_close_connection** (**connection_t** ∗conn)
- void **connection_close_immediate** (**connection_t** ∗conn)
- void **_connection_mark_for_close** (**connection_t** ∗conn, int line, const char ∗file)
- void **connection_expire_held_open** (void)
- int **connection_connect** (**connection_t** ∗conn, const char ∗address, const **tor_addr_t** ∗addr, uint16_t port, int ∗socket_error)
- int **connection_proxy_connect** (**connection_t** ∗conn, int type)
- int **connection_read_proxy_handshake** (**connection_t** ∗conn)
- int **retry_all_listeners** (**smartlist_t** ∗replaced_conns, **smartlist_t** ∗new_conns)
- ssize_t **connection_bucket_write_limit** (**connection_t** ∗conn, time_t now)
- int **global_write_bucket_low** (**connection_t** ∗conn, size_t attempt, int priority)
- void **connection_bucket_init** (void)
- void **connection_bucket_refill** (int seconds_elapsed, time_t now)
- int **connection_handle_read** (**connection_t** ∗conn)
- int **connection_fetch_from_buf** (char ∗string, size_t len, **connection_t** ∗conn)
- int **connection_wants_to_flush** (**connection_t** ∗conn)
- int **connection_outbuf_too_full** (**connection_t** ∗conn)
- int **connection_handle_write** (**connection_t** ∗conn, int force)
- void **_connection_write_to_buf_impl** (const char ∗string, size_t len, **connection_t** ∗conn, int zlib)
- static void **connection_write_to_buf** (const char ∗string, size_t len, **connection_t** ∗conn)
- static void **connection_write_to_buf_zlib** (const char ∗string, size_t len, **dir_connection_t** ∗conn, int done)
- **connection_t** ∗ **connection_get_by_global_id** (uint64_t id)
- **connection_t** ∗ **connection_get_by_type** (int type)
- **connection_t** ∗ **connection_get_by_type_purpose** (int type, int purpose)
- **connection_t** ∗ **connection_get_by_type_addr_port_purpose** (int type, const **tor_addr_t** ∗addr, uint16_t port, int purpose)
- **connection_t** ∗ **connection_get_by_type_state** (int type, int state)
- **connection_t** ∗ **connection_get_by_type_state_rendquery** (int type, int state, const char ∗rendquery)
- int **connection_is_listener** (**connection_t** ∗conn)
- int **connection_state_is_open** (**connection_t** ∗conn)

- int **connection_state_is_connecting** (**connection_t** ∗conn)
- char ∗ **alloc_http_authenticator** (const char ∗authenticator)
- void **assert_connection_ok** (**connection_t** ∗conn, time_t now)
- int **connection_or_nonopen_was_started_here** (**or_connection_t** ∗conn)
- void **connection_dump_buffer_mem_stats** (int severity)
- void **remove_file_if_very_old** (const char ∗fname, time_t now)
- void **_connection_mark_unattached_ap** (**edge_connection_t** ∗conn, int endreason, int line, const char ∗file)
- int **connection_edge_reached_eof** (**edge_connection_t** ∗conn)
- int **connection_edge_process_inbuf** (**edge_connection_t** ∗conn, int package_partial)
- int **connection_edge_destroy** (**circid_t** circ_id, **edge_connection_t** ∗conn)
- int **connection_edge_end** (**edge_connection_t** ∗conn, uint8_t reason)
- int **connection_edge_end_errno** (**edge_connection_t** ∗conn)
- int **connection_edge_finished_flushing** (**edge_connection_t** ∗conn)
- int **connection_edge_finished_connecting** (**edge_connection_t** ∗conn)
- int **connection_ap_handshake_send_begin** (**edge_connection_t** ∗ap_conn)
- int **connection_ap_handshake_send_resolve** (**edge_connection_t** ∗ap_conn)
- **edge_connection_t** ∗ **connection_ap_make_link** (char ∗address, uint16_t port, const char ∗digest, int use_begindir, int want_onehop)
- void **connection_ap_handshake_socks_reply** (**edge_connection_t** ∗conn, char ∗reply, size_t replylen, int endreason)
- void **connection_ap_handshake_socks_resolved** (**edge_connection_t** ∗conn, int answer_type, size_t answer_len, const char ∗answer, int ttl, time_t expires)
- int **connection_exit_begin_conn** (**cell_t** ∗cell, **circuit_t** ∗circ)
- int **connection_exit_begin_resolve** (**cell_t** ∗cell, **or_circuit_t** ∗circ)
- void **connection_exit_connect** (**edge_connection_t** ∗conn)
- int **connection_edge_is_rendezvous_stream** (**edge_connection_t** ∗conn)
- int **connection_ap_can_use_exit** (**edge_connection_t** ∗conn, **routerinfo_t** ∗exit, int excluded_means_no)
- void **connection_ap_expire_beginning** (void)
- void **connection_ap_attach_pending** (void)
- void **connection_ap_fail_onehop** (const char ∗failed_digest, **cpath_build_state_t** ∗build_state)
- void **circuit_discard_optional_exit_enclaves** (**extend_info_t** ∗info)
- int **connection_ap_detach_retriable** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ, int reason)
- int **connection_ap_process_transparent** (**edge_connection_t** ∗conn)
- int **address_is_invalid_destination** (const char ∗address, int client)
- void **addressmap_init** (void)
- void **addressmap_clean** (time_t now)
- void **addressmap_clear_configured** (void)
- void **addressmap_clear_transient** (void)
- void **addressmap_free_all** (void)
- int **addressmap_rewrite** (char ∗address, size_t maxlen, time_t ∗expires_out)
- int **addressmap_have_mapping** (const char ∗address, int update_timeout)
- void **addressmap_register** (const char ∗address, char ∗new_address, time_t expires, **addressmap_entry_source_t** source)
- int **parse_virtual_addr_network** (const char ∗val, int validate_only, char ∗∗msg)
- int **client_dns_incr_failures** (const char ∗address)
- void **client_dns_clear_failures** (const char ∗address)

- void **client_dns_set_addressmap** (const char *address, uint32_t val, const char *exitname, int ttl)
- const char * **addressmap_register_virtual_address** (int type, char *new_address)
- void **addressmap_get_mappings** (**smartlist_t** *sl, time_t min_expires, time_t max_expires, int want_expiry)
- int **connection_ap_handshake_rewrite_and_attach** (**edge_connection_t** *conn, **origin_circuit_t** *circ, **crypt_path_t** *cpath)
- int **hostname_is_noconnect_address** (const char *address)
- **hostname_type_t** **parse_extended_hostname** (char *address, int allowdotexit)
- void **connection_or_remove_from_identity_map** (**or_connection_t** *conn)
- void **connection_or_clear_identity_map** (void)
- **or_connection_t** * **connection_or_get_for_extend** (const char *digest, const **tor_addr_t** *target_addr, const char **msg_out, int *launch_out)
- void **connection_or_set_bad_connections** (void)
- int **connection_or_reached_eof** (**or_connection_t** *conn)
- int **connection_or_process_inbuf** (**or_connection_t** *conn)
- int **connection_or_flushed_some** (**or_connection_t** *conn)
- int **connection_or_finished_flushing** (**or_connection_t** *conn)
- int **connection_or_finished_connecting** (**or_connection_t** *conn)
- int **connection_or_digest_is_known_relay** (const char *id_digest)
- void **connection_or_connect_failed** (**or_connection_t** *conn, int reason, const char *msg)
- **or_connection_t** * **connection_or_connect** (const **tor_addr_t** *addr, uint16_t port, const char *id_digest)
- int **connection_tls_start_handshake** (**or_connection_t** *conn, int receiving)
- int **connection_tls_continue_handshake** (**or_connection_t** *conn)
- void **or_handshake_state_free** (**or_handshake_state_t** *state)
- int **connection_or_set_state_open** (**or_connection_t** *conn)
- void **connection_or_write_cell_to_buf** (const **cell_t** *cell, **or_connection_t** *conn)
- void **connection_or_write_var_cell_to_buf** (const **var_cell_t** *cell, **or_connection_t** *conn)
- int **connection_or_send_destroy** (**circid_t** circ_id, **or_connection_t** *conn, int reason)
- int **connection_or_send_netinfo** (**or_connection_t** *conn)
- int **connection_or_send_cert** (**or_connection_t** *conn)
- int **connection_or_send_link_auth** (**or_connection_t** *conn)
- int **connection_or_compute_link_auth_hmac** (**or_connection_t** *conn, char *hmac_out)
- int **is_or_protocol_version_known** (uint16_t version)
- void **cell_pack** (**packed_cell_t** *dest, const **cell_t** *src)
- void **var_cell_pack_header** (const **var_cell_t** *cell, char *hdr_out)
- **var_cell_t** * **var_cell_new** (uint16_t payload_len)
- void **var_cell_free** (**var_cell_t** *cell)
- void **control_update_global_event_mask** (void)
- void **control_adjust_event_log_severity** (void)
- int **connection_control_finished_flushing** (**control_connection_t** *conn)
- int **connection_control_reached_eof** (**control_connection_t** *conn)
- int **connection_control_process_inbuf** (**control_connection_t** *conn)
- int **control_event_is_interesting** (int event)

- int **control_event_circuit_status** (**origin_circuit_t** ∗circ, **circuit_status_event_t** e, int reason)
- int **control_event_stream_status** (**edge_connection_t** ∗conn, **stream_status_event_t** e, int reason)
- int **control_event_or_conn_status** (**or_connection_t** ∗conn, **or_conn_status_event_t** e, int reason)
- int **control_event_bandwidth_used** (uint32_t n_read, uint32_t n_written)
- int **control_event_stream_bandwidth** (**edge_connection_t** ∗edge_conn)
- int **control_event_stream_bandwidth_used** (void)
- void **control_event_logmsg** (int severity, unsigned int domain, const char ∗msg)
- int **control_event_descriptors_changed** (**smartlist_t** ∗routers)
- int **control_event_address_mapped** (const char ∗from, const char ∗to, time_t expires, const char ∗error)
- int **control_event_or_authdir_new_descriptor** (const char ∗action, const char ∗desc, size_t desclen, const char ∗msg)
- int **control_event_my_descriptor_changed** (void)
- int **control_event_networkstatus_changed** (**smartlist_t** ∗statuses)
- int **control_event_newconsensus** (const **networkstatus_t** ∗consensus)
- int **control_event_networkstatus_changed_single** (**routerstatus_t** ∗rs)
- int **control_event_general_status** (int severity, const char ∗format,...) CHECK_PRINTF(2
- int int **control_event_client_status** (int severity, const char ∗format,...) CHECK_PRINTF(2
- int int int **control_event_server_status** (int severity, const char ∗format,...) CHECK_PRINTF(2
- int int int int **control_event_guard** (const char ∗nickname, const char ∗digest, const char ∗status)
- int **control_event_buildtimeout_set** (const **circuit_build_times_t** ∗cbt, **buildtimeout_set_event_t** type)
- int **init_cookie_authentication** (int enabled)
- **smartlist_t** ∗ **decode_hashed_passwords** (**config_line_t** ∗passwords)
- void **disable_control_logging** (void)
- void **enable_control_logging** (void)
- void **control_event_bootstrap** (**bootstrap_status_t** status, int progress)
- void **control_event_bootstrap_problem** (const char ∗warn, int reason)
- void **control_event_clients_seen** (const char ∗controller_str)
- void **cpu_init** (void)
- void **cpuworkers_rotate** (void)
- int **connection_cpu_finished_flushing** (**connection_t** ∗conn)
- int **connection_cpu_reached_eof** (**connection_t** ∗conn)
- int **connection_cpu_process_inbuf** (**connection_t** ∗conn)
- int **assign_onionskin_to_cpuworker** (**connection_t** ∗cpuworker, **or_circuit_t** ∗circ, char ∗onionskin)
- int **directories_have_accepted_server_descriptor** (void)
- char ∗ **authority_type_to_string** (**authority_type_t** auth)
- void **directory_post_to_dirservers** (uint8_t dir_purpose, uint8_t router_purpose, **authority_type_t** type, const char ∗payload, size_t payload_len, size_t extrainfo_len)
- void **directory_get_from_dirserver** (uint8_t dir_purpose, uint8_t router_purpose, const char ∗resource, int pds_flags)
- void **directory_get_from_all_authorities** (uint8_t dir_purpose, uint8_t router_purpose, const char ∗resource)

- void **directory_initiate_command_routerstatus** (**routerstatus_t** *status, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since)
- void **directory_initiate_command_routerstatus_rend** (**routerstatus_t** *status, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since, const **rend_data_t** *rend_query)
- int **parse_http_response** (const char *headers, int *code, time_t *date, **compress_method_t** *compression, char **response)
- int **connection_dir_is_encrypted** (**dir_connection_t** *conn)
- int **connection_dir_reached_eof** (**dir_connection_t** *conn)
- int **connection_dir_process_inbuf** (**dir_connection_t** *conn)
- int **connection_dir_finished_flushing** (**dir_connection_t** *conn)
- int **connection_dir_finished_connecting** (**dir_connection_t** *conn)
- void **connection_dir_request_failed** (**dir_connection_t** *conn)
- void **directory_initiate_command** (const char *address, const **tor_addr_t** *addr, uint16_t or_port, uint16_t dir_port, int supports_conditional_consensus, int supports_begindir, const char *digest, uint8_t dir_purpose, uint8_t router_purpose, int anonymized_connection, const char *resource, const char *payload, size_t payload_len, time_t if_modified_since)
- int **dir_split_resource_into_fingerprints** (const char *resource, **smartlist_t** *fp_out, int *compressed_out, int flags)
- int **dir_split_resource_into_fingerprint_pairs** (const char *res, **smartlist_t** *pairs_out)
- char * **directory_dump_request_log** (void)
- void **note_request** (const char *key, size_t bytes)
- int **router_supports_extrainfo** (const char *identity_digest, int is_authority)
- time_t **download_status_increment_failure** (**download_status_t** *dls, int status_code, const char *item, int server, time_t now)
- void **download_status_reset** (**download_status_t** *dls)
- static int **download_status_is_ready** (**download_status_t** *dls, time_t now, int max_failures)
- static void **download_status_mark_impossible** (**download_status_t** *dl)
- int **connection_dirserv_flushed_some** (**dir_connection_t** *conn)
- int **dirserv_add_own_fingerprint** (const char *nickname, **crypto_pk_env_t** *pk)
- int **dirserv_load_fingerprint_file** (void)
- void **dirserv_free_fingerprint_list** (void)
- const char * **dirserv_get_nickname_by_digest** (const char *digest)
- enum **was_router_added_t** **dirserv_add_multiple_descriptors** (const char *desc, uint8_t purpose, const char *source, const char **msg)
- enum **was_router_added_t** **dirserv_add_descriptor** (**routerinfo_t** *ri, const char **msg, const char *source)
- void **dirserv_free_descriptors** (void)
- void **dirserv_set_router_is_running** (**routerinfo_t** *router, time_t now)
- int **list_server_status_v1** (**smartlist_t** *routers, char **router_status_out, int for_controller)
- int **dirserv_dump_directory_to_string** (char **dir_out, **crypto_pk_env_t** *private_key)
- int **directory_fetches_from_authorities** (**or_options_t** *options)
- int **directory_fetches_dir_info_early** (**or_options_t** *options)
- int **directory_fetches_dir_info_later** (**or_options_t** *options)

- int **directory_caches_v2_dir_info** (**or_options_t** *options)
- int **directory_caches_dir_info** (**or_options_t** *options)
- int **directory_permits_begindir_requests** (**or_options_t** *options)
- int **directory_permits_controller_requests** (**or_options_t** *options)
- int **directory_too_idle_to_fetch_descriptors** (**or_options_t** *options, time_t now)
- void **directory_set_dirty** (void)
- **cached_dir_t** * **dirserv_get_directory** (void)
- **cached_dir_t** * **dirserv_get_runningrouters** (void)
- **cached_dir_t** * **dirserv_get_consensus** (const char *flavor_name)
- void **dirserv_set_cached_directory** (const char *directory, time_t when, int is_running_routers)
- void **dirserv_set_cached_networkstatus_v2** (const char *directory, const char *identity, time_t published)
- void **dirserv_set_cached_consensus_networkstatus** (const char *consensus, const char *flavor_name, const **digests_t** *digests, time_t published)
- void **dirserv_clear_old_networkstatuses** (time_t cutoff)
- void **dirserv_clear_old_v1_info** (time_t now)
- void **dirserv_get_networkstatus_v2** (**smartlist_t** *result, const char *key)
- void **dirserv_get_networkstatus_v2_fingerprints** (**smartlist_t** *result, const char *key)
- int **dirserv_get_routerdesc_fingerprints** (**smartlist_t** *fps_out, const char *key, const char **msg, int for_unencrypted_conn, int is_extrainfo)
- int **dirserv_get_routerdescs** (**smartlist_t** *descs_out, const char *key, const char **msg)
- void **dirserv_orconn_tls_done** (const char *address, uint16_t or_port, const char *digest_rcvd, int as_advertised)
- void **dirserv_single_reachability_test** (time_t now, **routerinfo_t** *router)
- void **dirserv_test_reachability** (time_t now)
- int **authdir_wants_to_reject_router** (**routerinfo_t** *ri, const char **msg, int complain)
- int **dirserv_would_reject_router** (**routerstatus_t** *rs)
- int **dirserv_remove_old_statuses** (**smartlist_t** *fps, time_t cutoff)
- int **dirserv_have_any_serverdesc** (**smartlist_t** *fps, int spool_src)
- int **dirserv_have_any_microdesc** (const **smartlist_t** *fps)
- size_t **dirserv_estimate_data_size** (**smartlist_t** *fps, int is_serverdescs, int compressed)
- size_t **dirserv_estimate_microdesc_size** (const **smartlist_t** *fps, int compressed)
- int **routerstatus_format_entry** (char *buf, size_t buf_len, **routerstatus_t** *rs, const char *platform, routerstatus_format_type_t format)
- void **dirserv_free_all** (void)
- void **cached_dir_decref** (**cached_dir_t** *d)
- **cached_dir_t** * **new_cached_dir** (char *s, time_t published)
- int **dirserv_read_measured_bandwidths** (const char *from_file, **smartlist_t** *routerstatuses)
- void **dirvote_free_all** (void)
- char * **networkstatus_compute_consensus** (**smartlist_t** *votes, int total_authorities, **crypto_pk_env_t** *identity_key, **crypto_pk_env_t** *signing_key, const char *legacy_identity_key_digest, **crypto_pk_env_t** ***legacy_signing_key**, **consensus_flavor_t** flavor)

- int **networkstatus_add_detached_signatures** (**networkstatus_t** *target, **ns_detached_signatures_t** *sigs, const char **msg_out)
- char * **networkstatus_get_detached_signatures** (**smartlist_t** *consensuses)
- void **ns_detached_signatures_free** (**ns_detached_signatures_t** *s)
- **authority_cert_t** * **authority_cert_dup** (**authority_cert_t** *cert)
- void **dirvote_get_preferred_voting_intervals** (**vote_timing_t** *timing_out)
- time_t **dirvote_get_start_of_next_interval** (time_t now, int interval)
- void **dirvote_recalculate_timing** (**or_options_t** *options, time_t now)
- void **dirvote_act** (**or_options_t** *options, time_t now)
- struct **pending_vote_t** * **dirvote_add_vote** (const char *vote_body, const char **msg_out, int *status_out)
- int **dirvote_add_signatures** (const char *detached_signatures_body, const char *source, const char **msg_out)
- const char * **dirvote_get_pending_consensus** (**consensus_flavor_t** flav)
- const char * **dirvote_get_pending_detached_signatures** (void)
- const **cached_dir_t** * **dirvote_get_vote** (const char *fp, int flags)
- void **set_routerstatus_from_routerinfo** (**routerstatus_t** *rs, **routerinfo_t** *ri, time_t now, int naming, int exits_can_be_guards, int listbadexits, int listbaddirs)
- void **router_clear_status_flags** (**routerinfo_t** *ri)
- **networkstatus_t** * **dirserv_generate_networkstatus_vote_obj** (**crypto_pk_env_t** *private_key, **authority_cert_t** *cert)
- **microdesc_t** * **dirvote_create_microdescriptor** (const **routerinfo_t** *ri)
- ssize_t **dirvote_format_microdesc_vote_line** (char *out, size_t out_len, const **microdesc_t** *md)
- int **vote_routerstatus_find_microdesc_hash** (char *digest256_out, const **vote_routerstatus_t** *vrs, int method, digest_algorithm_t alg)
- **document_signature_t** * **voter_get_sig_by_algorithm** (const **networkstatus_voter_info_t** *voter, digest_algorithm_t alg)
- int **dns_init** (void)
- int **has_dns_init_failed** (void)
- void **dns_free_all** (void)
- uint32_t **dns_clip_ttl** (uint32_t ttl)
- int **dns_reset** (void)
- void **connection_dns_remove** (**edge_connection_t** *conn)
- void **assert_connection_edge_not_dns_pending** (**edge_connection_t** *conn)
- void **assert_all_pending_dns_resolves_ok** (void)
- void **dns_cancel_pending_resolve** (const char *question)
- int **dns_resolve** (**edge_connection_t** *exitconn)
- void **dns_launch_correctness_checks** (void)
- int **dns_seems_to_be_broken** (void)
- void **dns_reset_correctness_checks** (void)
- void **dump_dns_mem_usage** (int severity)
- void **dnsserv_configure_listener** (**connection_t** *conn)
- void **dnsserv_close_listener** (**connection_t** *conn)
- void **dnsserv_resolved** (**edge_connection_t** *conn, int answer_type, size_t answer_len, const char *answer, int ttl)
- void **dnsserv_reject_request** (**edge_connection_t** *conn)
- int **dnsserv_launch_request** (const char *name, int is_reverse)
- int **should_record_bridge_info** (**or_options_t** *options)
- int **geoip_load_file** (const char *filename, **or_options_t** *options)
- int **geoip_get_country_by_ip** (uint32_t ipaddr)

- int **geoip_get_n_countries** (void)
- const char * **geoip_get_country_name** (**country_t** num)
- int **geoip_is_loaded** (void)
- **country_t** **geoip_get_country** (const char *countrycode)
- void **geoip_note_client_seen** (**geoip_client_action_t** action, uint32_t addr, time_t now)
- void **geoip_remove_old_clients** (time_t cutoff)
- void **geoip_note_ns_response** (**geoip_client_action_t** action, **geoip_ns_response_t** response)
- time_t **geoip_get_history_start** (void)
- char * **geoip_get_client_history_dirreq** (time_t now, **geoip_client_action_t** action)
- char * **geoip_get_client_history_bridge** (time_t now, **geoip_client_action_t** action)
- char * **geoip_get_request_history** (time_t now, **geoip_client_action_t** action)
- int **getinfo_helper_geoip** (**control_connection_t** *control_conn, const char *question, char **answer)
- void **geoip_free_all** (void)
- void **geoip_start_dirreq** (uint64_t dirreq_id, size_t response_size, **geoip_client_action_t** action, **dirreq_type_t** type)
- void **geoip_change_dirreq_state** (uint64_t dirreq_id, **dirreq_type_t** type, **dirreq_state_t** new_state)
- void **geoip_dirreq_stats_init** (time_t now)
- void **geoip_dirreq_stats_write** (time_t now)
- void **geoip_entry_stats_init** (time_t now)
- void **geoip_entry_stats_write** (time_t now)
- void **geoip_bridge_stats_init** (time_t now)
- time_t **geoip_bridge_stats_write** (time_t now)
- const char * **geoip_get_bridge_stats_extrainfo** (time_t)
- const char * **geoip_get_bridge_stats_controller** (time_t)
- int **accounting_parse_options** (**or_options_t** *options, int validate_only)
- int **accounting_is_enabled** (**or_options_t** *options)
- void **configure_accounting** (time_t now)
- void **accounting_run_housekeeping** (time_t now)
- void **accounting_add_bytes** (size_t n_read, size_t n_written, int seconds)
- int **accounting_record_bandwidth_usage** (time_t now, **or_state_t** *state)
- void **hibernate_begin_shutdown** (void)
- int **we_are_hibernating** (void)
- void **consider_hibernation** (time_t now)
- int **getinfo_helper_accounting** (**control_connection_t** *conn, const char *question, char **answer)
- void **accounting_set_bandwidth_usage_from_state** (**or_state_t** *state)
- int **connection_add** (**connection_t** *conn)
- int **connection_remove** (**connection_t** *conn)
- void **connection_unregister_events** (**connection_t** *conn)
- int **connection_in_array** (**connection_t** *conn)
- void **add_connection_to_closeable_list** (**connection_t** *conn)
- int **connection_is_on_closeable_list** (**connection_t** *conn)
- **smartlist_t** * **get_connection_array** (void)
- void **connection_watch_events** (**connection_t** *conn, watchable_events_t events)
- int **connection_is_reading** (**connection_t** *conn)

- void **connection_stop_reading** (**connection_t** *conn)
- void **connection_start_reading** (**connection_t** *conn)
- int **connection_is_writing** (**connection_t** *conn)
- void **connection_stop_writing** (**connection_t** *conn)
- void **connection_start_writing** (**connection_t** *conn)
- void **connection_stop_reading_from_linked_conn** (**connection_t** *conn)
- void **directory_all_unreachable** (time_t now)
- void **directory_info_has_arrived** (time_t now, int from_cache)
- void **ip_address_changed** (int at_interface)
- void **dns_servers_relaunch_checks** (void)
- void **control_signal_act** (int the_signal)
- void **handle_signals** (int is_parent)
- int **try_locking** (**or_options_t** *options, int err_if_locked)
- int **have_lockfile** (void)
- void **release_lockfile** (void)
- void **tor_cleanup** (void)
- void **tor_free_all** (int postfork)
- int **tor_main** (int argc, char *argv[ ])
- **microdesc_cache_t** * **get_microdesc_cache** (void)
- **smartlist_t** * **microdescs_add_to_cache** (**microdesc_cache_t** *cache, const char *s, const char *eos, **saved_location_t** where, int no_save)
- **smartlist_t** * **microdescs_add_list_to_cache** (**microdesc_cache_t** *cache, **smartlist_t** *descriptors, **saved_location_t** where, int no_save)
- int **microdesc_cache_rebuild** (**microdesc_cache_t** *cache)
- int **microdesc_cache_reload** (**microdesc_cache_t** *cache)
- void **microdesc_cache_clear** (**microdesc_cache_t** *cache)
- **microdesc_t** * **microdesc_cache_lookup_by_digest256** (**microdesc_cache_t** *cache, const char *d)
- size_t **microdesc_average_size** (**microdesc_cache_t** *cache)
- void **microdesc_free** (**microdesc_t** *md)
- void **microdesc_free_all** (void)
- void **networkstatus_reset_warnings** (void)
- void **networkstatus_reset_download_failures** (void)
- int **router_reload_v2_networkstatus** (void)
- int **router_reload_consensus_networkstatus** (void)
- void **routerstatus_free** (**routerstatus_t** *rs)
- void **networkstatus_v2_free** (**networkstatus_v2_t** *ns)
- void **networkstatus_vote_free** (**networkstatus_t** *ns)
- **networkstatus_voter_info_t** * **networkstatus_get_voter_by_id** (**networkstatus_t** *vote, const char *identity)
- int **networkstatus_check_consensus_signature** (**networkstatus_t** *consensus, int warn)
- int **networkstatus_check_document_signature** (const **networkstatus_t** *consensus, **document_signature_t** *sig, const **authority_cert_t** *cert)
- char * **networkstatus_get_cache_filename** (const char *identity_digest)
- int **router_set_networkstatus_v2** (const char *s, time_t arrived_at, **v2_networkstatus_source_t** source, **smartlist_t** *requested_fingerprints)
- void **networkstatus_v2_list_clean** (time_t now)
- int **compare_digest_to_routerstatus_entry** (const void *_key, const void **_member)

- **routerstatus_t** ∗ **networkstatus_v2_find_entry** (**networkstatus_v2_t** ∗ns, const char ∗digest)
- **routerstatus_t** ∗ **networkstatus_vote_find_entry** (**networkstatus_t** ∗ns, const char ∗digest)
- int **networkstatus_vote_find_entry_idx** (**networkstatus_t** ∗ns, const char ∗digest, int ∗found_out)
- const **smartlist_t** ∗ **networkstatus_get_v2_list** (void)
- **download_status_t** ∗ **router_get_dl_status_by_descriptor_digest** (const char ∗d)
- **routerstatus_t** ∗ **router_get_consensus_status_by_id** (const char ∗digest)
- **routerstatus_t** ∗ **router_get_consensus_status_by_descriptor_digest** (const char ∗digest)
- **routerstatus_t** ∗ **router_get_consensus_status_by_nickname** (const char ∗nickname, int warn_if_unnamed)
- const char ∗ **networkstatus_get_router_digest_by_nickname** (const char ∗nickname)
- int **networkstatus_nickname_is_unnamed** (const char ∗nickname)
- void **networkstatus_consensus_download_failed** (int status_code)
- void **update_consensus_networkstatus_fetch_time** (time_t now)
- int **should_delay_dir_fetches** (**or_options_t** ∗options)
- void **update_networkstatus_downloads** (time_t now)
- void **update_certificate_downloads** (time_t now)
- int **consensus_is_waiting_for_certs** (void)
- **networkstatus_v2_t** ∗ **networkstatus_v2_get_by_digest** (const char ∗digest)
- **networkstatus_t** ∗ **networkstatus_get_latest_consensus** (void)
- **networkstatus_t** ∗ **networkstatus_get_live_consensus** (time_t now)
- **networkstatus_t** ∗ **networkstatus_get_reasonably_live_consensus** (time_t now)
- int **networkstatus_set_current_consensus** (const char ∗consensus, const char ∗flavor, unsigned flags)
- void **networkstatus_note_certs_arrived** (void)
- void **routers_update_all_from_networkstatus** (time_t now, int dir_version)
- void **routerstatus_list_update_from_consensus_networkstatus** (time_t now)
- void **routers_update_status_from_consensus_networkstatus** (**smartlist_t** ∗routers, int reset_failures)
- void **signed_descs_update_status_from_consensus_networkstatus** (**smartlist_t** ∗descs)
- char ∗ **networkstatus_getinfo_helper_single** (**routerstatus_t** ∗rs)
- char ∗ **networkstatus_getinfo_by_purpose** (const char ∗purpose_string, time_t now)
- void **networkstatus_dump_bridge_status_to_file** (time_t now)
- int32_t **get_net_param_from_list** (**smartlist_t** ∗net_params, const char ∗name, int default_val)
- int32_t **networkstatus_get_param** (**networkstatus_t** ∗ns, const char ∗param_name, int32_t default_val)
- int **getinfo_helper_networkstatus** (**control_connection_t** ∗conn, const char ∗question, char ∗∗answer)
- int32_t **networkstatus_get_bw_weight** (**networkstatus_t** ∗ns, const char ∗weight, int32_t default_val)
- const char ∗ **networkstatus_get_flavor_name** (**consensus_flavor_t** flav)
- int **networkstatus_parse_flavor_name** (const char ∗flavname)
- void **document_signature_free** (**document_signature_t** ∗sig)

- **document_signature_t** ∗ **document_signature_dup** (const **document_signature_t** ∗sig)
- void **networkstatus_free_all** (void)
- int **onion_pending_add** (**or_circuit_t** ∗circ, char ∗onionskin)
- **or_circuit_t** ∗ **onion_next_task** (char ∗∗onionskin_out)
- void **onion_pending_remove** (**or_circuit_t** ∗circ)
- int **onion_skin_create** (**crypto_pk_env_t** ∗router_key, **crypto_dh_env_t** ∗∗handshake_state_out, char ∗onion_skin_out)
- int **onion_skin_server_handshake** (const char ∗onion_skin, **crypto_pk_env_t** ∗private_key, **crypto_pk_env_t** ∗prev_private_key, char ∗handshake_reply_out, char ∗key_out, size_t key_out_len)
- int **onion_skin_client_handshake** (**crypto_dh_env_t** ∗handshake_state, const char ∗handshake_reply, char ∗key_out, size_t key_out_len)
- int **fast_server_handshake** (const char ∗key_in, char ∗handshake_reply_out, char ∗key_out, size_t key_out_len)
- int **fast_client_handshake** (const char ∗handshake_state, const char ∗handshake_reply_out, char ∗key_out, size_t key_out_len)
- void **clear_pending_onions** (void)
- int **firewall_is_fascist_or** (void)
- int **fascist_firewall_allows_address_or** (const **tor_addr_t** ∗addr, uint16_t port)
- int **fascist_firewall_allows_or** (**routerinfo_t** ∗ri)
- int **fascist_firewall_allows_address_dir** (const **tor_addr_t** ∗addr, uint16_t port)
- int **dir_policy_permits_address** (const **tor_addr_t** ∗addr)
- int **socks_policy_permits_address** (const **tor_addr_t** ∗addr)
- int **authdir_policy_permits_address** (uint32_t addr, uint16_t port)
- int **authdir_policy_valid_address** (uint32_t addr, uint16_t port)
- int **authdir_policy_baddir_address** (uint32_t addr, uint16_t port)
- int **authdir_policy_badexit_address** (uint32_t addr, uint16_t port)
- int **validate_addr_policies** (**or_options_t** ∗options, char ∗∗msg)
- void **policy_expand_private** (**smartlist_t** ∗∗policy)
- int **policies_parse_from_options** (**or_options_t** ∗options)
- **addr_policy_t** ∗ **addr_policy_get_canonical_entry** (**addr_policy_t** ∗ent)
- int **cmp_addr_policies** (**smartlist_t** ∗a, **smartlist_t** ∗b)
- **addr_policy_result_t** **compare_tor_addr_to_addr_policy** (const **tor_addr_t** ∗addr, uint16_t port, const **smartlist_t** ∗policy)
- **addr_policy_result_t** **compare_addr_to_addr_policy** (uint32_t addr, uint16_t port, const **smartlist_t** ∗policy)
- int **policies_parse_exit_policy** (**config_line_t** ∗cfg, **smartlist_t** ∗∗dest, int reject-private, const char ∗local_address, int add_default_policy)
- void **policies_set_router_exitpolicy_to_reject_all** (**routerinfo_t** ∗exitrouter)
- int **exit_policy_is_general_exit** (**smartlist_t** ∗policy)
- int **policy_is_reject_star** (const **smartlist_t** ∗policy)
- int **getinfo_helper_policies** (**control_connection_t** ∗conn, const char ∗question, char ∗∗answer)
- int **policy_write_item** (char ∗buf, size_t buflen, **addr_policy_t** ∗item, int format_for_desc)
- void **addr_policy_list_free** (**smartlist_t** ∗p)
- void **addr_policy_free** (**addr_policy_t** ∗p)
- void **policies_free_all** (void)
- char ∗ **policy_summarize** (**smartlist_t** ∗policy)
- const char ∗ **stream_end_reason_to_control_string** (int reason)

- const char * **stream_end_reason_to_string** (int reason)
- socks5_reply_status_t **stream_end_reason_to_socks5_response** (int reason)
- uint8_t **errno_to_stream_end_reason** (int e)
- const char * **orconn_end_reason_to_control_string** (int r)
- int **tls_error_to_orconn_end_reason** (int e)
- int **errno_to_orconn_end_reason** (int e)
- const char * **circuit_end_reason_to_control_string** (int reason)
- const char * **socks4_response_code_to_string** (uint8_t code)
- const char * **socks5_response_code_to_string** (uint8_t code)
- int **circuit_receive_relay_cell** (**cell_t** *cell, **circuit_t** *circ, **cell_direction_t** cell_direction)
- void **relay_header_pack** (char *dest, const **relay_header_t** *src)
- void **relay_header_unpack** (**relay_header_t** *dest, const char *src)
- int **relay_send_command_from_edge** (**streamid_t** stream_id, **circuit_t** *circ, uint8_t relay_command, const char *payload, size_t payload_len, **crypt_path_t** *cpath_layer)
- int **connection_edge_send_command** (**edge_connection_t** *fromconn, uint8_t relay_command, const char *payload, size_t payload_len)
- int **connection_edge_package_raw_inbuf** (**edge_connection_t** *conn, int package_partial)
- void **connection_edge_consider_sending_sendme** (**edge_connection_t** *conn)
- void **init_cell_pool** (void)
- void **free_cell_pool** (void)
- void **clean_cell_pool** (void)
- void **dump_cell_pool_usage** (int severity)
- void **cell_queue_clear** (**cell_queue_t** *queue)
- void **cell_queue_append** (**cell_queue_t** *queue, **packed_cell_t** *cell)
- void **cell_queue_append_packed_copy** (**cell_queue_t** *queue, const **cell_t** *cell)
- void **append_cell_to_circuit_queue** (**circuit_t** *circ, **or_connection_t** *orconn, **cell_t** *cell, **cell_direction_t** direction)
- void **connection_or_unlink_all_active_circs** (**or_connection_t** *conn)
- int **connection_or_flush_from_first_active_circuit** (**or_connection_t** *conn, int max, time_t now)
- void **assert_active_circuits_ok** (**or_connection_t** *orconn)
- void **make_circuit_inactive_on_conn** (**circuit_t** *circ, **or_connection_t** *conn)
- void **make_circuit_active_on_conn** (**circuit_t** *circ, **or_connection_t** *conn)
- int **append_address_to_payload** (char *payload_out, const **tor_addr_t** *addr)
- const char * **decode_address_from_payload** (**tor_addr_t** *addr_out, const char *payload, int payload_len)
- unsigned **cell_ewma_get_tick** (void)
- void **cell_ewma_set_scale_factor** (**or_options_t** *options, **networkstatus_t** *consensus)
- void **rep_hist_init** (void)
- void **rep_hist_note_connect_failed** (const char *nickname, time_t when)
- void **rep_hist_note_connect_succeeded** (const char *nickname, time_t when)
- void **rep_hist_note_disconnect** (const char *nickname, time_t when)
- void **rep_hist_note_connection_died** (const char *nickname, time_t when)
- void **rep_hist_note_extend_succeeded** (const char *from_name, const char *to_name)
- void **rep_hist_note_extend_failed** (const char *from_name, const char *to_name)
- void **rep_hist_dump_stats** (time_t now, int severity)

- void **rep_hist_note_bytes_read** (size_t num_bytes, time_t when)
- void **rep_hist_note_bytes_written** (size_t num_bytes, time_t when)
- void **rep_hist_note_exit_bytes_read** (uint16_t port, size_t num_bytes)
- void **rep_hist_note_exit_bytes_written** (uint16_t port, size_t num_bytes)
- void **rep_hist_note_exit_stream_opened** (uint16_t port)
- void **rep_hist_exit_stats_init** (time_t now)
- void **rep_hist_exit_stats_write** (time_t now)
- int **rep_hist_bandwidth_assess** (void)
- char * **rep_hist_get_bandwidth_lines** (int for_extrainfo)
- void **rep_hist_update_state** (**or_state_t** *state)
- int **rep_hist_load_state** (**or_state_t** *state, char **err)
- void **rep_history_clean** (time_t before)
- void **rep_hist_note_router_reachable** (const char *id, time_t when)
- void **rep_hist_note_router_unreachable** (const char *id, time_t when)
- int **rep_hist_record_mtbf_data** (time_t now, int missing_means_down)
- int **rep_hist_load_mtbf_data** (time_t now)
- time_t **rep_hist_downrate_old_runs** (time_t now)
- double **rep_hist_get_stability** (const char *id, time_t when)
- double **rep_hist_get_weighted_fractional_uptime** (const char *id, time_t when)
- long **rep_hist_get_weighted_time_known** (const char *id, time_t when)
- int **rep_hist_have_measured_enough_stability** (void)
- const char * **rep_hist_get_router_stability_doc** (time_t now)
- void **rep_hist_note_used_port** (time_t now, uint16_t port)
- **smartlist_t** * **rep_hist_get_predicted_ports** (time_t now)
- void **rep_hist_note_used_resolve** (time_t now)
- void **rep_hist_note_used_internal** (time_t now, int need_uptime, int need_capacity)
- int **rep_hist_get_predicted_internal** (time_t now, int *need_uptime, int *need_capacity)
- int **any_predicted_circuits** (time_t now)
- int **rep_hist_circbuilding_dormant** (time_t now)
- void **note_crypto_pk_op** (**pk_op_t** operation)
- void **dump_pk_ops** (int severity)
- void **rep_hist_free_all** (void)
- void **hs_usage_note_publish_total** (const char *service_id, time_t now)
- void **hs_usage_note_publish_novel** (const char *service_id, time_t now)
- void **hs_usage_note_fetch_total** (const char *service_id, time_t now)
- void **hs_usage_note_fetch_successful** (const char *service_id, time_t now)
- void **hs_usage_write_statistics_to_file** (time_t now)
- void **hs_usage_free_all** (void)
- void **rep_hist_buffer_stats_init** (time_t now)
- void **rep_hist_buffer_stats_add_circ** (**circuit_t** *circ, time_t end_of_interval)
- void **rep_hist_buffer_stats_write** (time_t now)
- void **rend_client_introcirc_has_opened** (**origin_circuit_t** *circ)
- void **rend_client_rendcirc_has_opened** (**origin_circuit_t** *circ)
- int **rend_client_introduction_acked** (**origin_circuit_t** *circ, const char *request, size_t request_len)
- void **rend_client_refetch_v2_renddesc** (const **rend_data_t** *rend_query)
- int **rend_client_remove_intro_point** (**extend_info_t** *failed_intro, const **rend_data_t** *rend_query)

- int **rend_client_rendezvous_acked** (**origin_circuit_t** *circ, const char *request, size_t request_len)
- int **rend_client_receive_rendezvous** (**origin_circuit_t** *circ, const char *request, size_t request_len)
- void **rend_client_desc_trynow** (const char *query)
- **extend_info_t** * **rend_client_get_random_intro** (const **rend_data_t** *rend_query)
- int **rend_client_send_introduction** (**origin_circuit_t** *introcirc, **origin_circuit_t** *rendcirc)
- int **rend_parse_service_authorization** (**or_options_t** *options, int validate_only)
- **rend_service_authorization_t** * **rend_client_lookup_service_authorization** (const char *onion_address)
- void **rend_service_authorization_free_all** (void)
- **rend_data_t** * **rend_data_dup** (const **rend_data_t** *request)
- static INLINE void **rend_data_free** (**rend_data_t** *data)
- int **rend_cmp_service_ids** (const char *one, const char *two)
- void **rend_process_relay_cell** (**circuit_t** *circ, const **crypt_path_t** *layer_hint, int command, size_t length, const char *payload)
- void **rend_service_descriptor_free** (**rend_service_descriptor_t** *desc)
- **rend_service_descriptor_t** * **rend_parse_service_descriptor** (const char *str, size_t len)
- int **rend_get_service_id** (**crypto_pk_env_t** *pk, char *out)
- void **rend_encoded_v2_service_descriptor_free** (**rend_encoded_v2_service_descriptor_t** *desc)
- void **rend_intro_point_free** (**rend_intro_point_t** *intro)
- void **rend_cache_init** (void)
- void **rend_cache_clean** (void)
- void **rend_cache_clean_v2_descs_as_dir** (void)
- void **rend_cache_free_all** (void)
- int **rend_valid_service_id** (const char *query)
- int **rend_cache_lookup_desc** (const char *query, int version, const char **desc, size_t *desc_len)
- int **rend_cache_lookup_entry** (const char *query, int version, **rend_cache_entry_t** **entry_out)
- int **rend_cache_lookup_v2_desc_as_dir** (const char *query, const char **desc)
- int **rend_cache_store** (const char *desc, size_t desc_len, int published)
- int **rend_cache_store_v2_desc_as_client** (const char *desc, const **rend_data_t** *rend_query)
- int **rend_cache_store_v2_desc_as_dir** (const char *desc)
- int **rend_cache_size** (void)
- int **rend_encode_v2_descriptors** (**smartlist_t** *descs_out, **rend_service_descriptor_t** *desc, time_t now, uint8_t period, **rend_auth_type_t** auth_type, **crypto_pk_env_t** *client_key, **smartlist_t** *client_cookies)
- int **rend_compute_v2_desc_id** (char *desc_id_out, const char *service_id, const char *descriptor_cookie, time_t now, uint8_t replica)
- int **rend_id_is_in_interval** (const char *a, const char *b, const char *c)
- void **rend_get_descriptor_id_bytes** (char *descriptor_id_out, const char *service_id, const char *secret_id_part)
- int **num_rend_services** (void)
- int **rend_config_services** (**or_options_t** *options, int validate_only)
- int **rend_service_load_keys** (void)

- void **rend_services_init** (void)
- void **rend_services_introduce** (void)
- void **rend_consider_services_upload** (time_t now)
- void **rend_hsdir_routers_changed** (void)
- void **rend_consider_descriptor_republication** (void)
- void **rend_service_intro_has_opened** (**origin_circuit_t** *circuit)
- int **rend_service_intro_established** (**origin_circuit_t** *circuit, const char *request, size_t request_len)
- void **rend_service_rendezvous_has_opened** (**origin_circuit_t** *circuit)
- int **rend_service_introduce** (**origin_circuit_t** *circuit, const char *request, size_t request_len)
- void **rend_service_relaunch_rendezvous** (**origin_circuit_t** *oldcirc)
- int **rend_service_set_connection_addr_port** (**edge_connection_t** *conn, **origin_circuit_t** *circ)
- void **rend_service_dump_stats** (int severity)
- void **rend_service_free_all** (void)
- int **rend_mid_establish_intro** (**or_circuit_t** *circ, const char *request, size_t request_len)
- int **rend_mid_introduce** (**or_circuit_t** *circ, const char *request, size_t request_len)
- int **rend_mid_establish_rendezvous** (**or_circuit_t** *circ, const char *request, size_t request_len)
- int **rend_mid_rendezvous** (**or_circuit_t** *circ, const char *request, size_t request_len)
- **crypto_pk_env_t** * **get_onion_key** (void)
- time_t **get_onion_key_set_at** (void)
- void **set_identity_key** (**crypto_pk_env_t** *k)
- **crypto_pk_env_t** * **get_identity_key** (void)
- int **identity_key_is_set** (void)
- **authority_cert_t** * **get_my_v3_authority_cert** (void)
- **crypto_pk_env_t** * **get_my_v3_authority_signing_key** (void)
- **authority_cert_t** * **get_my_v3_legacy_cert** (void)
- **crypto_pk_env_t** * **get_my_v3_legacy_signing_key** (void)
- void **dup_onion_keys** (**crypto_pk_env_t** **key, **crypto_pk_env_t** **last)
- void **rotate_onion_key** (void)
- **crypto_pk_env_t** * **init_key_from_file** (const char *fname, int generate, int severity)
- void **v3_authority_check_key_expiry** (void)
- int **init_keys** (void)
- int **check_whether_orport_reachable** (void)
- int **check_whether_dirport_reachable** (void)
- void **consider_testing_reachability** (int test_or, int test_dir)
- void **router_orport_found_reachable** (void)
- void **router_dirport_found_reachable** (void)
- void **router_perform_bandwidth_test** (int num_circs, time_t now)
- int **authdir_mode** (**or_options_t** *options)
- int **authdir_mode_v1** (**or_options_t** *options)
- int **authdir_mode_v2** (**or_options_t** *options)
- int **authdir_mode_v3** (**or_options_t** *options)
- int **authdir_mode_any_main** (**or_options_t** *options)
- int **authdir_mode_any_nonhidserv** (**or_options_t** *options)
- int **authdir_mode_handles_descs** (**or_options_t** *options, int purpose)

- int **authdir_mode_publishes_statuses** (**or_options_t** ∗options)
- int **authdir_mode_tests_reachability** (**or_options_t** ∗options)
- int **authdir_mode_bridge** (**or_options_t** ∗options)
- int **server_mode** (**or_options_t** ∗options)
- int **advertised_server_mode** (void)
- int **proxy_mode** (**or_options_t** ∗options)
- void **consider_publishable_server** (int force)
- void **router_upload_dir_desc_to_dirservers** (int force)
- void **mark_my_descriptor_dirty_if_older_than** (time_t when)
- void **mark_my_descriptor_dirty** (void)
- void **check_descriptor_bandwidth_changed** (time_t now)
- void **check_descriptor_ipaddress_changed** (time_t now)
- void **router_new_address_suggestion** (const char ∗suggestion, const **dir_connection_t** ∗d_conn)
- int **router_compare_to_my_exit_policy** (**edge_connection_t** ∗conn)
- **routerinfo_t** ∗ **router_get_my_routerinfo** (void)
- **extrainfo_t** ∗ **router_get_my_extrainfo** (void)
- const char ∗ **router_get_my_descriptor** (void)
- int **router_digest_is_me** (const char ∗digest)
- int **router_extrainfo_digest_is_me** (const char ∗digest)
- int **router_is_me** (**routerinfo_t** ∗router)
- int **router_fingerprint_is_me** (const char ∗fp)
- int **router_pick_published_address** (**or_options_t** ∗options, uint32_t ∗addr)
- int **router_rebuild_descriptor** (int force)
- int **router_dump_router_to_string** (char ∗s, size_t maxlen, **routerinfo_t** ∗router, **crypto_pk_env_t** ∗ident_key)
- int **extrainfo_dump_to_string** (char ∗s, size_t maxlen, **extrainfo_t** ∗extrainfo, **crypto_pk_env_t** ∗ident_key)
- int **is_legal_nickname** (const char ∗s)
- int **is_legal_nickname_or_hexdigest** (const char ∗s)
- int **is_legal_hexdigest** (const char ∗s)
- void **router_get_verbose_nickname** (char ∗buf, const **routerinfo_t** ∗router)
- void **routerstatus_get_verbose_nickname** (char ∗buf, const **routerstatus_t** ∗router)
- void **router_reset_warnings** (void)
- void **router_reset_reachability** (void)
- void **router_free_all** (void)
- const char ∗ **router_purpose_to_string** (uint8_t p)
- uint8_t **router_purpose_from_string** (const char ∗s)
- int **get_n_authorities** (**authority_type_t** type)
- int **trusted_dirs_reload_certs** (void)
- int **trusted_dirs_load_certs_from_string** (const char ∗contents, int from_store, int flush)
- void **trusted_dirs_flush_certs_to_disk** (void)
- **authority_cert_t** ∗ **authority_cert_get_newest_by_id** (const char ∗id_digest)
- **authority_cert_t** ∗ **authority_cert_get_by_sk_digest** (const char ∗sk_digest)
- **authority_cert_t** ∗ **authority_cert_get_by_digests** (const char ∗id_digest, const char ∗sk_digest)
- void **authority_cert_get_all** (**smartlist_t** ∗certs_out)
- void **authority_cert_dl_failed** (const char ∗id_digest, int status)
- void **authority_certs_fetch_missing** (**networkstatus_t** ∗status, time_t now)

- int **router_reload_router_list** (void)
- **smartlist_t** ∗ **router_get_trusted_dir_servers** (void)
- **routerstatus_t** ∗ **router_pick_directory_server** (**authority_type_t** type, int flags)
- **trusted_dir_server_t** ∗ **router_get_trusteddirserver_by_digest** (const char ∗d)
- **trusted_dir_server_t** ∗ **trusteddirserver_get_by_v3_auth_digest** (const char ∗d)
- **routerstatus_t** ∗ **router_pick_trusteddirserver** (**authority_type_t** type, int flags)
- int **router_get_my_share_of_directory_requests** (double ∗v2_share_out, double ∗v3_share_out)
- void **router_reset_status_download_failures** (void)
- void **routerlist_add_family** (**smartlist_t** ∗sl, **routerinfo_t** ∗router)
- int **routers_in_same_family** (**routerinfo_t** ∗r1, **routerinfo_t** ∗r2)
- void **add_nickname_list_to_smartlist** (**smartlist_t** ∗sl, const char ∗list, int must_be_running)
- int **router_nickname_is_in_list** (**routerinfo_t** ∗router, const char ∗list)
- **routerinfo_t** ∗ **routerlist_find_my_routerinfo** (void)
- **routerinfo_t** ∗ **router_find_exact_exit_enclave** (const char ∗address, uint16_t port)
- int **router_is_unreliable** (**routerinfo_t** ∗router, int need_uptime, int need_capacity, int need_guard)
- uint32_t **router_get_advertised_bandwidth** (**routerinfo_t** ∗router)
- uint32_t **router_get_advertised_bandwidth_capped** (**routerinfo_t** ∗router)
- **routerinfo_t** ∗ **routerlist_sl_choose_by_bandwidth** (**smartlist_t** ∗sl, **bandwidth_weight_rule_t** rule)
- **routerstatus_t** ∗ **routerstatus_sl_choose_by_bandwidth** (**smartlist_t** ∗sl, **bandwidth_weight_rule_t** rule)
- const char ∗ **bandwidth_weight_rule_to_string** (enum **bandwidth_weight_rule_t** rule)
- **routerinfo_t** ∗ **router_choose_random_node** (**smartlist_t** ∗excludedsmartlist, struct **routerset_t** ∗excludedset, **router_crn_flags_t** flags)
- **routerinfo_t** ∗ **router_get_by_nickname** (const char ∗nickname, int warn_if_unnamed)
- int **router_digest_version_as_new_as** (const char ∗digest, const char ∗cutoff)
- int **router_digest_is_trusted_dir_type** (const char ∗digest, **authority_type_t** type)
- int **router_addr_is_trusted_dir** (uint32_t addr)
- int **hexdigest_to_digest** (const char ∗hexdigest, char ∗digest)
- **routerinfo_t** ∗ **router_get_by_hexdigest** (const char ∗hexdigest)
- **routerinfo_t** ∗ **router_get_by_digest** (const char ∗digest)
- **signed_descriptor_t** ∗ **router_get_by_descriptor_digest** (const char ∗digest)
- **signed_descriptor_t** ∗ **router_get_by_extrainfo_digest** (const char ∗digest)
- **signed_descriptor_t** ∗ **extrainfo_get_by_descriptor_digest** (const char ∗digest)
- const char ∗ **signed_descriptor_get_body** (**signed_descriptor_t** ∗desc)
- const char ∗ **signed_descriptor_get_annotations** (**signed_descriptor_t** ∗desc)
- **routerlist_t** ∗ **router_get_routerlist** (void)
- void **routerinfo_free** (**routerinfo_t** ∗router)
- void **extrainfo_free** (**extrainfo_t** ∗extrainfo)
- void **routerlist_free** (**routerlist_t** ∗rl)
- void **dump_routerlist_mem_usage** (int severity)
- void **routerlist_remove** (**routerlist_t** ∗rl, **routerinfo_t** ∗ri, int make_old, time_t now)
- void **routerlist_free_all** (void)

- void **routerlist_reset_warnings** (void)
- void **router_set_status** (const char *digest, int up)
- static int **WRA_WAS_ADDED** (**was_router_added_t** s)
- static int **WRA_WAS_OUTDATED** (**was_router_added_t** s)
- static int **WRA_WAS_REJECTED** (**was_router_added_t** s)
- **was_router_added_t** **router_add_to_routerlist** (**routerinfo_t** *router, const char **msg, int from_cache, int from_fetch)
- **was_router_added_t** **router_add_extrainfo_to_routerlist** (**extrainfo_t** *ei, const char **msg, int from_cache, int from_fetch)
- void **routerlist_remove_old_routers** (void)
- int **router_load_single_router** (const char *s, uint8_t purpose, int cache, const char **msg)
- int **router_load_routers_from_string** (const char *s, const char *eos, **saved_location_t** saved_location, **smartlist_t** *requested_fingerprints, int descriptor_digests, const char *prepend_annotations)
- void **router_load_extrainfo_from_string** (const char *s, const char *eos, **saved_location_t** saved_location, **smartlist_t** *requested_fingerprints, int descriptor_digests)
- void **routerlist_retry_directory_downloads** (time_t now)
- int **router_exit_policy_all_routers_reject** (uint32_t addr, uint16_t port, int need_uptime)
- int **router_exit_policy_rejects_all** (**routerinfo_t** *router)
- **trusted_dir_server_t** * **add_trusted_dir_server** (const char *nickname, const char *address, uint16_t dir_port, uint16_t or_port, const char *digest, const char *v3_auth_digest, **authority_type_t** type)
- void **authority_cert_free** (**authority_cert_t** *cert)
- void **clear_trusted_dir_servers** (void)
- int **any_trusted_dir_is_v1_authority** (void)
- void **update_consensus_router_descriptor_downloads** (time_t now, int is_vote, **networkstatus_t** *consensus)
- void **update_router_descriptor_downloads** (time_t now)
- void **update_extrainfo_downloads** (time_t now)
- int **router_have_minimum_dir_info** (void)
- void **router_dir_info_changed** (void)
- const char * **get_dir_info_status_string** (void)
- int **count_loading_descriptors_progress** (void)
- void **router_reset_descriptor_download_failures** (void)
- int **router_differences_are_cosmetic** (**routerinfo_t** *r1, **routerinfo_t** *r2)
- int **routerinfo_incompatible_with_extrainfo** (**routerinfo_t** *ri, **extrainfo_t** *ei, **signed_descriptor_t** *sd, const char **msg)
- void **routerlist_assert_ok** (**routerlist_t** *rl)
- const char * **esc_router_info** (**routerinfo_t** *router)
- void **routers_sort_by_identity** (**smartlist_t** *routers)
- **routerset_t** * **routerset_new** (void)
- int **routerset_parse** (**routerset_t** *target, const char *s, const char *description)
- void **routerset_union** (**routerset_t** *target, const **routerset_t** *source)
- int **routerset_is_list** (const **routerset_t** *set)
- int **routerset_needs_geoip** (const **routerset_t** *set)
- int **routerset_contains_router** (const **routerset_t** *set, **routerinfo_t** *ri)
- int **routerset_contains_routerstatus** (const **routerset_t** *set, **routerstatus_t** *rs)
- int **routerset_contains_extendinfo** (const **routerset_t** *set, const **extend_info_t** *ei)

- void **routerset_get_all_routers** (**smartlist_t** *out, const **routerset_t** *routerset, int running_only)
- void **routersets_get_disjunction** (**smartlist_t** *target, const **smartlist_t** *source, const **routerset_t** *include, const **routerset_t** *exclude, int running_only)
- void **routerset_subtract_routers** (**smartlist_t** *out, const **routerset_t** *routerset)
- char * **routerset_to_string** (const **routerset_t** *routerset)
- void **routerset_refresh_countries** (**routerset_t** *target)
- int **routerset_equal** (const **routerset_t** *old, const **routerset_t** *new)
- void **routerset_free** (**routerset_t** *routerset)
- void **routerinfo_set_country** (**routerinfo_t** *ri)
- void **routerlist_refresh_countries** (void)
- void **refresh_all_country_info** (void)
- int **hid_serv_get_responsible_directories** (**smartlist_t** *responsible_dirs, const char *id)
- int **hid_serv_acting_as_directory** (void)
- int **hid_serv_responsible_for_desc_id** (const char *id)
- int **router_get_router_hash** (const char *s, size_t s_len, char *digest)
- int **router_get_dir_hash** (const char *s, char *digest)
- int **router_get_runningrouters_hash** (const char *s, char *digest)
- int **router_get_networkstatus_v2_hash** (const char *s, char *digest)
- int **router_get_networkstatus_v3_hash** (const char *s, char *digest, digest_algorithm_t algorithm)
- int **router_get_networkstatus_v3_hashes** (const char *s, **digests_t** *digests)
- int **router_get_extrainfo_hash** (const char *s, char *digest)
- int **router_append_dirobj_signature** (char *buf, size_t buf_len, const char *digest, size_t digest_len, **crypto_pk_env_t** *private_key)
- int **router_parse_list_from_string** (const char **s, const char *eos, **smartlist_t** *dest, **saved_location_t** saved_location, int is_extrainfo, int allow_annotations, const char *prepend_annotations)
- int **router_parse_routerlist_from_directory** (const char *s, **routerlist_t** **dest, **crypto_pk_env_t** *pkey, int check_version, int write_to_cache)
- int **router_parse_runningrouters** (const char *str)
- int **router_parse_directory** (const char *str)
- **routerinfo_t** * **router_parse_entry_from_string** (const char *s, const char *end, int cache_copy, int allow_annotations, const char *prepend_annotations)
- **extrainfo_t** * **extrainfo_parse_entry_from_string** (const char *s, const char *end, int cache_copy, struct digest_ri_map_t *routermap)
- **addr_policy_t** * **router_parse_addr_policy_item_from_string** (const char *s, int assume_action)
- **version_status_t** **tor_version_is_obsolete** (const char *myversion, const char *versionlist)
- int **tor_version_parse** (const char *s, **tor_version_t** *out)
- int **tor_version_as_new_as** (const char *platform, const char *cutoff)
- int **tor_version_compare** (**tor_version_t** *a, **tor_version_t** *b)
- void **sort_version_list** (**smartlist_t** *lst, int remove_duplicates)
- void **assert_addr_policy_ok** (**smartlist_t** *t)
- void **dump_distinct_digest_count** (int severity)
- int **compare_routerstatus_entries** (const void **_a, const void **_b)
- **networkstatus_v2_t** * **networkstatus_v2_parse_from_string** (const char *s)
- int **networkstatus_verify_bw_weights** (**networkstatus_t** *ns)

- **networkstatus_t** ∗ **networkstatus_parse_vote_from_string** (const char ∗s, const char ∗∗eos_out, **networkstatus_type_t** ns_type)
- **ns_detached_signatures_t** ∗ **networkstatus_parse_detached_signatures** (const char ∗s, const char ∗eos)
- **smartlist_t** ∗ **microdescs_parse_from_string** (const char ∗s, const char ∗eos, int allow_annotations, int copy_body)
- **authority_cert_t** ∗ **authority_cert_parse_from_string** (const char ∗s, const char ∗∗end_of_string)
- int **rend_parse_v2_service_descriptor** (**rend_service_descriptor_t** ∗∗parsed_out, char ∗desc_id_out, char ∗∗intro_points_encrypted_out, size_t ∗intro_points_encrypted_size_out, size_t ∗encoded_size_out, const char ∗∗next_out, const char ∗desc)
- int **rend_decrypt_introduction_points** (char ∗∗ipos_decrypted, size_t ∗ipos_decrypted_size, const char ∗descriptor_cookie, const char ∗ipos_encrypted, size_t ipos_encrypted_size)
- int **rend_parse_introduction_points** (**rend_service_descriptor_t** ∗parsed, const char ∗intro_points_encoded, size_t intro_points_encoded_size)
- int **rend_parse_client_keys** (strmap_t ∗parsed_clients, const char ∗str)
- void **tor_gettimeofday_cache_clear** (void)

Variables

- **circuit_build_times_t circ_times**
- uint64_t **stats_n_padding_cells_processed**
- uint64_t **stats_n_create_cells_processed**
- uint64_t **stats_n_created_cells_processed**
- uint64_t **stats_n_relay_cells_processed**
- uint64_t **stats_n_destroy_cells_processed**
- int **has_completed_circuit**
- uint64_t **stats_n_relay_cells_relayed**
- uint64_t **stats_n_relay_cells_delivered**
- uint64_t **stats_n_data_cells_packaged**
- uint64_t **stats_n_data_bytes_packaged**
- uint64_t **stats_n_data_cells_received**
- uint64_t **stats_n_data_bytes_received**

4.35.1 Detailed Description

Master header file for Tor-specific functionality.

4.35.2 Define Documentation

4.35.2.1 #define AP_CONN_STATE_CIRCUIT_WAIT 8

State for a SOCKS connection: waiting for a completed circuit.

Referenced by ap_stream_wants_exit_attention(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), conn_state_to_string(), connection_ap_attach_pending(), connection_ap_detach_retriable(), connection_ap_fail_onehop(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_ap_process_natd(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_init_accepted_conn(), directory_all_unreachable(), getinfo_helper_events(), rend_client_desc_trynow(), and rend_client_remove_intro_point().

4.35.2.2 #define AP_CONN_STATE_CONNECT_WAIT 9

State for a SOCKS connection: sent BEGIN, waiting for CONNECTED.

Referenced by conn_state_to_string(), connection_ap_handshake_send_begin(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), getinfo_helper_events(), and handle_control_attachstream().

4.35.2.3 #define AP_CONN_STATE_CONTROLLER_WAIT 7

The controller will attach this connection to a circuit; it isn't our job to do so.

Referenced by conn_state_to_string(), connection_ap_detach_retriable(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_process_socks(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), getinfo_helper_events(), and handle_control_attachstream().

4.35.2.4 #define AP_CONN_STATE_IS_UNATTACHED(s) ((s) <= AP_CONN_STATE_CIRCUIT_WAIT || (s) == AP_CONN_STATE_NATD_WAIT)

True iff the AP_CONN_STATE_* value **s** means that the corresponding edge connection is not attached to any circuit.

Referenced by connection_ap_expire_beginning().

4.35.2.5 #define AP_CONN_STATE_NATD_WAIT 12

State for a transparent natd connection: waiting for original destination.

Referenced by conn_state_to_string(), connection_ap_process_natd(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_init_accepted_conn(), and getinfo_helper_events().

4.35.2.6 #define AP_CONN_STATE_OPEN 11

State for a SOCKS connection: ready to send and receive.

Referenced by assert_connection_ok(), conn_state_to_string(), connection_ap_expire_beginning(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_state_is_open(), and getinfo_helper_events().

4.35.2.7 #define AP_CONN_STATE_RENDDESC_WAIT 6

State for a SOCKS connection: got a y.onion URL; waiting to receive rendezvous descriptor.

Referenced by circuit_get_open_circ_or_launch(), conn_state_to_string(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), getinfo_helper_events(), rend_client_desc_trynow(), and rend_client_remove_intro_point().

4.35.2.8 #define AP_CONN_STATE_RESOLVE_WAIT 10

State for a SOCKS connection: sent RESOLVE, waiting for RESOLVED.

Referenced by conn_state_to_string(), connection_ap_handshake_send_resolve(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), dnsserv_launch_request(), evdns_server_callback(), getinfo_helper_events(), and handle_control_attachstream().

4.35.2.9 #define AP_CONN_STATE_SOCKS_WAIT 5

State for a SOCKS connection: waiting for SOCKS request.

Referenced by conn_state_to_string(), connection_ap_handshake_process_socks(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_init_accepted_conn(), and getinfo_helper_events().

4.35.2.10 #define BW_WEIGHT_SCALE 10000

Precision multiplier for the Bw weights

Referenced by networkstatus_verify_bw_weights(), and smartlist_choose_by_bandwidth_weights().

4.35.2.11 #define CBT_BIN_WIDTH ((build_time_t)50)

Width of the histogram bins in milliseconds

Referenced by circuit_build_times_create_histogram(), circuit_build_times_mode(), and circuit_build_times_update_state().

4.35.2.12 #define CBT_DEFAULT_MAX_RECENT_TIMEOUT_COUNT (CBT_DEFAULT_RECENT_CIRCUITS*9/10)

Maximum count of timeouts that finish the first hop in the past RECENT_CIRCUITS before calculating a new timeout.

This tells us to abandon timeout history and set the timeout back to BUILD_TIMEOUT_INITIAL_VALUE.

4.35.2.13 #define CBT_DEFAULT_MIN_CIRCUITS_TO_OBSERVE 100

Minimum circuits before estimating a timeout

4.35.2.14 #define CBT_DEFAULT_QUANTILE_CUTOFF 80

Cutoff percentile on the CDF for our timeout estimation.

4.35.2.15 #define CBT_DEFAULT_RECENT_CIRCUITS 20

How many circuits count as recent when considering if the connection has gone gimpy or changed.

Referenced by circuit_build_times_new_consensus_params().

4.35.2.16 #define CBT_DEFAULT_TEST_FREQUENCY 60

How often in seconds should we build a test circuit

4.35.2.17 #define CBT_DEFAULT_TIMEOUT_INITIAL_VALUE (60*1000)

Initial circuit build timeout in milliseconds

4.35.2.18 #define CBT_DEFAULT_TIMEOUT_MIN_VALUE (2*1000)

Lowest allowable value for CircuitBuildTimeout in milliseconds

4.35.2.19 #define CBT_MAX_SYNTHETIC_QUANTILE 0.985

Maximum quantile to use to generate synthetic timeouts. We want to stay a bit short of 1.0, because longtail is loooooooooooooooooooooooooooooooooooooooooooooooooooooooong.

Referenced by circuit_build_times_add_timeout_worker(), and circuit_build_times_count_pretimeouts().

4.35.2.20 #define CBT_NCIRCUITS_TO_OBSERVE 1000

Total size of the circuit timeout history to accumulate. 1000 is approx 2.5 days worth of continual-use circuits.

Referenced by circuit_build_times_add_time(), circuit_build_times_max(), and circuit_build_times_rewind_history().

4.35.2.21 #define CBT_NETWORK_NONLIVE_DISCARD_COUNT (CBT_NETWORK_NONLIVE_TIMEOUT_COUNT*2)

This tells us when to toss out the last streak of N timeouts.

If instead we start getting cells, we switch back to computing the timeout from our saved history.

Referenced by circuit_build_times_network_check_live().

4.35.2.22 #define CBT_NETWORK_NONLIVE_TIMEOUT_COUNT (3)

Have we received a cell in the last N circ attempts?

This tells us when to temporarily switch back to BUILD_TIMEOUT_INITIAL_VALUE until we start getting cells, at which point we switch back to computing the timeout from our saved history.

Referenced by circuit_build_times_network_check_live().

4.35.2.23 #define CBT_SAVE_STATE_EVERY 10

Save state every 10 circuits

Referenced by circuit_build_times_add_time().

4.35.2.24 #define CELL_COMMAND_IS_VAR_LENGTH(x) ((x) == CELL_VERSIONS)

True iff the cell command **x** is one that implies a variable-length cell.

Referenced by fetch_var_cell_from_buf().

4.35.2.25 #define CELL_NETWORK_SIZE 512

Number of bytes in a cell transmitted over the network.

Referenced by connection_bucket_read_limit(), connection_bucket_write_limit(), connection_or_flush_from_first_active_circuit(), connection_or_flushed_some(), connection_or_process_cells_from_inbuf(), connection_or_write_cell_to_buf(), and router_perform_bandwidth_test().

4.35.2.26 #define CELL_PAYLOAD_SIZE 509

Number of bytes in a cell, minus cell header.

Referenced by cell_pack(), cell_unpack(), command_process_netinfo_cell(), connection_ap_handshake_send_begin(), connection_edge_package_raw_inbuf(), connection_outbuf_too_full(), relay_crypt_one_payload(), relay_digest_matches(), and relay_set_digest().

4.35.2.27 #define CIRCLAUNCH_IS_INTERNAL (1<<3)

Flag to set when the last hop of a circuit doesn't need to be an exit node.

Referenced by circuit_find_to_cannibalize(), circuit_get_open_circ_or_launch(), circuit_predict_and_launch_new(), consider_testing_reachability(), origin_circuit_init(), rend_client_introduction_acked(), rend_service_introduce(), rend_service_launch_establish_intro(), and rend_service_relaunch_rendezvous().

4.35.2.28 #define CIRCLAUNCH_NEED_CAPACITY (1<<2)

Flag to set when a circuit needs to be built of high-capacity nodes

Referenced by circuit_find_to_cannibalize(), circuit_get_open_circ_or_launch(), circuit_predict_and_launch_new(), consider_testing_reachability(), handle_control_extendcircuit(), origin_circuit_init(), rend_service_introduce(), and rend_service_relaunch_rendezvous().

4.35.2.29 #define CIRCLAUNCH_NEED_UPTIME (1<<1)

Flag to set when a circuit needs to be built of high-uptime nodes

Referenced by circuit_find_to_cannibalize(), circuit_get_open_circ_or_launch(), circuit_predict_and_launch_new(), origin_circuit_init(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.35.2.30 #define CIRCLAUNCH_ONEHOP_TUNNEL (1<<0)

Flag to set when a circuit should have only a single hop.

Referenced by circuit_get_open_circ_or_launch(), circuit_launch_by_extend_info(), and origin_circuit_init().

4.35.2.31 #define CIRCUIT_IS_ORIGIN(c) (CIRCUIT_PURPOSE_IS_ORIGIN((c) → purpose))

True iff the **circuit_t** (p. 40) **c** is actually an **origin_circuit_t** (p. 181).

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_deliver_create_cell(), circuit_detach_stream(), circuit_dump_by_conn(), circuit_dump_details(), circuit_enough_testing_circs(), circuit_expire_all_dirty_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_expire_old_circuits_serverside(), circuit_find_to_cannibalize(), circuit_free(), circuit_free_all(), circuit_get_by_circid_orconn_impl(), circuit_get_by_edge_conn(), circuit_get_by_global_id(), circuit_is_acceptable(), circuit_is_better(), circuit_mark_all_unused_circs(), circuit_n_conn_done(), circuit_package_relay_cell(), circuit_predict_and_launch_new(), circuit_receive_relay_cell(), circuit_resume_edge_reading(), circuit_stream_is_being_handled(), circuit_unlink_all_from_or_conn(), command_process_created_cell(), command_process_destroy_cell(), command_process_relay_cell(), connection_ap_handshake_attach_circuit(), connection_cpu_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_or_flush_from_first_active_circuit(), control_event_stream_status(), dns_found_answer(), dump_cell_pool_usage(), getinfo_helper_events(), relay_crypt(), relay_lookup_conn(), relay_send_command_from_edge(), rend_process_relay_cell(), rep_hist_buffer_stats_add_circ(), and set_streams_blocked_on_circ().

4.35.2.32 #define CIRCUIT_PURPOSE_C_ESTABLISH_REND 9

Client-side circuit purpose: at Alice, waiting for ack.

Referenced by choose_good_exit_server(), circuit_build_failed(), circuit_expire_building(), circuit_get_open_circ_or_launch(), circuit_has_opened(), circuit_is_acceptable(), circuit_launch_by_extend_info(), circuit_purpose_to_controller_string(), rend_client_rendcirc_has_opened(), rend_client_rendezvous_acked(), rend_client_send_establish_rendezvous(), and warn_if_last_router_excluded().

4.35.2.33 #define CIRCUIT_PURPOSE_C_GENERAL 5

Client-side circuit purpose: Normal circuit, with cpath.

Referenced by add_an_entry_guard(), choose_good_exit_server(), circuit_build_failed(), circuit_build_needed_circs(), circuit_expire_old_circuits_clientside(), circuit_find_to_cannibalize(), circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_has_opened(), circuit_is_acceptable(), circuit_is_better(), circuit_launch_by_extend_info(), circuit_predict_and_launch_new(), circuit_purpose_from_string(), circuit_purpose_to_controller_string(), circuit_stream_is_being_handled(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), handle_control_extendcircuit(), rend_service_intro_has_opened(), and warn_if_last_router_excluded().

4.35.2.34 #define CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT 7

Client-side circuit purpose: at Alice, sent INTRODUCE1 to intro point, waiting for ACK/NAK.

Referenced by _circuit_mark_for_close(), circuit_expire_building(), circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_is_better(), circuit_purpose_to_controller_string(), connection_ap_handshake_attach_circuit(), rend_client_introduction_acked(), and warn_if_last_router_excluded().

4.35.2.35 #define CIRCUIT_PURPOSE_C_INTRODUCE_ACKED 8

Client-side circuit purpose: at Alice, introduced and acked, closing.

Referenced by circuit_purpose_to_controller_string(), rend_client_introduction_acked(), and warn_if_last_router_excluded().

4.35.2.36 #define CIRCUIT_PURPOSE_C_INTRODUCING 6

Client-side circuit purpose: at Alice, connecting to intro point.

Referenced by circuit_build_failed(), circuit_expire_building(), circuit_get_open_circ_or_launch(), circuit_has_opened(), circuit_is_acceptable(), circuit_launch_by_extend_info(), circuit_purpose_to_controller_string(), connection_ap_handshake_attach_circuit(), rend_client_introcirc_has_opened(), rend_client_introduction_acked(), rend_client_send_introduction(), and warn_if_last_router_excluded().

4.35.2.37 #define CIRCUIT_PURPOSE_C_REND_JOINED 12

Client-side circuit purpose: at Alice, rendezvous established.

Referenced by circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_is_better(), circuit_purpose_to_controller_string(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), rend_client_receive_rendezvous(), and warn_if_last_router_excluded().

4.35.2.38 #define CIRCUIT_PURPOSE_C_REND_READY 10

Client-side circuit purpose: at Alice, waiting for Bob.

Referenced by circuit_expire_building(), circuit_is_acceptable(), circuit_purpose_to_controller_string(), connection_ap_handshake_attach_circuit(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_rendezvous_acked(), rend_client_send_introduction(), and warn_if_last_router_excluded().

4.35.2.39 #define CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED 11

Client-side circuit purpose: at Alice, waiting for Bob, INTRODUCE has been acknowledged.

Referenced by circuit_expire_building(), circuit_is_acceptable(), circuit_purpose_to_controller_string(), connection_ap_handshake_attach_circuit(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), and warn_if_last_router_excluded().

4.35.2.40 #define CIRCUIT_PURPOSE_CONTROLLER 18

A controller made this circuit and Tor should not use it.

Referenced by circuit_purpose_from_string(), circuit_purpose_to_controller_string(), and warn_if_last_router_excluded().

4.35.2.41 #define CIRCUIT_PURPOSE_INTRO_POINT 2

OR-side circuit purpose: At OR, from Bob, waiting for intro from Alices.

Referenced by circuit_get_intro_point(), circuit_purpose_to_controller_string(), rend_mid_establish_intro(), and warn_if_last_router_excluded().

4.35.2.42 #define CIRCUIT_PURPOSE_IS_CLIENT(p)

Value:

```
((p)> _CIRCUIT_PURPOSE_OR_MAX &&    \
   (p)<=_CIRCUIT_PURPOSE_C_MAX)
```

True iff the circuit purpose **p** is for a circuit that originated here to serve as a client. (Hidden services don't count here.)

Referenced by connection_edge_end(), and relay_send_end_cell_from_edge().

4.35.2.43 #define CIRCUIT_PURPOSE_IS_ESTABLISHED_REND(p)

Value:

```
((p) == CIRCUIT_PURPOSE_C_REND_JOINED ||     \
   (p) == CIRCUIT_PURPOSE_S_REND_JOINED)
```

True iff the circuit purpose **p** is for an established rendezvous circuit.

Referenced by relay_send_command_from_edge().

4.35.2.44 #define CIRCUIT_PURPOSE_IS_ORIGIN(p) ((p)>_CIRCUIT_PURPOSE_OR_MAX)

True iff the circuit purpose **p** is for a circuit that originated at this node.

Referenced by circuit_get_by_rend_query_and_purpose(), and circuit_get_next_by_pk_and_purpose().

4.35.2.45 #define CIRCUIT_PURPOSE_OR 1

OR-side circuit purpose: normal circuit, at OR.

Referenced by circuit_purpose_to_controller_string(), command_process_create_cell(), connection_edge_process_relay_cell(), connection_exit_begin_conn(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), and warn_if_last_router_excluded().

4.35.2.46 #define CIRCUIT_PURPOSE_REND_ESTABLISHED 4

OR-side circuit purpose: At OR, both circuits have this purpose.

Referenced by assert_circuit_ok(), circuit_purpose_to_controller_string(), circuit_receive_relay_cell(), rend_mid_rendezvous(), and warn_if_last_router_excluded().

4.35.2.47 #define CIRCUIT_PURPOSE_REND_POINT_WAITING 3

OR-side circuit purpose: At OR, from Alice, waiting for Bob.

Referenced by circuit_get_rendezvous(), circuit_purpose_to_controller_string(), rend_mid_establish_rendezvous(), and warn_if_last_router_excluded().

4.35.2.48 #define CIRCUIT_PURPOSE_S_CONNECT_REND 15

Hidden-service-side circuit purpose: at Bob, connecting to rend point.

Referenced by circuit_build_failed(), circuit_has_opened(), circuit_launch_by_extend_info(), circuit_purpose_to_controller_string(), rend_service_introduce(), rend_service_relaunch_rendezvous(), rend_service_rendezvous_has_opened(), and warn_if_last_router_excluded().

4.35.2.49 #define CIRCUIT_PURPOSE_S_ESTABLISH_INTRO 13

Hidden-service-side circuit purpose: at Bob, waiting for introductions.

Referenced by circuit_build_failed(), circuit_expire_building(), circuit_has_opened(), circuit_launch_by_extend_info(), circuit_purpose_to_controller_string(), count_established_intro_points(), find_intro_circuit(), new_route_len(), rend_config_services(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_launch_establish_intro(), and warn_if_last_router_excluded().

4.35.2.50 #define CIRCUIT_PURPOSE_S_INTRO 14

Hidden-service-side circuit purpose: at Bob, successfully established intro.

Referenced by circuit_purpose_to_controller_string(), count_established_intro_points(), find_intro_circuit(), rend_config_services(), rend_service_intro_established(), rend_service_introduce(), and rend_service_update_descriptor().

4.35.2.51 #define CIRCUIT_PURPOSE_S_REND_JOINED 16

Hidden-service-side circuit purpose: at Bob, rendezvous established.

Referenced by circuit_purpose_to_controller_string(), connection_edge_process_relay_cell(), connection_exit_begin_conn(), rend_service_rendezvous_has_opened(), rend_service_set_connection_addr_port(), and warn_if_last_router_excluded().

4.35.2.52 #define CIRCUIT_PURPOSE_TESTING 17

A testing circuit; not meant to be used for actual traffic.

Referenced by choose_good_entry_server(), circuit_build_failed(), circuit_enough_testing_circs(), circuit_has_opened(), circuit_launch_by_extend_info(), circuit_purpose_to_controller_string(), consider_testing_reachability(), new_route_len(), router_perform_bandwidth_test(), and warn_if_last_router_excluded().

4.35.2.53 #define CIRCUIT_PURPOSE_UNKNOWN 255

A catch-all for unrecognized purposes. Currently we don't expect to make or see any circuits with this purpose.

Referenced by circuit_purpose_from_string(), handle_control_extendcircuit(), and handle_control_setcircuitpurpose().

4.35.2.54 #define CIRCUIT_STATE_BUILDING 0

Circuit state: I'm the origin, still haven't done all my handshakes.

Referenced by circuit_extend_to_new_exit(), circuit_send_next_onion_skin(), circuit_state_to_string(), and handle_control_extendcircuit().

4.35.2.55 #define CIRCUIT_STATE_ONIONSKIN_PENDING 1

Circuit state: Waiting to process the onionskin.

Referenced by _circuit_mark_for_close(), circuit_state_to_string(), command_process_create_cell(), and command_process_relay_cell().

4.35.2.56 #define CIRCUIT_STATE_OPEN 3

Circuit state: onionskin(s) processed, ready to send/receive cells.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_build_failed(), circuit_dump_by_conn(), circuit_enough_testing_circs(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_find_to_cannibalize(), circuit_get_best(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_list_path_impl(), circuit_n_conn_done(), circuit_send_next_onion_skin(), circuit_set_state(), circuit_state_to_string(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_attach_circuit(), count_established_intro_points(), getinfo_helper_events(), handle_control_attachstream(), handle_control_extendcircuit(), onionskin_answer(), rend_config_services(), rend_service_launch_establish_intro(), and router_perform_bandwidth_test().

4.35.2.57 #define CIRCUIT_STATE_OR_WAIT 2

Circuit state: I'd like to deliver a create, but my n_conn is still connecting.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_extend(), circuit_get_all_pending_on_or_conn(), circuit_n_conn_done(), circuit_set_state(), circuit_state_to_string(), and origin_circuit_init().

4.35.2.58 #define CIRCWINDOW_INCREMENT 100

Amount to increment a circuit window when we get a circuit SENDME.

Referenced by circuit_consider_sending_sendme(), and connection_edge_process_relay_cell().

4.35.2.59 #define CIRCWINDOW_START 1000

Initial value for both sides of a circuit transmission window when the circuit is initialized. Measured in cells.

Referenced by circuit_consider_sending_sendme(), circuit_initial_package_window(), init_circuit_base(), onion_append_hop(), rend_client_receive_rendezvous(), rend_service_rendezvous_has_opened(), and router_perform_bandwidth_test().

4.35.2.60 #define CONN_IS_EDGE(x) ((x) → type == CONN_TYPE_EXIT || (x) → type == CONN_TYPE_AP)

True iff **x** is an edge connection.

Referenced by _connection_free(), _connection_write_to_buf_impl(), assert_connection_ok(), conn_read_callback(), conn_write_callback(), connection_about_to_close_connection(), connection_get_by_type_state_rendquery(), connection_handle_read_impl(), connection_handle_write_impl(), and hibernate_go_dormant().

4.35.2.61 #define CONN_LOG_PROTECT(conn, stmt)

Value:

```
STMT_BEGIN                                                    \
    int _log_conn_is_control = (conn && conn->type == CONN_TYPE_CONTROL); \
    if (_log_conn_is_control)                                   \
      disable_control_logging();                                \
  STMT_BEGIN stmt; STMT_END;                                    \
    if (_log_conn_is_control)                                   \
      enable_control_logging();                                 \
  STMT_END
```

Execute the statement **stmt**, which may log events concerning the connection **conn**. To prevent infinite loops, disable log messages being sent to controllers if **conn** is a control connection.

Stmt must not contain any return or goto statements.

Referenced by _connection_write_to_buf_impl(), connection_handle_write_impl(), and connection_read_to_buf().

4.35.2.62 #define CONN_TYPE_AP 7

A SOCKS proxy connection from the user application to the onion proxy.

Referenced by _connection_free(), _connection_mark_unattached_ap(), ap_stream_wants_exit_attention(), assert_connection_ok(), circuit_discard_optional_exit_enclaves(), conn_state_to_string(), conn_type_to_string(), connection_about_to_close_connection(), connection_add(), connection_ap_attach_pending(), connection_ap_can_use_exit(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_ap_handshake_process_socks(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_destroy(), connection_edge_process_relay_cell_not_open(), connection_edge_send_command(), connection_finished_flushing(), connection_get_by_type_state_rendquery(), connection_handle_listener_read(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init(), connection_init_accepted_conn(), connection_new(), connection_process_inbuf(), connection_reached_eof(), connection_read_to_buf(), connection_state_is_open(), control_event_stream_bandwidth_used(), directory_all_unreachable(), dnsserv_launch_request(), edge_connection_new(), evdns_server_callback(), get_stream(), getinfo_helper_events(), hibernate_go_dormant(), rend_client_desc_trynow(), and rend_client_remove_intro_point().

4.35.2.63 #define CONN_TYPE_AP_DNS_LISTENER 15

Type for sockets listening for DNS requests.

Referenced by assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_create_listener(), connection_handle_read_impl(), connection_is_listener(), dnsserv_close_listener(), dnsserv_configure_listener(), hibernate_begin(), and retry_all_listeners().

4.35.2.64 #define CONN_TYPE_AP_LISTENER 6

Type for sockets listening for SOCKS connections.

Referenced by assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_handle_read_impl(), connection_init_accepted_conn(), connection_is_listener(), hibernate_begin(), and retry_all_listeners().

4.35.2.65 #define CONN_TYPE_AP_NATD_LISTENER 14

Type for sockets listening for transparent connections redirected by natd.

Referenced by assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_handle_read_impl(), connection_init_accepted_conn(), connection_is_listener(), hibernate_begin(), and retry_all_listeners().

4.35.2.66 #define CONN_TYPE_AP_TRANS_LISTENER 13

Type for sockets listening for transparent connections redirected by pf or netfilter.

Referenced by assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_handle_read_impl(), connection_init_accepted_conn(), connection_is_listener(), hibernate_begin(), and retry_all_listeners().

4.35.2.67 #define CONN_TYPE_CONTROL 12

Type for connections from user interface process.

Referenced by _connection_free(), _connection_write_to_buf_impl(), assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_finished_flushing(), connection_free(), connection_free_all(), connection_handle_read_impl(), connection_init(), connection_init_accepted_conn(), connection_new(), connection_process_inbuf(), connection_reached_eof(), connection_state_is_open(), control_connection_new(), control_update_global_event_mask(), and send_control_event_string().

4.35.2.68 #define CONN_TYPE_CONTROL_LISTENER 11

Type for listening for connections from user interface process.

Referenced by _connection_free(), assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_create_listener(), connection_handle_listener_read(), connection_handle_read_impl(), connection_is_listener(), and retry_all_listeners().

4.35.2.69 #define CONN_TYPE_CPUWORKER 10

Connection from the main process to a CPU worker process.

Referenced by assert_connection_ok(), assign_onionskin_to_cpuworker(), conn_state_to_string(), conn_type_to_string(), connection_cpu_finished_flushing(), connection_cpu_process_inbuf(), connection_finished_flushing(), connection_process_inbuf(), connection_reached_eof(), cpuworkers_rotate(), cull_wedged_cpuworkers(), and spawn_cpuworker().

4.35.2.70 #define CONN_TYPE_DIR 9

Type for HTTP connections to the directory server.

Referenced by _connection_free(), any_pending_bridge_descriptor_fetches(), assert_connection_ok(), circuit_get_open_circ_or_launch(), conn_close_if_marked(), conn_state_to_string(), conn_type_to_string(), connection_about_to_close_connection(), connection_bucket_read_limit(), connection_bucket_write_limit(), connection_counts_as_relayed_traffic(), connection_dir_finished_connecting(), connection_dir_finished_flushing(), connection_dir_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_exit_connect_dir(), connection_expire_held_open(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_get_by_type_state_rendquery(), connection_handle_listener_read(), connection_handle_read_impl(), connection_init(), connection_init_accepted_conn(), connection_new(), connection_process_inbuf(), connection_reached_eof(), connection_state_is_connecting(), consider_testing_reachability(), control_event_stream_status(), dir_connection_new(), directory_handle_command(), directory_send_command(), launch_direct_bridge_descriptor_fetch(), list_pending_downloads(), router_pick_trusteddirserver_impl(), run_connection_housekeeping(), update_consensus_networkstatus_downloads(), and update_v2_networkstatus_cache_downloads().

4.35.2.71 #define CONN_TYPE_DIR_LISTENER 8

Type for sockets listening for HTTP connections to the directory server.

Referenced by assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_handle_read_impl(), connection_is_listener(), hibernate_begin(), and retry_all_listeners().

4.35.2.72 #define CONN_TYPE_EXIT 5

A TCP connection from an onion router to a stream's destination.

Referenced by _connection_free(), assert_circuit_ok(), assert_connection_ok(), conn_close_if_marked(), conn_state_to_string(), conn_type_to_string(), connection_about_to_close_connection(), connection_buckets_decrement(), connection_dns_remove(), connection_edge_finished_connecting(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_expire_held_open(), connection_finished_connecting(), connection_finished_flushing(), connection_get_by_type_state_rendquery(), connection_init(), connection_new(), connection_process_inbuf(), connection_reached_eof(), connection_state_is_connecting(), connection_state_is_open(), connection_unlink(), edge_connection_new(), and hibernate_go_dormant().

4.35.2.73 #define CONN_TYPE_OR 4

A bidirectional TLS connection transmitting a sequence of cells. May be from an OR to an OR, or from an OP to an OR.

Referenced by _connection_free(), _connection_write_to_buf_impl(), assert_connection_ok(), circuit_dump_by_conn(), conn_close_if_marked(), conn_state_to_string(), conn_type_to_string(), connection_about_to_close_connection(), connection_counts_as_relayed_traffic(), connection_cpu_process_inbuf(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init(), connection_init_accepted_conn(), connection_new(), connection_or_clear_identity_map(), connection_or_get_for_extend(), connection_or_nonopen_was_started_here(), connection_process_inbuf(), connection_reached_eof(), connection_state_is_connecting(), connection_state_is_open(), connection_unlink(), dumpstats(), getinfo_helper_events(), hibernate_go_dormant(), or_connection_new(), and run_connection_housekeeping().

4.35.2.74 #define CONN_TYPE_OR_LISTENER 3

Type for sockets listening for OR connections.

Referenced by assert_connection_ok(), conn_state_to_string(), conn_type_to_string(), connection_handle_read_impl(), connection_is_listener(), hibernate_begin(), and retry_all_listeners().

4.35.2.75 #define CONTROL_CONN_STATE_NEEDAUTH 2

State for a control connection: Waiting for authentication; speaking protocol v1.

Referenced by conn_state_to_string(), connection_control_process_inbuf(), and connection_init_accepted_conn().

4.35.2.76 #define CONTROL_CONN_STATE_OPEN 1

State for a control connection: Authenticated and accepting v1 commands.

Referenced by conn_state_to_string(), connection_control_process_inbuf(), connection_state_is_open(), handle_control_authenticate(), is_valid_initial_command(), and send_control_event_string().

4.35.2.77 #define CPUWORKER_STATE_BUSY_ONION 2

State for a connection to a cpuworker process that's processing a handshake.

Referenced by assign_onionskin_to_cpuworker(), conn_state_to_string(), connection_cpu_process_inbuf(), and cull_wedged_cpuworkers().

4.35.2.78 #define CPUWORKER_STATE_IDLE 1

State for a connection to a cpuworker process that's idle.

Referenced by assign_onionskin_to_cpuworker(), conn_state_to_string(), connection_cpu_process_inbuf(), connection_cpu_reached_eof(), cpuworkers_rotate(), and spawn_cpuworker().

4.35.2.79 #define DEFAULT_CLIENT_NICKNAME "client"

Name to use in client TLS certificates if no nickname is given. Once Tor 0.1.2.x is obsolete, we can remove this.

Referenced by dirserv_load_fingerprint_file().

4.35.2.80 #define DEFAULT_DNS_TTL (30*60)

How long do we cache/tell clients to cache DNS records when no TTL is known?

Referenced by client_dns_set_addressmap_impl(), and dns_resolve_impl().

4.35.2.81 #define DIR_CONN_IS_SERVER(conn) ((conn) → purpose == DIR_PURPOSE_SERVER)

True iff the purpose of **conn** means that it's a server-side directory connection.

Referenced by connection_counts_as_relayed_traffic(), and run_connection_housekeeping().

4.35.2.82 #define DIR_CONN_STATE_CLIENT_FINISHED 4

State for connection to directory server: happy and finished.

Referenced by conn_state_to_string(), connection_about_to_close_connection(), and connection_dir_reached_eof().

4.35.2.83 #define DIR_CONN_STATE_CLIENT_READING 3

State for connection to directory server: reading HTTP response.

Referenced by conn_state_to_string(), connection_dir_finished_flushing(), and connection_dir_reached_eof().

4.35.2.84 #define DIR_CONN_STATE_CLIENT_SENDING 2

State for connection to directory server: sending HTTP request.

Referenced by conn_state_to_string(), connection_dir_finished_connecting(), connection_dir_finished_flushing(), and directory_initiate_command_rend().

4.35.2.85 #define DIR_CONN_STATE_CONNECTING 1

State for connection to directory server: waiting for connect().

Referenced by conn_state_to_string(), connection_dir_finished_connecting(), connection_state_is_connecting(), and directory_initiate_command_rend().

4.35.2.86 #define DIR_CONN_STATE_SERVER_COMMAND_WAIT 5

State for connection at directory server: waiting for HTTP request.

Referenced by conn_state_to_string(), connection_dir_process_inbuf(), connection_exit_connect_dir(), and connection_init_accepted_conn().

4.35.2.87 #define DIR_CONN_STATE_SERVER_WRITING 6

State for connection at directory server: sending HTTP response.

Referenced by conn_state_to_string(), connection_dir_finished_flushing(), connection_dirserv_flushed_some(), connection_flushed_some(), directory_handle_command_get(), and directory_handle_command_post().

4.35.2.88 #define DIR_ENTRY_RECORD_USAGE_RETAIN_IPS (24*60*60)

Time interval: Flush geoip data to disk this often.

Referenced by geoip_bridge_stats_write().

4.35.2.89 #define DIR_PURPOSE_FETCH_CERTIFICATE 15

A connection to a directory server: download one or more directory authority certificates.

Referenced by authority_certs_fetch_missing(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_request_failed(), connection_edge_process_relay_cell_not_open(), dir_conn_purpose_to_string(), directory_get_from_dirserver(), directory_send_command(), and purpose_needs_anonymity().

4.35.2.90 #define DIR_PURPOSE_FETCH_CONSENSUS 14

A connection to a directory server: download a v3 networkstatus consensus.

Referenced by connection_dir_client_reached_eof(), connection_dir_request_failed(), connection_edge_process_relay_cell_not_open(), dir_conn_purpose_to_string(), directory_get_from_dirserver(), directory_send_command(), purpose_needs_anonymity(), and update_consensus_networkstatus_downloads().

4.35.2.91 #define DIR_PURPOSE_FETCH_DETACHED_SIGNATURES 13

A connection to a directory server: download a v3 detached signatures object for a consensus.

Referenced by connection_dir_client_reached_eof(), connection_dir_request_failed(), dir_conn_purpose_to_string(), directory_get_from_all_authorities(), directory_get_from_dirserver(), directory_send_command(), dirvote_fetch_missing_signatures(), and purpose_needs_anonymity().

4.35.2.92 #define DIR_PURPOSE_FETCH_EXTRAINFO 7

A connection to a directory server: download one or more extra-info documents.

Referenced by connection_dir_client_reached_eof(), connection_dir_download_routerdesc_failed(), connection_dir_request_failed(), dir_conn_purpose_to_string(), directory_get_from_dirserver(), directory_send_command(), list_pending_descriptor_downloads(), purpose_needs_anonymity(), router_pick_trusteddirserver_impl(), and update_extrainfo_downloads().

4.35.2.93 #define DIR_PURPOSE_FETCH_RENDDESC 3

A connection to a directory server: download a rendezvous descriptor.

Referenced by body_is_plausible(), connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), and directory_get_from_dirserver().

4.35.2.94 #define DIR_PURPOSE_FETCH_RENDDESC_V2 18

A connection to a hidden service directory server: download a v2 rendezvous descriptor.

Referenced by connection_about_to_close_connection(), connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), directory_get_from_hs_dir(), and directory_send_command().

4.35.2.95 #define DIR_PURPOSE_FETCH_SERVERDESC 6

A connection to a directory server: download one or more server descriptors.

Referenced by any_pending_bridge_descriptor_fetches(), connection_dir_client_reached_eof(), connection_dir_download_routerdesc_failed(), connection_dir_request_failed(), connection_edge_process_relay_cell_not_open(), consider_testing_reachability(), dir_conn_purpose_to_string(), directory_get_from_dirserver(), directory_send_command(), fetch_bridge_descriptors(), launch_direct_bridge_descriptor_fetch(), launch_router_descriptor_downloads(), list_pending_descriptor_downloads(), purpose_needs_anonymity(), router_pick_trusteddirserver_impl(), run_connection_housekeeping(), and update_router_descriptor_downloads().

4.35.2.96 #define DIR_PURPOSE_FETCH_STATUS_VOTE 12

A connection to a directory server: download one or more v3 networkstatus votes.

Referenced by connection_dir_client_reached_eof(), connection_dir_request_failed(), dir_conn_purpose_to_string(), directory_get_from_all_authorities(), directory_get_from_dirserver(), directory_send_command(), and purpose_needs_anonymity().

4.35.2.97 #define DIR_PURPOSE_FETCH_V2_NETWORKSTATUS 5

A connection to a directory server: download one or more v2 network-status objects

Referenced by connection_dir_client_reached_eof(), connection_dir_request_failed(), dir_conn_purpose_to_string(), directory_get_from_dirserver(), directory_send_command(), purpose_needs_anonymity(), and update_v2_networkstatus_cache_downloads().

4.35.2.98 #define DIR_PURPOSE_HAS_FETCHED_RENDDESC 4

A connection to a directory server: set after a rendezvous descriptor is downloaded.

Referenced by connection_dir_client_reached_eof().

4.35.2.99 #define DIR_PURPOSE_IS_UPLOAD(p)

Value:

```
((p)==DIR_PURPOSE_UPLOAD_DIR ||              \
   (p)==DIR_PURPOSE_UPLOAD_RENDDESC ||          \
   (p)==DIR_PURPOSE_UPLOAD_VOTE ||              \
   (p)==DIR_PURPOSE_UPLOAD_SIGNATURES)
```

True iff **p** is a purpose corresponding to uploading data to a directory server.

Referenced by control_event_stream_status().

4.35.2.100 #define DIR_PURPOSE_SERVER 16

Purpose for connection at a directory server.

Referenced by conn_close_if_marked(), connection_exit_connect_dir(), connection_expire_held_open(), and connection_init_accepted_conn().

4.35.2.101 #define DIR_PURPOSE_UPLOAD_DIR 8

A connection to a directory server: upload a server descriptor.

Referenced by connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), directory_post_to_dirservers(), directory_send_command(), purpose_needs_anonymity(), and router_upload_dir_desc_to_dirservers().

4.35.2.102 #define DIR_PURPOSE_UPLOAD_RENDDESC 9

A connection to a directory server: upload a rendezvous descriptor.

Referenced by connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), and directory_send_command().

4.35.2.103 #define DIR_PURPOSE_UPLOAD_RENDDESC_V2 17

A connection to a hidden service directory server: upload a v2 rendezvous descriptor.

Referenced by connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), directory_post_to_hs_dir(), and directory_send_command().

4.35.2.104 #define DIR_PURPOSE_UPLOAD_SIGNATURES 11

A connection to a directory server: upload a v3 consensus signature

Referenced by connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), directory_send_command(), dirvote_compute_consensuses(), and purpose_needs_anonymity().

4.35.2.105 #define DIR_PURPOSE_UPLOAD_VOTE 10

A connection to a directory server: upload a v3 networkstatus vote.

Referenced by connection_dir_client_reached_eof(), dir_conn_purpose_to_string(), directory_send_command(), dirvote_perform_vote(), and purpose_needs_anonymity().

4.35.2.106 #define DIR_RECORD_USAGE_GRANULARITY 8

Round all GeoIP results to the next multiple of this value, to avoid leaking information.

Referenced by geoip_get_client_history_dirreq().

4.35.2.107 #define DIR_RECORD_USAGE_MIN_OBSERVATION_TIME (12*60*60)

How long do we have to have observed per-country request history before we are willing to talk about it?

Referenced by geoip_get_client_history_dirreq().

4.35.2.108 #define DOWNCAST(to, ptr) ((to*)SUBTYPE_P(ptr, to, _base))

Helper macro: Given a pointer to to._base, of type from*, return &to.

Referenced by TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CIRCUIT(), TO_OR_CONN(), and TO_ORIGIN_CIRCUIT().

4.35.2.109 #define download_status_failed(dls, sc)

Value:

```
download_status_increment_failure((dls), (sc), NULL,                 \
                                  get_options()->DirPort, time(NULL))
```

Increment the failure count of the **download_status_t** (p. 88) **dls**, with the optional status code **sc**.

Referenced by authority_cert_dl_failed(), connection_dir_download_networkstatus_failed(), dir_networkstatus_download_failed(), fetch_bridge_descriptors(), networkstatus_consensus_download_failed(), networkstatus_set_current_consensus(), router_set_networkstatus_v2(), and update_consensus_networkstatus_downloads().

4.35.2.110 #define END_CIRC_AT_ORIGIN -1

Catch-all "other" reason for closing origin circuits.

Referenced by _circuit_mark_for_close(), circuit_end_reason_to_control_string(), and circuit_testing_opened().

4.35.2.111 #define END_CIRC_REASON_FLAG_REMOTE 512

Bitwise-OR this with the argument to circuit_mark_for_close() or **control_event_circuit_status()** (p. 493) to indicate that the reason was passed through from a destroy or truncate cell.

Referenced by _circuit_mark_for_close(), circuit_end_reason_to_control_string(), circuit_truncated(), command_process_destroy_cell(), and control_event_circuit_status().

4.35.2.112 #define END_CIRC_REASON_NOPATH -2

We couldn't build a path for this circuit.

Referenced by circuit_end_reason_to_control_string(), and circuit_establish_circuit().

4.35.2.113 #define END_STREAM_REASON_CANT_ATTACH 257

We were unable to attach the connection to any circuit at all.

Referenced by connection_ap_attach_pending(), connection_ap_expire_beginning(), connection_ap_make_link(), connection_edge_package_raw_inbuf(), rend_client_desc_trynow(), stream_end_reason_to_control_string(), and stream_end_reason_to_socks5_response().

4.35.2.114 #define END_STREAM_REASON_CANT_FETCH_ORIG_DEST 260

This is a transparent proxy connection, but we can't extract the original target address:port.

Referenced by connection_ap_process_transparent().

4.35.2.115 #define END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED 1024

Bitwise-or this with the argument to control_event_stream_status to indicate that we already sent a CLOSED stream event.

Referenced by _circuit_mark_for_close(), connection_edge_destroy(), and control_event_stream_status().

4.35.2.116 #define END_STREAM_REASON_FLAG_ALREADY_SOCKS_REPLIED 2048

Bitwise-or this with endreason to indicate that we already sent a socks reply, and no further reply needs to be sent from connection_mark_unattached_ap().

Referenced by _connection_mark_unattached_ap(), connection_ap_handshake_process_socks(), and connection_edge_process_relay_cell_not_open().

4.35.2.117 #define END_STREAM_REASON_FLAG_REMOTE 512

Bitwise-or this with the argument to control_event_stream_status to indicate that the reason came from an END cell.

Referenced by connection_ap_process_end_not_open(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), and control_event_stream_status().

4.35.2.118 #define END_STREAM_REASON_INVALID_NATD_DEST 261

This is a connection on the NATD port, and the destination IP:Port was either ill-formed or out-of-range.

Referenced by connection_ap_process_natd().

4.35.2.119 #define END_STREAM_REASON_MASK 511

Bitwise-and this value with endreason to mask out all flags.

Referenced by stream_end_reason_to_control_string(), and stream_end_reason_to_socks5_response().

4.35.2.120 #define END_STREAM_REASON_NET_UNREACHABLE 258

We can't connect to any directories at all, so we killed our streams before they can time out.

Referenced by directory_all_unreachable(), stream_end_reason_to_control_string(), and stream_end_reason_to_socks5_response().

4.35.2.121 #define END_STREAM_REASON_SOCKSPROTOCOL 259

This is a SOCKS connection, and the client used (or misused) the SOCKS protocol in a way we couldn't handle.

Referenced by connection_ap_handshake_process_socks(), stream_end_reason_to_control_string(), and stream_end_reason_to_socks5_response().

4.35.2.122 #define EXIT_CONN_STATE_CONNECTING 2

State for an exit connection: waiting for connect() to finish.

Referenced by conn_state_to_string(), connection_edge_finished_connecting(), connection_edge_process_inbuf(), connection_exit_begin_conn(), connection_exit_connect(), connection_state_is_connecting(), and dns_found_answer().

4.35.2.123 #define EXIT_CONN_STATE_OPEN 3

State for an exit connection: open and ready to transmit data.

Referenced by conn_state_to_string(), connection_edge_finished_connecting(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_exit_connect(), connection_exit_connect_dir(), and connection_state_is_open().

4.35.2.124 #define EXIT_CONN_STATE_RESOLVEFAILED 4

State for an exit connection: waiting to be removed.

Referenced by conn_state_to_string(), connection_exit_begin_conn(), connection_exit_begin_resolve(), dns_cancel_pending_resolve(), and dns_found_answer().

4.35.2.125 #define EXIT_CONN_STATE_RESOLVING 1

State for an exit connection: waiting for response from DNS farm.

Referenced by conn_state_to_string(), connection_about_to_close_connection(), connection_dns_remove(), and dns_resolve().

4.35.2.126 #define EXIT_PURPOSE_CONNECT 1

This exit stream wants to do an ordinary connect.

Referenced by assert_connection_ok(), connection_exit_begin_conn(), and dns_found_answer().

4.35.2.127 #define EXIT_PURPOSE_RESOLVE 2

This exit stream wants to do a resolve (either normal or reverse).

Referenced by assert_connection_ok(), connection_exit_begin_resolve(), and dns_resolve().

4.35.2.128 #define get_datadir_fname(sub1) get_datadir_fname2_suffix((sub1), NULL, NULL)

Return a newly allocated string containing datadir/sub1. See get_datadir_fname2_suffix.

Referenced by dirserv_load_fingerprint_file(), dirvote_compute_consensuses(), dump_desc(), geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), get_cookie_file(), get_microdesc_cache(), init_keys(), load_bridge_stats(), load_stats_file(), networkstatus_set_current_consensus(), or_state_load(), or_state_save(), read_bandwidth_usage(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_load_mtbf_data(), rep_hist_record_mtbf_data(), router_rebuild_store(), router_reload_consensus_networkstatus(), router_reload_router_list_impl(), router_reload_v2_networkstatus(), and trusted_dirs_reload_certs().

4.35.2.129 #define get_datadir_fname2(sub1, sub2) get_datadir_fname2_suffix((sub1), (sub2), NULL)

Return a newly allocated string containing datadir/sub1/sub2. See get_datadir_fname2_suffix.

Referenced by geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), init_keys(), load_authority_keyset(), load_bridge_stats(), networkstatus_get_cache_filename(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rotate_onion_key(), and router_reload_v2_networkstatus().

4.35.2.130 #define get_datadir_fname_suffix(sub1, suffix) get_datadir_fname2_suffix((sub1), NULL, (suffix))

Return a newly allocated string containing datadir/sub1suffix. See get_datadir_fname2_suffix.

Referenced by router_rebuild_store(), router_reload_router_list_impl(), and signed_desc_append_to_journal().

4.35.2.131 #define IMPOSSIBLE_TO_DOWNLOAD 255

If n_download_failures is this high, the download can never happen.

Referenced by download_status_increment_failure(), download_status_mark_impossible(), and fetch_bridge_descriptors().

4.35.2.132 #define LEGAL_NICKNAME_CHARACTERS "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

Legal characters in a nickname.

Referenced by get_default_nickname(), and is_legal_nickname().

4.35.2.133 #define LISTENER_STATE_READY 0

State for any listener connection.

Referenced by assert_connection_ok(), conn_state_to_string(), and connection_create_listener().

4.35.2.134 #define LOG_FN_CONN(conn, args) CONN_LOG_PROTECT(conn, log_fn args)

Log information about the connection **conn**, protecting it as with CONN_LOG_PROTECT. Example:

LOG_FN_CONN(conn, (LOG_DEBUG, "Socket %d wants to write", conn->s));

Referenced by conn_close_if_marked(), conn_write_callback(), connection_bucket_refill(), connection_consider_empty_read_buckets(), and connection_consider_empty_write_buckets().

4.35.2.135 #define LOG_PROTOCOL_WARN

Value:

```
(get_options()->ProtocolWarnings ? \
                          LOG_WARN : LOG_INFO)
```

4.35.2.136 #define MAX_BUF_SIZE ((1<<24)-1)

Maximum size, in bytes, for resized buffers.

4.35.2.137 #define MAX_DESCRIPTOR_UPLOAD_SIZE 20000

Maximum size, in bytes, of a single router descriptor uploaded to us as a directory authority. Caches and clients fetch whatever descriptors the authorities tell them to fetch, and don't care about size.

Referenced by dirserv_add_descriptor().

4.35.2.138 #define MAX_DIR_DL_SIZE MAX_BUF_SIZE

Maximum size, in bytes, for any directory object that we've downloaded.

Referenced by connection_dir_client_reached_eof().

4.35.2.139 #define MAX_DIR_UL_SIZE MAX_BUF_SIZE

Maximum size, in bytes, for any directory object that we're accepting as an upload.

Referenced by directory_handle_command().

4.35.2.140 #define MAX_DNS_ENTRY_AGE (30*60)

How long do we keep DNS cache entries before purging them (regardless of their TTL)?

Referenced by client_dns_incr_failures(), and dns_get_expiry_ttl().

4.35.2.141 #define MAX_DNS_TTL (3*60*60)

How long can a TTL be before we stop believing it?

Referenced by dns_clip_ttl().

4.35.2.142 #define MAX_EXITPOLICY_SUMMARY_LEN (1000)

Maximum length of an exit policy summary.

Referenced by policy_summarize().

4.35.2.143 #define MAX_EXTRAINFO_UPLOAD_SIZE 50000

Maximum size of a single extrainfo document, as above.

Referenced by dirserv_add_extrainfo(), and router_rebuild_descriptor().

4.35.2.144 #define MAX_FLAG_LINE_LEN 96

Length of "r Authority BadDirectory BadExit Exit Fast Guard HSDir Named Running Stable Unnamed V2Dir Valid

".

4.35.2.145 #define MAX_HEADERS_SIZE 50000

For HTTP parsing: Maximum number of bytes we'll accept in the headers of an HTTP request or response.

Referenced by connection_dir_client_reached_eof(), connection_read_https_proxy_response(), and directory_handle_command().

4.35.2.146 #define MAX_HEX_NICKNAME_LEN (HEX_DIGEST_LEN+1)

Length of a router identity encoded as a hexadecimal digest, plus possible dollar sign.

Referenced by rend_service_introduce().

4.35.2.147 #define MAX_NETWORKSTATUS_AGE (10*24*60*60)

How old do we allow a v2 network-status to get before removing it completely?

Referenced by networkstatus_v2_list_clean(), and update_router_descriptor_cache_downloads_v2().

4.35.2.148 #define MAX_NICKNAME_LEN 19

Length of longest allowable configured nickname.

Referenced by circuit_list_path_impl(), dirserv_load_fingerprint_file(), entry_guards_parse_state(), entry_guards_update_state(), get_default_nickname(), init_keys(), is_legal_nickname(), list_single_server_status(), rend_mid_introduce(), rend_service_introduce(), router_get_verbose_nickname(), and routerstatus_get_verbose_nickname().

4.35.2.149 #define MAX_POLICY_LINE_LEN (3+MAX_EXITPOLICY_SUMMARY_LEN)

Maximum length of an exit policy summary line.

4.35.2.150 #define MAX_RELAY_EARLY_CELLS_PER_CIRCUIT 8

Largest number of relay_early cells that we can send on a given circuit.

Referenced by or_circuit_new(), and origin_circuit_new().

4.35.2.151 #define MAX_ROUTERDESC_DOWNLOAD_FAILURES 8

How many times will we try to download a router's descriptor before giving up?

Referenced by dir_routerdesc_download_failed(), and update_extrainfo_downloads().

4.35.2.152 #define MAX_SSL_KEY_LIFETIME (2*60*60)

How often do we rotate TLS contexts?

Referenced by init_keys(), and run_scheduled_events().

4.35.2.153 #define MAX_V_LINE_LEN 128

Maximum allowable length of a version line in a networkstatus.

Referenced by routerstatus_format_entry().

4.35.2.154 #define MAX_VERBOSE_NICKNAME_LEN (1+HEX_DIGEST_LEN+1+MAX_NICKNAME_LEN)

Maximum length of verbose router identifier: dollar sign, hex ID digest, equal sign or tilde, nickname.

Referenced by circuit_list_path_impl(), client_dns_set_addressmap_impl(), control_event_descriptors_changed(), control_event_guard(), getinfo_helper_entry_guards(), list_server_status_v1(), and orconn_target_get_name().

4.35.2.155 #define MAX_WEIGHT_LINE_LEN (12+10+10+10+1)

Length of "w" line for weighting. Currently at most "w Bandwidth=<uint32t> Measured=<uint32t>\n"

4.35.2.156 #define MIN_CIRCUITS_HANDLING_STREAM 2

How many circuits do we want simultaneously in-progress to handle a given stream?

Referenced by ap_stream_wants_exit_attention(), and circuit_remove_handled_ports().

4.35.2.157 #define MIN_DIST_SECONDS 20

Lowest allowable value for DistSeconds.

Referenced by networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

4.35.2.158 #define MIN_DNS_TTL (60)

How small can a TTL be before we stop believing it? Provides rudimentary pinning.

Referenced by dns_clip_ttl(), and dns_get_expiry_ttl().

4.35.2.159 #define MIN_ONION_KEY_LIFETIME (7*24*60*60)

How often do we rotate onion keys?

Referenced by run_scheduled_events().

4.35.2.160 #define MIN_VOTE_INTERVAL 300

Smallest allowable voting interval.

Referenced by networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

4.35.2.161 #define MIN_VOTE_SECONDS 20

Lowest allowable value for VoteSeconds.

Referenced by networkstatus_compute_consensus(), and networkstatus_parse_vote_from_string().

4.35.2.162 #define N_CONSENSUS_FLAVORS ((int)(FLAV_MICRODESC)+1)

How many different consensus flavors are there?

Referenced by dirvote_clear_pending_consensuses(), dirvote_compute_consensuses(), and dirvote_fetch_missing_signatures().

4.35.2.163 #define OLD_ROUTER_DESC_MAX_AGE (60*60*24*5)

How old do we let a saved descriptor get before force-removing it?

Referenced by client_would_use_router(), networkstatus_set_current_consensus(), router_add_to_routerlist(), and routerlist_remove_old_routers().

4.35.2.164 #define ONIONSKIN_CHALLENGE_LEN

Value:

```
(PKCS1_OAEP_PADDING_OVERHEAD+\
                                 CIPHER_KEY_LEN+\
                                 DH_KEY_LEN)
```

4.35.2.165 #define OR_CONN_STATE_CONNECTING 1

State for a connection to an OR: waiting for connect() to finish.

Referenced by conn_state_to_string(), connection_handle_read_impl(), connection_or_connect(), connection_or_finished_connecting(), connection_state_is_connecting(), and run_connection_housekeeping().

4.35.2.166 #define OR_CONN_STATE_OPEN 7

State for a connection to an OR: Ready to send/receive cells.

Referenced by assert_connection_ok(), conn_close_if_marked(), conn_state_to_string(), connection_about_to_close_connection(), connection_bucket_read_limit(), connection_bucket_refill(), connection_bucket_should_increase(), connection_bucket_write_limit(), connection_buckets_decrement(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_or_finished_flushing(), connection_or_get_for_extend(), connection_or_group_set_badness(), connection_or_process_inbuf(), connection_or_set_state_open(), connection_state_is_open(), and getinfo_helper_events().

4.35.2.167 #define OR_CONN_STATE_OR_HANDSHAKING 6

State for a connection to an OR: We're done with our SSL handshake, but we haven't yet negotiated link protocol versions and sent a netinfo cell.

Referenced by command_process_cell(), command_process_netinfo_cell(), command_process_var_cell(), command_process_versions_cell(), conn_state_to_string(), connection_or_finished_flushing(), connection_or_process_inbuf(), and connection_tls_finish_handshake().

4.35.2.168 #define OR_CONN_STATE_PROXY_HANDSHAKING 2

State for a connection to an OR: waiting for proxy handshake to complete

Referenced by assert_connection_ok(), conn_state_to_string(), connection_handle_write_impl(), connection_or_finished_connecting(), connection_or_finished_flushing(), connection_or_process_inbuf(), and connection_read_to_buf().

4.35.2.169 #define OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING 4

State for a connection to an OR: We're doing a second SSL handshake for renegotiation purposes.

Referenced by conn_state_to_string(), connection_handle_write_impl(), connection_read_to_buf(), and connection_tls_continue_handshake().

4.35.2.170 #define OR_CONN_STATE_TLS_HANDSHAKING 3

State for a connection to an OR or client: SSL is handshaking, not done yet.

Referenced by conn_state_to_string(), connection_about_to_close_connection(), connection_handle_write_impl(), connection_read_to_buf(), connection_tls_continue_handshake(), and connection_tls_start_handshake().

4.35.2.171 #define OR_CONN_STATE_TLS_SERVER_RENEGOTIATING 5

State for a connection at an OR: We're waiting for the client to renegotiate.

Referenced by conn_state_to_string(), connection_handle_write_impl(), and connection_tls_continue_handshake().

4.35.2.172 #define PDS_ALLOW_SELF (1<<0)

Flag to indicate that we should not automatically be willing to use ourself to answer a directory request. Passed to router_pick_directory_server (et al).

Referenced by directory_get_from_dirserver(), router_get_my_share_of_directory_requests(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.35.2.173 #define PDS_IGNORE_FASCISTFIREWALL (1<<2)

Flag to indicate that we should not exclude directory servers that our ReachableAddress settings would exclude. This usually means that we're going to connect to the server over Tor, and so we don't need to worry about our firewall telling us we can't. Passed to router_pick_directory_server (et al).

Referenced by directory_get_from_dirserver(), router_get_my_share_of_directory_requests(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.35.2.174 #define PDS_NO_EXISTING_SERVERDESC_FETCH (1<<3)

Flag to indicate that we should not use any directory authority to which we have an existing directory connection for downloading server descriptors or extrainfo documents.

Passed to router_pick_directory_server (et al)

[XXXX NOTE: This option is only implemented for pick_trusteddirserver, not pick_directory_server. If we make it work on pick_directory_server too, we could conservatively make it only prevent multiple fetches to the same authority, or we could aggressively make it prevent multiple fetches to _any_ single directory server.]

Referenced by directory_get_from_dirserver(), launch_router_descriptor_downloads(), router_pick_trusteddirserver(), router_pick_trusteddirserver_impl(), and update_extrainfo_downloads().

4.35.2.175 #define PDS_RETRY_IF_NO_SERVERS (1<<1)

Flag to indicate that if no servers seem to be up, we should mark all directory servers as up and try again. Passed to router_pick_directory_server (et al).

Referenced by launch_router_descriptor_downloads(), router_pick_directory_server(), router_pick_trusteddirserver(), update_consensus_networkstatus_downloads(), update_extrainfo_downloads(), update_router_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.35.2.176 #define RELAY_HEADER_SIZE (1+2+2+4+2)

Number of bytes in a relay cell's header (not including general cell header).

Referenced by circuit_extend(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_exit_begin_resolve(), and relay_send_command_from_edge().

4.35.2.177 #define RELAY_PAYLOAD_SIZE (CELL_PAYLOAD_SIZE-RELAY_HEADER_SIZE)

Largest number of bytes that can fit in a relay cell payload.

Referenced by connection_ap_handshake_send_begin(), connection_bucket_read_limit(), connection_bucket_write_limit(), connection_edge_end(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell(), dumpstats(), relay_send_command_from_edge(), rend_client_send_introduction(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), send_resolved_cell(), and send_resolved_hostname_cell().

4.35.2.178 #define REMAP_STREAM_SOURCE_CACHE 1

Reason for remapping an AP connection's address: we have a cached answer.

Referenced by control_event_stream_status().

4.35.2.179 #define REMAP_STREAM_SOURCE_EXIT 2

Reason for remapping an AP connection's address: the exit node told us an answer.

Referenced by control_event_stream_status(), and remap_event_helper().

4.35.2.180 #define REND_BASIC_AUTH_CLIENT_ENTRY_LEN

Value:

```
(REND_BASIC_AUTH_CLIENT_ID_LEN \
                                        + CIPHER_KEY_LEN)
```

Length of client entry consisting of client identifier and encrypted session key for hidden service authorization type 'basic'.

Referenced by rend_decrypt_introduction_points(), and rend_encrypt_v2_intro_points_basic().

4.35.2.181 #define REND_BASIC_AUTH_CLIENT_ID_LEN 4

Length of client identifier in encrypted introduction points for hidden service authorization type 'basic'.

Referenced by rend_decrypt_introduction_points(), and rend_encrypt_v2_intro_points_basic().

4.35.2.182 #define REND_BASIC_AUTH_CLIENT_MULTIPLE 16

Multiple of the number of clients to which the real number of clients is padded with fake clients for hidden service authorization type 'basic'.

Referenced by rend_decrypt_introduction_points(), and rend_encrypt_v2_intro_points_basic().

4.35.2.183 #define REND_CLIENTNAME_MAX_LEN 16

Maximum length of authorized client names for a hidden service.

Referenced by rend_config_services().

4.35.2.184 #define REND_COOKIE_LEN DIGEST_LEN

Length of the rendezvous cookie that is used to connect circuits at the rendezvous point.

Referenced by circuit_get_rendezvous(), rend_client_send_establish_rendezvous(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_introduce(), and rend_service_rendezvous_has_opened().

4.35.2.185 #define REND_DESC_COOKIE_LEN 16

Length of the descriptor cookie that is used for client authorization to hidden services.

Referenced by directory_get_from_hs_dir(), get_secret_id_part_bytes(), rend_check_authorization(), rend_decrypt_introduction_points(), rend_encrypt_v2_intro_points_basic(), rend_parse_client_keys(), rend_parse_service_authorization(), rend_service_introduce(), and rend_service_load_keys().

4.35.2.186 #define REND_DESC_COOKIE_LEN_BASE64 22

Length of the base64-encoded descriptor cookie that is used for exchanging client authorization between hidden service and client.

Referenced by directory_get_from_hs_dir(), rend_parse_client_keys(), rend_parse_service_authorization(), and rend_service_load_keys().

4.35.2.187 #define REND_DESC_ID_V2_LEN_BASE32 32

Length of v2 descriptor ID (32 base32 chars = 160 bits).

Referenced by directory_get_from_hs_dir(), directory_handle_command_get(), directory_post_to_hs_dir(), directory_send_command(), lookup_last_hid_serv_request(), rend_cache_clean_v2_descs_as_dir(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store_v2_desc_as_dir(), rend_encode_v2_descriptors(), and rend_parse_v2_service_descriptor().

4.35.2.188 #define REND_DESC_MAX_SIZE (20 * 1024)

Maximum size of v2 hidden service descriptors.

Referenced by rend_parse_v2_service_descriptor().

4.35.2.189 #define REND_INTRO_POINT_ID_LEN_BASE32 32

Length of the base32-encoded hash of an introduction point's identity key.

Referenced by rend_encode_v2_intro_points(), and rend_parse_introduction_points().

4.35.2.190 #define REND_LEGAL_CLIENTNAME_CHARACTERS "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01 _"

Legal characters for use in authorized client names for a hidden service.

Referenced by rend_config_services(), and rend_parse_client_keys().

4.35.2.191 #define REND_NUMBER_OF_CONSECUTIVE_REPLICAS 3

Number of consecutive replicas for a descriptor.

Referenced by hid_serv_get_responsible_directories().

4.35.2.192 #define REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS 2

Number of non-consecutive replicas (i.e. distributed somewhere in the ring) for a descriptor.

Referenced by rend_client_refetch_v2_renddesc(), rend_compute_v2_desc_id(), and rend_encode_v2_descriptors().

4.35.2.193 #define REND_REPLAY_TIME_INTERVAL (60 ∗ 60)

Time interval for tracking possible replays of INTRODUCE2 cells. Incoming cells with timestamps half of this interval in the past or future are dropped immediately.

Referenced by clean_accepted_intros(), and rend_service_introduce().

4.35.2.194 #define REND_SECRET_ID_PART_LEN_BASE32 32

Length of the base32-encoded secret ID part of versioned hidden service descriptors.

Referenced by rend_encode_v2_descriptors(), and rend_parse_v2_service_descriptor().

4.35.2.195 #define REND_SERVICE_ADDRESS_LEN (16+1+5)

Length of 'y.onion' including '.onion' URL.

Referenced by rend_parse_service_authorization().

4.35.2.196 #define REND_SERVICE_ID_LEN 10

Length of a binary-encoded rendezvous service ID.

Referenced by rend_compute_v2_desc_id(), rend_get_descriptor_id_bytes(), rend_get_service_id(), rend_mid_establish_intro(), rend_mid_introduce(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

4.35.2.197 #define REND_SERVICE_ID_LEN_BASE32 16

Length of 'y' portion of 'y.onion' URL.

Referenced by connection_about_to_close_connection(), parse_extended_hostname(), rend_cache_lookup_entry(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_compute_v2_desc_id(), rend_get_service_id(), rend_mid_establish_intro(), rend_mid_introduce(), rend_parse_service_authorization(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), rend_service_set_connection_addr_port(), rend_valid_service_id(), and upload_service_descriptor().

4.35.2.198 #define REND_TIME_PERIOD_OVERLAPPING_V2_DESCS (60∗60)

Time period within which two sets of v2 descriptors will be uploaded in parallel.

Referenced by upload_service_descriptor().

4.35.2.199 #define REND_TIME_PERIOD_V2_DESC_VALIDITY (24*60*60)

Time period for which a v2 descriptor will be valid.

Referenced by get_seconds_valid(), get_time_period(), and rend_encode_v2_descriptors().

4.35.2.200 #define ROUTER_ANNOTATION_BUF_LEN 256

The max size we expect router descriptor annotations we create to be. We'll accept larger ones if we see them on disk, but we won't create any that are larger than this.

Referenced by dirserv_add_multiple_descriptors(), and router_load_single_router().

4.35.2.201 #define ROUTER_MAX_AGE (60*60*48)

How old do we allow a router to get before removing it from the router list? In seconds.

Referenced by routerlist_remove_old_routers().

4.35.2.202 #define ROUTER_MAX_AGE_TO_PUBLISH (60*60*20)

How old can a router get before we (as a server) will no longer consider it live? In seconds.

Referenced by authdir_wants_to_reject_router(), connection_dirserv_add_servers_to_outbuf(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_routerdescs(), dirserv_have_any_serverdesc(), generate_v2_networkstatus_opinion(), list_server_status_v1(), networkstatus_getinfo_by_purpose(), router_is_active(), and set_cached_dir().

4.35.2.203 #define ROUTER_PURPOSE_BRIDGE 2

Tor should use this router only for bridge positions in circuits.

Referenced by any_pending_bridge_descriptor_fetches(), authdir_mode_handles_descs(), bridges_retry_helper(), dir_routerdesc_download_failed(), directory_command_should_use_begindir(), directory_get_from_dirserver(), directory_handle_command_post(), dirserv_orconn_tls_done(), dirserv_test_reachability(), entry_guard_set_status(), entry_is_live(), fetch_bridge_descriptors(), launch_direct_bridge_descriptor_fetch(), learned_bridge_descriptor(), networkstatus_getinfo_by_purpose(), purpose_needs_anonymity(), router_purpose_from_string(), router_purpose_to_string(), router_rebuild_descriptor(), router_upload_dir_desc_to_dirservers(), and routerlist_descriptors_added().

4.35.2.204 #define ROUTER_PURPOSE_CONTROLLER 1

Tor should avoid using this router for circuit-building.

Referenced by router_purpose_from_string(), and router_purpose_to_string().

4.35.2.205 #define ROUTER_PURPOSE_GENERAL 0

Tor can use this router for general positions in circuits.

Referenced by authdir_mode_handles_descs(), consider_testing_reachability(), directory_get_from_hs_dir(), directory_handle_command_post(), directory_post_to_hs_dir(), dirserv_add_multiple_descriptors(), dirvote_compute_consensuses(), dirvote_perform_vote(), entry_is_live(), handle_control_postdescriptor(), initiate_descriptor_downloads(), load_downloaded_routers(), router_add_running_routers_to_smartlist(), router_add_to_routerlist(), router_parse_entry_from_string(), router_purpose_from_string(), router_purpose_to_string(), router_rebuild_descriptor(), router_upload_dir_desc_to_dirservers(), routerlist_insert_old(), routerlist_remove(), routerlist_replace(), routers_update_status_from_consensus_networkstatus(), update_consensus_networkstatus_downloads(), update_router_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.35.2.206 #define ROUTER_PURPOSE_UNKNOWN 255

Tor should not use this router; it was marked in cached-descriptors with a purpose we didn't recognize.

Referenced by handle_control_postdescriptor(), networkstatus_getinfo_by_purpose(), and router_purpose_from_string().

4.35.2.207 #define RS_ENTRY_LEN

Value:

```
( /* first line */                                             \
   MAX_NICKNAME_LEN+BASE64_DIGEST_LEN*2+ISO_TIME_LEN+INET_NTOA_BUF_LEN+ \
   5*2 /* ports */ + 10 /* punctuation */ +                       \
   /* second line */                                              \
   MAX_FLAG_LINE_LEN +                                            \
   /* weight line */                                              \
   MAX_WEIGHT_LINE_LEN +                                          \
   /* p line. */                                                  \
   MAX_POLICY_LINE_LEN +                                          \
   /* v line. */                                                  \
   MAX_V_LINE_LEN                                                 \
   )
```

Amount of space to allocate for each entry: r, s, and v lines.

Referenced by format_networkstatus_vote(), generate_v2_networkstatus_opinion(), and networkstatus_getinfo_helper_single().

4.35.2.208 #define SOCKS4_NETWORK_LEN 8

Number of bytes in a SOCKS4 header.

Referenced by connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), and fetch_from_buf_socks().

4.35.2.209 #define SOCKS_COMMAND_CONNECT 0x01

Please open a TCP connection to this addr:port.

Referenced by circuit_get_open_circ_or_launch(), connection_ap_can_use_exit(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_make_link(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_proxy_connect(), connection_send_socks5_connect(), and fetch_from_buf_socks().

4.35.2.210 #define SOCKS_COMMAND_IS_RESOLVE(c)

Value:

```
((c)==SOCKS_COMMAND_RESOLVE || \
                                   (c)==SOCKS_COMMAND_RESOLVE_PTR)
```

4.35.2.211 #define SOCKS_COMMAND_RESOLVE 0xF0

Please turn this FQDN into an IP address, privately.

Referenced by connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_resolve(), dnsserv_launch_request(), dnsserv_resolved(), evdns_server_callback(), and fetch_from_buf_socks().

4.35.2.212 #define SOCKS_COMMAND_RESOLVE_PTR 0xF1

Please turn this IP address into an FQDN, privately.

Referenced by connection_ap_can_use_exit(), dnsserv_launch_request(), dnsserv_resolved(), evdns_server_callback(), and fetch_from_buf_socks().

4.35.2.213 #define STREAMWINDOW_INCREMENT 50

Amount to increment a stream window when we get a stream SENDME.

Referenced by connection_edge_consider_sending_sendme(), and connection_edge_process_relay_cell().

4.35.2.214 #define STREAMWINDOW_START 500

Initial value on both sides of a stream transmission window when the stream is initialized. Measured in cells.

Referenced by connection_ap_handshake_send_begin(), connection_edge_consider_sending_sendme(), and connection_exit_begin_conn().

4.35.2.215 #define TIMEOUT_UNTIL_UNREACHABILITY_COMPLAINT (20*60)

How long to test reachability before complaining to the user.

Referenced by inform_testing_reachability(), run_scheduled_events(), and second_elapsed_callback().

4.35.2.216 #define TO_CIRCUIT(x) (&((x) → _base))

Convert a circuit subtype to a **circuit_t** (p. 40).

Referenced by _circuit_mark_for_close(), cell_ewma_to_circuit(), circuit_establish_circuit(), circuit_extend_to_new_exit(), circuit_get_next_by_pk_and_purpose(), circuit_has_opened(), circuit_receive_relay_cell(), circuit_send_next_onion_skin(), circuit_set_-

p_circid_orconn(), circuit_testing_opened(), circuit_truncated(), command_process_create_cell(), connection_ap_detach_retriable(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_exit_begin_resolve(), handle_control_closecircuit(), handle_control_extendcircuit(), link_apconn_to_circ(), onion_pending_add(), onionskin_answer(), or_circuit_new(), origin_circuit_init(), origin_circuit_new(), rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_rendezvous_acked(), rend_client_send_establish_rendezvous(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and router_perform_bandwidth_test().

4.35.2.217 #define TO_CONN(c) (&(((c) → _base)))

Cast a **connection_t** (p. 53) subtype pointer to a **connection_t** (p. 53)

Referenced by _circuit_mark_for_close(), _connection_mark_unattached_ap(), assert_all_pending_dns_resolves_ok(), circuit_consider_stop_edge_reading(), circuit_dump_by_conn(), circuit_free_all(), circuit_resume_edge_reading_helper(), command_process_netinfo_cell(), command_process_versions_cell(), connection_ap_handshake_process_socks(), connection_ap_handshake_socks_reply(), connection_ap_make_link(), connection_control_finished_flushing(), connection_control_process_inbuf(), connection_control_reached_eof(), connection_dir_finished_flushing(), connection_dir_is_encrypted(), connection_dir_process_inbuf(), connection_dir_reached_eof(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dns_remove(), connection_edge_consider_sending_sendme(), connection_edge_destroy(), connection_edge_finished_connecting(), connection_edge_finished_flushing(), connection_edge_package_raw_inbuf(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_reached_eof(), connection_edge_send_command(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect(), connection_exit_connect_dir(), connection_new(), connection_or_connect(), connection_or_finished_connecting(), connection_or_finished_flushing(), connection_or_flush_from_first_active_circuit(), connection_or_process_cells_from_inbuf(), connection_or_process_inbuf(), connection_or_reached_eof(), connection_or_set_state_open(), connection_or_tls_renegotiated_cb(), connection_or_write_cell_to_buf(), connection_or_write_var_cell_to_buf(), connection_printf_to_buf(), connection_tls_continue_handshake(), connection_tls_start_handshake(), connection_write_str_to_buf(), control_connection_new(), control_event_stream_status(), dir_connection_new(), directory_handle_command(), directory_handle_command_get(), directory_initiate_command_rend(), directory_send_command(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve(), dns_resolve_impl(), dnsserv_launch_request(), edge_connection_new(), evdns_server_callback(), handle_control_authenticate(), handle_control_getconf(), handle_control_getinfo(), handle_control_mapaddress(), handle_control_protocolinfo(), handle_control_signal(), or_connection_new(), purge_expired_resolves(), rend_client_desc_trynow(), send_control_event_string(), set_streams_blocked_on_circ(), write_http_response_header_impl(), and write_http_status_line().

4.35.2.218 #define USABLE_CONSENSUS_FLAVOR FLAV_NS

Which consensus flavor do we actually want to use to build circuits?

Referenced by consensus_is_waiting_for_certs(), and networkstatus_set_current_consensus().

4.35.2.219 #define VAR_CELL_HEADER_SIZE 5

Length of a header on a variable-length cell.

Referenced by connection_or_write_var_cell_to_buf(), and fetch_var_cell_from_buf().

4.35.3 Typedef Documentation

4.35.3.1 typedef uint32_t build_time_t

A build_time_t is milliseconds

4.35.3.2 typedef uint16_t circid_t

Identifies a circuit on an or_connection

4.35.3.3 typedef int16_t country_t

A signed integer representing a country code.

4.35.3.4 typedef uint16_t streamid_t

Identifies a stream on a circuit

4.35.4 Enumeration Type Documentation

4.35.4.1 enum addr_policy_action_t

What action type does an address policy indicate: accept or reject?

4.35.4.2 enum addr_policy_result_t

Outcome of applying an address policy to an address.

Enumerator:

ADDR_POLICY_ACCEPTED The address was accepted

ADDR_POLICY_REJECTED The address was rejected

ADDR_POLICY_PROBABLY_ACCEPTED Part of the address was unknown, but as far as we can tell, it was accepted.

ADDR_POLICY_PROBABLY_REJECTED Part of the address was unknown, but as far as we can tell, it was rejected.

4.35.4.3 enum addressmap_entry_source_t

Enumerates possible origins of a client-side address mapping.

Enumerator:

ADDRMAPSRC_CONTROLLER We're remapping this address because the controller told us to.

ADDRMAPSRC_TORRC We're remapping this address because our configuration (via torrc, the command line, or a SETCONF command) told us to.

ADDRMAPSRC_TRACKEXIT We're remapping this address because we have TrackHostExit configured, and we want to remember to use the same exit next time.

ADDRMAPSRC_DNS We're remapping this address because we got a DNS resolution from a Tor server that told us what its value was.

4.35.4.4 enum authority_type_t

Bitfield enum type listing types of directory authority/directory server.

Enumerator:

V1_AUTHORITY Serves/signs v1 directory information: Big lists of routers, and short routerstatus documents.

V2_AUTHORITY Serves/signs v2 directory information: i.e. v2 networkstatus documents

V3_AUTHORITY Serves/signs v3 directory information: votes, consensuses, certs

HIDSERV_AUTHORITY Serves hidden service descriptors.

BRIDGE_AUTHORITY Serves bridge descriptors.

EXTRAINFO_CACHE Serves extrainfo documents. (XXX Not precisely an authority type)

4.35.4.5 enum bandwidth_weight_rule_t

Possible ways to weight routers when choosing one randomly. See **routerlist_sl_choose_by_bandwidth()** (p. 1008) for more information.

4.35.4.6 enum bootstrap_status_t

Enum describing various stages of bootstrapping, for use with controller bootstrap status events. The values range from 0 to 100.

4.35.4.7 enum buildtimeout_set_event_t

Used to indicate the type of a buildtime event

4.35.4.8 enum cell_direction_t

Used to indicate which way a cell is going on a circuit.

Enumerator:

CELL_DIRECTION_IN The cell is moving towards the origin.

CELL_DIRECTION_OUT The cell is moving away from the origin.

4.35.4.9 enum circ_id_type_t

Possible rules for generating circuit IDs on an OR connection.

Enumerator:

CIRC_ID_TYPE_LOWER Pick from 0..1<<15-1.

CIRC_ID_TYPE_HIGHER Pick from 1<<15..1<<16-1.

CIRC_ID_TYPE_NEITHER The other side of a connection is an OP: never create circuits to it, and let it use any circuit ID it wants.

4.35.4.10 enum circuit_status_event_t

Used to indicate the type of a circuit event passed to the controller. The various types are defined in control-spec.txt

4.35.4.11 enum consensus_flavor_t

Enumerates recognized flavors of a consensus networkstatus document. All flavors of a consensus are generated from the same set of votes, but they present different types information to different versions of Tor.

4.35.4.12 enum dirreq_state_t

Possible states for either direct or tunneled directory requests that are relevant for determining network status download times.

Enumerator:

DIRREQ_IS_FOR_NETWORK_STATUS Found that the client requests a network status; applies to both direct and tunneled requests; initial state of a request that we are measuring.

DIRREQ_FLUSHING_DIR_CONN_FINISHED Finished writing a network status to the directory connection; applies to both direct and tunneled requests; completes a direct request.

DIRREQ_END_CELL_SENT END cell sent to circuit that initiated a tunneled request.

DIRREQ_CIRC_QUEUE_FLUSHED Flushed last cell from queue of the circuit that initiated a tunneled request to the outbuf of the OR connection.

DIRREQ_OR_CONN_BUFFER_FLUSHED Flushed last byte from buffer of the OR connection belonging to the circuit that initiated a tunneled request; completes a tunneled request.

4.35.4.13 enum dirreq_type_t

Directory requests that we are measuring can be either direct or tunneled.

4.35.4.14 enum download_schedule_t

Enumeration: what kind of download schedule are we using for a given object?

4.35.4.15 enum geoip_client_action_t

Indicates an action that we might be noting geoip statistics on. Note that if we're noticing CONNECT, we're a bridge, and if we're noticing the others, we're not.

Enumerator:

GEOIP_CLIENT_CONNECT We've noticed a connection as a bridge relay or entry guard.

GEOIP_CLIENT_NETWORKSTATUS We've served a networkstatus consensus as a directory server.

GEOIP_CLIENT_NETWORKSTATUS_V2 We've served a v2 networkstatus consensus as a directory server.

4.35.4.16 enum geoip_ns_response_t

Indicates either a positive reply or a reason for rejectng a network status request that will be included in geoip statistics.

Enumerator:

GEOIP_SUCCESS Request is answered successfully.

GEOIP_REJECT_NOT_ENOUGH_SIGS V3 network status is not signed by a sufficient number of requested authorities.

GEOIP_REJECT_UNAVAILABLE Requested network status object is unavailable.

GEOIP_REJECT_NOT_FOUND Requested network status not found.

GEOIP_REJECT_NOT_MODIFIED Network status has not been modified since If-Modified-Since time.

GEOIP_REJECT_BUSY Directory is busy.

4.35.4.17 enum hostname_type_t

Possible return values for parse_extended_hostname.

4.35.4.18 enum invalid_router_usage_t

Bitfield type: things that we're willing to use invalid routers for.

4.35.4.19 enum networkstatus_type_t

Enumerates the possible seriousness values of a networkstatus document.

4.35.4.20 enum or_conn_status_event_t

Used to indicate the type of an OR connection event passed to the controller. The various types are defined in control-spec.txt

4.35.4.21 enum pk_op_t

Possible public/private key operations in Tor: used to keep track of where we're spending our time.

4.35.4.22 enum rend_auth_type_t

Client authorization type that a hidden service performs.

4.35.4.23 enum router_crn_flags_t

Flags to be passed to control **router_choose_random_node()** (p. 987) to indicate what kind of nodes to pick according to what algorithm.

4.35.4.24 enum saved_location_t

Enum used to remember where a **signed_descriptor_t** (p. 227) is stored and how to manage the memory for signed_descriptor_body.

Enumerator:

SAVED_NOWHERE The descriptor isn't stored on disk at all: the copy in memory is canonical; the saved_offset field is meaningless.

SAVED_IN_CACHE The descriptor is stored in the cached_routers file: the signed_descriptor_body is meaningless; the signed_descriptor_len and saved_offset are used to index into the mmaped cache file.

SAVED_IN_JOURNAL The descriptor is stored in the cached_routers.new file: the signed_descriptor_body and saved_offset fields are both set.

4.35.4.25 enum setopt_err_t

An error from **options_trial_assign()** (p. 397) or **options_init_from_string()** (p. 395).

4.35.4.26 enum store_type_t

Allowable types of **desc_store_t** (p. 75).

4.35.4.27 enum stream_status_event_t

Used to indicate the type of a stream event passed to the controller. The various types are defined in control-spec.txt

4.35.4.28 enum v2_networkstatus_source_t

Location where we found a v2 networkstatus.

4.35.4.29 enum version_status_t

Possible statuses of a version of Tor, given opinions from the directory servers.

Enumerator:

VS_RECOMMENDED This version is listed as recommended.

VS_OLD This version is older than any recommended version.

VS_NEW This version is newer than any recommended version.

VS_NEW_IN_SERIES This version is newer than any recommended version in its series, but later recommended versions exist.

VS_UNRECOMMENDED This version is not recommended (general case).

VS_EMPTY The version list was empty; no agreed-on versions.

VS_UNKNOWN We have no idea.

4.35.4.30 enum was_router_added_t

Return value for **router_add_to_routerlist()** (p. 985) and **dirserv_add_descriptor()** (p. 594)

4.35.5 Function Documentation

4.35.5.1 circuit_t∗ _circuit_get_global_list (void)

Return the head of the global linked list of circuits.

Referenced by count_established_intro_points(), dump_cell_pool_usage(), getinfo_helper_events(), rend_config_services(), and rep_hist_buffer_stats_write().

4.35.5.2 void _circuit_mark_for_close (circuit_t ∗ *circ*, int *reason*, int *line*, const char ∗ *file*)

Mark **circ** to be closed next time we call **circuit_close_all_marked()** (p. 331). Do any cleanup needed:

- If state is onionskin_pending, remove circ from the onion_pending list.
- If circ isn't open yet: call **circuit_build_failed()** (p. 342) if we're the origin, and in either case call **circuit_rep_hist_note_result()** (p. 315) to note stats.
- If purpose is C_INTRODUCE_ACK_WAIT, remove the intro point we just tried from our list of intro points for that service descriptor.
- Send appropriate destroys and edge_destroys for conns and streams attached to circ.
- If circ->rend_splice is set (we are the midpoint of a joined rendezvous stream), then mark the other circuit to close as well.

References or_circuit_t::_base, edge_connection_t::_base, assert_circuit_ok(), origin_circuit_t::build_state, build_state_get_exit_nickname(), cpath_build_state_t::chosen_exit, circuit_build_failed(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, circuit_rep_hist_note_result(), CIRCUIT_STATE_ONIONSKIN_PENDING, CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, connection_edge_destroy(), connection_or_send_destroy(), control_event_circuit_status(), edge_connection_t::edge_has_sent_end, END_CIRC_AT_ORIGIN, END_CIRC_REASON_FLAG_REMOTE, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED, LD_BUG, LD_REND, connection_t::marked_for_close, circuit_t::marked_for_close, circuit_t::marked_for_close_file, circuit_t::n_circ_id, circuit_t::n_conn, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, rend_data_t::onion_address, onion_pending_remove(), or_circuit_t::p_circ_id, or_circuit_t::p_conn, origin_circuit_t::p_streams, circuit_t::purpose, rend_client_remove_intro_point(), origin_circuit_t::rend_data, or_circuit_t::rend_splice, or_circuit_t::resolving_streams, safe_str_client(), smartlist_remove(), circuit_t::state, TO_CIRCUIT, TO_CONN, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_assert.

4.35.5.3 void _connection_mark_for_close (connection_t * *conn*, int *line*, const char * *file*)

Mark **conn** to be closed next time we loop through **conn_close_if_marked()** (p. 693) in **main.c** (p. 690).

References add_connection_to_closeable_list(), assert_connection_ok(), LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, connection_t::timestamp_lastwritten, tor_assert, and tor_fragile_assert.

Referenced by _connection_mark_unattached_ap().

4.35.5.4 void _connection_mark_unattached_ap (edge_connection_t * *conn*, int *endreason*, int *line*, const char * *file*)

An AP stream has failed/finished. If it hasn't already sent back a socks reply, send one now (based on endreason). Also set has_sent_end to 1, and mark the conn.

References edge_connection_t::_base, _connection_mark_for_close(), socks_request_t::command, CONN_TYPE_AP, connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SOCKS_REPLIED, socks_request_t::has_finished, connection_t::hold_open_until_flushed, LD_BUG, connection_t::marked_for_close, edge_connection_t::socks_request, TO_CONN, tor_assert, and connection_t::type.

4.35.5.5 void _connection_write_to_buf_impl (const char * *string*, size_t *len*, connection_t * *conn*, int *zlib*)

Append **len** bytes of **string** onto **conn**'s outbuf, and ask it to start writing.

If **zlib** is nonzero, this is a directory connection that should get its contents compressed or decompressed as they're written. If zlib is negative, this is the last data to be compressed, and the connection's zlib state should be flushed.

If it's an OR conn and an entire TLS record is ready, then try to flush the record now. Similarly, if it's a local control connection and a 64k chunk is ready, try to flush it all, so we don't end up with many megabytes of controller info queued at once.

References buf_datalen(), circuit_get_by_edge_conn(), CONN_IS_EDGE, CONN_LOG_PROTECT, CONN_TYPE_CONTROL, CONN_TYPE_OR, connection_close_immediate(), connection_is_rate_limited(), connection_start_writing(), connection_t::hold_open_until_flushed, connection_t::in_flushed_some, LD_BUG, LD_NET, connection_t::marked_for_close, MIN_TLS_FLUSHLEN, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::s, TO_DIR_CONN(), TO_EDGE_CONN(), tor_fragile_assert, connection_t::type, write_to_buf(), write_to_buf_zlib(), and dir_connection_t::zlib_state.

4.35.5.6 void accounting_add_bytes (size_t *n_read*, size_t *n_written*, int *seconds*)

Called from **main.c** (p. 690) to tell us that **seconds** seconds have passed, **n_read** bytes have been read, and **n_written** bytes have been written.

References n_bytes_read_in_interval, n_bytes_written_in_interval, and n_seconds_active_in_interval.

Referenced by second_elapsed_callback().

4.35.5.7 int accounting_is_enabled (or_options_t * *options*)

If we want to manage the accounting system and potentially hibernate, return 1, else return 0.

References or_options_t::AccountingMax.

Referenced by decide_to_advertise_dirport(), do_hup(), getinfo_helper_accounting(), options_act(), or_state_save(), run_scheduled_events(), second_elapsed_callback(), and tor_cleanup().

4.35.5.8 int accounting_parse_options (or_options_t * *options*, int *validate_only*)

Configure accounting start/end time settings based on options->AccountingStart. Return 0 on success, -1 on failure. If **validate_only** is true, do not change the current settings.

References or_options_t::AccountingStart, cfg_start_day, cfg_unit, LD_CONFIG, smartlist_create(), smartlist_free(), smartlist_split_string(), tor_assert, tor_free, and tor_parse_long().

Referenced by options_act().

4.35.5.9 int accounting_record_bandwidth_usage (time_t *now*, or_state_t * *state*)

Save all our bandwidth tracking information to disk. Return 0 on success, -1 on failure.

References or_state_t::AccountingBytesReadInInterval, or_state_t::AccountingBytesWrittenInInterval, or_state_t::AccountingExpectedUsage, or_state_t::AccountingIntervalStart, or_state_t::AccountingSecondsActive, expected_bandwidth_usage, get_options(), interval_start_time, n_bytes_read_in_interval, n_bytes_written_in_interval, n_seconds_active_in_interval, and or_state_mark_dirty().

Referenced by accounting_run_housekeeping(), do_hup(), hibernate_begin(), hibernate_go_dormant(), and tor_cleanup().

4.35.5.10 void accounting_run_housekeeping (time_t *now*)

Invoked once per second. Checks whether it is time to hibernate, record bandwidth used, etc.

References accounting_record_bandwidth_usage(), configure_accounting(), get_or_state(), interval_end_time, LD_FS, and time_to_record_bandwidth_usage().

Referenced by hibernate_end_time_elapsed(), or_state_save(), and run_scheduled_events().

4.35.5.11 void add_connection_to_closeable_list (connection_t ∗ *conn*)

Schedule **conn** to be closed.

References assert_connection_ok(), connection_t::marked_for_close, smartlist_add(), smartlist_isin(), and tor_assert.

Referenced by _connection_mark_for_close().

4.35.5.12 void add_nickname_list_to_smartlist (smartlist_t ∗ *sl*, const char ∗ *list*, int *must_be_running*)

Given a (possibly NULL) comma-and-whitespace separated list of nicknames, see which nicknames in **list** name routers in our routerlist, and add the routerinfos for those routers to **sl**. If **must_be_running**, only include routers that we think are running. Warn if any non-Named routers are specified by nickname.

References is_legal_nickname_or_hexdigest(), routerinfo_t::is_running, LD_CONFIG, router_get_by_nickname(), router_get_consensus_status_by_nickname(), router_have_minimum_dir_info(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), smartlist_string_isin(), tor_assert, tor_free, and warned_nicknames.

4.35.5.13 trusted_dir_server_t∗ add_trusted_dir_server (const char ∗ *nickname*, const char ∗ *address*, uint16_t *dir_port*, uint16_t *or_port*, const char ∗ *digest*, const char ∗ *v3_auth_digest*, authority_type_t *type*)

Add to the list of authoritative directory servers one at **address**:**port**, with identity key **digest**. If **address** is NULL, add ourself. Return the new trusted directory server entry on success or NULL if we couldn't add it.

References routerstatus_t::addr, trusted_dir_server_t::addr, trusted_dir_server_t::address, trusted_dir_server_t::description, trusted_dir_server_t::digest, DIGEST_LEN, routerstatus_t::dir_port, trusted_dir_server_t::dir_port, trusted_dir_server_t::fake_status, get_options(), routerstatus_t::identity_digest, trusted_dir_server_t::is_running, LD_CONFIG, routerstatus_t::nickname, trusted_dir_server_t::nickname, routerstatus_t::or_port, trusted_dir_server_t::or_port, resolve_my_address(), router_dir_info_changed(), smartlist_add(), smartlist_create(), tor_lookup_hostname(), tor_snprintf(), trusted_dir_servers, trusted_dir_server_t::type, V3_AUTHORITY, trusted_dir_server_t::v3_identity_digest, routerstatus_t::version_supports_begindir, and routerstatus_t::version_supports_conditional_consensus.

Referenced by init_keys(), and parse_dir_server_line().

4.35.5.14 void addr_policy_free (addr_policy_t ∗ *p*)

Release all storage held by **p**.

References addr_policy_t::is_canonical, addr_policy_t::refcnt, tor_assert, and tor_free.

Referenced by addr_policy_list_free(), exit_policy_remove_redundancies(), load_policy_from_option(), policy_expand_private(), router_add_exit_policy(), and routerset_free().

4.35.5.15 void addr_policy_list_free (smartlist_t * *lst*)

Release all storage held by **p**.

References addr_policy_free(), and smartlist_free().

Referenced by load_policy_from_option(), parse_addr_policy(), parse_reachable_addresses(), policies_free_all(), policies_set_router_exitpolicy_to_reject_all(), routerinfo_free(), and validate_addr_policies().

4.35.5.16 int address_is_invalid_destination (const char * *address*, int *client*)

Return 1 if **address** has funny characters in it like colons. Return 0 if it's fine, or if we're configured to allow it anyway. **client** should be true if we're using this address as a client; false if we're using it as a server.

References get_options().

Referenced by config_register_addressmaps(), dns_resolve_impl(), and handle_control_mapaddress().

4.35.5.17 void addressmap_clean (time_t *now*)

Clean out entries from the addressmap cache that were added long enough ago that they are no longer valid.

References addressmap_get_mappings().

Referenced by circuit_build_needed_circs().

4.35.5.18 void addressmap_clear_configured (void)

Remove all entries from the addressmap that were set via the configuration file or the command line.

References addressmap_get_mappings().

Referenced by config_register_addressmaps().

4.35.5.19 void addressmap_clear_transient (void)

Remove all entries from the addressmap that are set to expire, ever.

References addressmap_get_mappings().

Referenced by do_hup(), signal_callback(), and signewnym_impl().

4.35.5.20 void addressmap_free_all (void)

Free all the elements in the addressmap, and free the addressmap itself.

References addressmap, addressmap_ent_free(), addressmap_virtaddress_ent_free(), strmap_free(), and virtaddress_reversemap.

Referenced by tor_free_all().

4.35.5.21 void addressmap_get_mappings (smartlist_t * *sl*, time_t *min_expires*, time_t *max_expires*, int *want_expiry*)

Iterate over all address mappings which have expiry times between min_expires and max_expires, inclusive. If sl is provided, add an "old-addr new-addr expiry" string to sl for each mapping, omitting the expiry time if want_expiry is false. If sl is NULL, remove the mappings.

References addressmap, addressmap_ent_remove(), addressmap_init(), addressmap_entry_t::expires, format_iso_time(), addressmap_entry_t::new_address, smartlist_add(), strmap_iter_done(), strmap_iter_get(), strmap_iter_init(), strmap_iter_next(), strmap_iter_next_rmv(), and tor_snprintf().

Referenced by addressmap_clean(), addressmap_clear_configured(), addressmap_clear_transient(), and getinfo_helper_events().

4.35.5.22 int addressmap_have_mapping (const char * *address*, int *update_expiry*)

Return 1 if **address** is already registered, else return 0. If address is already registered, and **update_expires** is non-zero, then update the expiry time on the mapping with update_expires if it is a mapping created by TrackHostExits.

References addressmap, ADDRMAPSRC_TRACKEXIT, addressmap_entry_t::expires, addressmap_entry_t::source, and strmap_get_lc().

Referenced by consider_recording_trackhost(), and fetch_from_buf_socks().

4.35.5.23 void addressmap_init (void)

Initialize addressmap.

References addressmap, and virtaddress_reversemap.

Referenced by addressmap_get_mappings(), and tor_init().

4.35.5.24 void addressmap_register (const char * *address*, char * *new_address*, time_t *expires*, addressmap_entry_source_t *source*)

Register a request to map **address** to **new_address**, which will expire on **expires** (or 0 if never expires from config file, 1 if never expires from controller, 2 if never expires (virtual address mapping) from the controller.)

new_address should be a newly dup'ed string, which we'll use or free as appropriate. We will leave address alone.

If **new_address** is NULL, or equal to **address**, remove any mappings that exist from **address**.

References address_is_in_virtual_range(), addressmap, addressmap_ent_remove(), addressmap_virtaddress_remove(), control_event_address_mapped(), addressmap_entry_t::expires, LD_APP, LD_CONFIG, addressmap_entry_t::new_address, addressmap_entry_t::num_resolve_failures, safe_str_client(), addressmap_entry_t::source, strmap_get(), strmap_remove(), strmap_set(), and tor_free.

Referenced by addressmap_register_virtual_address(), client_dns_set_addressmap_impl(), config_register_addressmaps(), and handle_control_mapaddress().

4.35.5.25 const char* addressmap_register_virtual_address (int *type*, char * *new_address*)

A controller has requested that we map some address of type **type** to the address **new_address**. Choose an address that is unlikely to be used, and map it, and return it in a newly allocated string. If another address of the same type is already mapped to **new_address**, try to return a copy of that address.

The string in **new_address** may be freed, or inserted into a map as appropriate.

References addressmap, addressmap_get_virtual_address(), addressmap_register(), ADDRMAPSRC_CONTROLLER, virtaddress_entry_t::hostname_address, virtaddress_entry_t::ipv4_address, LD_APP, LD_BUG, addressmap_entry_t::new_address, safe_str_client(), strmap_get(), strmap_set(), tor_assert, tor_free, and virtaddress_reversemap.

Referenced by connection_ap_handshake_rewrite_and_attach(), and handle_control_mapaddress().

4.35.5.26 int addressmap_rewrite (char * *address*, size_t *maxlen*, time_t * *expires_out*)

Look at address, and rewrite it until it doesn't want any more rewrites; but don't get into an infinite loop. Don't write more than maxlen chars into address. Return true if the address changed; false otherwise. Set **expires_out* to the expiry time of the result, or to **time_max** if the result does not expire.

References addressmap, escaped_safe_str_client(), addressmap_entry_t::expires, LD_APP, LD_CONFIG, addressmap_entry_t::new_address, strmap_get(), and tor_free.

Referenced by connection_ap_process_end_not_open().

4.35.5.27 int advertised_server_mode (void)

Return true iff we have published our descriptor lately.

References server_is_advertised.

Referenced by directory_caches_dir_info(), directory_fetches_from_authorities(), and run_scheduled_events().

4.35.5.28 char* alloc_http_authenticator (const char * *authenticator*)

Allocates a base64'ed authenticator for use in http or https auth, based on the input string **authenticator**. Returns it if success, else returns NULL.

References base64_encode(), and tor_free.

Referenced by connection_proxy_connect(), directory_handle_command_get(), and directory_send_command().

4.35.5.29 int any_bridge_descriptors_known (void)

Return 1 if any of our entry guards have descriptors that are marked with purpose 'bridge' and are running. Else return 0.

We use this function to decide if we're ready to start building circuits through our bridges, or if we need to wait until the directory "server/authority" requests finish.

References choose_random_entry(), get_options(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), control_event_bootstrap_problem(), learned_bridge_descriptor(), and should_delay_dir_fetches().

4.35.5.30 int any_pending_bridge_descriptor_fetches (void)

Return 1 if there are any directory conns fetching bridge descriptors that aren't marked for close. We use this to guess if we should tell the controller that we have a problem.

References CONN_TYPE_DIR, DIR_PURPOSE_FETCH_SERVERDESC, get_connection_array(), LD_DIR, ROUTER_PURPOSE_BRIDGE, and TO_DIR_CONN().

Referenced by control_event_bootstrap_problem().

4.35.5.31 int any_predicted_circuits (time_t *now*)

Any ports used lately? These are pre-seeded if we just started up or if we're running a hidden service.

References PREDICTED_CIRCS_RELEVANCE_TIME, and predicted_internal_time.

Referenced by directory_info_has_arrived(), options_act(), rep_hist_circbuilding_dormant(), and run_scheduled_events().

4.35.5.32 int any_trusted_dir_is_v1_authority (void)

Return 1 if any trusted dir server supports v1 directories, else return 0.

References get_n_authorities(), trusted_dir_servers, and V1_AUTHORITY.

4.35.5.33 int append_address_to_payload (char * *payload_out*, const tor_addr_t * *addr*)

Append an encoded value of **addr** to **payload_out**, which must have at least 18 bytes of free space. The encoding is, as specified in tor-spec.txt: RESOLVED_TYPE_IPV4 or RESOLVED_TYPE_IPV6 [1 byte] LENGTH [1 byte] ADDRESS [length bytes] Return the number of bytes added, or -1 on error

References tor_addr_family(), and tor_addr_to_ipv4n().

Referenced by connection_or_send_netinfo().

4.35.5.34 void append_cell_to_circuit_queue (circuit_t * *circ*, or_connection_t * *orconn*, cell_t * *cell*, cell_direction_t *direction*)

Add **cell** to the queue of **circ** writing to **orconn** transmitting in **direction**.

References or_connection_t::_base, approx_time(), buf_datalen(), CELL_DIRECTION_OUT, cell_queue_append_packed_copy(), CELL_QUEUE_HIGHWATER_SIZE, cell_t::command, connection_or_flush_from_first_active_circuit(), LD_GENERAL, or_connection_t::link_proto, make_circuit_active_on_conn(), cell_queue_t::n, circuit_t::n_conn_cells, connection_t::outbuf, or_circuit_t::p_conn_cells, set_streams_blocked_on_circ(), circuit_t::streams_blocked_on_n_conn, circuit_t::streams_blocked_on_p_conn, and TO_OR_CIRCUIT().

Referenced by circuit_deliver_create_cell(), circuit_package_relay_cell(), circuit_receive_relay_cell(), and onionskin_answer().

4.35.5.35 void assert_active_circuits_ok (or_connection_t ∗ *orconn*)

Fail with an assert if the active circuits ring on **orconn** is corrupt.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuits, cell_ewma_t::heap_index, cell_ewma_t::is_for_p_conn, circuit_t::n_cell_ewma, circuit_t::n_conn, next_circ_on_conn_p(), or_circuit_t::p_cell_ewma, prev_circ_on_conn_p(), TO_OR_CIRCUIT(), and tor_assert.

4.35.5.36 void assert_addr_policy_ok (smartlist_t ∗ *lst*)

Log and exit if **t** is malformed

References tor_assert.

4.35.5.37 void assert_all_pending_dns_resolves_ok (void)

Log an error and abort if any connection waiting for a DNS resolve is corrupted.

References edge_connection_t::_base, assert_connection_ok(), pending_connection_t::conn, connection_in_array(), pending_connection_t::next, connection_t::s, TO_CONN, and tor_assert.

4.35.5.38 void assert_buf_ok (buf_t ∗ *buf*)

Log an error and exit if **buf** is corrupted.

References BUFFER_MAGIC, chunk_t::data, chunk_t::datalen, buf_t::datalen, buf_t::head, buf_t::magic, chunk_t::mem, chunk_t::memlen, chunk_t::next, buf_t::tail, and tor_assert.

Referenced by assert_connection_ok().

4.35.5.39 void assert_circuit_ok (const circuit_t ∗ *c*)

Verify that circuit **c** has all of its invariants correct. Trigger an assert if anything is invalid.

References edge_connection_t::_base, assert_cpath_ok(), circuit_get_by_circid_orconn_impl(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_REND_ESTABLISHED, CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, CONN_TYPE_EXIT, origin_circuit_t::cpath, circuit_t::deliver_window, circuit_t::magic, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, circuit_t::n_conn_onionskin, or_circuit_t::n_crypto, or_circuit_t::n_digest, circuit_t::n_hop, or_circuit_t::n_streams, edge_connection_t::next_stream, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::p_crypto, or_circuit_t::p_digest, circuit_t::package_window, circuit_t::purpose, or_circuit_t::rend_splice, smartlist_isin(), circuit_t::state, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and connection_t::type.

Referenced by _circuit_mark_for_close(), circuit_package_relay_cell(), connection_ap_handshake_attach_circuit(), connection_exit_begin_conn(), and connection_exit_begin_resolve().

4.35.5.40 void assert_connection_edge_not_dns_pending (edge_connection_t * *conn*)

Log an error and abort if conn is waiting for a DNS resolve.

References edge_connection_t::_base, connection_t::address, pending_connection_t::conn, pending_connection_t::next, cached_resolve_t::pending_connections, and tor_assert.

Referenced by connection_unlink().

4.35.5.41 void assert_connection_ok (connection_t * *conn*, time_t *now*)

Verify that connection **conn** has all of its invariants correct. Trigger an assert if anything is invalid.

References connection_t::address, AP_CONN_STATE_OPEN, assert_buf_ok(), assert_cpath_layer_ok(), edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, edge_connection_t::chosen_exit_retries, CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_EXIT, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, connection_is_listener(), connection_is_writing(), edge_connection_t::cpath_layer, EXIT_PURPOSE_CONNECT, EXIT_PURPOSE_RESOLVE, socks_request_t::has_finished, connection_t::hold_open_until_flushed, connection_t::inbuf, connection_t::linked, connection_t::linked_conn, LISTENER_STATE_READY, connection_t::magic, connection_t::marked_for_close, OR_CONN_STATE_OPEN, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::purpose, connection_t::s, edge_connection_t::socks_request, connection_t::state, or_connection_t::tls, TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, connection_t::type, and connection_t::write_blocked_on_bw.

Referenced by _connection_mark_for_close(), add_connection_to_closeable_list(), assert_all_pending_dns_resolves_ok(), command_process_netinfo_cell(), conn_close_if_marked(), conn_read_callback(), conn_write_callback(), connection_close_immediate(), connection_dns_remove(), connection_or_finished_flushing(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), and rend_client_desc_trynow().

4.35.5.42 void assert_cpath_layer_ok (const crypt_path_t * *cp*)

Verify that cpath layer **cp** has all of its invariants correct. Trigger an assert if anything is invalid.

References crypt_path_t::b_crypto, crypt_path_t::deliver_window, crypt_path_t::dh_handshake_state, crypt_path_t::f_crypto, LD_BUG, crypt_path_t::magic, crypt_path_t::package_window, crypt_path_t::state, and tor_assert.

Referenced by assert_connection_ok(), and assert_cpath_ok().

4.35.5.43 int assign_onionskin_to_cpuworker (connection_t ∗ *cpuworker*, or_circuit_t ∗ *circ*, char ∗ *onionskin*)

Try to tell a cpuworker to perform the public key operations necessary to respond to **onionskin** for the circuit **circ**.

If **cpuworker** is defined, assert that he's idle, and use him. Else, look for an idle cpuworker and use him. If none idle, queue task onto the pending onion list and return. Return 0 if we successfully assign the task, or -1 on failure.

References or_connection_t::_base, CONN_TYPE_CPUWORKER, connection_get_by_type_state(), CPUWORKER_STATE_BUSY_ONION, CPUWORKER_STATE_IDLE, cull_wedged_cpuworkers(), connection_t::global_identifier, LD_OR, num_cpuworkers, num_cpuworkers_busy, onion_pending_add(), or_circuit_t::p_circ_id, or_circuit_t::p_conn, spawn_enough_cpuworkers(), connection_t::state, TAG_LEN, tag_pack(), connection_t::timestamp_lastwritten, tor_assert, and tor_free.

Referenced by command_process_create_cell(), and process_pending_task().

4.35.5.44 int authdir_mode (or_options_t ∗ *options*)

Return true iff we believe ourselves to be an authoritative directory server.

References or_options_t::AuthoritativeDir.

Referenced by authdir_mode_bridge(), authdir_mode_v1(), authdir_mode_v2(), authdir_mode_v3(), decide_if_publishable_server(), decide_to_advertise_dirport(), directory_handle_command_post(), global_write_bucket_low(), init_keys(), options_validate(), rep_history_clean(), router_digest_is_trusted_dir_type(), router_rebuild_descriptor(), update_consensus_router_descriptor_downloads(), and update_router_descriptor_cache_downloads_v2().

4.35.5.45 int authdir_mode_any_main (or_options_t ∗ *options*)

Return true iff we are a v1, v2, or v3 directory authority.

References or_options_t::V1AuthoritativeDir, or_options_t::V2AuthoritativeDir, and or_options_t::V3AuthoritativeDir.

Referenced by authdir_mode_any_nonhidserv(), authdir_mode_handles_descs(), and options_validate().

4.35.5.46 int authdir_mode_any_nonhidserv (or_options_t ∗ *options*)

Return true if we believe ourselves to be any kind of authoritative directory beyond just a hidserv authority.

References authdir_mode_any_main(), and or_options_t::BridgeAuthoritativeDir.

Referenced by authdir_mode_handles_descs(), authdir_mode_publishes_statuses(), and launch_router_descriptor_downloads().

4.35.5.47 int authdir_mode_bridge (or_options_t ∗ *options*)

Return true iff we believe ourselves to be a bridge authoritative directory server.

References authdir_mode(), and or_options_t::BridgeAuthoritativeDir.

Referenced by authdir_mode_publishes_statuses(), directory_handle_command_post(), dirserv_orconn_tls_done(), dirserv_pick_cached_dir_obj(), dirserv_test_reachability(), and networkstatus_getinfo_by_purpose().

4.35.5.48 int authdir_mode_handles_descs (or_options_t * *options*, int *purpose*)

Return true iff we are an authoritative directory server that is authoritative about receiving and serving descriptors of type **purpose** its dirport. Use -1 for "any purpose".

References authdir_mode_any_main(), authdir_mode_any_nonhidserv(), or_options_t::BridgeAuthoritativeDir, ROUTER_PURPOSE_BRIDGE, and ROUTER_PURPOSE_GENERAL.

Referenced by authdir_mode_tests_reachability(), directory_handle_command_post(), do_hup(), and router_add_to_routerlist().

4.35.5.49 int authdir_mode_publishes_statuses (or_options_t * *options*)

Return true iff we are an authoritative directory server that publishes its own network statuses.

References authdir_mode_any_nonhidserv(), and authdir_mode_bridge().

Referenced by list_server_status_v1().

4.35.5.50 int authdir_mode_tests_reachability (or_options_t * *options*)

Return true iff we are an authoritative directory server that tests reachability of the descriptors it learns about.

References authdir_mode_handles_descs().

Referenced by connection_about_to_close_connection(), connection_or_check_valid_tls_handshake(), connection_or_connect_failed(), directory_handle_command_get(), run_scheduled_events(), and tor_cleanup().

4.35.5.51 int authdir_mode_v1 (or_options_t * *options*)

Return true iff we believe ourselves to be a v1 authoritative directory server.

References authdir_mode(), and or_options_t::V1AuthoritativeDir.

Referenced by dirserv_pick_cached_dir_obj(), router_parse_directory(), and router_parse_runningrouters().

4.35.5.52 int authdir_mode_v2 (or_options_t * *options*)

Return true iff we believe ourselves to be a v2 authoritative directory server.

References authdir_mode(), and or_options_t::V2AuthoritativeDir.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_pick_cached_dir_obj(), routers_update_status_from_consensus_networkstatus(), should_generate_v2_networkstatus(), and update_v2_networkstatus_cache_downloads().

4.35.5.53 int authdir_mode_v3 (or_options_t * *options*)

Return true iff we believe ourselves to be a v3 authoritative directory server.

References authdir_mode(), and or_options_t::V3AuthoritativeDir.

Referenced by directory_handle_command_post(), dirvote_act(), dirvote_recalculate_timing(), init_keys(), options_act(), routers_update_status_from_consensus_networkstatus(), run_scheduled_events(), update_consensus_networkstatus_downloads(), and v3_authority_check_key_expiry().

4.35.5.54 int authdir_policy_baddir_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** should be marked as a bad dir, based on **authdir_baddir_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.35.5.55 int authdir_policy_badexit_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** should be marked as a bad exit, based on **authdir_badexit_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.35.5.56 int authdir_policy_permits_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** is permitted to publish to our directory, based on **authdir_reject_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.35.5.57 int authdir_policy_valid_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** is considered valid in our directory, based on **authdir_invalid_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.35.5.58 int authdir_wants_to_reject_router (routerinfo_t * *ri*, const char ** *msg*, int *complain*)

Check whether we, as a directory server, want to accept **ri**. If so, set its is_valid,named,running fields and return 0. Otherwise, return -1.

If the router is rejected, set ***msg** to an explanation of why.

If **complain** then explain at log-level 'notice' why we refused a descriptor; else explain at log-level 'info'.

References routerinfo_t::address, routerinfo_t::cache_info, routerinfo_t::contact_info, dirserv_router_get_status(), dirserv_router_has_valid_address(), esc_router_info(), FP_BADDIR, FP_BADEXIT, FP_INVALID, FP_NAMED, FP_REJECT, routerinfo_t::is_bad_directory, routerinfo_t::is_bad_exit, routerinfo_t::is_named, routerinfo_t::is_valid, LD_DIRSERV, routerinfo_t::nickname, signed_descriptor_t::published_on, ROUTER_ALLOW_SKEW, ROUTER_MAX_AGE_TO_PUBLISH, and tor_assert.

Referenced by router_add_to_routerlist().

4.35.5.59 void authority_cert_dl_failed (const char ∗ *id_ digest*, int *status*)

Called when an attempt to download a certificate with the authority with ID **id_digest** fails with HTTP response code **status**: remember the failure, so we don't try again immediately.

References digestmap_get(), cert_list_t::dl_status, download_status_failed, and trusted_dir_certs.

Referenced by connection_dir_download_cert_failed(), and trusted_dirs_load_certs_from_string().

4.35.5.60 authority_cert_t∗ authority_cert_dup (authority_cert_t ∗ *cert*)

Allocate and return a new **authority_cert_t** (p. 14) with the same contents as **cert**.

References authority_cert_t::cache_info, crypto_pk_dup_key(), authority_cert_t::identity_key, signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, authority_cert_t::signing_key, and tor_assert.

4.35.5.61 void authority_cert_free (authority_cert_t ∗ *cert*)

Free storage held in **cert**.

References authority_cert_t::cache_info, crypto_free_pk_env(), authority_cert_t::identity_key, signed_descriptor_t::signed_descriptor_body, authority_cert_t::signing_key, and tor_free.

Referenced by authority_cert_parse_from_string(), load_authority_keyset(), networkstatus_vote_free(), router_free_all(), routerlist_free_all(), trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

4.35.5.62 void authority_cert_get_all (smartlist_t ∗ *certs_ out*)

Add every known **authority_cert_t** (p. 14) to **certs_out**.

References smartlist_add(), tor_assert, and trusted_dir_certs.

Referenced by directory_handle_command_get().

4.35.5.63 authority_cert_t∗ authority_cert_get_by_digests (const char ∗ *id_ digest*, const char ∗ *sk_ digest*)

Return the v3 authority certificate with signing key matching **sk_digest**, for the authority with identity digest **id_digest**. Return NULL if no such authority is known.

References cert_list_t::certs, DIGEST_LEN, digestmap_get(), and trusted_dir_certs.

Referenced by authority_cert_parse_from_string(), authority_certs_fetch_missing(), directory_handle_command_get(), dirvote_add_vote(), networkstatus_add_detached_signatures(), and networkstatus_check_consensus_signature().

4.35.5.64 authority_cert_t∗ authority_cert_get_by_sk_digest (const char ∗ *sk_digest*)

Return the newest v3 authority certificate whose directory signing key has digest **sk_digest**. Return NULL if no such certificate is known.

References DIGEST_LEN, get_my_v3_authority_cert(), get_my_v3_legacy_cert(), authority_cert_t::signing_key_digest, and trusted_dir_certs.

Referenced by directory_handle_command_get().

4.35.5.65 authority_cert_t∗ authority_cert_get_newest_by_id (const char ∗ *id_digest*)

Return the newest v3 authority certificate whose v3 authority identity key has digest **id_digest**. Return NULL if no such authority is known, or it has no certificate.

References authority_cert_t::cache_info, cert_list_t::certs, digestmap_get(), signed_descriptor_t::published_on, and trusted_dir_certs.

Referenced by directory_handle_command_get().

4.35.5.66 authority_cert_t∗ authority_cert_parse_from_string (const char ∗ *s*, const char ∗∗ *end_of_string*)

Parse a key certificate from **s**; point **end-of-string** to the first character after the certificate.

References authority_cert_t::addr, directory_token_t::args, authority_cert_free(), authority_cert_get_by_digests(), base16_decode(), authority_cert_t::cache_info, check_signature_token(), crypto_pk_get_digest(), DIGEST_LEN, authority_cert_t::dir_port, dump_desc(), eat_whitespace(), escaped(), authority_cert_t::expires, find_opt_by_keyword(), signed_descriptor_t::identity_digest, authority_cert_t::identity_key, authority_cert_t::is_cross_certified, directory_token_t::key, LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, parse_addr_port(), parse_iso_time(), signed_descriptor_t::published_on, router_get_hash_impl(), signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, authority_cert_t::signing_key, authority_cert_t::signing_key_digest, smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, tor_free, tor_inet_aton(), and directory_token_t::tp.

Referenced by load_authority_keyset(), networkstatus_parse_vote_from_string(), and trusted_dirs_load_certs_from_string().

4.35.5.67 void authority_certs_fetch_missing (networkstatus_t ∗ *status*, time_t *now*)

Try to download any v3 authority certificates that we may be missing. If **status** is provided, try to get all the ones that were used to sign **status**. Additionally, try to have a non-expired certificate for every V3 authority in trusted_dir_servers. Don't fetch certificates we already have.

References authority_cert_get_by_digests(), cert_list_t::certs, DIGEST_LEN, digestmap_get(), digestmap_new(), DIR_PURPOSE_FETCH_CERTIFICATE, directory_caches_dir_info(), cert_list_t::dl_status, download_status_is_ready(), download_status_reset(), authority_cert_t::expires, ftime_definitely_after(), get_cert_list(), get_options(), hex_str(), LD_DIR, list_pending_downloads(), MAX_CERT_DL_FAILURES, should_delay_dir_fetches(), smartlist_add(), smartlist_create(), smartlist_digest_isin(), trusted_dir_servers, trusteddirserver_get_by_v3_auth_digest(), V3_AUTHORITY, and networkstatus_t::voters.

Referenced by networkstatus_set_current_consensus(), and update_certificate_downloads().

4.35.5.68 char∗ authority_type_to_string (authority_type_t *auth*)

Return a newly allocated string describing **auth**.

References BRIDGE_AUTHORITY, HIDSERV_AUTHORITY, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), V1_AUTHORITY, V2_AUTHORITY, and V3_AUTHORITY.

Referenced by directory_post_to_dirservers().

4.35.5.69 const char∗ bandwidth_weight_rule_to_string (bandwidth_weight_rule_t *rule*)

Return a string corresponding to a bandwidht_weight_rule_t

Referenced by smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.35.5.70 void bridge_add_from_config (const tor_addr_t ∗ *addr*, uint16_t *port*, char ∗ *digest*)

Remember a new bridge at **addr**:**port**. If **digest** is set, it tells us the identity key too.

References bridge_info_t::addr, DIGEST_LEN, bridge_info_t::fetch_status, bridge_info_t::identity, bridge_info_t::port, download_status_t::schedule, smartlist_add(), smartlist_create(), and tor_addr_copy().

Referenced by parse_bridge_line().

4.35.5.71 int bridges_known_but_down (void)

Do we know any descriptors for our bridges, and are they all down?

References bridges_retry_helper().

Referenced by circuit_get_open_circ_or_launch().

4.35.5.72 void bridges_retry_all (void)

Mark all down known bridges up.

References bridges_retry_helper().

Referenced by circuit_get_open_circ_or_launch().

4.35.5.73 size_t buf_allocation (const buf_t * *buf*)

Return the total length of all chunks used in **buf**.

References buf_t::head, chunk_t::memlen, and chunk_t::next.

Referenced by connection_dump_buffer_mem_stats(), and dumpstats().

4.35.5.74 void buf_clear (buf_t * *buf*)

Remove all data from **buf**.

References chunk_free_unchecked(), buf_t::datalen, buf_t::head, chunk_t::next, and buf_t::tail.

Referenced by buf_free(), connection_close_immediate(), fetch_from_buf_socks(), and fetch_from_buf_socks_client().

4.35.5.75 size_t buf_datalen (const buf_t * *buf*)

Return the number of bytes stored in **buf**

References buf_t::datalen.

Referenced by _connection_free(), _connection_write_to_buf_impl(), append_cell_to_circuit_queue(), conn_close_if_marked(), connection_cpu_process_inbuf(), connection_dir_process_inbuf(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dirserv_flushed_some(), connection_dump_buffer_mem_stats(), connection_edge_package_raw_inbuf(), connection_edge_reached_eof(), connection_handle_read_impl(), connection_handle_write_impl(), connection_or_flushed_some(), connection_or_process_cells_from_inbuf(), connection_read_to_buf(), connection_should_read_from_linked_conn(), dumpstats(), and run_connection_housekeeping().

4.35.5.76 void buf_dump_freelist_sizes (int *severity*)

Describe the current status of the freelists at log level **severity**.

References chunk_freelist_t::alloc_size, LD_MM, and n_freelist_miss.

Referenced by dumpmemusage().

4.35.5.77 void buf_free (buf_t * *buf*)

Release storage held by **buf**.

References buf_clear(), buf_t::magic, and tor_free.

Referenced by _connection_free().

4.35.5.78 buf_t* buf_new (void)

Allocate and return a new buffer with default capacity.

References BUFFER_MAGIC, buf_t::default_chunk_size, and buf_t::magic.

Referenced by buf_new_with_capacity(), and connection_init().

4.35.5.79 buf_t∗ buf_new_with_capacity (size_t *size*)

Create and return a new buf with default chunk capacity **size**.

References buf_new(), buf_t::default_chunk_size, and preferred_chunk_size().

4.35.5.80 void buf_shrink (buf_t ∗ *buf*)

Resize buf so it won't hold extra memory that we haven't been using lately.

Referenced by run_scheduled_events().

4.35.5.81 void buf_shrink_freelists (int *free_all*)

Remove from the freelists most chunks that have not been used since the last call to **buf_shrink_freelists()** (p. 289).

References chunk_freelist_t::alloc_size, assert_freelist_ok(), chunk_freelist_t::cur_length, chunk_freelist_t::head, LD_BUG, LD_MM, chunk_freelist_t::lowest_length, chunk_freelist_t::n_free, chunk_t::next, chunk_freelist_t::slack, tor_assert, and tor_free.

Referenced by run_scheduled_events(), and tor_free_all().

4.35.5.82 size_t buf_slack (const buf_t ∗ *buf*)

Return the number of bytes that can be added to **buf** without performing any additional allocation.

References CHUNK_REMAINING_CAPACITY(), and buf_t::tail.

Referenced by connection_read_to_buf().

4.35.5.83 const char∗ build_state_get_exit_nickname (cpath_build_state_t ∗ *state*)

Return the nickname for the chosen exit router in **state**. If there is no chosen exit, or if we don't know the **routerinfo_t** (p. 206) for the chosen exit, return NULL.

References cpath_build_state_t::chosen_exit, and extend_info_t::nickname.

Referenced by _circuit_mark_for_close(), circuit_build_failed(), circuit_launch_by_extend_info(), circuit_list_path_impl(), and onion_extend_cpath().

4.35.5.84 routerinfo_t∗ build_state_get_exit_router (cpath_build_state_t ∗ *state*)

Return the **routerinfo_t** (p. 206) for the chosen exit router in **state**. If there is no chosen exit, or if we don't know the **routerinfo_t** (p. 206) for the chosen exit, return NULL.

References cpath_build_state_t::chosen_exit, extend_info_t::identity_digest, and router_get_by_digest().

Referenced by choose_good_entry_server(), choose_good_middle_server(), choose_random_entry(), circuit_is_acceptable(), and circuit_stream_is_being_handled().

4.35.5.85 void cached_dir_decref (cached_dir_t * *d*)

Decrement the reference count on **d**, and free it if it no longer has any references.

References clear_cached_dir(), cached_dir_t::refcnt, and tor_free.

Referenced by _connection_free(), _free_cached_dir(), connection_dirserv_add_dir_bytes_to_outbuf(), dirserv_clear_old_networkstatuses(), dirserv_clear_old_v1_info(), dirserv_free_all(), dirserv_regenerate_directory(), dirserv_set_cached_consensus_networkstatus(), dirserv_set_cached_directory(), dirserv_set_cached_networkstatus_v2(), dirvote_add_vote(), and dirvote_clear_votes().

4.35.5.86 unsigned cell_ewma_get_tick (void)

Compute and return the current cell_ewma tick.

References approx_time(), and EWMA_TICK_LEN.

Referenced by init_circuit_base(), or_circuit_new(), and or_connection_new().

4.35.5.87 void cell_ewma_set_scale_factor (or_options_t * *options*, networkstatus_t * *consensus*)

Adjust the global cell scale factor based on **options**

References or_options_t::CircuitPriorityHalflife, EWMA_DEFAULT_HALFLIFE, EWMA_TICK_LEN, LD_OR, and networkstatus_get_param().

Referenced by networkstatus_set_current_consensus(), and options_act().

4.35.5.88 void cell_pack (packed_cell_t * *dst*, const cell_t * *src*)

Pack the **cell_t** (p. 30) host-order structure **src** into network-order in the buffer **dest**. See torspec.txt for details about the wire format.

Note that this function doesn't touch **dst**->next: the caller should set it or clear it as appropriate.

References packed_cell_t::body, CELL_PAYLOAD_SIZE, cell_t::circ_id, cell_t::command, and cell_t::payload.

Referenced by connection_or_write_cell_to_buf(), and packed_cell_copy().

4.35.5.89 void cell_queue_append (cell_queue_t * *queue*, packed_cell_t * *cell*)

Append **cell** to the end of **queue**.

References cell_queue_t::head, cell_queue_t::n, packed_cell_t::next, cell_queue_t::tail, and tor_assert.

Referenced by cell_queue_append_packed_copy().

4.35.5.90 void cell_queue_append_packed_copy (cell_queue_t * *queue*, const cell_t * *cell*)

Append a newly allocated copy of **cell** to the end of **queue**

References cell_queue_append(), insertion_time_elem_t::counter, insertion_time_queue_t::first, get_options(), insertion_time_elem_t::insertion_time, cell_queue_t::insertion_times, insertion_time_queue_t::last, insertion_time_elem_t::next, and packed_cell_copy().

Referenced by append_cell_to_circuit_queue().

4.35.5.91 void cell_queue_clear (cell_queue_t * *queue*)

Remove and free every cell in **queue**.

References insertion_time_queue_t::first, cell_queue_t::head, cell_queue_t::insertion_times, cell_queue_t::n, insertion_time_elem_t::next, packed_cell_t::next, packed_cell_free_unchecked(), cell_queue_t::tail, and tor_free.

Referenced by circuit_free().

4.35.5.92 void check_descriptor_bandwidth_changed (time_t *now*)

Check whether bandwidth has changed a lot since the last time we announced bandwidth. If so, mark our descriptor dirty.

References routerinfo_t::bandwidthcapacity, LD_GENERAL, mark_my_descriptor_dirty(), MAX_BANDWIDTH_CHANGE_FREQ, rep_hist_bandwidth_assess(), and we_are_hibernating().

Referenced by run_scheduled_events().

4.35.5.93 void check_descriptor_ipaddress_changed (time_t *now*)

Check whether our own address as defined by the Address configuration has changed. This is for routers that get their address from a service like dyndns. If our address has changed, mark our descriptor dirty.

References routerinfo_t::addr, get_options(), ip_address_changed(), LD_CONFIG, log_addr_has_changed(), and resolve_my_address().

Referenced by run_scheduled_events().

4.35.5.94 int check_whether_dirport_reachable (void)

Return 1 if we don't have a dirport configured, or if it's reachable.

References or_options_t::AssumeReachable, can_reach_dir_port, or_options_t::DirPort, get_options(), and we_are_hibernating().

Referenced by consider_testing_reachability(), decide_to_advertise_dirport(), getinfo_helper_events(), rep_hist_circbuilding_dormant(), and second_elapsed_callback().

4.35.5.95 int check_whether_orport_reachable (void)

Return 1 if ORPort is known reachable; else return 0.

References or_options_t::AssumeReachable, can_reach_or_port, and get_options().

Referenced by circuit_send_next_onion_skin(), circuit_testing_failed(), circuit_testing_opened(), consider_testing_reachability(), decide_if_publishable_server(), getinfo_helper_-

events(), rep_hist_circbuilding_dormant(), and second_elapsed_callback().

4.35.5.96 routerinfo_t* choose_random_entry (cpath_build_state_t * *state*)

Pick a live (up and listed) entry guard from entry_guards. If **state** is non-NULL, this is for a specific circuit – make sure not to pick this circuit's exit or any node in the exit's family. If **state** is NULL, we're looking for a random guard (likely a bridge).

References add_an_entry_guard(), build_state_get_exit_router(), entry_guards_changed(), entry_guards_prepend_from_config(), entry_is_live(), entry_list_is_constrained(), or_options_t::EntryNodes, get_options(), LD_CIRC, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, or_options_t::NumEntryGuards, pick_entry_guards(), routerlist_add_family(), routerlist_sl_choose_by_bandwidth(), routerset_contains_router(), smartlist_add(), smartlist_choose(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_isin(), or_options_t::StrictNodes, and tor_fragile_assert.

Referenced by any_bridge_descriptors_known(), choose_good_entry_server(), and directory_get_from_dirserver().

4.35.5.97 int circuit_all_predicted_ports_handled (time_t *now*, int * *need_uptime*, int * *need_capacity*)

Return 1 if we already have circuits present or on the way for all anticipated ports. Return 0 if we should make more.

If we're returning 0, set need_uptime and need_capacity to indicate any requirements that the unhandled ports have.

References circuit_get_unhandled_ports(), get_options(), or_options_t::LongLivedPorts, smartlist_free(), smartlist_string_num_isin(), tor_assert, and tor_free.

Referenced by circuit_predict_and_launch_new().

4.35.5.98 int circuit_append_new_exit (origin_circuit_t * *circ*, extend_info_t * *exit*)

Give **circ** a new exit destination to **exit**, and add a hop to the cpath reflecting this. Don't send the next extend cell – the caller will do this if it wants to.

References origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, origin_circuit_t::cpath, cpath_build_state_t::desired_path_len, extend_info_dup(), extend_info_free(), onion_append_hop(), and tor_assert.

Referenced by circuit_extend_to_new_exit(), and handle_control_extendcircuit().

4.35.5.99 void circuit_build_failed (origin_circuit_t * *circ*)

Called whenever a circuit could not be successfully built.

References or_connection_t::_base, origin_circuit_t::_base, connection_t::address, origin_circuit_t::build_state, build_state_get_exit_nickname(), circuit_discard_optional_exit_enclaves(), circuit_increment_failure_count(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, CIRCUIT_STATE_OPEN, circuit_testing_-

failed(), connection_ap_fail_onehop(), origin_circuit_t::cpath, entry_guard_register_connect_status(), escaped(), crypt_path_t::extend_info, extend_info_t::identity_digest, or_connection_t::is_bad_for_new_circs, LD_OR, LD_REND, circuit_t::n_conn, connection_t::port, crypt_path_t::prev, circuit_t::purpose, rend_service_relaunch_rendezvous(), circuit_t::state, and crypt_path_t::state.

Referenced by _circuit_mark_for_close().

4.35.5.100 void circuit_build_needed_circs (time_t *now*)

This function is called once a second, if router_have_min_dir_info() is true. Its job is to make sure all services we offer have enough circuits available. Some services just want enough circuits for current tasks, whereas others want a minimum set of idle circuits hanging around.

References addressmap_clean(), circuit_expire_old_circuits_clientside(), circuit_launch_by_router(), circuit_predict_and_launch_new(), CIRCUIT_PURPOSE_C_GENERAL, circuit_reset_failure_count(), connection_ap_attach_pending(), or_options_t::DisablePredictedCircuits, get_options(), or_options_t::NewCircuitPeriod, proxy_mode(), rend_services_introduce(), and TESTING_CIRCUIT_INTERVAL.

Referenced by run_scheduled_events().

4.35.5.101 int circuit_build_times_add_time (circuit_build_times_t ∗ *cbt*, build_time_t *time*)

Add a new timeout value **time** to the set of build times. Time units are milliseconds.

circuit_build_times **cbt is a circular array, so loop around when array is full.**

References or_options_t::AvoidDiskWrites, circuit_build_times_t::build_times_idx, CBT_NCIRCUITS_TO_OBSERVE, CBT_SAVE_STATE_EVERY, circuit_build_times_t::circuit_build_times, get_options(), get_or_state(), LD_CIRC, or_state_mark_dirty(), tor_assert, circuit_build_times_t::total_build_times, and unit_tests.

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_shuffle_and_store_array(), and circuit_send_next_onion_skin().

4.35.5.102 int circuit_build_times_add_timeout (circuit_build_times_t ∗ *cbt*, int *did_onehop*, time_t *start_time*)

Store a timeout as a synthetic value.

Returns true if the store was successful and we should possibly update our timeout estimate.

References circuit_build_times_add_timeout_worker(), circuit_build_times_count_pretimeouts(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), circuit_build_times_network_timeout(), circuit_build_times_t::have_computed_timeout, LD_CIRC, circuit_build_times_t::pre_timeouts, and circuit_build_times_t::total_build_times.

Referenced by circuit_expire_building().

4.35.5.103 void circuit_build_times_init (circuit_build_times_t ∗ *cbt*)

Initialize the buildtimes structure for first use.

Sets the initial timeout value based to either the config setting or BUILD_TIMEOUT_INITIAL_VALUE.

References circuit_build_times_get_initial_timeout(), control_event_buildtimeout_set(), circuit_build_times_t::liveness, network_liveness_t::num_recent_circs, circuit_build_times_t::timeout_ms, and network_liveness_t::timeouts_after_firsthop.

Referenced by circuit_build_times_parse_state().

4.35.5.104 int circuit_build_times_needs_circuits (circuit_build_times_t ∗ *cbt*)

Returns true if we need circuits to be built

References circuit_build_times_t::total_build_times.

Referenced by circuit_build_times_needs_circuits_now(), and circuit_expire_old_circuits_clientside().

4.35.5.105 int circuit_build_times_needs_circuits_now (circuit_build_times_t ∗ *cbt*)

Returns true if we should build a timeout test circuit right now.

References approx_time(), circuit_build_times_needs_circuits(), and circuit_build_times_t::last_circ_at.

Referenced by circuit_predict_and_launch_new().

4.35.5.106 int circuit_build_times_network_check_live (circuit_build_times_t ∗ *cbt*)

Returns false if the network has not received a cell or tls handshake in the past NETWORK_NOTLIVE_TIMEOUT_COUNT circuits.

Also has the side effect of rewinding the circuit time history in the case of recent liveness changes.

References approx_time(), CBT_NETWORK_NONLIVE_DISCARD_COUNT, CBT_NETWORK_NONLIVE_TIMEOUT_COUNT, circuit_build_times_get_initial_timeout(), circuit_build_times_rewind_history(), control_event_buildtimeout_set(), LD_CIRC, circuit_build_times_t::liveness, network_liveness_t::net_suspended, network_liveness_t::network_last_live, network_liveness_t::nonlive_discarded, network_liveness_t::nonlive_timeouts, circuit_build_times_t::timeout_ms, and tor_lround().

Referenced by circuit_build_times_add_timeout().

4.35.5.107 void circuit_build_times_network_circ_success (circuit_build_times_t ∗ *cbt*)

Called to indicate that we completed a circuit. Because this circuit succeeded, it doesn't count as a timeout-after-the-first-hop.

References network_liveness_t::after_firsthop_idx, circuit_build_times_t::liveness, network_liveness_t::num_recent_circs, and network_liveness_t::timeouts_after_firsthop.

Referenced by circuit_send_next_onion_skin().

4.35.5.108 void circuit_build_times_network_is_live (circuit_build_times_t * *cbt*)

Called to indicate that the network showed some signs of liveness.

This function is called every time we receive a cell. Avoid syscalls, events, and other high-intensity work.

References approx_time(), circuit_build_times_t::liveness, network_liveness_t::network_last_live, network_liveness_t::nonlive_discarded, and network_liveness_t::nonlive_timeouts.

Referenced by connection_or_process_cells_from_inbuf(), connection_or_set_state_open(), and connection_tls_finish_handshake().

4.35.5.109 void circuit_build_times_new_consensus_params (circuit_build_times_t * *cbt*, networkstatus_t * *ns*)

This function is called when we get a consensus update.

It checks to see if we have changed any consensus parameters that require reallocation or discard of previous stats.

References network_liveness_t::after_firsthop_idx, CBT_DEFAULT_RECENT_CIRCUITS, LD_CIRC, circuit_build_times_t::liveness, networkstatus_get_param(), network_liveness_t::num_recent_circs, network_liveness_t::timeouts_after_firsthop, tor_assert, and tor_free.

Referenced by networkstatus_set_current_consensus().

4.35.5.110 int circuit_build_times_parse_state (circuit_build_times_t * *cbt*, or_state_t * *state*, char ** *msg*)

Load histogram from **state**, shuffling the resulting array after we do so. Use this result to estimate parameters and calculate the timeout.

Returns -1 and sets msg on error. Msg must be freed by the caller.

References or_state_t::BuildtimeHistogram, circuit_build_times_t::circuit_build_times, circuit_build_times_init(), circuit_build_times_set_timeout(), circuit_build_times_shuffle_and_store_array(), LD_CIRC, config_line_t::next, smartlist_create(), smartlist_free(), smartlist_split_string(), tor_assert, tor_free, tor_parse_ulong(), circuit_build_times_t::total_build_times, or_state_t::TotalBuildTimes, and config_line_t::value.

Referenced by or_state_set().

4.35.5.111 void circuit_build_times_set_timeout (circuit_build_times_t * *cbt*)

Estimate a new timeout based on history and set our timeout variable accordingly.

References circuit_build_times_t::alpha, circuit_build_times_calculate_timeout(), circuit_build_times_count_pretimeouts(), circuit_build_times_update_alpha(), control_event_buildtimeout_set(), circuit_build_times_t::have_computed_timeout, LD_CIRC, circuit_build_times_t::timeout_ms, tor_lround(), circuit_build_times_t::total_build_times, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_parse_state(), circuit_expire_building(), and circuit_send_next_onion_skin().

4.35.5.112 void circuit_build_times_update_state (circuit_build_times_t * *cbt*, or_state_t * *state*)

Output a histogram of current circuit build times to the **or_state_t** (p. 178) state structure.

References or_options_t::AvoidDiskWrites, or_state_t::BuildtimeHistogram, CBT_BIN_WIDTH, circuit_build_times_create_histogram(), config_free_lines(), get_options(), get_or_state(), config_line_t::key, config_line_t::next, or_state_mark_dirty(), tor_free, tor_snprintf(), circuit_build_times_t::total_build_times, or_state_t::TotalBuildTimes, unit_tests, and config_line_t::value.

Referenced by or_state_save().

4.35.5.113 void circuit_close_all_marked (void)

Detach from the global circuit list, and deallocate, all circuits that have been marked for close.

References circuit_free(), circuit_t::marked_for_close, and circuit_t::next.

Referenced by run_scheduled_events().

4.35.5.114 int circuit_conforms_to_options (const origin_circuit_t * *circ*, const or_options_t * *options*)

Check whether, according to the policies in **options**, the circuit **circ** makes sense.

References origin_circuit_t::cpath, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, crypt_path_t::extend_info, crypt_path_t::next, and routerset_contains_extendinfo().

4.35.5.115 int circuit_count_pending_on_or_conn (or_connection_t * *or_conn*)

Return the number of circuits in state OR_WAIT, waiting for the given connection.

References circuit_get_all_pending_on_or_conn(), LD_CIRC, or_connection_t::nickname, smartlist_create(), and smartlist_free().

Referenced by control_event_or_conn_status().

4.35.5.116 void circuit_detach_stream (circuit_t * *circ*, edge_connection_t * *conn*)

If the stream **conn** is a member of any of the linked lists of **circ**, then remove it from the list.

References CIRCUIT_IS_ORIGIN, edge_connection_t::cpath_layer, LD_BUG, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, origin_circuit_t::p_streams, or_circuit_t::resolving_streams, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and tor_fragile_assert.

Referenced by connection_about_to_close_connection(), connection_ap_detach_retriable(), connection_exit_connect(), dns_cancel_pending_resolve(), dns_found_answer(), handle_control_attachstream(), and purge_expired_resolves().

4.35.5.117 void circuit_discard_optional_exit_enclaves (extend_info_t * *info*)

A circuit failed to finish on its last hop **info**. If there are any streams waiting with this exit node in mind, but they don't absolutely require it, make them give up on it.

References socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, edge_connection_t::chosen_exit_retries, clear_trackexithost_mappings(), CONN_TYPE_AP, consider_plaintext_ports(), escaped_safe_str_client(), get_connection_array(), LD_APP, extend_info_t::nickname, socks_request_t::port, router_get_by_nickname(), safe_str_client(), edge_connection_t::socks_request, TO_EDGE_CONN(), tor_assert, and tor_free.

Referenced by circuit_build_failed().

4.35.5.118 void circuit_dump_by_conn (connection_t * *conn*, int *severity*)

Log, at severity **severity**, information about each circuit that is connected to **conn**.

References connection_t::addr, extend_info_t::addr, circuit_dump_details(), CIRCUIT_IS_ORIGIN, CIRCUIT_STATE_OPEN, connection_t::conn_array_index, CONN_TYPE_OR, DIGEST_LEN, extend_info_t::identity_digest, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, circuit_t::n_hop, circuit_t::next, edge_connection_t::next_stream, or_circuit_t::p_circ_id, connection_t::port, extend_info_t::port, circuit_t::state, TO_CONN, TO_OR_CIRCUIT(), TO_OR_CONN(), TO_ORIGIN_CIRCUIT(), tor_addr_eq, and connection_t::type.

Referenced by dumpstats().

4.35.5.119 const char* circuit_end_reason_to_control_string (int *reason*)

Convert a numeric reason for destroying a circuit into a string for a CIRCUIT event.

References END_CIRC_AT_ORIGIN, END_CIRC_REASON_FLAG_REMOTE, END_CIRC_REASON_NOPATH, and LD_BUG.

Referenced by control_event_circuit_status().

4.35.5.120 int circuit_enough_testing_circs (void)

Return 1 if we've already exercised our bandwidth, or if we have fewer than NUM_PARALLEL_TESTING_CIRCS testing circuits established or on the way. Else return 0.

References CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_TESTING, CIRCUIT_STATE_OPEN, have_performed_bandwidth_test, circuit_t::marked_for_close, circuit_t::next, NUM_PARALLEL_TESTING_CIRCS, circuit_t::purpose, and circuit_t::state.

Referenced by circuit_testing_opened(), consider_testing_reachability(), and rep_hist_circbuilding_dormant().

4.35.5.121 origin_circuit_t* circuit_establish_circuit (uint8_t *purpose*, extend_info_t * *exit*, int *flags*)

Build a new circuit for **purpose**. If **exit** is defined, then use that as your exit router, else choose a suitable exit node.

Also launch a connection to the first OR in the chosen path, if it's not open already.

References circuit_handle_first_hop(), control_event_circuit_status(), END_CIRC_-REASON_NOPATH, onion_pick_cpath_exit(), onion_populate_cpath(), origin_circuit_-init(), and TO_CIRCUIT.

Referenced by circuit_launch_by_extend_info().

4.35.5.122 void circuit_expire_all_dirty_circs (void)

Go through the circuitlist; for each circuit that starts at us and is dirty, frob its timestamp_dirty so we won't use it for any new streams.

This is useful for letting the user change pseudonyms, so new streams will not be linkable to old streams.

References CIRCUIT_IS_ORIGIN, get_options(), circuit_t::marked_for_close, or_options_-t::MaxCircuitDirtiness, circuit_t::next, and circuit_t::timestamp_dirty.

Referenced by circuit_note_clock_jumped(), do_hup(), options_act(), and signewnym_impl().

4.35.5.123 void circuit_expire_building (time_t *now*)

Close all circuits that start at us, aren't open, and were born at least CircuitBuildTimeout seconds ago.

References or_connection_t::_base, connection_t::address, origin_circuit_t::build_state, circ_times, circuit_build_times_add_timeout(), circuit_build_times_set_timeout(), CIRCUIT_IS_ORIGIN, circuit_log_path(), CIRCUIT_PURPOSE_C_ESTABLISH_-REND, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_-C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_-C_REND_READY_INTRO_ACKED, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_STATE_OPEN, circuit_state_to_string(), origin_circuit_t::cpath, LD_CIRC, LD_REND, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, circuit_-t::next, cpath_build_state_t::onehop_tunnel, connection_t::port, circuit_t::purpose, crypt_-path_t::state, circuit_t::state, circuit_build_times_t::timeout_ms, circuit_t::timestamp_-created, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by run_scheduled_events().

4.35.5.124 void circuit_expire_old_circuits_serverside (time_t *now*)

Find each non-origin circuit that has been unused for too long, has no streams on it, used a create_fast, and ends here: mark it for close.

References CIRCUIT_IS_ORIGIN, IDLE_ONE_HOP_CIRC_TIMEOUT, or_circuit_t::is_-first_hop, LD_CIRC, circuit_t::marked_for_close, circuit_t::n_conn, or_circuit_t::n_streams, circuit_t::next, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::resolving_-streams, or_connection_t::timestamp_last_added_nonpadding, and TO_OR_CIRCUIT().

Referenced by run_scheduled_events().

4.35.5.125 int circuit_extend (cell_t ∗ *cell*, circuit_t ∗ *circ*)

Take the 'extend' **cell**, pull out addr/port plus the onion skin and identity digest for the next hop. If we're already connected, pass the onion skin to the next hop using a create cell; otherwise launch a new OR connection, and **circ** will notice when the connection succeeds or fails.

Return -1 if we want to warn and tear down the circuit, else return 0.

References or_connection_t::_base, connection_t::address, circuit_deliver_create_cell(), circuit_set_state(), CIRCUIT_STATE_OR_WAIT, connection_or_connect(), connection_or_get_for_extend(), DIGEST_LEN, extend_info_alloc(), fmt_addr(), get_options(), get_uint16(), get_uint32(), LD_CIRC, LD_OR, LD_PROTOCOL, relay_header_t::length, circuit_t::n_conn, circuit_t::n_conn_onionskin, circuit_t::n_hop, cell_t::payload, connection_t::port, RELAY_HEADER_SIZE, relay_header_unpack(), server_mode(), TO_OR_CIRCUIT(), tor_addr_from_ipv4h, tor_assert, and tor_digest_is_zero().

Referenced by connection_edge_process_relay_cell().

4.35.5.126 int circuit_extend_to_new_exit (origin_circuit_t * *circ*, extend_info_t * *exit*)

Take an open **circ**, and add a new hop at the end, based on **info**. Set its state back to CIRCUIT_STATE_BUILDING, and then send the next extend cell to begin connecting to that hop.

References circuit_append_new_exit(), circuit_send_next_onion_skin(), circuit_set_state(), CIRCUIT_STATE_BUILDING, LD_CIRC, extend_info_t::nickname, TO_CIRCUIT, and warn_if_last_router_excluded().

Referenced by circuit_launch_by_extend_info(), and rend_client_introduction_acked().

4.35.5.127 origin_circuit_t* circuit_find_to_cannibalize (uint8_t *purpose*, extend_info_t * *info*, int *flags*)

Return a circuit that is open, is CIRCUIT_PURPOSE_C_GENERAL, has a timestamp_dirty value of 0, has flags matching the CIRCLAUNCH_* flags in **flags**, and if info is defined, does not already use info as any of its hops; or NULL if no circuit fits this description.

The **purpose** argument (currently ignored) refers to the purpose of the circuit we want to create, not the purpose of the circuit we want to cannibalize.

If !CIRCLAUNCH_NEED_UPTIME, prefer returning non-uptime circuits.

References origin_circuit_t::build_state, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_STATE_OPEN, origin_circuit_t::cpath, DIGEST_LEN, crypt_path_t::extend_info, extend_info_t::identity_digest, cpath_build_state_t::is_internal, LD_CIRC, circuit_t::marked_for_close, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, crypt_path_t::next, circuit_t::next, circuit_t::purpose, origin_circuit_t::remaining_relay_early_cells, router_get_by_digest(), routers_in_same_family(), circuit_t::state, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by circuit_launch_by_extend_info().

4.35.5.128 int circuit_finish_handshake (origin_circuit_t * *circ*, uint8_t *reply_type*, const char * *reply*)

A created or extended cell came back to us on the circuit, and it included **reply** as its body. (If **reply_type** is CELL_CREATED, the body contains (the second DH key, plus KH). If **reply_type** is CELL_CREATED_FAST, the body contains a secret y and a hash H(x|y).)

Calculate the appropriate keys and digests, make sure KH is correct, and initialize this hop of the cpath.

Return - reason if we want to mark circ for close, else return 0.

References CIPHER_KEY_LEN, circuit_init_cpath_crypto(), circuit_log_path(), control_event_circuit_status(), origin_circuit_t::cpath, crypto_dh_free(), crypt_path_t::dh_handshake_state, DIGEST_LEN, fast_client_handshake(), crypt_path_t::fast_handshake_state, crypt_path_t::handshake_digest, LD_CIRC, LD_PROTOCOL, onion_next_hop_in_cpath(), onion_skin_client_handshake(), crypt_path_t::state, and tor_assert.

Referenced by command_process_created_cell(), and connection_edge_process_relay_cell().

4.35.5.129 void circuit_free_all (void)

Release all storage held by circuits.

References circuit_free(), CIRCUIT_IS_ORIGIN, connection_free(), circuit_t::next, edge_connection_t::next_stream, or_circuit_t::resolving_streams, smartlist_free(), TO_CONN, and TO_OR_CIRCUIT().

Referenced by tor_free_all().

4.35.5.130 void circuit_get_all_pending_on_or_conn (smartlist_t * *out*, or_connection_t * *or_conn*)

Append to **out** all circuits in state OR_WAIT waiting for the given connection.

References or_connection_t::_base, connection_t::addr, CIRCUIT_STATE_OR_WAIT, DIGEST_LEN, or_connection_t::identity_digest, connection_t::port, smartlist_add(), tor_addr_eq, tor_assert, and tor_digest_is_zero().

Referenced by circuit_count_pending_on_or_conn(), and circuit_n_conn_done().

4.35.5.131 circuit_t* circuit_get_by_circid_orconn (circid_t *circ_id*, or_connection_t * *conn*)

Return a circ such that:

- circ->n_circ_id or circ->p_circ_id is equal to **circ_id**, and
- circ is attached to **conn**, either as p_conn or n_conn.
- circ is not marked for close. Return NULL if no such circuit exists.

References circuit_get_by_circid_orconn_impl(), and circuit_t::marked_for_close.

Referenced by command_process_created_cell(), command_process_destroy_cell(), command_process_relay_cell(), and connection_cpu_process_inbuf().

4.35.5.132 circuit_t* circuit_get_by_edge_conn (edge_connection_t * *conn*)

Return the circuit that a given edge connection is using.

References CIRCUIT_IS_ORIGIN, circuit_t::magic, edge_connection_t::on_circuit, and tor_assert.

Referenced by _connection_write_to_buf_impl(), connection_about_to_close_connection(), connection_ap_expire_beginning(), connection_edge_consider_sending_sendme(), connection_edge_end(), connection_edge_package_raw_inbuf(), connection_exit_connect(), control_event_stream_status(), dns_cancel_pending_resolve(), dns_found_answer(), getinfo_helper_events(), handle_control_attachstream(), and purge_expired_resolves().

4.35.5.133 origin_circuit_t* circuit_get_by_global_id (uint32_t *id*)

Return the circuit whose global ID is **id**, or NULL if no such circuit exists.

References CIRCUIT_IS_ORIGIN, circuit_t::marked_for_close, circuit_t::next, and TO_ORIGIN_CIRCUIT().

Referenced by get_circ().

4.35.5.134 origin_circuit_t* circuit_get_by_rend_query_and_purpose (const char * *rend_query*, uint8_t *purpose*)

Return a circ such that:

- circ->rend_data->query is equal to **rend_query**, and
- circ->purpose is equal to **purpose**.

Return NULL if no such circuit exists.

References CIRCUIT_PURPOSE_IS_ORIGIN, circuit_t::marked_for_close, circuit_t::next, rend_data_t::onion_address, circuit_t::purpose, rend_cmp_service_ids(), origin_circuit_t::rend_data, TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by rend_client_introduction_acked().

4.35.5.135 crypt_path_t* circuit_get_cpath_hop (origin_circuit_t * *circ*, int *hopnum*)

Return the **hopnum**th hop in **circ**->cpath, or NULL if there aren't that many hops in the list.

References origin_circuit_t::cpath, and crypt_path_t::next.

Referenced by handle_control_attachstream().

4.35.5.136 int circuit_get_cpath_len (origin_circuit_t * *circ*)

Return the number of hops in circuit's path.

References origin_circuit_t::cpath, and crypt_path_t::next.

Referenced by handle_control_attachstream(), and onion_extend_cpath().

4.35.5.137 or_circuit_t* circuit_get_intro_point (const char * *digest*)

Return the circuit waiting for intro cells of the given digest. Return NULL if no such circuit is found.

References circuit_get_by_rend_token_and_purpose(), CIRCUIT_PURPOSE_INTRO_POINT, and DIGEST_LEN.

Referenced by rend_mid_establish_intro(), and rend_mid_introduce().

4.35.5.138 origin_circuit_t∗ circuit_get_next_by_pk_and_purpose (origin_circuit_t ∗ *start*, const char ∗ *digest*, uint8_t *purpose*)

Return the first circuit originating here in global_circuitlist after **start** whose purpose is **purpose**, and where **digest** (if set) matches the rend_pk_digest field. Return NULL if no circuit is found. If **start** is NULL, begin at the start of the list.

References CIRCUIT_PURPOSE_IS_ORIGIN, DIGEST_LEN, circuit_t::marked_for_close, circuit_t::next, circuit_t::purpose, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by find_intro_circuit(), and router_perform_bandwidth_test().

4.35.5.139 or_circuit_t∗ circuit_get_rendezvous (const char ∗ *cookie*)

Return the circuit waiting for a rendezvous with the provided cookie. Return NULL if no such circuit is found.

References circuit_get_by_rend_token_and_purpose(), CIRCUIT_PURPOSE_REND_POINT_WAITING, and REND_COOKIE_LEN.

Referenced by rend_mid_establish_rendezvous(), and rend_mid_rendezvous().

4.35.5.140 int circuit_handle_first_hop (origin_circuit_t ∗ *circ*)

Start establishing the first hop of our circuit. Figure out what OR we should connect to, and if necessary start the connection to it. If we're already connected, then send the 'create' cell. Return 0 for ok, -reason if circ should be marked-for-close.

References origin_circuit_t::_base, extend_info_t::addr, origin_circuit_t::build_state, circuit_send_next_onion_skin(), connection_or_connect(), connection_or_get_for_extend(), control_event_bootstrap(), origin_circuit_t::cpath, crypt_path_t::extend_info, extend_info_dup(), fmt_addr(), extend_info_t::identity_digest, LD_CIRC, circuit_t::n_conn, circuit_t::n_hop, extend_info_t::nickname, cpath_build_state_t::onehop_tunnel, onion_next_hop_in_cpath(), extend_info_t::port, safe_str_client(), and tor_assert.

Referenced by circuit_establish_circuit(), and handle_control_extendcircuit().

4.35.5.141 void circuit_has_opened (origin_circuit_t ∗ *circ*)

The circuit **circ** has just become open. Take the next step: for rendezvous circuits, we pass circ to the appropriate function in rendclient or rendservice. For general circuits, we call connection_ap_attach_pending, which looks for pending streams that could use circ.

References CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, circuit_testing_opened(), connection_ap_attach_pending(), control_event_circuit_status(), rend_client_introcirc_has_opened(), rend_client_rendcirc_has_opened(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), and TO_CIRCUIT.

Referenced by circuit_send_next_onion_skin(), and rend_service_intro_has_opened().

4.35.5.142 int circuit_id_in_use_on_orconn (circid_t *circ_id*, or_connection_t ∗ *conn*)

Return true iff the circuit ID **circ_id** is currently used by a circuit, marked or not, on **conn**.

References circuit_get_by_circid_orconn_impl().

Referenced by command_process_create_cell(), and get_unique_circ_id_by_conn().

4.35.5.143 int circuit_init_cpath_crypto (crypt_path_t ∗ *cpath*, const char ∗ *key_data*, int *reverse*)

Initialize cpath->{f|b}_{crypto|digest} from the key material in key_data. key_data must contain CPATH_KEY_MATERIAL bytes, which are used as follows:

- 20 to initialize f_digest
- 20 to initialize b_digest
- 16 to key f_crypto
- 16 to key b_crypto

(If 'reverse' is true, then f_XX and b_XX are swapped.)

References crypt_path_t::b_crypto, crypt_path_t::b_digest, CIPHER_KEY_LEN, crypto_create_init_cipher(), crypto_digest_add_bytes(), crypto_new_digest_env(), DIGEST_LEN, crypt_path_t::f_crypto, crypt_path_t::f_digest, LD_BUG, and tor_assert.

Referenced by circuit_finish_handshake(), onionskin_answer(), rend_client_receive_rendezvous(), and rend_service_introduce().

4.35.5.144 int32_t circuit_initial_package_window (void)

Pick a reasonable package_window to start out for our circuits. Originally this was hard-coded at 1000, but now the consensus votes on the answer. See proposal 168.

References CIRCWINDOW_START, and networkstatus_get_param().

Referenced by init_circuit_base(), onion_append_hop(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

4.35.5.145 origin_circuit_t∗ circuit_launch_by_extend_info (uint8_t *purpose*, extend_info_t ∗ *extend_info*, int *flags*)

Launch a new circuit with purpose **purpose** and exit node **extend_info** (or NULL to select a random exit node). If flags contains CIRCLAUNCH_NEED_UPTIME, choose among routers with high uptime. If CIRCLAUNCH_NEED_CAPACITY is set, choose among routers with high bandwidth. If CIRCLAUNCH_IS_INTERNAL is true, the last hop need not be an exit node. If CIRCLAUNCH_ONEHOP_TUNNEL is set, the circuit will have only one hop. Return the newly allocated circuit on success, or NULL on failure.

References origin_circuit_t::_base, origin_circuit_t::build_state, build_state_get_exit_nickname(), CIRCLAUNCH_ONEHOP_TUNNEL, circuit_establish_circuit(), circuit_extend_to_new_exit(), circuit_find_to_cannibalize(), CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_TESTING, did_circs_fail_last_period, LD_BUG, LD_CIRC, MAX_CIRCUIT_FAILURES, n_circuit_failures, circuit_t::purpose, router_have_minimum_dir_info(), circuit_t::timestamp_created, tor_assert, and tor_fragile_assert.

Referenced by circuit_get_open_circ_or_launch(), circuit_launch_by_router(), rend_client_introduction_acked(), rend_service_introduce(), rend_service_launch_establish_intro(), and rend_service_relaunch_rendezvous().

4.35.5.146 origin_circuit_t* circuit_launch_by_router (uint8_t *purpose*, routerinfo_t * *exit*, int *flags*)

Launch a new circuit; see **circuit_launch_by_extend_info()** (p. 346) for details on arguments.

References circuit_launch_by_extend_info(), extend_info_free(), and extend_info_from_router().

Referenced by circuit_build_needed_circs(), circuit_predict_and_launch_new(), consider_testing_reachability(), and handle_control_extendcircuit().

4.35.5.147 char* circuit_list_path (origin_circuit_t * *circ*, int *verbose*)

If **verbose** is false, allocate and return a comma-separated list of the currently built elements of **circuit_t** (p. 40). If **verbose** is true, also list information about link status in a more verbose format using spaces.

References circuit_list_path_impl().

Referenced by circuit_log_path().

4.35.5.148 char* circuit_list_path_for_controller (origin_circuit_t * *circ*)

Allocate and return a comma-separated list of the currently built elements of **circuit_t** (p. 40), giving each as a verbose nickname.

References circuit_list_path_impl().

Referenced by control_event_circuit_status(), and getinfo_helper_events().

4.35.5.149 void circuit_log_path (int *severity*, unsigned int *domain*, origin_circuit_t * *circ*)

Log, at severity **severity**, the nicknames of each router in circ's cpath. Also log the length of the cpath, and the intended exit point.

References circuit_list_path(), tor_free, and tor_log().

Referenced by circuit_dump_details(), circuit_expire_building(), circuit_finish_handshake(), connection_ap_handshake_attach_circuit(), connection_ap_process_end_not_open(), connection_edge_process_relay_cell_not_open(), and warn_if_last_router_excluded().

4.35.5.150 void circuit_mark_all_unused_circs (void)

Go through the circuitlist; mark-for-close each circuit that starts at us but has not yet been used.

References CIRCUIT_IS_ORIGIN, circuit_t::marked_for_close, circuit_t::next, and circuit_t::timestamp_dirty.

Referenced by circuit_note_clock_jumped(), and options_act().

4.35.5.151 void circuit_n_conn_done (or_connection_t * *or_conn*, int *status*)

Find any circuits that are waiting on **or_conn** to become open and get them to send their create cells forward.

Status is 1 if connect succeeded, or 0 if connect failed.

References or_connection_t::_base, connection_t::addr, connection_t::address, circuit_deliver_create_cell(), circuit_get_all_pending_on_or_conn(), CIRCUIT_IS_ORIGIN, circuit_send_next_onion_skin(), circuit_set_state(), CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, DIGEST_LEN, extend_info_free(), or_connection_t::identity_digest, LD_CIRC, or_connection_t::nickname, connection_t::port, smartlist_create(), smartlist_free(), TO_ORIGIN_CIRCUIT(), tor_addr_eq, tor_assert, tor_digest_is_zero(), and tor_free.

Referenced by connection_about_to_close_connection(), and connection_or_set_state_open().

4.35.5.152 void circuit_note_clock_jumped (int *seconds_elapsed*)

Our clock just jumped by **seconds_elapsed**. Assume something has also gone wrong with our network: notify the user, and abandon all not-yet-used circuits.

References circuit_expire_all_dirty_circs(), circuit_mark_all_unused_circs(), control_event_client_status(), control_event_general_status(), get_options(), has_completed_circuit, LD_GENERAL, server_mode(), and tor_log().

Referenced by second_elapsed_callback().

4.35.5.153 const char* circuit_purpose_to_controller_string (uint8_t *purpose*)

Map a circuit purpose to a string suitable to be displayed to a controller.

References CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCE_ACKED, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_PURPOSE_CONTROLLER, CIRCUIT_PURPOSE_INTRO_POINT, CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_ESTABLISHED, CIRCUIT_PURPOSE_REND_POINT_WAITING, CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, CIRCUIT_PURPOSE_S_REND_JOINED, CIRCUIT_PURPOSE_TESTING, and tor_snprintf().

Referenced by control_event_circuit_status(), and getinfo_helper_events().

4.35.5.154 int circuit_receive_relay_cell (cell_t * *cell*, circuit_t * *circ*, cell_direction_t *cell_direction*)

Receive a relay cell:

- Crypt it (encrypt if headed toward the origin or if we **are** the origin; decrypt if we're headed toward the exit).
- Check if recognized (if exitward).
- If recognized and the digest checks out, then find if there's a stream that the cell is intended for, and deliver it to the right connection_edge.
- If not recognized, then we need to relay it: append it to the appropriate cell_queue on **circ**.

Return -**reason** on failure.

References or_circuit_t::_base, append_cell_to_circuit_queue(), CELL_DIRECTION_IN, CELL_DIRECTION_OUT, cell_t::circ_id, CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_REND_ESTABLISHED, circuit_receive_relay_cell(), cell_t::command, connection_edge_process_relay_cell(), LD_BUG, LD_OR, LD_PROTOCOL, LD_REND, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, or_circuit_t::p_circ_id, or_circuit_t::p_conn, circuit_t::purpose, relay_crypt(), relay_lookup_conn(), or_circuit_t::rend_splice, TO_CIRCUIT, TO_OR_CIRCUIT(), and tor_assert.

Referenced by circuit_receive_relay_cell(), and command_process_relay_cell().

4.35.5.155 void circuit_remove_handled_ports (smartlist_t * *needed_ports*)

Remove any elements in **needed_ports** that are handled by an open or in-progress circuit.

References circuit_stream_is_being_handled(), LD_CIRC, MIN_CIRCUITS_HANDLING_STREAM, smartlist_del(), tor_assert, and tor_free.

Referenced by circuit_get_unhandled_ports().

4.35.5.156 void circuit_rep_hist_note_result (origin_circuit_t * *circ*)

Tell the rep(utation)hist(ory) module about the status of the links in circ. Hops that have become OPEN are marked as successfully extended; the _first_ hop that isn't open (if any) is marked as unable to extend.

References routerinfo_t::cache_info, origin_circuit_t::cpath, crypt_path_t::extend_info, get_options(), extend_info_t::identity_digest, signed_descriptor_t::identity_digest, crypt_path_t::next, rep_hist_note_extend_failed(), rep_hist_note_extend_succeeded(), router_get_by_digest(), router_get_my_routerinfo(), server_mode(), and crypt_path_t::state.

Referenced by _circuit_mark_for_close(), and circuit_send_next_onion_skin().

4.35.5.157 void circuit_reset_failure_count (int *timeout*)

Reset the failure count for opening general circuits. This means we will try MAX_CIRCUIT_FAILURES times more (if necessary) before stopping again.

References did_circs_fail_last_period, MAX_CIRCUIT_FAILURES, and n_circuit_failures.

Referenced by circuit_build_needed_circs(), and circuit_send_next_onion_skin().

4.35.5.158 int circuit_send_next_onion_skin (origin_circuit_t * *circ*)

This is the backbone function for building circuits.

If circ's first hop is closed, then we need to build a create cell and send it forward.

Otherwise, we need to build a relay extend cell and send it forward.

Return -reason if we want to tear down circ, else return 0.

References origin_circuit_t::_base, extend_info_t::addr, origin_circuit_t::build_state, check_whether_orport_reachable(), circuit_build_times_add_time(), circuit_build_times_network_circ_success(), circuit_build_times_set_timeout(), circuit_deliver_create_cell(), circuit_has_opened(), circuit_rep_hist_note_result(), circuit_reset_failure_count(), circuit_set_state(), CIRCUIT_STATE_BUILDING, CIRCUIT_STATE_OPEN, consider_testing_reachability(), control_event_bootstrap(), control_event_client_status(), origin_circuit_t::cpath, crypto_rand(), crypt_path_t::dh_handshake_state, DIGEST_LEN, crypt_path_t::extend_info, crypt_path_t::fast_handshake_state, get_options(), has_completed_circuit, circuit_t::highres_created, extend_info_t::identity_digest, or_connection_t::identity_digest, inform_testing_reachability(), LD_BUG, LD_CIRC, LD_GENERAL, circuit_t::n_conn, routerinfo_t::nickname, cpath_build_state_t::onehop_tunnel, extend_info_t::onion_key, onion_next_hop_in_cpath(), onion_skin_create(), extend_info_t::port, crypt_path_t::prev, relay_send_command_from_edge(), router_get_by_digest(), server_mode(), set_uint16(), set_uint32(), should_use_create_fast_for_circuit(), circuit_t::state, crypt_path_t::state, TO_CIRCUIT, tor_addr_family(), tor_addr_to_ipv4n(), tor_assert, tor_gettimeofday(), and tv_mdiff().

Referenced by circuit_extend_to_new_exit(), circuit_handle_first_hop(), circuit_n_conn_done(), command_process_created_cell(), connection_edge_process_relay_cell(), and handle_control_extendcircuit().

4.35.5.159 void circuit_set_n_circid_orconn (circuit_t * *circ*, circid_t *id*, or_connection_t * *conn*)

Set the n_conn field of a circuit **circ**, along with the corresponding circuit ID, and add the circuit as appropriate to the (orconn,id)->circuit map.

References bool_eq, CELL_DIRECTION_OUT, cell_queue_t::n, circuit_t::n_conn_cells, circuit_t::next_active_on_n_conn, and tor_assert.

Referenced by circuit_deliver_create_cell(), circuit_free(), circuit_unlink_all_from_or_conn(), command_process_destroy_cell(), and connection_edge_process_relay_cell().

4.35.5.160 void circuit_set_p_circid_orconn (or_circuit_t * *circ*, circid_t *id*, or_connection_t * *conn*)

Set the p_conn field of a circuit **circ**, along with the corresponding circuit ID, and add the circuit as appropriate to the (orconn,id)->circuit map.

References bool_eq, CELL_DIRECTION_IN, cell_queue_t::n, or_circuit_t::next_active_on_p_conn, or_circuit_t::p_conn_cells, TO_CIRCUIT, and tor_assert.

Referenced by circuit_free(), circuit_unlink_all_from_or_conn(), command_process_destroy_cell(), and or_circuit_new().

4.35.5.161 void circuit_set_state (circuit_t ∗ *circ*, uint8_t *state*)

Change the state of **circ** to **state**, adding it to or removing it from lists as appropriate.

References CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, circuit_t::n_conn_onionskin, smartlist_add(), smartlist_create(), smartlist_remove(), circuit_t::state, and tor_assert.

Referenced by circuit_extend(), circuit_extend_to_new_exit(), circuit_n_conn_done(), circuit_send_next_onion_skin(), command_process_create_cell(), handle_control_extendcircuit(), onionskin_answer(), and origin_circuit_init().

4.35.5.162 const char∗ circuit_state_to_string (int *state*)

Function to make circ->state human-readable

References CIRCUIT_STATE_BUILDING, CIRCUIT_STATE_ONIONSKIN_PENDING, CIRCUIT_STATE_OPEN, CIRCUIT_STATE_OR_WAIT, LD_BUG, and tor_snprintf().

Referenced by circuit_dump_details(), circuit_expire_building(), and rend_service_dump_stats().

4.35.5.163 int circuit_stream_is_being_handled (edge_connection_t ∗ *conn*, uint16_t *port*, int *min*)

Return 1 if at least **min** general-purpose non-internal circuits will have an acceptable exit node for exit stream **conn** if it is defined, else for "∗:port". Else return 0.

References ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, origin_circuit_t::build_state, build_state_get_exit_router(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_C_GENERAL, compare_addr_to_addr_policy(), connection_ap_can_use_exit(), routerinfo_t::exit_policy, get_options(), cpath_build_state_t::is_internal, circuit_t::marked_for_close, cpath_build_state_t::need_uptime, circuit_t::next, cpath_build_state_t::onehop_tunnel, socks_request_t::port, circuit_t::purpose, smartlist_string_num_isin(), edge_connection_t::socks_request, circuit_t::timestamp_dirty, and TO_ORIGIN_CIRCUIT().

Referenced by ap_stream_wants_exit_attention(), and circuit_remove_handled_ports().

4.35.5.164 int circuit_truncated (origin_circuit_t ∗ *circ*, crypt_path_t ∗ *layer*)

We received a relay truncated cell on circ.

Since we don't ask for truncates currently, getting a truncated means that a connection broke or an extend failed. For now, just give up: for circ to close, and return 0.

References circuit_free_cpath_node(), origin_circuit_t::cpath, END_CIRC_REASON_FLAG_REMOTE, LD_APP, LD_CIRC, crypt_path_t::next, edge_connection_t::next_stream, origin_circuit_t::p_streams, TO_CIRCUIT, and tor_assert.

Referenced by connection_edge_process_relay_cell().

4.35.5.165 void circuit_unlink_all_from_or_conn (or_connection_t ∗ *conn*, int *reason*)

For each circuit that has **conn** as n_conn or p_conn, unlink the circuit from the orconn,circid map, and mark it for close if it hasn't been marked already.

References CIRCUIT_IS_ORIGIN, circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), connection_or_unlink_all_active_circs(), circuit_t::marked_for_close, circuit_t::n_conn, circuit_t::next, or_circuit_t::p_conn, and TO_OR_CIRCUIT().

Referenced by connection_about_to_close_connection().

4.35.5.166 void clean_cell_pool (void)

Free excess storage in cell pool.

References tor_assert.

Referenced by run_scheduled_events().

4.35.5.167 void clear_bridge_list (void)

Initialize the bridge list to empty, creating it if needed.

References smartlist_clear(), smartlist_create(), and tor_free.

Referenced by entry_guards_free_all(), and options_act().

4.35.5.168 void clear_pending_onions (void)

Remove all circuits from the pending list. Called from tor_free_all.

References onion_queue_t::next, ol_length, onion_queue_t::onionskin, and tor_free.

Referenced by tor_free_all().

4.35.5.169 void clear_trusted_dir_servers (void)

Remove all members from the list of trusted dir servers.

References router_dir_info_changed(), smartlist_clear(), smartlist_create(), trusted_dir_server_free(), and trusted_dir_servers.

Referenced by consider_adding_dir_authorities().

4.35.5.170 void client_dns_clear_failures (const char ∗ *address*)

If **address** is in the client DNS addressmap, reset the number of resolve failures we have on record for it. This is used when we fail a stream because it won't resolve: otherwise future attempts on that address will only try once.

References addressmap, addressmap_entry_t::num_resolve_failures, and strmap_get().

Referenced by connection_ap_process_end_not_open().

4.35.5.171 int client_dns_incr_failures (const char * *address*)

An attempt to resolve **address** failed at some OR. Increment the number of resolve failures we have on record for it, and then return that number.

References addressmap, addressmap_entry_t::expires, LD_APP, MAX_DNS_ENTRY_AGE, addressmap_entry_t::num_resolve_failures, safe_str_client(), strmap_get(), and strmap_set().

Referenced by connection_ap_process_end_not_open().

4.35.5.172 void client_dns_set_addressmap (const char * *address*, uint32_t *val*, const char * *exitname*, int *ttl*)

Record the fact that **address** resolved to **val**. We can now use this in subsequent streams via **addressmap_rewrite()** (p. 434) so we can more correctly choose an exit that will allow **address**.

If **exitname** is defined, then append the addresses with ".exitname.exit" before registering the mapping.

If **ttl** is nonnegative, the mapping will be valid for **ttl**seconds; otherwise, we use the default.

References client_dns_set_addressmap_impl(), INET_NTOA_BUF_LEN, tor_assert, tor_inet_aton(), and tor_inet_ntoa().

Referenced by connection_ap_handshake_socks_resolved(), connection_ap_process_end_not_open(), and connection_edge_process_relay_cell_not_open().

4.35.5.173 int cmp_addr_policies (smartlist_t * *a*, smartlist_t * *b*)

Like **cmp_single_addr_policy()** (p. 1033) above, but looks at the whole set of policies in each case.

References cmp_single_addr_policy().

Referenced by router_differences_are_cosmetic().

4.35.5.174 void command_process_cell (cell_t * *cell*, or_connection_t * *conn*)

Process a **cell** that was just received on **conn**. Keep internal statistics about how many of each cell we've processed so far this second, and the total number of microseconds it took to process each type of cell.

References or_connection_t::_base, cell_t::command, LD_OR, LD_PROTOCOL, OR_CONN_STATE_OR_HANDSHAKING, connection_t::state, stats_n_create_cells_processed, stats_n_created_cells_processed, stats_n_destroy_cells_processed, stats_n_netinfo_cells_processed, stats_n_padding_cells_processed, stats_n_relay_cells_processed, and tor_fragile_assert.

Referenced by connection_or_process_cells_from_inbuf().

4.35.5.175 void command_process_var_cell (var_cell_t * *cell*, or_connection_t * *conn*)

Process a **cell** that was just received on **conn**. Keep internal statistics about how many of each cell we've processed so far this second, and the total number of microseconds it took to process each type of cell.

References or_connection_t::_base, var_cell_t::command, LD_BUG, LD_OR, OR_CONN_STATE_OR_HANDSHAKING, connection_t::state, stats_n_versions_cells_processed, and tor_fragile_assert.

Referenced by connection_or_process_cells_from_inbuf().

4.35.5.176 addr_policy_result_t compare_addr_to_addr_policy (uint32_t *addr*, uint16_t *port*, const smartlist_t ∗ *policy*)

As compare_tor_addr_to_addr_policy, but instead of a **tor_addr_t** (p. 237), takes in host order.

References compare_tor_addr_to_addr_policy(), and tor_addr_from_ipv4h.

Referenced by circuit_stream_is_being_handled(), connection_ap_can_use_exit(), router_exit_policy_all_routers_reject(), and router_handles_some_port().

4.35.5.177 int compare_digest_to_routerstatus_entry (const void ∗ *_key*, const void ∗∗ *_member*)

Helper for bsearching a list of **routerstatus_t** (p. 218) pointers: compare a digest in the key to the identity digest of a **routerstatus_t** (p. 218).

References DIGEST_LEN, and routerstatus_t::identity_digest.

Referenced by measured_bw_line_apply(), networkstatus_v2_find_entry(), networkstatus_vote_find_entry(), networkstatus_vote_find_entry_idx(), and router_get_consensus_status_by_id().

4.35.5.178 int compare_routerstatus_entries (const void ∗∗ *_a*, const void ∗∗ *_b*)

Helper to sort a smartlist of pointers to **routerstatus_t** (p. 218)

References DIGEST_LEN, and routerstatus_t::identity_digest.

Referenced by dirserv_read_measured_bandwidths(), and networkstatus_v2_parse_from_string().

4.35.5.179 addr_policy_result_t compare_tor_addr_to_addr_policy (const tor_addr_t ∗ *addr*, uint16_t *port*, const smartlist_t ∗ *policy*)

Decide whether a given addr:port is definitely accepted, definitely rejected, probably accepted, or probably rejected by a given policy. If **addr** is 0, we don't know the IP of the target address. If **port** is 0, we don't know the port of the target address. (At least one of **addr** and **port** must be provided. If you want to know whether a policy would definitely reject an unknown address:port, use **policy_is_reject_star()** (p. 957).)

We could do better by assuming that some ranges never match typical addresses (127.0.0.1, and so on). But we'll try this for now.

References ADDR_POLICY_ACCEPTED, compare_known_tor_addr_to_addr_policy(), compare_known_tor_addr_to_addr_policy_noport(), compare_unknown_tor_addr_to_addr_policy(), tor_addr_is_null(), and tor_assert.

Referenced by addr_policy_permits_tor_addr(), compare_addr_to_addr_policy(), router_compare_to_my_exit_policy(), router_find_exact_exit_enclave(), and routerset_contains().

4.35.5.180 void config_free_all (void)

Release all memory and resources held by global configuration structures.

References config_free(), config_free_lines(), or_options_free(), and tor_free.

Referenced by tor_free_all().

4.35.5.181 void config_free_lines (config_line_t * *front*)

Free all the configuration lines on the linked list **front**.

References config_line_t::key, config_line_t::next, tor_free, and config_line_t::value.

Referenced by circuit_build_times_update_state(), config_dump(), config_free(), config_free_all(), config_get_commandlines(), config_get_lines(), control_setconf_helper(), dirserv_load_fingerprint_file(), entry_guards_update_state(), option_clear(), option_is_same(), option_reset(), options_dup(), options_init_from_string(), and or_state_load().

4.35.5.182 int config_get_lines (const char * *string*, config_line_t ** *result*)

Helper: parse the config string and strdup into key/value strings. Set *result to the list, or NULL if parsing the string failed. Return 0 on success, -1 on failure. Warn and ignore any misformatted lines.

References config_free_lines(), parse_config_line_from_str(), and tor_free.

Referenced by control_setconf_helper(), dirserv_load_fingerprint_file(), options_init_from_string(), and or_state_load().

4.35.5.183 void configure_accounting (time_t *now*)

Initialize the accounting subsystem.

References accounting_set_wakeup_time(), interval_end_time, interval_start_time, LD_ACCT, read_bandwidth_usage(), reset_accounting(), start_of_accounting_period_after(), and start_of_accounting_period_containing().

Referenced by accounting_run_housekeeping(), and options_act().

4.35.5.184 const char* conn_state_to_string (int *type*, int *state*)

Return the human-readable name for the connection state **state** for the connection type **type**

References AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_RESOLVE_WAIT, AP_CONN_STATE_SOCKS_WAIT, CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_CPUWORKER,

CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_EXIT, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, conn_type_to_string(), CONTROL_CONN_STATE_NEEDAUTH, CONTROL_CONN_STATE_OPEN, CPUWORKER_STATE_BUSY_ONION, CPUWORKER_STATE_IDLE, DIR_CONN_STATE_CLIENT_FINISHED, DIR_CONN_STATE_CLIENT_READING, DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_CONNECTING, DIR_CONN_STATE_SERVER_COMMAND_WAIT, DIR_CONN_STATE_SERVER_WRITING, EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, EXIT_CONN_STATE_RESOLVEFAILED, EXIT_CONN_STATE_RESOLVING, LD_BUG, LISTENER_STATE_READY, OR_CONN_STATE_CONNECTING, OR_CONN_STATE_OPEN, OR_CONN_STATE_OR_HANDSHAKING, OR_CONN_STATE_PROXY_HANDSHAKING, OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING, OR_CONN_STATE_TLS_HANDSHAKING, OR_CONN_STATE_TLS_SERVER_RENEGOTIATING, and tor_snprintf().

Referenced by _connection_free(), connection_ap_expire_beginning(), connection_buckets_decrement(), connection_close_immediate(), connection_edge_process_inbuf(), connection_edge_process_relay_cell_not_open(), connection_expire_held_open(), and dumpstats().

4.35.5.185 const char∗ conn_type_to_string (int *type*)

Return the human-readable name for the connection type **type**

References CONN_TYPE_AP, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_CPUWORKER, CONN_TYPE_DIR, CONN_TYPE_DIR_LISTENER, CONN_TYPE_EXIT, CONN_TYPE_OR, CONN_TYPE_OR_LISTENER, LD_BUG, and tor_snprintf().

Referenced by _connection_free(), check_sockaddr_family_match(), conn_close_if_marked(), conn_read_callback(), conn_state_to_string(), conn_write_callback(), connection_add(), connection_buckets_decrement(), connection_close_immediate(), connection_create_listener(), connection_expire_held_open(), connection_remove(), dumpstats(), and options_act_reversible().

4.35.5.186 void connection_about_to_close_connection (connection_t ∗ *conn*)

Do any cleanup needed:

- Directory conns that failed to fetch a rendezvous descriptor need to inform pending rendezvous streams.
- OR conns need to call rep_hist_note_∗() to record status.
- AP conns need to send a socks reject if necessary.
- Exit conns need to call **connection_dns_remove()** (p. 632) if necessary.
- AP and Exit conns need to send an end cell if they can.
- DNS conns need to fail any resolves that are pending on them.
- OR and edge connections need to be unlinked from circuits.

References authdir_mode_tests_reachability(), circuit_detach_stream(), circuit_get_by_edge_conn(), circuit_n_conn_done(), circuit_unlink_all_from_or_conn(), CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_dir_request_failed(), connection_dns_remove(), connection_or_nonopen_was_started_here(), control_event_bootstrap_problem(), control_event_or_conn_status(), control_event_stream_bandwidth(), control_event_stream_status(), DIR_CONN_STATE_CLIENT_FINISHED, DIR_PURPOSE_FETCH_RENDDESC_V2, edge_connection_t::dns_server_request, dnsserv_reject_request(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, entry_guard_register_connect_status(), EXIT_CONN_STATE_RESOLVING, get_options(), socks_request_t::has_finished, connection_t::hold_open_until_flushed, or_options_t::HttpsProxy, or_connection_t::identity_digest, LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, rend_data_t::onion_address, OR_CONN_STATE_OPEN, OR_CONN_STATE_TLS_HANDSHAKING, orconn_end_reason_to_control_string(), connection_t::purpose, rend_client_refetch_v2_renddesc(), dir_connection_t::rend_data, REND_SERVICE_ID_LEN_BASE32, rep_hist_note_connect_failed(), rep_hist_note_connection_died(), rep_hist_note_disconnect(), edge_connection_t::socks_request, connection_t::state, or_connection_t::tls_error, tls_error_to_orconn_end_reason(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CONN(), tor_assert, tor_digest_is_zero(), tor_fragile_assert, and connection_t::type.

Referenced by connection_unlink().

4.35.5.187 int connection_add (connection_t * *conn*)

Add **conn** to the array of connections that we can poll on. The connection's socket must be set; the connection starts out non-reading and non-writing.

References connection_t::address, connection_t::conn_array_index, conn_read_callback(), CONN_TYPE_AP, conn_type_to_string(), conn_write_callback(), LD_NET, connection_t::linked, connection_t::read_event, connection_t::s, smartlist_add(), TO_EDGE_CONN(), tor_assert, tor_event_new(), tor_libevent_get_base(), connection_t::type, and connection_t::write_event.

Referenced by connection_ap_make_link(), connection_connect(), connection_create_listener(), connection_exit_connect_dir(), connection_handle_listener_read(), directory_initiate_command_rend(), dnsserv_launch_request(), evdns_server_callback(), and spawn_cpuworker().

4.35.5.188 void connection_ap_attach_pending (void)

Tell any AP streams that are waiting for a new circuit to try again, either attaching to an available circ or launching a new one.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, CONN_TYPE_AP, connection_ap_handshake_attach_circuit(), END_STREAM_REASON_CANT_ATTACH, get_connection_array(), connection_t::marked_for_close, and TO_EDGE_CONN().

Referenced by circuit_build_needed_circs(), circuit_has_opened(), rend_client_introcirc_has_opened(), rend_client_receive_rendezvous(), and rend_client_rendezvous_acked().

4.35.5.189 int connection_ap_can_use_exit (edge_connection_t * *conn*, routerinfo_t * *exit*, int *excluded_means_no*)

Return 1 if router **exit** is likely to allow stream **conn** to exit from it, or 0 if it probably will not allow it. (We might be uncertain if conn's destination address has not yet been resolved.)

If **excluded_means_no** is 1 and Exclude*Nodes is set and excludes this relay, return 0.

References edge_connection_t::_base, or_options_t::_ExcludeExitNodesUnion, ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, socks_request_t::address, routerinfo_t::cache_info, edge_connection_t::chosen_exit_name, socks_request_t::command, compare_addr_to_addr_policy(), CONN_TYPE_AP, DIGEST_LEN, routerinfo_t::exit_policy, get_options(), routerinfo_t::has_old_dnsworkers, signed_descriptor_t::identity_digest, policy_is_reject_star(), socks_request_t::port, router_get_by_nickname(), routerset_contains_router(), SOCKS_COMMAND_CONNECT, SOCKS_COMMAND_RESOLVE_PTR, edge_connection_t::socks_request, or_options_t::StrictNodes, tor_assert, tor_inet_aton(), connection_t::type, and edge_connection_t::use_begindir.

Referenced by choose_good_exit_server_general(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_stream_is_being_handled(), and connection_ap_handshake_attach_circuit().

4.35.5.190 int connection_ap_detach_retriable (edge_connection_t ∗ *conn*, origin_circuit_t ∗ *circ*, int *reason*)

The AP connection **conn** has just failed while attaching or sending a BEGIN or resolving on **circ**, but another circuit might work. Detach the circuit, and either reattach it, launch a new circuit, tell the controller, or give up as a appropriate.

Returns -1 on err, 1 on success, 0 on not-yet-sure.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, circuit_detach_stream(), connection_ap_handshake_attach_circuit(), control_event_stream_status(), get_options(), connection_t::state, connection_t::timestamp_lastread, TO_CIRCUIT, and edge_connection_t::use_begindir.

Referenced by connection_ap_expire_beginning(), and connection_ap_process_end_not_open().

4.35.5.191 void connection_ap_expire_beginning (void)

Find all general-purpose AP streams waiting for a response that sent their begin/resolve cell too long ago. Detach from their current circuit, and mark their current circuit as unsuitable for new streams. Then call **connection_ap_handshake_attach_circuit()** (p. 349) to attach to a new circuit (if available) or launch a new one.

For rendezvous streams, simply give up after SocksTimeout seconds (with no retry attempt).

References edge_connection_t::_base, connection_t::addr, socks_request_t::address, AP_CONN_STATE_IS_UNATTACHED, AP_CONN_STATE_OPEN, circuit_get_by_edge_conn(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_REND_JOINED, compute_retry_timeout(), conn_state_to_string(), CONN_TYPE_AP, connection_ap_detach_retriable(), connection_edge_end(), edge_connection_t::cpath_layer, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_CANT_ATTACH, crypt_path_t::extend_info, get_connection_array(), get_options(), LD_APP, LD_REND, connection_t::marked_for_close, or_options_t::MaxCircuitDirtiness, extend_info_t::nickname, edge_connection_t::num_socks_retries, socks_request_t::port, connection_t::port, circuit_t::purpose, safe_str_client(), edge_connection_t::socks_request, or_options_t::SocksTimeout, connection_t::state, connection_t::timestamp_created, circuit_t::timestamp_dirty, connection_t::timestamp_lastread, TO_EDGE_CONN(), TO_ORIGIN_CIRCUIT(), tor_addr_is_null(), and tor_assert.

Referenced by run_scheduled_events().

4.35.5.192 void connection_ap_fail_onehop (const char ∗ *failed_digest*, cpath_build_state_t ∗ *build_state*)

Tell any AP streams that are waiting for a one-hop tunnel to **failed_digest** that they are going to fail.

References extend_info_t::addr, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, cpath_build_state_t::chosen_exit, edge_connection_t::chosen_exit_name, CONN_TYPE_AP, DIGEST_LEN, get_connection_array(), hexdigest_to_digest(), LD_APP, socks_request_t::port, extend_info_t::port, edge_connection_t::socks_request, TO_EDGE_CONN(), tor_addr_eq, tor_addr_from_str(), tor_digest_is_zero(), and edge_connection_t::want_onehop.

Referenced by circuit_build_failed().

4.35.5.193 int connection_ap_handshake_attach_chosen_circuit (edge_connection_t ∗ *conn*, origin_circuit_t ∗ *circ*, crypt_path_t ∗ *cpath*)

Attempt to attach the connection **conn** to **circ**, and send a begin or resolve cell as appropriate. Return values are as for connection_ap_handshake_attach_circuit. The stream will exit from the hop indicated by **cpath**, or from the last hop in circ's cpath if **cpath** is NULL.

References origin_circuit_t::_base, edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, CIRCUIT_STATE_OPEN, socks_request_t::command, connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), consider_recording_trackhost(), link_apconn_to_circ(), SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, circuit_t::state, connection_t::state, circuit_t::timestamp_dirty, tor_assert, and edge_connection_t::use_begindir.

Referenced by connection_ap_handshake_attach_circuit().

4.35.5.194 int connection_ap_handshake_attach_circuit (edge_connection_t ∗ *conn*)

Try to find a safe live circuit for CONN_TYPE_AP connection conn. If we don't find one: if conn cannot be handled by any known nodes, warn and return -1 (conn needs to die, and is maybe already marked); else launch new circuit (if necessary) and return 0. Otherwise, associate conn with a safe live circuit, do the right next step, and return 1.

References origin_circuit_t::_base, edge_connection_t::_base, connection_t::addr, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, assert_circuit_ok(), edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, circuit_get_open_circ_or_launch(), CIRCUIT_IS_ORIGIN, circuit_log_path(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCUIT_STATE_OPEN, connection_ap_can_use_exit(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_send_begin(), connection_edge_is_rendezvous_stream(), edge_connection_t::cpath_layer, get_options(), LD_APP, LD_CIRC, LD_REND, link_apconn_to_circ(), circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::next, rend_data_t::onion_address, socks_request_t::port, connection_t::port, circuit_t::purpose, rend_client_send_introduction(), rend_cmp_service_ids(), edge_connection_t::rend_data, origin_circuit_t::rend_data, router_get_by_nickname(), safe_str_client(), edge_connection_t::socks_request, or_options_t::SocksTimeout, connection_t::state, connection_t::timestamp_created, circuit_t::timestamp_dirty, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), tor_addr_is_null(), tor_assert, tor_free, and edge_connection_t::want_onehop.

Referenced by connection_ap_attach_pending(), connection_ap_detach_retriable(), connection_ap_make_link(), and rend_client_desc_trynow().

4.35.5.195 int connection_ap_handshake_rewrite_and_attach (edge_connection_t * *conn*, origin_circuit_t * *circ*, crypt_path_t * *cpath*)

Connection **conn** just finished its socks handshake, or the controller asked us to take care of it. If **circ** is defined, then that's where we'll want to attach it. Otherwise we have to figure it out ourselves.

First, parse whether it's a .exit address, remap it, and so on. Then if it's for a general circuit, try to attach it to a circuit (or launch one as needed), else if it's for a rendezvous circuit, fetch a rendezvous descriptor first (or attach/launch a circuit if the rendezvous descriptor is already here and fresh enough).

The stream will exit from the hop indicated by **cpath**, or from the last hop in circ's cpath if **cpath** is NULL.

References socks_request_t::address, addressmap_register_virtual_address(), or_options_t::AutomapHostsOnResolve, or_options_t::AutomapHostsSuffixes, socks_request_t::command, escaped_safe_str_client(), get_options(), LD_APP, socks_request_t::port, safe_str_client(), SOCKS_COMMAND_RESOLVE, edge_connection_t::socks_request, strcasecmpend(), tor_assert, tor_inet_aton(), and tor_strlower().

Referenced by connection_ap_handshake_process_socks(), connection_ap_process_natd(), connection_ap_process_transparent(), dnsserv_launch_request(), evdns_server_callback(), and handle_control_attachstream().

4.35.5.196 int connection_ap_handshake_send_begin (edge_connection_t * *ap_conn*)

Write a relay begin cell, using destaddr and destport from ap_conn's socks_request field, and send it down circ.

If ap_conn is broken, mark it for close and return -1. Else return 0.

References origin_circuit_t::_base, edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, origin_circuit_t::build_state, CELL_PAYLOAD_SIZE, CIRCUIT_PURPOSE_C_GENERAL, socks_request_t::command, CONN_TYPE_AP, connection_edge_send_command(), control_event_stream_status(), edge_connection_t::deliver_window, get_unique_stream_id_by_circ(), LD_APP, circuit_t::n_circ_id, edge_connection_t::on_circuit, cpath_build_state_t::onehop_tunnel, edge_connection_t::package_window, socks_request_t::port, circuit_t::purpose, RELAY_PAYLOAD_SIZE, connection_t::s, edge_connection_t::socks_request, connection_t::state, edge_connection_t::stream_id, STREAMWINDOW_START, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), tor_assert, tor_snprintf(), connection_t::type, and edge_connection_t::use_begindir.

Referenced by connection_ap_handshake_attach_chosen_circuit(), and connection_ap_handshake_attach_circuit().

4.35.5.197 int connection_ap_handshake_send_resolve (edge_connection_t * *ap_conn*)

Write a relay resolve cell, using destaddr and destport from ap_conn's socks_request field, and send it down circ.

If ap_conn is broken, mark it for close and return -1. Else return 0.

References origin_circuit_t::_base, edge_connection_t::_base, connection_t::address, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RESOLVE_WAIT, CIRCUIT_PURPOSE_C_GENERAL, socks_request_t::command, CONN_TYPE_AP, connection_edge_send_command(), control_event_stream_status(), get_unique_stream_id_by_circ(), LD_APP, LD_BUG, circuit_t::n_circ_id, edge_connection_t::on_circuit, circuit_t::purpose, REVERSE_LOOKUP_NAME_BUF_LEN, connection_t::s, safe_str_client(), SOCKS_COMMAND_RESOLVE, edge_connection_t::socks_request, connection_t::state, edge_connection_t::stream_id, TO_CIRCUIT, TO_ORIGIN_CIRCUIT(), tor_addr_parse_reverse_lookup_name(), tor_addr_to_reverse_lookup_name(), tor_assert, tor_free, and connection_t::type.

Referenced by connection_ap_handshake_attach_chosen_circuit().

4.35.5.198 void connection_ap_handshake_socks_reply (edge_connection_t * *conn*, char * *reply*, size_t *replylen*, int *endreason*)

Send a socks reply to stream **conn**, using the appropriate socks version, etc, and mark **conn** as completed with SOCKS handshaking.

If **reply** is defined, then write **replylen** bytes of it to conn and return, else reply based on **endreason** (one of END_STREAM_REASON_*). If **reply** is undefined, **endreason** can't be 0 or REASON_DONE. Send endreason to the controller, if appropriate.

References control_event_stream_status(), socks_request_t::has_finished, LD_BUG, SOCKS4_NETWORK_LEN, edge_connection_t::socks_request, socks_request_t::socks_version, stream_end_reason_to_socks5_response(), TO_CONN, and tor_assert.

Referenced by _connection_mark_unattached_ap(), connection_ap_handshake_process_socks(), connection_ap_handshake_socks_resolved(), and connection_edge_process_relay_cell_not_open().

4.35.5.199 void connection_ap_handshake_socks_resolved (edge_connection_t * *conn*, int *answer_type*, size_t *answer_len*, const char * *answer*, int *ttl*, time_t *expires*)

Send an answer to an AP connection that has requested a DNS lookup via SOCKS. The type should be one of RESOLVED_TYPE_(IPV4|IPV6|HOSTNAME) or -1 for unreachable; the answer should be in the format specified in the socks extensions document. **ttl** is the ttl for the answer, or -1 on certain errors or for values that didn't come via DNS. **expires** is a time when the answer expires, or -1 or TIME_MAX if there's a good TTL.

References socks_request_t::address, edge_connection_t::chosen_exit_name, client_dns_set_addressmap(), client_dns_set_reverse_addressmap(), connection_ap_handshake_socks_reply(), edge_connection_t::dns_server_request, dnsserv_resolved(), get_uint32(), socks_request_t::has_finished, edge_connection_t::is_dns_request, set_uint16(), SOCKS4_NETWORK_LEN, edge_connection_t::socks_request, socks_request_t::socks_version, tell_controller_about_resolved_result(), and tor_free.

Referenced by _connection_mark_unattached_ap(), and connection_edge_process_relay_cell_not_open().

4.35.5.200 edge_connection_t* connection_ap_make_link (char * *address*, uint16_t *port*, const char * *digest*, int *use_begindir*, int *want_onehop*)

Make an AP **connection_t** (p. 53), make a new linked connection pair, and attach one side to the conn, connection_add it, initialize it to circuit_wait, and call connection_ap_handshake_attach_circuit(conn) on it.

Return the other end of the linked connection pair, or -1 if error.

References edge_connection_t::_base, connection_t::addr, connection_t::address, socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, base16_encode(), edge_connection_t::chosen_exit_name, socks_request_t::command, CONN_TYPE_AP, connection_add(), connection_ap_handshake_attach_circuit(), connection_free(), control_event_stream_status(), DIGEST_LEN, edge_connection_new(), END_STREAM_REASON_CANT_ATTACH, socks_request_t::has_finished, HEX_DIGEST_LEN, LD_APP, connection_t::linked, connection_t::marked_for_close, connection_t::port, socks_request_t::port, safe_str_client(), SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, socks_request_t::socks_version, connection_t::state, TO_CONN, tor_addr_make_unspec(), tor_assert, edge_connection_t::use_begindir, and edge_connection_t::want_onehop.

Referenced by directory_initiate_command_rend().

4.35.5.201 int connection_ap_process_transparent (edge_connection_t * *conn*)

connection_init_accepted_conn() (p. 420) found a new trans AP conn. Get the original destination and send it to **connection_ap_handshake_rewrite_and_attach()** (p. 439).

Return -1 if an unexpected error with conn (and it should be marked for close), else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CONTROLLER_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_ap_get_original_destination(), connection_ap_handshake_rewrite_and_attach(), control_event_stream_status(), END_STREAM_REASON_CANT_FETCH_ORIG_DEST, get_options(), socks_request_t::has_finished, LD_APP, or_options_t::LeaveStreamsUnattached, SOCKS_COMMAND_CONNECT, edge_connection_t::socks_request, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_init_accepted_conn().

4.35.5.202 void connection_bucket_init (void)

Initialize the global read bucket to options->BandwidthBurst.

References or_options_t::BandwidthBurst, get_options(), or_options_t::RelayBandwidthBurst, and or_options_t::RelayBandwidthRate.

Referenced by do_main_loop().

4.35.5.203 void connection_bucket_refill (int *seconds_elapsed*, time_t *now*)

A second has rolled over; increment buckets appropriately.

References or_connection_t::bandwidthburst, or_options_t::BandwidthBurst, or_connection_t::bandwidthrate, or_options_t::BandwidthRate, connection_bucket_refill_helper(), connection_bucket_should_increase(), connection_counts_as_relayed_traffic(), connection_start_reading(), connection_start_writing(), get_connection_array(), get_options(), LD_NET, LOG_FN_CONN, OR_CONN_STATE_OPEN, or_connection_t::read_bucket, or_options_t::RelayBandwidthBurst, or_options_t::RelayBandwidthRate, TO_OR_CONN(), tor_assert, and or_connection_t::write_bucket.

Referenced by second_elapsed_callback().

4.35.5.204 ssize_t connection_bucket_write_limit (connection_t * *conn*, time_t *now*)

How many bytes at most can we write onto this connection?

References CELL_NETWORK_SIZE, CONN_TYPE_DIR, connection_bucket_round_robin(), connection_counts_as_relayed_traffic(), connection_is_rate_limited(), OR_CONN_STATE_OPEN, connection_t::outbuf_flushlen, RELAY_PAYLOAD_SIZE, connection_t::state, TO_OR_CONN(), connection_t::type, and or_connection_t::write_bucket.

Referenced by conn_close_if_marked(), and connection_handle_write_impl().

4.35.5.205 void connection_close_immediate (connection_t * *conn*)

Close the underlying socket for **conn**, so we don't try to flush it. Must be used in conjunction with (right before) connection_mark_for_close().

References assert_connection_ok(), buf_clear(), CONN_IS_CLOSED, conn_state_to_string(), conn_type_to_string(), connection_is_listener(), connection_unregister_events(), LD_BUG, LD_NET, connection_t::linked, connection_t::linked_conn_is_closed, connection_t::outbuf, connection_t::outbuf_flushlen, connection_t::s, connection_t::state, tor_close_socket(), tor_fragile_assert, and connection_t::type.

Referenced by _connection_write_to_buf_impl(), conn_write_callback(), connection_dir_reached_eof(), connection_exit_connect_dir(), connection_handle_read_impl(), connection_handle_write_impl(), and options_act_reversible().

4.35.5.206 int connection_connect (connection_t * *conn*, const char * *address*, const tor_addr_t * *addr*, uint16_t *port*, int * *socket_error*)

Take conn, make a nonblocking socket; try to connect to addr:port (they arrive in *host order*). If fail, return -1 and if applicable put your best guess about errno into ***socket_error**. Else assign s to conn->s: if connected return 1, if EAGAIN return 0.

address is used to make the logs useful.

On success, add conn to the list of polled connections.

References client_check_address_changed(), connection_add(), or_options_t::ConstrainedSockets, or_options_t::ConstrainedSockSize, escaped_safe_str_client(), get_n_open_sockets(), get_options(), LD_CONFIG, LD_NET, or_options_t::OutboundBindAddress, connection_t::s, server_mode(), set_constrained_socket_buffers(), set_socket_nonblocking(), tor_addr_family(), tor_addr_is_loopback(), tor_addr_to_sockaddr(), tor_assert, tor_close_socket(), tor_inet_aton(), tor_open_socket(), and warn_too_many_conns().

Referenced by connection_exit_connect(), connection_or_connect(), and directory_initiate_-

command_rend().

4.35.5.207 int connection_control_finished_flushing (control_connection_t * *conn*)

Called when **conn** has no more bytes left on its outbuf.

References connection_stop_writing(), TO_CONN, and tor_assert.

Referenced by connection_finished_flushing().

4.35.5.208 int connection_control_process_inbuf (control_connection_t * *conn*)

Called when data has arrived on a v1 control connection: Try to fetch commands from conn->inbuf, and execute them.

References control_connection_t::_base, connection_printf_to_buf(), connection_stop_reading(), connection_write_str_to_buf(), CONTROL_CONN_STATE_NEEDAUTH, CONTROL_CONN_STATE_OPEN, fetch_from_buf_line(), handle_control_attachstream(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_closestream(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_loadconf(), handle_control_mapaddress(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_redirectstream(), handle_control_resetconf(), handle_control_resolve(), handle_control_saveconf(), handle_control_setcircuitpurpose(), handle_control_setconf(), handle_control_setevents(), handle_control_signal(), handle_control_usefeature(), connection_t::hold_open_until_flushed, connection_t::inbuf, control_connection_t::incoming_cmd, control_connection_t::incoming_cmd_cur_len, control_connection_t::incoming_cmd_len, is_valid_initial_command(), MAX_COMMAND_LINE_LENGTH, peek_buf_has_control0_command(), set_uint16(), connection_t::state, TO_CONN, and tor_assert.

Referenced by connection_process_inbuf().

4.35.5.209 int connection_control_reached_eof (control_connection_t * *conn*)

Called when **conn** has gotten its socket closed.

References LD_CONTROL, TO_CONN, and tor_assert.

Referenced by connection_reached_eof().

4.35.5.210 int connection_cpu_finished_flushing (connection_t * *conn*)

Called when we're done sending a request to a cpuworker.

References CONN_TYPE_CPUWORKER, connection_stop_writing(), tor_assert, and connection_t::type.

Referenced by connection_finished_flushing().

4.35.5.211 int connection_cpu_process_inbuf (connection_t * *conn*)

Called when we get data from a cpuworker. If the answer is not complete, wait for a complete answer. If the answer is complete, process it as appropriate.

References buf_datalen(), circuit_get_by_circid_orconn(), CIRCUIT_IS_ORIGIN, CONN_TYPE_CPUWORKER, CONN_TYPE_OR, connection_fetch_from_buf(), connection_get_by_global_id(), CPUWORKER_STATE_BUSY_ONION, CPUWORKER_STATE_IDLE, connection_t::inbuf, last_rotation_time, LD_OR, LEN_ONION_RESPONSE, connection_t::marked_for_close, num_cpuworkers, num_cpuworkers_busy, onionskin_answer(), process_pending_task(), spawn_enough_cpuworkers(), connection_t::state, TAG_LEN, tag_unpack(), connection_t::timestamp_created, TO_OR_CIRCUIT(), TO_OR_CONN(), tor_assert, and connection_t::type.

Referenced by connection_process_inbuf().

4.35.5.212 int connection_cpu_reached_eof (connection_t ∗ *conn*)

If the cpuworker closes the connection, mark it as closed and spawn a new one as needed.

References CPUWORKER_STATE_IDLE, LD_GENERAL, num_cpuworkers, num_cpuworkers_busy, spawn_enough_cpuworkers(), and connection_t::state.

Referenced by connection_reached_eof().

4.35.5.213 int connection_dir_finished_connecting (dir_connection_t ∗ *conn*)

Connected handler for directory connections: begin sending data to the server

References dir_connection_t::_base, connection_t::address, CONN_TYPE_DIR, DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_CONNECTING, LD_HTTP, connection_t::port, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_finished_connecting().

4.35.5.214 int connection_dir_finished_flushing (dir_connection_t ∗ *conn*)

Write handler for directory connections; called when all data has been flushed. Close the connection or wait for a response as appropriate.

References dir_connection_t::_base, CONN_TYPE_DIR, connection_stop_writing(), DIR_CONN_STATE_CLIENT_READING, DIR_CONN_STATE_CLIENT_SENDING, DIR_CONN_STATE_SERVER_WRITING, DIRREQ_FLUSHING_DIR_CONN_FINISHED, geoip_change_dirreq_state(), LD_BUG, LD_DIR, LD_DIRSERV, connection_t::state, TO_CONN, tor_assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_finished_flushing().

4.35.5.215 int connection_dir_is_encrypted (dir_connection_t ∗ *conn*)

Return true iff anything we say on **conn** is being encrypted before we send it to the client/server.

References TO_CONN.

Referenced by connection_dirserv_add_servers_to_outbuf(), and directory_handle_command_get().

4.35.5.216 int connection_dir_process_inbuf (dir_connection_t ∗ *conn*)

Read handler for directory connections. (That's connections *to* directory servers and connections *at* directory servers.)

References dir_connection_t::_base, buf_datalen(), CONN_TYPE_DIR, DIR_CONN_STATE_SERVER_COMMAND_WAIT, directory_handle_command(), connection_t::inbuf, connection_t::inbuf_reached_eof, LD_HTTP, MAX_DIRECTORY_OBJECT_SIZE, connection_t::state, TO_CONN, tor_assert, and connection_t::type.

Referenced by connection_process_inbuf().

4.35.5.217 int connection_dir_reached_eof (dir_connection_t ∗ *conn*)

Called when a directory connection reaches EOF.

References dir_connection_t::_base, connection_close_immediate(), connection_dir_client_reached_eof(), DIR_CONN_STATE_CLIENT_FINISHED, DIR_CONN_STATE_CLIENT_READING, LD_HTTP, connection_t::state, and TO_CONN.

Referenced by connection_reached_eof(), and run_connection_housekeeping().

4.35.5.218 void connection_dir_request_failed (dir_connection_t ∗ *conn*)

Called when we are unable to complete the client's request to a directory server due to a network error: Mark the router as down and try again if possible.

References dir_connection_t::_base, connection_t::address, connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), connection_dir_download_routerdesc_failed(), DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, directory_conn_is_self_reachability_test(), entry_list_is_constrained(), get_options(), dir_connection_t::identity_digest, LD_DIR, networkstatus_consensus_download_failed(), connection_t::purpose, and router_set_status().

Referenced by connection_about_to_close_connection(), and directory_initiate_command_rend().

4.35.5.219 int connection_dirserv_flushed_some (dir_connection_t ∗ *conn*)

Called whenever we have flushed some directory data in state SERVER_WRITING.

References dir_connection_t::_base, buf_datalen(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), DIR_CONN_STATE_SERVER_WRITING, dir_connection_t::dir_spool_src, DIRSERV_BUFFER_MIN, connection_t::outbuf, connection_t::state, and tor_assert.

Referenced by connection_flushed_some(), and directory_handle_command_get().

4.35.5.220 void connection_dns_remove (edge_connection_t ∗ *conn*)

Remove **conn** from the list of connections waiting for conn->address.

References edge_connection_t::_base, connection_t::address, assert_connection_ok(), pending_connection_t::conn, CONN_TYPE_EXIT, escaped_safe_str(), EXIT_CONN_STATE_RESOLVING, LD_BUG, pending_connection_t::next, cached_resolve_t::pending_connections, connection_t::s, connection_t::state, TO_CONN, tor_assert, tor_free, and connection_t::type.

Referenced by connection_about_to_close_connection().

4.35.5.221 void connection_dump_buffer_mem_stats (int *severity*)

Log how many bytes are used by buffers of different kinds and sizes.

References buf_allocation(), buf_datalen(), and get_connection_array().

Referenced by dumpmemusage().

4.35.5.222 void connection_edge_consider_sending_sendme (edge_connection_t ∗ *conn*)

Called when we've just received a relay data cell, or when we've just finished flushing all bytes to stream **conn**.

If conn->outbuf is not too full, and our deliver window is low, send back a suitable number of stream-level sendme cells.

References edge_connection_t::_base, circuit_get_by_edge_conn(), connection_edge_send_command(), connection_outbuf_too_full(), edge_connection_t::cpath_layer, edge_connection_t::deliver_window, LD_APP, connection_t::outbuf_flushlen, STREAMWINDOW_INCREMENT, STREAMWINDOW_START, and TO_CONN.

Referenced by connection_edge_finished_flushing(), and connection_edge_process_relay_cell().

4.35.5.223 int connection_edge_destroy (circid_t *circ_id*, edge_connection_t ∗ *conn*)

This edge needs to be closed, because its circuit has closed. Mark it for close and return 0.

References edge_connection_t::_base, CONN_TYPE_AP, control_event_stream_bandwidth(), control_event_stream_status(), edge_connection_t::cpath_layer, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED, connection_t::hold_open_until_flushed, LD_EDGE, connection_t::marked_for_close, edge_connection_t::on_circuit, TO_CONN, and connection_t::type.

Referenced by _circuit_mark_for_close().

4.35.5.224 int connection_edge_end (edge_connection_t ∗ *conn*, uint8_t *reason*)

Send a relay end cell from stream **conn** down conn's circuit, and remember that we've done so. If this is not a client connection, set the relay end cell's reason for closing as **reason**.

Return -1 if this function has already been called on this conn, else return 0.

References edge_connection_t::_base, connection_t::addr, edge_connection_t::address_ttl, circuit_get_by_edge_conn(), CIRCUIT_PURPOSE_IS_CLIENT, connection_edge_is_-

rendezvous_stream(), connection_edge_send_command(), dns_clip_ttl(), edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, LD_BUG, LD_EDGE, circuit_t::marked_for_close, connection_t::marked_for_close, connection_t::marked_for_close_file, circuit_t::purpose, RELAY_PAYLOAD_SIZE, connection_t::s, set_uint32(), tor_addr_family(), tor_addr_to_ipv4n(), and tor_fragile_assert.

Referenced by connection_ap_expire_beginning(), connection_edge_end_errno(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_reached_eof(), connection_exit_connect(), connection_exit_connect_dir(), dns_cancel_pending_resolve(), dns_found_answer(), handle_control_attachstream(), hibernate_go_dormant(), and purge_expired_resolves().

4.35.5.225 int connection_edge_end_errno (edge_connection_t * *conn*)

An error has just occurred on an operation on an edge connection **conn**. Extract the errno; convert it to an end reason, and send an appropriate relay end cell to the other end of the connection's circuit.

References edge_connection_t::_base, connection_edge_end(), errno_to_stream_end_reason(), connection_t::s, and tor_assert.

Referenced by conn_read_callback(), connection_exit_connect(), connection_handle_read_impl(), and connection_handle_write_impl().

4.35.5.226 int connection_edge_finished_connecting (edge_connection_t * *edge_conn*)

Connected handler for exit connections: start writing pending data, deliver 'CONNECTED' relay cells as appropriate, and check any pending data that may have been received.

References edge_connection_t::_base, connection_t::addr, connection_t::address, edge_connection_t::address_ttl, CONN_TYPE_EXIT, connection_edge_is_rendezvous_stream(), connection_edge_process_inbuf(), connection_edge_send_command(), connection_start_writing(), connection_wants_to_flush(), connection_watch_events(), dns_clip_ttl(), escaped_safe_str(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, fmt_addr(), edge_connection_t::package_window, connection_t::port, rep_hist_note_exit_stream_opened(), safe_str(), set_uint32(), connection_t::state, TO_CONN, tor_addr_family(), tor_addr_to_ipv4n(), tor_assert, and connection_t::type.

Referenced by connection_finished_connecting().

4.35.5.227 int connection_edge_finished_flushing (edge_connection_t * *conn*)

Connection **conn** has finished writing and has no bytes left on its outbuf.

If it's in state 'open', stop writing, consider responding with a sendme, and return. Otherwise, stop writing and return.

If **conn** is broken, mark it for close and return -1, else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_SOCKS_WAIT, connection_edge_consider_sending_sendme(), connection_stop_writing(), EXIT_CONN_STATE_OPEN, LD_BUG, connection_t::state, TO_CONN, tor_assert, and tor_fragile_assert.

Referenced by connection_finished_flushing().

4.35.5.228 int connection_edge_is_rendezvous_stream (edge_connection_t ∗ *conn*)

Return 1 if **conn** is a rendezvous stream, or 0 if it is a general stream.

References edge_connection_t::rend_data, and tor_assert.

Referenced by ap_stream_wants_exit_attention(), circuit_get_open_circ_or_launch(), connection_ap_handshake_attach_circuit(), connection_ap_process_end_not_open(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_begin_conn(), connection_exit_connect(), relay_lookup_conn(), and write_stream_target_to_buf().

4.35.5.229 int connection_edge_package_raw_inbuf (edge_connection_t ∗ *conn*, int *package_partial*)

While conn->inbuf has an entire relay payload of bytes on it, and the appropriate package windows aren't empty, grab a cell and send it down the circuit.

Return -1 (and send a RELAY_COMMAND_END cell if necessary) if conn should be marked for close, else return 0.

References edge_connection_t::_base, buf_datalen(), CELL_PAYLOAD_SIZE, circuit_consider_stop_edge_reading(), circuit_get_by_edge_conn(), connection_edge_send_command(), connection_fetch_from_buf(), connection_stop_reading(), edge_connection_t::cpath_layer, edge_connection_t::end_reason, END_STREAM_REASON_CANT_ATTACH, connection_t::inbuf, LD_APP, LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, crypt_path_t::package_window, circuit_t::package_window, edge_connection_t::package_window, RELAY_PAYLOAD_SIZE, connection_t::s, TO_CONN, and tor_assert.

Referenced by circuit_resume_edge_reading_helper(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), and connection_edge_process_relay_cell_not_open().

4.35.5.230 int connection_edge_process_inbuf (edge_connection_t ∗ *conn*, int *package_partial*)

Handle new bytes on conn->inbuf based on state:

- If it's waiting for socks info, try to read another step of the socks handshake out of conn->inbuf.
- If it's waiting for the original destination, fetch it.
- If it's open, then package more relay cells from the stream.
- Else, leave the bytes on inbuf alone for now.

Mark and return -1 if there was an unexpected error with the conn, else return 0.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_CONTROLLER_WAIT, AP_CONN_STATE_NATD_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RENDDESC_WAIT, AP_CONN_STATE_RESOLVE_WAIT, AP_CONN_STATE_SOCKS_WAIT, conn_state_to_string(), connection_ap_handshake_process_socks(), connection_ap_process_natd(), connection_edge_end(), connection_edge_package_raw_inbuf(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, LD_BUG, LD_EDGE, connection_t::state, TO_CONN, tor_assert, tor_fragile_assert, and connection_t::type.

Referenced by connection_edge_finished_connecting(), and connection_process_inbuf().

4.35.5.231 int connection_edge_reached_eof (edge_connection_t * *conn*)

There was an EOF. Send an end and mark the connection for close.

References edge_connection_t::_base, buf_datalen(), connection_edge_end(), connection_state_is_open(), socks_request_t::has_finished, connection_t::inbuf, LD_EDGE, connection_t::marked_for_close, connection_t::s, edge_connection_t::socks_request, and TO_CONN.

Referenced by connection_reached_eof().

4.35.5.232 int connection_edge_send_command (edge_connection_t * *fromconn*, uint8_t *relay_command*, const char * *payload*, size_t *payload_len*)

Make a relay cell out of **relay_command** and **payload**, and send it onto the open circuit **circ**. **fromconn** is the stream that's sending the relay cell, or NULL if it's a control cell. **cpath_layer** is NULL for OR->OP cells, or the destination hop for OP->OR cells.

If you can't send the cell, mark the circuit for close and return -1. Else return 0.

References edge_connection_t::_base, CONN_TYPE_AP, edge_connection_t::cpath_layer, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, LD_APP, LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, edge_connection_t::on_circuit, relay_send_command_from_edge(), edge_connection_t::stream_id, TO_CONN, tor_assert, and connection_t::type.

Referenced by connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_consider_sending_sendme(), connection_edge_end(), connection_edge_finished_connecting(), connection_edge_package_raw_inbuf(), connection_exit_connect(), connection_exit_connect_dir(), send_resolved_cell(), and send_resolved_hostname_cell().

4.35.5.233 int connection_exit_begin_conn (cell_t * *cell*, circuit_t * *circ*)

A relay 'begin' or 'begin_dir' cell has arrived, and either we are an exit hop for the circuit, or we are the origin and it is a rendezvous begin.

Launch a new exit connection and initialize things appropriately.

If it's a rendezvous stream, call **connection_exit_connect()** (p. 445) on it.

For general streams, call **dns_resolve()** (p. 635) on it first, and only call **connection_exit_connect()** (p. 445) if the dns answer is already known.

Note that we don't call **connection_add()** (p. 694) on the new stream! We wait for **connection_exit_connect()** (p. 445) to do that.

Return -(some circuit end reason) if we want to tear down **circ**. Else return 0.

References edge_connection_t::_base, or_connection_t::_base, connection_t::addr, connection_t::address, assert_circuit_ok(), CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_S_REND_JOINED, relay_header_t::command, CONN_TYPE_EXIT, connection_edge_is_rendezvous_stream(), connection_exit_connect(), connection_exit_connect_dir(), connection_free(), connection_or_digest_is_known_relay(), origin_circuit_t::cpath, edge_connection_t::cpath_layer, edge_connection_t::deliver_window, directory_permits_begindir_requests(), circuit_t::dirreq_id, dns_resolve(), edge_connection_new(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_RESOLVEFAILED, EXIT_PURPOSE_CONNECT, get_options(), or_connection_t::identity_digest, or_circuit_t::is_first_hop, LD_BUG, LD_PROTOCOL, LD_REND, relay_header_t::length, edge_connection_t::next_stream, edge_connection_t::on_circuit, or_circuit_t::p_conn, origin_circuit_t::p_streams, edge_connection_t::package_window, parse_addr_port(), cell_t::payload, connection_t::port, crypt_path_t::prev, connection_t::purpose, circuit_t::purpose, or_connection_t::real_addr, RELAY_HEADER_SIZE, relay_header_unpack(), relay_send_end_cell_from_edge(), origin_circuit_t::rend_data, edge_connection_t::rend_data, rend_data_dup(), rend_service_set_connection_addr_port(), safe_str(), server_mode(), connection_t::state, edge_connection_t::stream_id, relay_header_t::stream_id, STREAMWINDOW_START, TO_CONN, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_addr_copy(), tor_addr_is_null(), tor_assert, tor_dup_addr(), tor_free, tor_strlower(), and we_are_hibernating().

Referenced by connection_edge_process_relay_cell().

4.35.5.234 int connection_exit_begin_resolve (cell_t * *cell*, or_circuit_t * *circ*)

Called when we receive a RELAY_COMMAND_RESOLVE cell 'cell' along the circuit **circ**; begin resolving the hostname, and (eventually) reply with a RESOLVED cell.

References edge_connection_t::_base, connection_t::address, assert_circuit_ok(), CONN_TYPE_EXIT, connection_free(), dns_resolve(), edge_connection_new(), EXIT_CONN_STATE_RESOLVEFAILED, EXIT_PURPOSE_RESOLVE, relay_header_t::length, connection_t::marked_for_close, edge_connection_t::on_circuit, cell_t::payload, connection_t::port, connection_t::purpose, RELAY_HEADER_SIZE, relay_header_unpack(), connection_t::state, relay_header_t::stream_id, edge_connection_t::stream_id, TO_CIRCUIT, and TO_CONN.

Referenced by connection_edge_process_relay_cell().

4.35.5.235 void connection_exit_connect (edge_connection_t * *edge_ conn*)

Connect to conn's specified addr and port. If it worked, conn has now been added to the connection_array.

Send back a connected cell. Include the resolved IP of the destination address, but *only* if it's a general exit stream. (Rendezvous streams must not reveal what IP they connected to.)

References connection_t::addr, connection_t::address, edge_connection_t::address_ttl, circuit_detach_stream(), circuit_get_by_edge_conn(), connection_connect(), connection_edge_end(), connection_edge_end_errno(), connection_edge_is_rendezvous_stream(), connection_edge_send_command(), connection_free(), connection_wants_to_flush(), connection_watch_events(), dns_clip_ttl(), escaped_safe_str_client(), EXIT_CONN_STATE_CONNECTING, EXIT_CONN_STATE_OPEN, LD_BUG, connection_t::port, router_compare_to_my_exit_policy(), set_uint32(), connection_t::state, TO_CONN, tor_addr_family(), and tor_addr_to_ipv4n().

Referenced by connection_exit_begin_conn(), and dns_found_answer().

4.35.5.236 void connection_expire_held_open (void)

Find each connection that has hold_open_until_flushed set to 1 but hasn't written in the past 15 seconds, and set hold_open_until_flushed to 0. This means it will get cleaned up in the next loop through close_if_marked() in **main.c** (p. 690).

References conn_state_to_string(), CONN_TYPE_DIR, CONN_TYPE_EXIT, conn_type_to_string(), DIR_PURPOSE_SERVER, get_connection_array(), LD_NET, and tor_assert.

Referenced by run_scheduled_events().

4.35.5.237 int connection_fetch_from_buf (char * *string*, size_t *len*, connection_t * *conn*)

A pass-through to fetch_from_buf.

References fetch_from_buf(), and connection_t::inbuf.

Referenced by connection_cpu_process_inbuf(), connection_edge_package_raw_inbuf(), and connection_or_process_cells_from_inbuf().

4.35.5.238 void connection_free (connection_t * *conn*)

Make sure **conn** isn't in any of the global conn lists; then free it.

References _connection_free(), CONN_TYPE_CONTROL, connection_in_array(), connection_is_on_closeable_list(), connection_or_remove_from_identity_map(), connection_start_reading(), connection_unregister_events(), control_update_global_event_mask(), control_connection_t::event_mask, LD_BUG, connection_t::linked_conn, connection_t::marked_for_close, connection_t::reading_from_linked_conn, TO_CONTROL_CONN(), TO_OR_CONN(), tor_assert, tor_digest_is_zero(), tor_fragile_assert, and connection_t::type.

Referenced by circuit_free_all(), connection_ap_make_link(), connection_create_listener(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_exit_connect(), connection_exit_connect_dir(), connection_handle_listener_read(), connection_or_connect(), connection_unlink(), directory_initiate_command_rend(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve(), dnsserv_launch_request(), evdns_server_callback(), purge_expired_resolves(), and spawn_cpuworker().

4.35.5.239 void connection_free_all (void)

Call **_connection_free()** (p. 408) on every connection in our array, and release all storage held by **connection.c** (p. 406). This is used by cpuworkers and dnsworkers when they fork, so they don't keep resources held open (especially sockets).

Don't do the checks in **connection_free()** (p. 417), because they will fail.

References _connection_free(), CONN_TYPE_CONTROL, connection_or_clear_identity_map(), control_update_global_event_mask(), get_connection_array(), smartlist_free(), TO_CONTROL_CONN(), and tor_free.

Referenced by tor_free_all().

4.35.5.240 connection_t* connection_get_by_global_id (uint64_t *id*)

Return the stream with id **id** if it is not already marked for close.

References get_connection_array().

Referenced by connection_cpu_process_inbuf(), and get_stream().

4.35.5.241 connection_t* connection_get_by_type (int *type*)

Return a connection of type **type** that is not marked for close.

References get_connection_array().

Referenced by circuit_get_open_circ_or_launch(), hibernate_begin(), and hibernate_go_dormant().

4.35.5.242 connection_t* connection_get_by_type_addr_port_purpose (int *type*, const tor_addr_t * *addr*, uint16_t *port*, int *purpose*)

Return a connection with given type, address, port, and purpose; or NULL if no such connection exists.

References get_connection_array(), and tor_addr_eq.

Referenced by consider_testing_reachability(), launch_direct_bridge_descriptor_fetch(), router_pick_trusteddirserver_impl(), and update_v2_networkstatus_cache_downloads().

4.35.5.243 connection_t* connection_get_by_type_purpose (int *type*, int *purpose*)

Return an open, non-marked connection of a given type and purpose, or NULL if no such connection exists.

References get_connection_array().

Referenced by update_consensus_networkstatus_downloads(), and update_v2_networkstatus_cache_downloads().

4.35.5.244 connection_t* connection_get_by_type_state (int *type*, int *state*)

Return a connection of type **type** that is in state **state**, and that is not marked for close.

References get_connection_array().

Referenced by assign_onionskin_to_cpuworker(), cpuworkers_rotate(), and directory_all_unreachable().

4.35.5.245 connection_t* connection_get_by_type_state_rendquery (int *type*, int *state*, const char * *rendquery*)

Return a connection of type **type** that has rendquery equal to **rendquery**, and that is not marked for close. If state is non-zero, conn must be of that state too.

References CONN_IS_EDGE, CONN_TYPE_AP, CONN_TYPE_DIR, CONN_TYPE_EXIT, get_connection_array(), rend_cmp_service_ids(), TO_DIR_CONN(), TO_EDGE_CONN(), and tor_assert.

Referenced by rend_client_remove_intro_point().

4.35.5.246 int connection_in_array (connection_t * *conn*)

Return true iff conn is in the current poll array.

References smartlist_isin().

Referenced by assert_all_pending_dns_resolves_ok(), and connection_free().

4.35.5.247 int connection_is_listener (connection_t * *conn*)

Return 1 if **conn** is a listener conn, else return 0.

References CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR_LISTENER, CONN_TYPE_OR_LISTENER, and connection_t::type.

Referenced by _connection_free(), assert_connection_ok(), connection_close_immediate(), connection_handle_write_impl(), connection_init(), and dumpstats().

4.35.5.248 int connection_is_on_closeable_list (connection_t * *conn*)

Return 1 if conn is on the closeable list, else return 0.

References smartlist_isin().

Referenced by connection_free().

4.35.5.249 int connection_is_reading (connection_t * *conn*)

Return true iff **conn** is listening for read events.

References connection_t::read_event, connection_t::reading_from_linked_conn, and tor_assert.

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), and set_streams_blocked_on_circ().

4.35.5.250 int connection_is_writing (connection_t * *conn*)

Return true iff **conn** is listening for write events.

References tor_assert, connection_t::write_event, and connection_t::writing_to_linked_conn.

Referenced by assert_connection_ok(), and connection_read_to_buf().

4.35.5.251 void connection_link_connections (connection_t * *conn_a*, connection_t * *conn_b*)

Create a link between **conn_a** and **conn_b**.

References connection_t::linked, connection_t::linked_conn, connection_t::s, and tor_assert.

Referenced by connection_exit_connect_dir(), and directory_initiate_command_rend().

4.35.5.252 connection_t∗ connection_new (int *type*, int *socket_family*)

Allocate, initialize, and return a new **connection_t** (p. 53) subtype of **type** to make or receive connections of address family **socket_family**. The type should be one of the CONN_TYPE_∗ constants.

References CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, connection_init(), control_connection_new(), dir_connection_new(), edge_connection_new(), or_connection_new(), and TO_CONN.

Referenced by connection_create_listener(), connection_handle_listener_read(), and spawn_cpuworker().

4.35.5.253 void connection_or_clear_identity_map (void)

Remove all entries from the identity-to-orconn map, and clear all identities in OR conns.

References CONN_TYPE_OR, DIGEST_LEN, get_connection_array(), and TO_OR_CONN().

Referenced by connection_free_all().

4.35.5.254 or_connection_t∗ connection_or_connect (const tor_addr_t ∗ _*addr*, uint16_t *port*, const char ∗ *id_digest*)

Launch a new OR connection to **addr**:**port** and expect to handshake with an OR with identity digest **id_digest**.

If **id_digest** is me, do nothing. If we're already connected to it, return that connection. If the connect() is in progress, set the new conn's state to 'connecting' and return it. If connect() succeeds, call **connection_tls_start_handshake()** (p. 459) on it.

This function is called from router_retry_connections(), for ORs connecting to ORs, and **circuit_establish_circuit()** (p. 312), for OPs connecting to ORs.

Return the launched conn, or NULL if it failed.

References or_connection_t::_base, connection_t::address, connection_connect(), connection_free(), connection_or_connect_failed(), connection_or_finished_connecting(), connection_or_init_conn_from_address(), connection_watch_events(), control_event_or_conn_status(), entry_guard_register_connect_status(), errno_to_orconn_end_reason(), get_options(), or_options_t::HttpsProxy, or_options_t::HttpsProxyAddr, or_options_t::HttpsProxyPort, or_connection_t::identity_digest, LD_PROTOCOL, OR_CONN_STATE_CONNECTING, or_connection_new(), router_digest_is_me(), server_mode(), or_options_t::Socks4Proxy, or_options_t::Socks4ProxyAddr, or_options_t::Socks4ProxyPort, or_options_t::Socks5Proxy, or_options_t::Socks5ProxyAddr, or_options_t::Socks5ProxyPort, connection_t::state, TO_CONN, tor_addr_copy(), and tor_assert.

Referenced by circuit_extend(), circuit_handle_first_hop(), and dirserv_single_reachability_test().

4.35.5.255 void connection_or_connect_failed (or_connection_t * *conn*, int *reason*, const char * *msg*)

conn is in the 'connecting' state, and it failed to complete a TCP connection. Send notifications appropriately.

reason specifies the or_conn_end_reason for the failure; **msg** specifies the strerror-style error message.

References authdir_mode_tests_reachability(), control_event_bootstrap_problem(), control_event_or_conn_status(), and get_options().

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), connection_or_connect(), and run_connection_housekeeping().

4.35.5.256 int connection_or_digest_is_known_relay (const char * *id_digest*)

Return 1 if identity digest **id_digest** is known to be a currently or recently running relay. Otherwise return 0.

References router_get_by_digest(), and router_get_consensus_status_by_id().

Referenced by connection_exit_begin_conn(), and connection_or_init_conn_from_address().

4.35.5.257 int connection_or_finished_connecting (or_connection_t * *or_conn*)

Connected handler for OR connections: begin the TLS handshake.

References connection_t::address, connection_proxy_connect(), connection_start_reading(), connection_tls_start_handshake(), control_event_bootstrap(), get_options(), LD_HANDSHAKE, OR_CONN_STATE_CONNECTING, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::port, connection_t::state, TO_CONN, and tor_assert.

Referenced by connection_finished_connecting(), and connection_or_connect().

4.35.5.258 int connection_or_finished_flushing (or_connection_t * *conn*)

Connection **conn** has finished writing and has no bytes left on its outbuf.

Otherwise it's in state "open": stop writing and return.

If **conn** is broken, mark it for close and return -1, else return 0.

References or_connection_t::_base, assert_connection_ok(), connection_stop_writing(), LD_BUG, OR_CONN_STATE_OPEN, OR_CONN_STATE_OR_HANDSHAKING, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::state, TO_CONN, tor_assert, and tor_fragile_assert.

Referenced by connection_finished_flushing().

4.35.5.259 int connection_or_flush_from_first_active_circuit (or_connection_t * *conn*, int *max*, time_t *now*)

Pull as many cells as possible (but no more than **max**) from the queue of the first active circuit on **conn**, and write them to **conn**->outbuf. Return the number of cells written. Advance the active circuit pointer to the next active circuit in the ring.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuit_pqueue_last_recalibrated, or_connection_t::active_circuits, add_cell_ewma_to_conn(), packed_cell_t::body, cell_ewma_tick_from_timeval(), cell_ewma_to_circuit(), CELL_NETWORK_SIZE, CELL_QUEUE_LOWWATER_SIZE, cell_queue_pop(), CIRCUIT_IS_ORIGIN, insertion_time_elem_t::counter, DIRREQ_CIRC_QUEUE_FLUSHED, ewma_scale_factor, insertion_time_queue_t::first, geoip_change_dirreq_state(), get_options(), cell_queue_t::head, insertion_time_elem_t::insertion_time, cell_queue_t::insertion_times, insertion_time_queue_t::last, LD_BUG, LD_GENERAL, make_circuit_inactive_on_conn(), cell_queue_t::n, circuit_t::n_conn, circuit_t::n_conn_cells, insertion_time_elem_t::next, next_circ_on_conn_p(), or_circuit_t::p_conn_cells, packed_cell_free_unchecked(), pop_first_cell_ewma_from_conn(), or_circuit_t::processed_cells, scale_active_circuits(), set_streams_blocked_on_circ(), circuit_t::streams_blocked_on_n_conn, circuit_t::streams_blocked_on_p_conn, or_connection_t::timestamp_last_added_nonpadding, TO_CONN, TO_OR_CIRCUIT(), tor_assert, and or_circuit_t::total_cell_waiting_time.

Referenced by append_cell_to_circuit_queue(), and connection_or_flushed_some().

4.35.5.260 int connection_or_flushed_some (or_connection_t * *conn*)

Called whenever we have flushed some data on an or_conn: add more data from active circuits.

References or_connection_t::_base, or_connection_t::active_circuits, approx_time(), buf_datalen(), CELL_NETWORK_SIZE, connection_or_flush_from_first_active_circuit(), OR_CONN_HIGHWATER, OR_CONN_LOWWATER, and connection_t::outbuf.

Referenced by connection_flushed_some().

4.35.5.261 or_connection_t* connection_or_get_for_extend (const char * *digest*, const tor_addr_t * *target_addr*, const char ** *msg_out*, int * *launch_out*)

Return the OR connection we should use to extend a circuit to the router whose identity is **digest**, and whose address we believe (or have been told in an extend cell) is **target_addr**. If there is no good connection, set ***msg_out** to a message describing the connection's state and our next action, and set **launch_out** to a boolean for whether we should launch a new connection or not.

References or_connection_t::_base, approx_time(), CONN_TYPE_OR, connection_or_is_better(), DIGEST_LEN, digestmap_get(), or_connection_t::identity_digest, or_connection_t::is_bad_for_new_circs, or_connection_t::is_canonical, or_connection_t::link_proto, connection_t::magic, connection_t::marked_for_close, or_connection_t::next_with_same_id, OR_CONN_STATE_OPEN, orconn_identity_map, or_connection_t::real_addr, connection_t::state, tor_addr_compare(), tor_assert, and connection_t::type.

Referenced by circuit_extend(), and circuit_handle_first_hop().

4.35.5.262 int connection_or_nonopen_was_started_here (or_connection_t * *conn*)

Return 1 if we initiated this connection, or 0 if it started out as an incoming connection.

References or_connection_t::_base, CONN_TYPE_OR, or_connection_t::handshake_state, or_handshake_state_t::started_here, or_connection_t::tls, tor_assert, tor_tls_is_server(), and connection_t::type.

Referenced by connection_about_to_close_connection(), connection_or_set_state_open(), connection_tls_finish_handshake(), and onionskin_answer().

4.35.5.263 int connection_or_process_inbuf (or_connection_t * *conn*)

Handle any new bytes that have come in on connection **conn**. If conn is in 'open' state, hand it to **connection_or_process_cells_from_inbuf()** (p. 455) (else do nothing).

References or_connection_t::_base, connection_or_process_cells_from_inbuf(), connection_read_proxy_handshake(), connection_tls_start_handshake(), OR_CONN_STATE_OPEN, OR_CONN_STATE_OR_HANDSHAKING, OR_CONN_STATE_PROXY_HANDSHAKING, connection_t::state, TO_CONN, and tor_assert.

Referenced by connection_process_inbuf().

4.35.5.264 int connection_or_reached_eof (or_connection_t * *conn*)

We've received an EOF from **conn**. Mark it for close and return.

References LD_OR, and TO_CONN.

Referenced by connection_reached_eof().

4.35.5.265 void connection_or_remove_from_identity_map (or_connection_t * *conn*)

If conn is listed in orconn_identity_map, remove it, and clear conn->identity_digest. Otherwise do nothing.

References DIGEST_LEN, digestmap_get(), digestmap_remove(), digestmap_set(), or_connection_t::identity_digest, LD_BUG, or_connection_t::next_with_same_id, or_connection_t::nickname, orconn_identity_map, tor_assert, and tor_digest_is_zero().

Referenced by _connection_free(), connection_free(), connection_or_set_identity_digest(), and connection_unlink().

4.35.5.266 int connection_or_send_destroy (circid_t *circ_id*, or_connection_t * *conn*, int *reason*)

Write a destroy cell with circ ID **circ_id** and reason **reason** onto OR connection **conn**. Don't perform range-checking on reason: we may want to propagate reasons from other cells.

Return 0.

References cell_t::circ_id, cell_t::command, connection_or_write_cell_to_buf(), LD_OR, cell_t::payload, and tor_assert.

Referenced by _circuit_mark_for_close(), command_process_create_cell(), and connection_edge_process_relay_cell().

4.35.5.267 int connection_or_send_netinfo (or_connection_t * *conn*)

Send a NETINFO cell on **conn**, telling the other server what we know about their address, our address, and the current time.

References or_connection_t::_base, routerinfo_t::addr, connection_t::addr, append_address_to_payload(), cell_t::command, connection_or_write_cell_to_buf(), cell_t::payload, router_get_my_routerinfo(), set_uint32(), and tor_addr_from_ipv4h.

Referenced by command_process_versions_cell().

4.35.5.268 void connection_or_set_bad_connections (void)

Go through all the OR connections, and set the is_bad_for_new_circs flag on:

- all connections that are too old.
- all open non-canonical connections for which a canonical connection exists to the same router.
- all open canonical connections for which a 'better' canonical connection exists to the same router.
- all open non-canonical connections for which a 'better' non-canonical connection exists to the same router at the same address.

See **connection_or_is_better()** (p. 455) for our idea of what makes one OR connection better than another.

References connection_or_group_set_badness(), and orconn_identity_map.

Referenced by run_scheduled_events().

4.35.5.269 int connection_or_set_state_open (or_connection_t * *conn*)

Set **conn**'s state to OR_CONN_STATE_OPEN, and tell other subsystems as appropriate. Called when we are done with all TLS and OR handshaking.

References or_connection_t::_base, circ_times, circuit_build_times_network_is_live(), circuit_n_conn_done(), connection_or_nonopen_was_started_here(), connection_start_reading(), control_event_or_conn_status(), entry_guard_register_connect_status(), GEOIP_CLIENT_CONNECT, geoip_note_client_seen(), or_connection_t::handshake_state, or_connection_t::identity_digest, LD_OR, OR_CONN_STATE_OPEN, or_handshake_state_free(), rep_hist_note_connect_succeeded(), router_get_by_digest(), router_set_status(), connection_t::state, TO_CONN, tor_addr_family(), and tor_addr_to_ipv4h().

Referenced by command_process_netinfo_cell(), and connection_tls_finish_handshake().

4.35.5.270 void connection_or_unlink_all_active_circs (or_connection_t * *orconn*)

Remove all circuits from the list of circuits with pending cells on **conn**.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuits, next_circ_on_conn_p(), prev_circ_on_conn_p(), and smartlist_clear().

Referenced by circuit_unlink_all_from_or_conn().

4.35.5.271 void connection_or_write_cell_to_buf (const cell_t * *cell*, or_connection_t * *conn*)

Pack **cell** into wire-format, and write it onto **conn**'s outbuf. For cells that use or affect a circuit, this should only be called by **connection_or_flush_from_first_active_circuit()** (p. 877).

References approx_time(), packed_cell_t::body, CELL_NETWORK_SIZE, cell_pack(), cell_t::command, or_connection_t::timestamp_last_added_nonpadding, TO_CONN, and tor_assert.

Referenced by connection_or_send_destroy(), connection_or_send_netinfo(), and run_connection_housekeeping().

4.35.5.272 void connection_or_write_var_cell_to_buf (const var_cell_t ∗ *cell*, or_connection_t ∗ *conn*)

Pack a variable-length **cell** into wire-format, and write it onto **conn**'s outbuf. Right now, this *DOES NOT* support cells that affect a circuit.

References approx_time(), var_cell_t::command, var_cell_t::payload, var_cell_t::payload_len, or_connection_t::timestamp_last_added_nonpadding, TO_CONN, tor_assert, VAR_CELL_HEADER_SIZE, and var_cell_pack_header().

Referenced by connection_or_send_versions().

4.35.5.273 int connection_outbuf_too_full (connection_t ∗ *conn*)

Are there too many bytes on edge connection **conn**'s outbuf to send back a relay-level sendme yet? Return 1 if so, 0 if not. Used by **connection_edge_consider_sending_sendme()** (p. 868).

References CELL_PAYLOAD_SIZE, and connection_t::outbuf_flushlen.

Referenced by connection_edge_consider_sending_sendme().

4.35.5.274 int connection_proxy_connect (connection_t ∗ *conn*, int *type*)

Write a proxy request of **type** (socks4, socks5, https) to conn for conn->addr:conn->port, authenticating with the auth details given in the configuration (if available). SOCKS 5 and HTTP CONNECT proxies support authentication.

Returns -1 if conn->addr is incompatible with the proxy protocol, and 0 otherwise.

Use **connection_read_proxy_handshake()** (p. 423) to complete the handshake.

References connection_t::addr, alloc_http_authenticator(), connection_proxy_state_to_string(), fmt_addr(), get_options(), or_options_t::HttpsProxyAuthenticator, LD_BUG, LD_NET, LD_OR, connection_t::port, connection_t::proxy_state, or_options_t::Socks5ProxyUsername, SOCKS_COMMAND_CONNECT, tor_addr_family(), tor_addr_to_ipv4n(), tor_assert, tor_fragile_assert, tor_free, and tor_snprintf().

Referenced by connection_or_finished_connecting().

4.35.5.275 int connection_read_proxy_handshake (connection_t ∗ *conn*)

Call this from connection_∗_process_inbuf() to advance the proxy handshake.

No matter what proxy protocol is used, if this function returns 1, the handshake is complete, and the data remaining on inbuf may contain the start of the communication with the requested server.

Returns 0 if the current buffer contains an incomplete response, and -1 on error.

References connection_t::address, connection_proxy_state_to_string(), connection_read_https_proxy_response(), connection_send_socks5_connect(), escaped(), fetch_from_buf_socks_client(), get_options(), connection_t::inbuf, LD_BUG, LD_NET, connection_t::port, connection_t::proxy_state, or_options_t::Socks5ProxyPassword, or_options_t::Socks5ProxyUsername, tor_assert, tor_fragile_assert, and tor_free.

Referenced by connection_or_process_inbuf().

4.35.5.276 int connection_remove (connection_t ∗ *conn*)

Remove the connection from the global list, and remove the corresponding poll entry. Calling this function will shift the last connection (if any) into the position occupied by conn.

References connection_t::conn_array_index, conn_type_to_string(), connection_unregister_events(), LD_NET, connection_t::s, smartlist_del(), tor_assert, and connection_t::type.

Referenced by connection_unlink().

4.35.5.277 void connection_start_reading (connection_t ∗ *conn*)

Tell the main loop to start notifying **conn** of any read events.

References connection_should_read_from_linked_conn(), connection_start_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::read_event, connection_t::reading_from_linked_conn, connection_t::s, and tor_assert.

Referenced by circuit_resume_edge_reading_helper(), connection_bucket_refill(), connection_create_listener(), connection_edge_process_relay_cell(), connection_exit_connect_dir(), connection_free(), connection_init_accepted_conn(), connection_or_finished_connecting(), connection_or_set_state_open(), connection_tls_continue_handshake(), connection_tls_start_handshake(), connection_unlink(), connection_watch_events(), directory_initiate_command_rend(), set_streams_blocked_on_circ(), and spawn_cpuworker().

4.35.5.278 void connection_start_writing (connection_t ∗ *conn*)

Tell the main loop to start notifying **conn** of any write events.

References connection_should_read_from_linked_conn(), connection_start_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::linked_conn, connection_t::s, tor_assert, connection_t::write_event, and connection_t::writing_to_linked_conn.

Referenced by _connection_write_to_buf_impl(), connection_bucket_refill(), connection_edge_finished_connecting(), connection_read_to_buf(), connection_tls_continue_handshake(), and connection_watch_events().

4.35.5.279 int connection_state_is_connecting (connection_t ∗ *conn*)

Return 1 if conn is in 'connecting' state, else return 0.

References CONN_TYPE_DIR, CONN_TYPE_EXIT, CONN_TYPE_OR, DIR_CONN_STATE_CONNECTING, EXIT_CONN_STATE_CONNECTING, connection_t::marked_for_close, OR_CONN_STATE_CONNECTING, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_handle_write_impl().

4.35.5.280 int connection_state_is_open (connection_t ∗ *conn*)

Return 1 if **conn** is in state "open" and is not marked for close, else return 0.

References AP_CONN_STATE_OPEN, CONN_TYPE_AP, CONN_TYPE_CONTROL, CONN_TYPE_EXIT, CONN_TYPE_OR, CONTROL_CONN_STATE_OPEN, EXIT_CONN_STATE_OPEN, connection_t::marked_for_close, OR_CONN_STATE_OPEN, connection_t::state, tor_assert, and connection_t::type.

Referenced by connection_edge_process_relay_cell(), connection_edge_reached_eof(), and run_connection_housekeeping().

4.35.5.281 void connection_stop_reading (connection_t ∗ *conn*)

Tell the main loop to stop notifying **conn** of any read events.

References connection_stop_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::read_event, connection_t::reading_from_linked_conn, connection_t::s, and tor_assert.

Referenced by circuit_consider_stop_edge_reading(), connection_consider_empty_read_buckets(), connection_control_process_inbuf(), connection_edge_package_raw_inbuf(), connection_watch_events(), and set_streams_blocked_on_circ().

4.35.5.282 void connection_stop_reading_from_linked_conn (connection_t ∗ *conn*)

Tell the main loop to stop reading bytes into **conn** from its linked connection, if is currently doing so. Called by connection_stop_reading, connection_stop_writing, and connection_read.

References connection_t::active_on_link, connection_t::linked, smartlist_isin(), smartlist_remove(), and tor_assert.

Referenced by connection_handle_read_impl(), connection_stop_reading(), and connection_stop_writing().

4.35.5.283 void connection_stop_writing (connection_t ∗ *conn*)

Tell the main loop to stop notifying **conn** of any write events.

References connection_stop_reading_from_linked_conn(), LD_NET, connection_t::linked, connection_t::linked_conn, connection_t::s, tor_assert, connection_t::write_event, and connection_t::writing_to_linked_conn.

Referenced by connection_consider_empty_write_buckets(), connection_control_finished_flushing(), connection_cpu_finished_flushing(), connection_dir_finished_flushing(), connection_edge_finished_flushing(), connection_handle_write_impl(), connection_or_finished_flushing(), connection_tls_continue_handshake(), and connection_watch_events().

4.35.5.284 int connection_tls_continue_handshake (or_connection_t ∗ *conn*)

Move forward with the tls handshake. If it finishes, hand **conn** to **connection_tls_finish_handshake()** (p. 459).

Return -1 if **conn** is broken, else return 0.

References or_connection_t::_base, CASE_TOR_TLS_ERROR_ANY, connection_or_tls_renegotiated_cb(), connection_start_reading(), connection_start_writing(), connection_stop_writing(), connection_tls_finish_handshake(), LD_OR, OR_CONN_STATE_TLS_CLIENT_RENEGOTIATING, OR_CONN_STATE_TLS_HANDSHAKING, OR_CONN_STATE_TLS_SERVER_RENEGOTIATING, connection_t::state, or_connection_t::tls, TO_CONN, tor_assert, tor_tls_err_to_string(), tor_tls_handshake(), tor_tls_is_server(), tor_tls_renegotiate(), tor_tls_set_renegotiate_callback(), and tor_tls_used_v1_handshake().

Referenced by connection_handle_write_impl(), connection_read_to_buf(), and connection_tls_start_handshake().

4.35.5.285 int connection_tls_start_handshake (or_connection_t ∗ *conn*, int *receiving*)

Begin the tls handshake with **conn**. **receiving** is 0 if we initiated the connection, else it's 1.

Assign a new tls object to conn->tls, begin reading on **conn**, and pass **conn** to **connection_tls_continue_handshake()** (p. 458).

Return -1 if **conn** is broken, else return 0.

References or_connection_t::_base, connection_t::address, connection_start_reading(), connection_tls_continue_handshake(), escaped_safe_str(), LD_BUG, LD_HANDSHAKE, note_crypto_pk_op(), OR_CONN_STATE_TLS_HANDSHAKING, connection_t::s, connection_t::state, or_connection_t::tls, TO_CONN, tor_tls_new(), and tor_tls_set_logged_address().

Referenced by connection_init_accepted_conn(), connection_or_finished_connecting(), and connection_or_process_inbuf().

4.35.5.286 void connection_unregister_events (connection_t ∗ *conn*)

Tell libevent that we don't care about **conn** any more.

References connection_t::dns_server_port, dnsserv_close_listener(), LD_BUG, connection_t::read_event, connection_t::s, tor_free, and connection_t::write_event.

Referenced by connection_close_immediate(), connection_free(), and connection_remove().

4.35.5.287 int connection_wants_to_flush (connection_t ∗ *conn*)

Return conn->outbuf_flushlen: how many bytes conn wants to flush from its outbuf.

References connection_t::outbuf_flushlen.

Referenced by conn_close_if_marked(), connection_edge_finished_connecting(), connection_exit_connect(), connection_handle_read_impl(), and connection_handle_write_impl().

4.35.5.288 void connection_watch_events (connection_t ∗ *conn*, watchable_events_t *events*)

Set the event mask on **conn** to **events**. (The event mask is a bitmask whose bits are READ_EVENT and WRITE_EVENT)

References connection_start_reading(), connection_start_writing(), connection_stop_reading(), and connection_stop_writing().

Referenced by connection_edge_finished_connecting(), connection_exit_connect(), connection_or_connect(), and directory_initiate_command_rend().

4.35.5.289 int consensus_is_waiting_for_certs (void)

Return 1 if we have a consensus but we don't have enough certificates to start using it yet.

References consensus_waiting_for_certs_t::consensus, and USABLE_CONSENSUS_FLAVOR.

Referenced by connection_edge_process_relay_cell_not_open().

4.35.5.290 void consider_hibernation (time_t *now*)

Consider our environment and decide if it's time to start/stop hibernating.

References or_options_t::AccountingMax, format_local_iso_time(), get_options(), hibernate_begin(), hibernate_end_time, hibernate_end_time_elapsed(), hibernate_go_dormant(), hibernate_hard_limit_reached(), hibernate_soft_limit_reached(), hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_EXITING, HIBERNATE_STATE_LIVE, HIBERNATE_STATE_LOWBANDWIDTH, interval_wakeup_time, LD_ACCT, LD_GENERAL, shutdown_time, tor_assert, and tor_cleanup().

Referenced by run_scheduled_events().

4.35.5.291 void consider_publishable_server (int *force*)

Initiate server descriptor upload as reasonable (if server is publishable, etc). **force** is as for router_upload_dir_desc_to_dirservers.

We need to rebuild the descriptor if it's dirty even if we're not uploading, because our reachability testing *uses* our descriptor to determine what IP address and ports to test.

References decide_if_publishable_server(), get_options(), router_rebuild_descriptor(), router_upload_dir_desc_to_dirservers(), server_mode(), and set_server_advertised().

Referenced by run_scheduled_events().

4.35.5.292 void consider_testing_reachability (int *test_or*, int *test_dir*)

Some time has passed, or we just got new directory information. See if we currently believe our ORPort or DirPort to be unreachable. If so, launch a new test for it.

For ORPort, we simply try making a circuit that ends at ourselves. Success is noticed in **onionskin_answer()** (p. 324).

For DirPort, we make a connection via Tor to our DirPort and ask for our own server descriptor. Success is noticed in **connection_dir_client_reached_eof()** (p. 569).

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::cache_info, check_whether_dirport_reachable(), check_whether_orport_reachable(), CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, circuit_enough_testing_circs(), circuit_launch_by_router(), CIRCUIT_PURPOSE_TESTING, CONN_TYPE_DIR, connection_get_by_type_addr_port_purpose(), routerinfo_t::dir_port, DIR_PURPOSE_FETCH_SERVERDESC, directory_initiate_command(), signed_descriptor_t::identity_digest, LD_CIRC, routerinfo_t::or_port, router_get_my_routerinfo(), ROUTER_PURPOSE_GENERAL, and tor_addr_from_ipv4h.

Referenced by circuit_send_next_onion_skin(), circuit_testing_opened(), directory_info_has_arrived(), and run_scheduled_events().

4.35.5.293 void control_adjust_event_log_severity (void)

Adjust the log severities that result in control_event_logmsg being called to match the severity of log messages that any controllers are interested in.

References change_callback_log_severity(), control_event_logmsg(), EVENT_IS_INTERESTING, and event_to_log_severity().

Referenced by options_act_reversible().

4.35.5.294 control_connection_t* control_connection_new (int *socket_family*)

Allocate and return a new **control_connection_t** (p. 63), initialized as by **connection_init()** (p. 420).

References CONN_TYPE_CONTROL, connection_init(), LD_CONTROL, and TO_CONN.

Referenced by connection_new().

4.35.5.295 int control_event_address_mapped (const char * *from*, const char * *to*, time_t *expires*, const char * *error*)

Called when an address mapping on **from** from changes to **to**. **expires** values less than 3 are special; see **connection_edge.c** (p. 429). If **error** is non-NULL, it is an error code describing the failure mode of the mapping.

References ALL_FORMATS, EVENT_IS_INTERESTING, format_iso_time(), format_local_iso_time(), and send_control_event().

Referenced by addressmap_register(), handle_control_resolve(), and tell_controller_about_resolved_result().

4.35.5.296 int control_event_bandwidth_used (uint32_t *n_read*, uint32_t *n_written*)

A second or more has elapsed: tell any interested control connections how much bandwidth we used.

References ALL_FORMATS, EVENT_IS_INTERESTING, and send_control_event().

Referenced by second_elapsed_callback().

4.35.5.297 void control_event_bootstrap (bootstrap_status_t *status*, int *progress*)

Called when Tor has made progress at bootstrapping its directory information and initial circuits.

status is the new status, that is, what task we will be doing next. **percent** is zero if we just started this task, else it represents progress on the task.

References BOOTSTRAP_MSG_LEN, bootstrap_status_to_string(), control_event_client_status(), last_sent_bootstrap_message, LD_CONTROL, and tor_snprintf().

Referenced by circuit_handle_first_hop(), circuit_send_next_onion_skin(), connection_edge_process_relay_cell_not_open(), connection_or_finished_connecting(), do_main_loop(), load_downloaded_routers(), and update_router_have_minimum_dir_info().

4.35.5.298 void control_event_bootstrap_problem (const char * *warn*, int *reason*)

Called when Tor has failed to make bootstrapping progress in a way that indicates a problem. **warn** gives a hint as to why, and **reason** provides an "or_conn_end_reason" tag.

References any_bridge_descriptors_known(), any_pending_bridge_descriptor_fetches(), BOOTSTRAP_MSG_LEN, bootstrap_status_to_string(), control_event_client_status(), get_options(), last_sent_bootstrap_message, LD_CONTROL, orconn_end_reason_to_control_string(), and tor_snprintf().

Referenced by connection_about_to_close_connection(), connection_or_check_valid_tls_handshake(), and connection_or_connect_failed().

4.35.5.299 int control_event_buildtimeout_set (const circuit_build_times_t * *cbt*, buildtimeout_set_event_t *type*)

Called when we compute a new circuitbuildtimeout

References ALL_FORMATS, circuit_build_times_t::alpha, control_event_is_interesting(), send_control_event(), circuit_build_times_t::timeout_ms, circuit_build_times_t::total_build_times, and circuit_build_times_t::Xm.

Referenced by circuit_build_times_init(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), and circuit_build_times_set_timeout().

4.35.5.300 int control_event_circuit_status (origin_circuit_t * *circ*, circuit_status_event_t *tp*, int *reason_code*)

Something has happened to circuit **circ**: tell any interested control connections.

References origin_circuit_t::_base, ALL_FORMATS, circuit_end_reason_to_control_string(), circuit_list_path_for_controller(), circuit_purpose_to_controller_string(), END_CIRC_REASON_FLAG_REMOTE, EVENT_IS_INTERESTING, origin_circuit_t::global_identifier, LD_BUG, circuit_t::purpose, send_control_event(), tor_assert, tor_free, and tor_snprintf().

Referenced by _circuit_mark_for_close(), circuit_establish_circuit(), circuit_finish_handshake(), circuit_has_opened(), and handle_control_extendcircuit().

4.35.5.301 void control_event_clients_seen (const char * *controller_str*)

We just generated a new summary of which countries we've seen clients from recently. Send a copy to the controller in case it wants to display it for the user.

References send_control_event().

Referenced by geoip_bridge_stats_write().

4.35.5.302 int control_event_descriptors_changed (smartlist_t ∗ *routers*)

Called whenever we receive new router descriptors: tell any interested control connections. **routers** is a list of routerinfo_t's.

References ALL_FORMATS, EVENT_IS_INTERESTING, MAX_VERBOSE_NICKNAME_LEN, router_get_verbose_nickname(), send_control_event_string(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

Referenced by dirserv_add_descriptor(), and routerlist_descriptors_added().

4.35.5.303 int int int int control_event_guard (const char ∗ *nickname*, const char ∗ *digest*, const char ∗ *status*)

Called when the status of an entry guard with the given **nickname** and identity **digest** has changed to **status**: tells any controllers that care.

References ALL_FORMATS, base16_encode(), DIGEST_LEN, EVENT_IS_INTERESTING, HEX_DIGEST_LEN, MAX_VERBOSE_NICKNAME_LEN, router_get_by_digest(), router_get_verbose_nickname(), send_control_event(), and tor_snprintf().

Referenced by add_an_entry_guard(), control_event_guard_deferred(), entry_guard_set_status(), remove_dead_entry_guards(), and remove_obsolete_entry_guards().

4.35.5.304 int control_event_is_interesting (int *event*)

Return true iff the event with code **c** is being sent to any current control connection. This is useful if the amount of work needed to prepare to call the appropriate control_event_...() function is high.

References EVENT_IS_INTERESTING.

Referenced by control_event_buildtimeout_set(), control_event_newconsensus(), and notify_control_networkstatus_changed().

4.35.5.305 int control_event_my_descriptor_changed (void)

Our own router descriptor has changed; tell any controllers that care.

References ALL_FORMATS, and send_control_event().

Referenced by router_rebuild_descriptor().

4.35.5.306 int control_event_networkstatus_changed (smartlist_t ∗ *statuses*)

Called when the routerstatus_ts **statuses** have changed: sends an NS event to any controller that cares.

References control_event_networkstatus_changed_helper().

Referenced by control_event_networkstatus_changed_single(), and notify_control_networkstatus_changed().

4.35.5.307 int control_event_networkstatus_changed_single (routerstatus_t * *rs*)

Called when a single local_routerstatus_t has changed: Sends an NS event to any controller that cares.

References control_event_networkstatus_changed(), EVENT_IS_INTERESTING, smartlist_add(), smartlist_create(), and smartlist_free().

Referenced by router_set_status().

4.35.5.308 int control_event_newconsensus (const networkstatus_t * *consensus*)

Called when we get a new consensus networkstatus. Sends a NEWCONSENSUS event consisting of an NS-style line for each relay in the consensus.

References control_event_is_interesting(), control_event_networkstatus_changed_helper(), and networkstatus_t::routerstatus_list.

Referenced by notify_control_networkstatus_changed().

4.35.5.309 int control_event_or_authdir_new_descriptor (const char * *action*, const char * *desc*, size_t *desclen*, const char * *msg*)

The authoritative dirserver has received a new descriptor that has passed basic syntax checks and is properly self-signed.

Notify any interested party of the new descriptor and what has been done with it, and also optionally give an explanation/reason.

References ALL_FORMATS, EVENT_IS_INTERESTING, send_control_event_string(), tor_free, tor_snprintf(), and write_escaped_data().

Referenced by dirserv_add_descriptor().

4.35.5.310 int control_event_or_conn_status (or_connection_t * *conn*, or_conn_status_event_t *tp*, int *reason*)

Called when the status of an OR connection **conn** changes: tell any interested control connections. **tp** is the new status for the connection. If **conn** has just closed or failed, then **reason** may be the reason why.

References ALL_FORMATS, circuit_count_pending_on_or_conn(), EVENT_IS_INTERESTING, LD_BUG, or_connection_t::n_circuits, orconn_end_reason_to_control_string(), orconn_target_get_name(), send_control_event(), and tor_snprintf().

Referenced by connection_about_to_close_connection(), connection_init_accepted_conn(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_connect_failed(), and connection_or_set_state_open().

4.35.5.311 int control_event_stream_bandwidth (edge_connection_t * *edge_conn*)

Print out STREAM_BW event for a single conn

References edge_connection_t::_base, ALL_FORMATS, EVENT_IS_INTERESTING, connection_t::global_identifier, edge_connection_t::n_read, edge_connection_t::n_written, and send_control_event().

Referenced by connection_about_to_close_connection(), and connection_edge_destroy().

4.35.5.312 int control_event_stream_bandwidth_used (void)

A second or more has elapsed: tell any interested control connections how much bandwidth streams have used.

References edge_connection_t::_base, ALL_FORMATS, CONN_TYPE_AP, EVENT_IS_INTERESTING, get_connection_array(), connection_t::global_identifier, edge_connection_t::n_read, edge_connection_t::n_written, send_control_event(), and TO_EDGE_CONN().

Referenced by second_elapsed_callback().

4.35.5.313 int control_event_stream_status (edge_connection_t * *conn*, stream_status_event_t *tp*, int *reason_code*)

Something has happened to the stream associated with AP connection **conn**: tell any interested control connections.

References edge_connection_t::_base, ALL_FORMATS, circuit_get_by_edge_conn(), CIRCUIT_IS_ORIGIN, socks_request_t::command, CONN_TYPE_DIR, DIR_PURPOSE_IS_UPLOAD, END_STREAM_REASON_FLAG_ALREADY_SENT_CLOSED, END_STREAM_REASON_FLAG_REMOTE, EVENT_IS_INTERESTING, origin_circuit_t::global_identifier, connection_t::global_identifier, edge_connection_t::is_dns_request, LD_BUG, connection_t::purpose, REMAP_STREAM_SOURCE_CACHE, REMAP_STREAM_SOURCE_EXIT, send_control_event(), edge_connection_t::socks_request, stream_end_reason_to_control_string(), TO_CONN, TO_ORIGIN_CIRCUIT(), tor_assert, tor_free, tor_snprintf(), connection_t::type, edge_connection_t::use_begindir, and write_stream_target_to_buf().

Referenced by connection_about_to_close_connection(), connection_ap_detach_retriable(), connection_ap_handshake_process_socks(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_reply(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_edge_destroy(), evdns_server_callback(), and remap_event_helper().

4.35.5.314 void control_signal_act (int *the_signal*)

Used to implement the SIGNAL control command: if we accept **the_signal** as a remote pseudo-signal, act on it.

References LD_BUG, and signal_callback().

Referenced by handle_control_signal().

4.35.5.315 void control_update_global_event_mask (void)

Set **global_event_mask**∗ to the bitwise OR of each live control connection's event_mask field.

References CONN_TYPE_CONTROL, get_connection_array(), global_event_mask, STATE_IS_OPEN, and TO_CONTROL_CONN().

Referenced by connection_free(), connection_free_all(), and handle_control_setevents().

4.35.5.316 int count_loading_descriptors_progress (void)

We just fetched a new set of descriptors. Compute how far through the "loading descriptors" bootstrapping phase we are, so we can inform the controller of our progress.

References count_usable_descriptors(), get_options(), and networkstatus_get_reasonably_live_consensus().

Referenced by connection_edge_process_relay_cell_not_open(), and load_downloaded_routers().

4.35.5.317 void cpu_init (void)

Initialize the cpuworker subsystem.

References cpuworkers_rotate().

Referenced by do_main_loop().

4.35.5.318 void cpuworkers_rotate (void)

Called when the onion key has changed and we need to spawn new cpuworkers. Close all currently idle cpuworkers, and mark the last rotation time as now.

References CONN_TYPE_CPUWORKER, connection_get_by_type_state(), CPUWORKER_STATE_IDLE, get_options(), last_rotation_time, num_cpuworkers, server_mode(), and spawn_enough_cpuworkers().

Referenced by cpu_init(), do_hup(), options_act(), and run_scheduled_events().

4.35.5.319 const char∗ decode_address_from_payload (tor_addr_t ∗ *addr_out*, const char ∗ *payload*, int *payload_len*)

Given **payload_len** bytes at **payload**, starting with an address encoded as by **append_address_to_payload()** (p. 823), try to decode the address into ∗**addr_out**. Return the next byte in the payload after the address on success, or NULL on failure.

References get_uint32(), tor_addr_from_ipv4n(), tor_addr_from_ipv6_bytes(), and tor_addr_make_unspec().

Referenced by command_process_netinfo_cell().

4.35.5.320 smartlist_t∗ decode_hashed_passwords (config_line_t ∗ *passwords*)

Decode the hashed, base64'd passwords stored in **passwords**. Return a smartlist of acceptable passwords (unterminated strings of length S2K_SPECIFIER_LEN+DIGEST_LEN) on success, or NULL on failure.

References base16_decode(), base64_decode(), DIGEST_LEN, config_line_t::next, S2K_SPECIFIER_LEN, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), tor_assert, tor_free, and config_line_t::value.

Referenced by handle_control_authenticate().

4.35.5.321 dir_connection_t∗ dir_connection_new (int *socket_family*)

Allocate and return a new **dir_connection_t** (p. 79), initialized as by **connection_init()** (p. 420).

References CONN_TYPE_DIR, connection_init(), and TO_CONN.

Referenced by connection_exit_connect_dir(), connection_new(), and directory_initiate_command_rend().

4.35.5.322 int dir_policy_permits_address (const tor_addr_t ∗ *addr*)

Return 1 if **addr** is permitted to connect to our dir port, based on **dir_policy**. Else return 0.

References addr_policy_permits_tor_addr().

Referenced by connection_handle_listener_read().

4.35.5.323 int dir_split_resource_into_fingerprint_pairs (const char ∗ *res*, smartlist_t ∗ *pairs_out*)

Divide a string **res** of the form FP1-FP2+FP3-FP4...[.z], where each FP is a hex-encoded fingerprint, into a sequence of distinct sorted **fp_pair_t** (p. 100). Skip malformed pairs. On success, return 0 and add those **fp_pair_t** (p. 100) into **pairs_out**. On failure, return -1.

References _compare_pairs(), _tor_free(), base16_decode(), DIGEST_LEN, escaped(), fp_pair_t::first, HEX_DIGEST_LEN, LD_DIR, fp_pair_t::second, smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_sort(), smartlist_split_string(), smartlist_uniq(), and tor_free.

Referenced by directory_handle_command_get().

4.35.5.324 int dir_split_resource_into_fingerprints (const char ∗ *resource*, smartlist_t ∗ *fp_out*, int ∗ *compressed_out*, int *flags*)

Given a directory **resource** request, containing zero or more strings separated by plus signs, followed optionally by ".z", store the strings, in order, into **fp_out**. If **compressed_out** is non-NULL, set it to 1 if the resource ends in ".z", else set it to 0.

If (flags & DSR_HEX), then delete all elements that aren't hex digests, and decode the rest. If (flags & DSR_BASE64), then use "-" rather than "+" as a separator, delete all the elements that aren't base64-encoded digests, and decode the rest. If (flags & DSR_DIGEST256), these digests should be 256 bits long; else they should be 160.

If (flags & DSR_SORT_UNIQ), then sort the list and remove all duplicates.

References base16_decode(), base64_decode(), BASE64_DIGEST256_LEN, BASE64_DIGEST_LEN, DIGEST256_LEN, DIGEST_LEN, escaped(), HEX_DIGEST256_LEN, HEX_DIGEST_LEN, LD_DIR, smartlist_add_all(), smartlist_create(), smartlist_del_keeporder(), smartlist_free(), smartlist_sort_digests(), smartlist_sort_digests256(), smartlist_sort_strings(), smartlist_split_string(), smartlist_uniq_digests(), smartlist_uniq_digests256(), smartlist_uniq_strings(), tor_assert, and tor_free.

Referenced by client_likes_consensus(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), directory_handle_command_get(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), and list_pending_downloads().

4.35.5.325 int directories_have_accepted_server_descriptor (void)

Return true iff any trusted directory authority has accepted our server descriptor.

We consider any authority sufficient because waiting for all of them means it never happens while any authority is down; we don't go for something more complex in the middle (like >1/3 or >1/2 or >=1/2) because that doesn't seem necessary yet.

References or_options_t::_PublishServerDescriptor, get_options(), and router_get_trusted_dir_servers().

Referenced by connection_dir_client_reached_eof(), and getinfo_helper_events().

4.35.5.326 void directory_all_unreachable (time_t *now*)

We've just tried every dirserver we know about, and none of them were reachable. Assume the network is down. Change state so next time an application connection arrives we'll delay it and try another directory fetch. Kill off all the circuit_wait streams that are waiting now, since they will all timeout anyway.

References socks_request_t::address, AP_CONN_STATE_CIRCUIT_WAIT, CONN_TYPE_AP, connection_get_by_type_state(), control_event_general_status(), END_STREAM_REASON_NET_UNREACHABLE, LD_NET, socks_request_t::port, safe_str_client(), edge_connection_t::socks_request, stats_n_seconds_working, and TO_EDGE_CONN().

Referenced by directory_get_from_dirserver().

4.35.5.327 int directory_caches_dir_info (or_options_t * *options*)

Return 1 if we want to keep descriptors, networkstatuses, etc around and we're willing to serve them to others. Else return 0.

References advertised_server_mode(), or_options_t::BridgeRelay, or_options_t::DirPort, or_options_t::RefuseUnknownExits, and server_mode().

Referenced by authority_certs_fetch_missing(), directory_too_idle_to_fetch_descriptors(), networkstatus_set_current_consensus(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), signed_desc_digest_is_recognized(), and trusted_dirs_load_certs_from_string().

4.35.5.328 int directory_caches_v2_dir_info (or_options_t * *options*)

Return 1 if we want to cache v2 dir info (each status file).

References or_options_t::DirPort.

Referenced by add_networkstatus_to_cache(), networkstatus_v2_list_clean(), options_act_reversible(), router_reload_v2_networkstatus(), and router_set_networkstatus_v2().

4.35.5.329 int directory_fetches_dir_info_early (or_options_t * *options*)

Return 1 if we should fetch new networkstatuses, descriptors, etc on the "mirror" schedule rather than the "client" schedule.

References directory_fetches_from_authorities().

Referenced by launch_router_descriptor_downloads(), options_act(), update_consensus_networkstatus_fetch_time(), update_networkstatus_downloads(), update_router_descriptor_cache_downloads_v2(), and update_router_descriptor_downloads().

4.35.5.330 int directory_fetches_dir_info_later (or_options_t * *options*)

Return 1 if we should fetch new networkstatuses, descriptors, etc on a very passive schedule – waiting long enough for ordinary clients to probably have the info we want. These would include bridge users, and maybe others in the future e.g. if a Tor client uses another Tor client as a directory guard.

References or_options_t::UseBridges.

Referenced by options_act(), and update_consensus_networkstatus_fetch_time().

4.35.5.331 int directory_fetches_from_authorities (or_options_t * *options*)

Return 1 if we fetch our directory material directly from the authorities, rather than from a mirror.

References advertised_server_mode(), or_options_t::BridgeRelay, routerinfo_t::dir_port, or_options_t::DirPort, or_options_t::FetchDirInfoEarly, or_options_t::RefuseUnknownExits, router_get_my_routerinfo(), router_pick_published_address(), and server_mode().

Referenced by dir_routerdesc_download_failed(), directory_command_should_use_begindir(), directory_fetches_dir_info_early(), directory_get_from_dirserver(), and directory_info_has_arrived().

4.35.5.332 void directory_get_from_all_authorities (uint8_t *dir_purpose*, uint8_t *router_purpose*, const char * *resource*)

As directory_get_from_dirserver, but initiates a request to *every* directory authority other than ourself. Only for use by authorities when searching for missing information while voting.

References DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_STATUS_VOTE, directory_initiate_command_routerstatus(), router_digest_is_me(), router_get_trusted_dir_servers(), tor_assert, and V3_AUTHORITY.

Referenced by dirvote_fetch_missing_signatures().

4.35.5.333 void directory_get_from_dirserver (uint8_t *dir_purpose*, uint8_t *router_purpose*, const char * *resource*, int *pds_flags*)

Start a connection to a random running directory server, using connection purpose **dir_purpose**, intending to fetch descriptors of purpose **router_purpose**, and requesting **resource**. Use **pds_flags** as arguments to **router_pick_directory_server()** (p. 1001) or **router_pick_trusteddirserver()** (p. 1001).

References routerinfo_t::addr, routerinfo_t::address, BRIDGE_AUTHORITY, routerinfo_t::cache_info, choose_random_entry(), dir_conn_purpose_to_string(), DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_DETACHED_SIGNATURES, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_RENDDESC, DIR_PURPOSE_FETCH_SERVERDESC, DIR_PURPOSE_FETCH_STATUS_VOTE, DIR_PURPOSE_FETCH_V2_NETWORKSTATUS, directory_all_unreachable(), directory_fetches_from_authorities(), directory_initiate_command(), directory_initiate_command_routerstatus(), EXTRAINFO_CACHE, or_-

options_t::FetchServerDescriptors, get_options(), HIDSERV_AUTHORITY, signed_descriptor_t::identity_digest, LD_BUG, LD_DIR, networkstatus_get_latest_consensus(), routerinfo_t::or_port, PDS_ALLOW_SELF, PDS_IGNORE_FASCISTFIREWALL, PDS_NO_EXISTING_SERVERDESC_FETCH, purpose_needs_anonymity(), router_pick_directory_server(), router_pick_trusteddirserver(), ROUTER_PURPOSE_BRIDGE, tor_addr_from_ipv4h, or_options_t::UseBridges, V2_AUTHORITY, V3_AUTHORITY, and networkstatus_t::valid_after.

Referenced by connection_dir_download_networkstatus_failed(), fetch_bridge_descriptors(), initiate_descriptor_downloads(), update_consensus_networkstatus_downloads(), update_router_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.35.5.334 void directory_info_has_arrived (time_t *now*, int *from_cache*)

This function is called whenever we successfully pull down some new network statuses or server descriptors.

References any_predicted_circuits(), consider_testing_reachability(), directory_fetches_from_authorities(), directory_too_idle_to_fetch_descriptors(), or_options_t::DownloadExtraInfo, entry_guards_compute_status(), get_dir_info_status_string(), get_options(), has_completed_circuit, LD_DIR, router_have_minimum_dir_info(), server_mode(), update_extrainfo_downloads(), update_router_descriptor_downloads(), and we_are_hibernating().

Referenced by connection_dir_client_reached_eof(), and do_main_loop().

4.35.5.335 void directory_initiate_command (const char ∗ *address*, const tor_addr_t ∗ *_addr*, uint16_t *or_port*, uint16_t *dir_port*, int *supports_conditional_consensus*, int *supports_begindir*, const char ∗ *digest*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char ∗ *resource*, const char ∗ *payload*, size_t *payload_len*, time_t *if_modified_since*)

Helper for directory_initiate_command_routerstatus: send the command to a server whose address is **address**, whose IP is **addr**, whose directory port is **dir_port**, whose tor version **supports_begindir**, and whose identity key digest is **digest**.

References directory_initiate_command_rend().

Referenced by consider_testing_reachability(), directory_get_from_dirserver(), and launch_direct_bridge_descriptor_fetch().

4.35.5.336 void directory_initiate_command_routerstatus (routerstatus_t ∗ *status*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char ∗ *resource*, const char ∗ *payload*, size_t *payload_len*, time_t *if_modified_since*)

Launch a new connection to the directory server **status** to upload or download a server or rendezvous descriptor. **dir_purpose** determines what kind of directory connection we're launching, and must be one of DIR_PURPOSE_{FETCH|UPLOAD}_{DIR|RENDDESC|RENDDESC_V2}. **router_purpose** specifies the descriptor purposes we have in mind (currently only used for FETCH_DIR).

When uploading, **payload** and **payload_len** determine the content of the HTTP post. Otherwise, **payload** should be NULL.

When fetching a rendezvous descriptor, **resource** is the service ID we want to fetch.

References directory_initiate_command_routerstatus_rend().

Referenced by directory_get_from_all_authorities(), directory_get_from_dirserver(), directory_post_to_dirservers(), directory_post_to_hs_dir(), initiate_descriptor_downloads(), and update_v2_networkstatus_cache_downloads().

4.35.5.337 void directory_initiate_command_routerstatus_rend (routerstatus_t ∗ *status*, uint8_t *dir_purpose*, uint8_t *router_purpose*, int *anonymized_connection*, const char ∗ *resource*, const char ∗ *payload*, size_t *payload_len*, time_t *if_modified_since*, const rend_data_t ∗ *rend_query*)

Same as **directory_initiate_command_routerstatus()** (p. 577), but accepts rendezvous data to fetch a hidden service descriptor.

References routerstatus_t::addr, routerinfo_t::address, routerstatus_t::dir_port, directory_initiate_command_rend(), routerstatus_t::identity_digest, INET_NTOA_BUF_LEN, LD_DIR, routerstatus_t::nickname, routerstatus_t::or_port, router_get_by_digest(), tor_addr_from_ipv4h, tor_inet_ntoa(), routerstatus_t::version_supports_begindir, and routerstatus_t::version_supports_conditional_consensus.

Referenced by directory_get_from_hs_dir(), and directory_initiate_command_routerstatus().

4.35.5.338 int directory_permits_begindir_requests (or_options_t ∗ *options*)

Return 1 if we want to allow remote people to ask us directory requests via the "begin_dir" interface, which doesn't require having any separate port open.

References or_options_t::BridgeRelay, and or_options_t::DirPort.

Referenced by connection_exit_begin_conn().

4.35.5.339 int directory_permits_controller_requests (or_options_t ∗ *options*)

Return 1 if we want to allow controllers to ask us directory requests via the controller interface, which doesn't require having any separate port open.

References or_options_t::DirPort.

4.35.5.340 void directory_post_to_dirservers (uint8_t *dir_purpose*, uint8_t *router_purpose*, authority_type_t *type*, const char ∗ *payload*, size_t *payload_len*, size_t *extrainfo_len*)

Start a connection to every suitable directory authority, using connection purpose 'purpose' and uploading the payload 'payload' (length 'payload_len'). dir_purpose should be one of 'DIR_PURPOSE_UPLOAD_DIR' or 'DIR_PURPOSE_UPLOAD_RENDDESC'.

type specifies what sort of dir authorities (V1, V2, HIDSERV, BRIDGE) we should upload to.

If **extrainfo_len** is nonzero, the first **payload_len** bytes of **payload** hold a router descriptor, and the next **extrainfo_len** bytes of **payload** hold an extra-info document. Upload the descriptor to all authorities, and the extra-info document to all authorities that support it.

References tor_addr_t::addr, authority_type_to_string(), DIR_PURPOSE_UPLOAD_DIR, directory_initiate_command_routerstatus(), fascist_firewall_allows_address_dir(), LD_DIR, purpose_needs_anonymity(), router_get_trusted_dir_servers(), router_supports_extrainfo(), tor_addr_from_ipv4h, tor_assert, and tor_free.

Referenced by dirvote_compute_consensuses(), dirvote_perform_vote(), and router_upload_dir_desc_to_dirservers().

4.35.5.341 void directory_set_dirty (void)

Mark the directory as **dirty** – when we're next asked for a directory, we will rebuild it instead of reusing the most recently generated one.

References cached_dir_t::dir, cached_dir_t::published, runningrouters_is_dirty, the_directory_is_dirty, and the_v2_networkstatus_is_dirty.

Referenced by connection_tls_finish_handshake(), directory_remove_invalid(), and router_add_to_routerlist().

4.35.5.342 int directory_too_idle_to_fetch_descriptors (or_options_t ∗ *options*, time_t *now*)

Return 1 if we have no need to fetch new descriptors. This generally happens when we're not a dir cache and we haven't built any circuits lately.

References directory_caches_dir_info(), or_options_t::FetchUselessDescriptors, and rep_hist_circbuilding_dormant().

Referenced by directory_info_has_arrived(), update_consensus_router_descriptor_downloads(), and update_router_have_minimum_dir_info().

4.35.5.343 enum was_router_added_t dirserv_add_descriptor (routerinfo_t ∗ *ri*, const char ∗∗ *msg*, const char ∗ *source*)

Examine the parsed server descriptor in **ri** and maybe insert it into the list of server descriptors. Set ∗**msg** to a message that should be passed back to the origin of this descriptor, or NULL if there is no such message. Use **source** to produce better log messages.

Return the status of the operation

This function is only called when fresh descriptors are posted, not when we re-load the cache.

References routerinfo_t::cache_info, control_event_descriptors_changed(), control_event_or_authdir_new_descriptor(), signed_descriptor_t::identity_digest, routerinfo_t::is_valid, LD_DIR, LD_DIRSERV, MAX_DESCRIPTOR_UPLOAD_SIZE, routerinfo_t::nickname, signed_descriptor_t::published_on, router_add_to_routerlist(), router_differences_are_cosmetic(), router_get_by_digest(), router_is_me(), routerinfo_free(), signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), tor_free, WRA_WAS_ADDED(), and WRA_WAS_REJECTED().

Referenced by dirserv_add_multiple_descriptors(), and init_keys().

4.35.5.344 enum was_router_added_t dirserv_add_multiple_descriptors (const char * *desc*, uint8_t *purpose*, const char * *source*, const char ** *msg*)

As for **dirserv_add_descriptor()** (p. 594), but accepts multiple documents, and returns the most severe error that occurred for any one of them.

References dirserv_add_descriptor(), dirserv_add_extrainfo(), escaped(), format_iso_time(), ROUTER_ANNOTATION_BUF_LEN, router_parse_list_from_string(), ROUTER_PURPOSE_GENERAL, router_purpose_to_string(), SAVED_NOWHERE, smartlist_clear(), smartlist_create(), smartlist_free(), tor_assert, tor_snprintf(), WRA_MORE_SEVERE(), and WRA_WAS_ADDED().

Referenced by directory_handle_command_post().

4.35.5.345 int dirserv_add_own_fingerprint (const char * *nickname*, crypto_pk_env_t * *pk*)

Add the nickname and fingerprint for this OR to the global list of recognized identity key fingerprints.

References add_fingerprint_to_dir(), authdir_config_new(), crypto_pk_get_fingerprint(), FINGERPRINT_LEN, and LD_BUG.

Referenced by init_keys().

4.35.5.346 void dirserv_clear_old_networkstatuses (time_t *cutoff*)

Remove any v2 networkstatus from the directory cache that was published before **cutoff**.

References cached_dir_decref(), cached_v2_networkstatus, file_status(), LD_DIR, networkstatus_get_cache_filename(), and tor_free.

Referenced by networkstatus_v2_list_clean().

4.35.5.347 void dirserv_clear_old_v1_info (time_t *now*)

Remove any v1 info from the directory cache that was published too long ago.

References cached_dir_decref(), clear_cached_dir(), MAX_V1_DIRECTORY_AGE, MAX_V1_RR_AGE, and cached_dir_t::published.

Referenced by networkstatus_v2_list_clean().

4.35.5.348 int dirserv_dump_directory_to_string (char ** *dir_out*, crypto_pk_env_t * *private_key*)

Generate a new v1 directory and write it into a newly allocated string. Point ***dir_out** to the allocated string. Sign the directory with **private_key**. Return 0 on success, -1 on failure. If **complete** is set, give us all the descriptors; otherwise leave out non-running and non-valid ones.

References crypto_pk_write_public_key_to_string(), DIGEST_LEN, format_iso_time(), format_versions_list(), get_options(), LD_BUG, note_crypto_pk_op(), router_append_dirobj_signature(), router_get_dir_hash(), tor_assert, tor_free, and tor_snprintf().

Referenced by dirserv_regenerate_directory().

4.35.5.349 size_t dirserv_estimate_data_size (smartlist_t * *fps*, int *is_serverdescs*, int *compressed*)

Return an approximate estimate of the number of bytes that will be needed to transmit the server descriptors (if is_serverdescs – they can be either d/ or fp/ queries) or networkstatus objects (if !is_serverdescs) listed in **fps**. If **compressed** is set, we guess how large the data will be after compression.

The return value is an estimate; it might be larger or smaller.

References routerinfo_t::cache_info, cached_dir_t::dir_len, cached_dir_t::dir_z_len, lookup_cached_dir_by_fp(), router_get_my_routerinfo(), signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by directory_handle_command_get().

4.35.5.350 size_t dirserv_estimate_microdesc_size (const smartlist_t * *fps*, int *compressed*)

Given a list of microdescriptor hashes, guess how many bytes will be needed to transmit them, and return the guess.

Referenced by directory_handle_command_get().

4.35.5.351 void dirserv_free_all (void)

Release all storage used by the directory server.

References _free_cached_dir(), cached_consensuses, cached_dir_decref(), cached_v2_networkstatus, clear_cached_dir(), digestmap_free(), dirserv_free_fingerprint_list(), and strmap_free().

Referenced by tor_free_all().

4.35.5.352 void dirserv_free_fingerprint_list (void)

Clear the current fingerprint list.

References _tor_free(), digestmap_free(), authdir_config_t::fp_by_name, authdir_config_t::status_by_digest, strmap_free(), and tor_free.

Referenced by dirserv_free_all(), and dirserv_load_fingerprint_file().

4.35.5.353 networkstatus_t* dirserv_generate_networkstatus_vote_obj (crypto_pk_env_t * *private_key*, authority_cert_t * *cert*)

Return a new networkstatus_t* containing our current opinion. (For v3 authorities)

References or_options_t::AuthDirListBadDirs, or_options_t::AuthDirListBadExits, or_options_t::ContactInfo, crypto_pk_get_digest(), DIGEST_LEN, dirserv_set_router_is_running(), format_versions_list(), get_options(), authority_cert_t::identity_key, LD_BUG, LD_NET, or_options_t::NamingAuthoritativeDir, or_options_t::RecommendedClientVersions, or_options_t::RecommendedServerVersions, resolve_my_address(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, routerlist_t::routers, tor_assert, tor_dup_ip(), tor_free, and or_options_t::VersioningAuthoritativeDir.

Referenced by dirvote_perform_vote().

4.35.5.354 cached_dir_t* dirserv_get_consensus (const char * *flavor_name*)

Return the latest downloaded consensus networkstatus in encoded, signed, optionally compressed format, suitable for sending to clients.

References cached_consensuses, and strmap_get().

Referenced by networkstatus_set_current_consensus().

4.35.5.355 cached_dir_t* dirserv_get_directory (void)

Return the most recently generated encoded signed v1 directory, generating a new one as necessary. If not a v1 authoritative directory may return NULL if no directory is yet cached.

References dirserv_pick_cached_dir_obj(), dirserv_regenerate_directory(), the_directory_is_dirty, and V1_AUTHORITY.

Referenced by directory_handle_command_get(), and directory_handle_command_post().

4.35.5.356 void dirserv_get_networkstatus_v2 (smartlist_t * *result*, const char * *key*)

Look for a network status object as specified by **key**, which should be either "authority" (to find a network status generated by us), a hex identity digest (to find a network status generated by given directory), or "all" (to return all the v2 network status objects we have).

References base16_encode(), cached_v2_networkstatus, DIGEST_LEN, digestmap_get(), digestmap_new(), dirserv_get_networkstatus_v2_fingerprints(), generate_v2_networkstatus_opinion(), HEX_DIGEST_LEN, LD_DIRSERV, router_digest_is_me(), should_generate_v2_networkstatus(), smartlist_add(), smartlist_create(), smartlist_free(), tor_assert, and tor_free.

4.35.5.357 void dirserv_get_networkstatus_v2_fingerprints (smartlist_t * *result*, const char * *key*)

Given the portion of a networkstatus request URL after "tor/status/" in **key**, append to **result** the digests of the identity keys of the networkstatus objects that the client has requested.

References authdir_mode_v2(), routerinfo_t::cache_info, cached_v2_networkstatus, DIGEST_LEN, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_new(), digestmap_size(), dir_split_resource_into_fingerprints(), generate_v2_networkstatus_opinion(), get_options(), signed_descriptor_t::identity_digest, LD_DIRSERV, router_get_my_routerinfo(), router_get_trusted_dir_servers(), should_generate_v2_networkstatus(), smartlist_add(), smartlist_sort_digests(), strcmpstart(), tor_assert, and V2_AUTHORITY.

Referenced by directory_handle_command_get(), and dirserv_get_networkstatus_v2().

4.35.5.358 const char* dirserv_get_nickname_by_digest (const char * *digest*)

If we are an authoritative dirserver, and the list of approved servers contains one whose identity key digest is **digest**, return that router's nickname. Otherwise return NULL.

References digestmap_get(), router_status_t::nickname, authdir_config_t::status_by_digest, and tor_assert.

Referenced by connection_or_init_conn_from_address().

4.35.5.359 int dirserv_get_routerdesc_fingerprints (smartlist_t * *fps_out*, const char * *key*, const char ** *msg*, int *for_unencrypted_conn*, int *is_extrainfo*)

As **dirserv_get_routerdescs()** (p. 599), but instead of getting **signed_descriptor_t** (p. 227) pointers, adds copies of digests to fps_out, and doesn't use the /tor/server/ prefix. For a /d/ request, adds descriptor digests; for other requests, adds identity digests.

References routerinfo_t::cache_info, DIGEST_LEN, dir_split_resource_into_fingerprints(), extrainfo_get_by_descriptor_digest(), get_signed_descriptor_by_fp(), signed_descriptor_t::identity_digest, router_get_by_descriptor_digest(), router_get_my_routerinfo(), router_get_routerlist(), routerlist_t::routers, signed_descriptor_t::send_unencrypted, smartlist_add(), strcmpstart(), and tor_free.

Referenced by directory_handle_command_get().

4.35.5.360 int dirserv_get_routerdescs (smartlist_t * *descs_out*, const char * *key*, const char ** *msg*)

Add a **signed_descriptor_t** (p. 227) to **descs_out** for each router matching **key**. The key should be either

- "/tor/server/authority" for our own routerinfo;
- "/tor/server/all" for all the routerinfos we have, concatenated;
- "/tor/server/fp/FP" where FP is a plus-separated sequence of hex identity digests; or
- "/tor/server/d/D" where D is a plus-separated sequence of server descriptor digests, in hex.

Return 0 if we found some matching descriptors, or -1 if we do not have any descriptors, no matching descriptors, or if we did not recognize the key (URL). If -1 is returned ***msg** will be set to an appropriate error message.

XXXX rename this function. It's only called from the controller. XXXX in fact, refactor this function, merging as much as possible.

References routerinfo_t::cache_info, dir_split_resource_into_fingerprints(), signed_descriptor_t::published_on, router_digest_is_me(), router_get_by_descriptor_digest(), router_get_by_digest(), router_get_my_routerinfo(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, routerlist_t::routers, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), and tor_free.

4.35.5.361 cached_dir_t* dirserv_get_runningrouters (void)

Set ***rr** to the most recently generated encoded signed running-routers list, generating a new one as necessary. Return the size of the directory on success, and 0 on failure.

References dirserv_pick_cached_dir_obj(), generate_runningrouters(), runningrouters_is_dirty, and V1_AUTHORITY.

Referenced by directory_handle_command_get().

4.35.5.362 int dirserv_have_any_microdesc (const smartlist_t * *fps*)

Return true iff any of the 256-bit elements in **fps** is the digest of a microdescriptor we have.

Referenced by directory_handle_command_get().

4.35.5.363 int dirserv_have_any_serverdesc (smartlist_t * *fps*, int *spool_src*)

Return true iff we have any of the documents (extrainfo or routerdesc) specified by the fingerprints in **fps** and **spool_src**. Used to decide whether to send a 404.

References extrainfo_get_by_descriptor_digest(), get_signed_descriptor_by_fp(), router_get_by_descriptor_digest(), and ROUTER_MAX_AGE_TO_PUBLISH.

Referenced by directory_handle_command_get().

4.35.5.364 int dirserv_load_fingerprint_file (void)

Load the nickname->fingerprint mappings stored in the approved-routers file. The file format is line-based, with each non-blank holding one nickname, some space, and a fingerprint for that nickname. On success, replace the current fingerprint list with the new list and return 0. On failure, leave the current fingerprint list untouched, and return -1.

References add_fingerprint_to_dir(), authdir_config_new(), base16_decode(), config_free_lines(), config_get_lines(), DEFAULT_CLIENT_NICKNAME, DIGEST_LEN, directory_remove_invalid(), dirserv_free_fingerprint_list(), get_datadir_fname, get_options(), HEX_DIGEST_LEN, is_legal_nickname(), LD_CONFIG, LD_FS, LD_GENERAL, MAX_NICKNAME_LEN, or_options_t::NamingAuthoritativeDir, config_line_t::next, read_file_to_str(), RFTS_IGNORE_MISSING, tor_free, and tor_strstrip().

Referenced by do_hup(), and init_keys().

4.35.5.365 void dirserv_orconn_tls_done (const char * *address*, uint16_t *or_port*, const char * *digest_rcvd*, int *as_advertised*)

Called when a TLS handshake has completed successfully with a router listening at **address**:**or_port**, and has yielded a certificate with digest **digest_rcvd**.

Also, if as_advertised is 1, then inform the reachability checker that we could get to this guy.

References authdir_mode_bridge(), DIGEST_LEN, get_options(), LD_DIRSERV, rep_hist_note_router_reachable(), router_get_routerlist(), ROUTER_PURPOSE_BRIDGE, routerlist_t::routers, and tor_assert.

Referenced by connection_or_check_valid_tls_handshake().

4.35.5.366 int dirserv_read_measured_bandwidths (const char * *from_file*, smartlist_t * *routerstatuses*)

Read the measured bandwidth file and apply it to the list of routerstatuses. Returns -1 on error, 0 otherwise.

References compare_routerstatus_entries(), escaped(), LD_CONFIG, LD_DIRSERV, measured_bw_line_apply(), measured_bw_line_parse(), smartlist_sort(), and tor_parse_ulong().

Referenced by options_validate().

4.35.5.367 int dirserv_remove_old_statuses (smartlist_t * *fps*, time_t *cutoff*)

Remove from **fps** every networkstatus key where both a) we have a networkstatus document and b) it is not newer than **cutoff**.

Return 1 if any items were present at all; else return 0.

References lookup_cached_dir_by_fp(), cached_dir_t::published, and tor_free.

Referenced by directory_handle_command_get().

4.35.5.368 void dirserv_set_cached_consensus_networkstatus (const char * *networkstatus*, const char * *flavor_name*, const digests_t * *digests*, time_t *published*)

Replace the v3 consensus networkstatus of type **flavor_name** that we're serving with **networkstatus**, published at **published**. No validation is performed.

References cached_consensuses, cached_dir_decref(), cached_dir_t::digests, new_cached_dir(), and strmap_set().

Referenced by networkstatus_set_current_consensus().

4.35.5.369 void dirserv_set_cached_directory (const char * *directory*, time_t *published*, int *is_running_routers*)

If we have no cached v1 directory, or it is older than **published**, then replace it with **directory**, published at **published**.

If **published** is too old, do nothing.

If **is_running_routers**, this is really a v1 running_routers document rather than a v1 directory.

References cached_dir_decref(), MAX_V1_DIRECTORY_AGE, MAX_V1_RR_AGE, new_cached_dir(), and set_cached_dir().

Referenced by dirserv_regenerate_directory(), router_parse_directory(), and router_parse_runningrouters().

4.35.5.370 void dirserv_set_cached_networkstatus_v2 (const char * *networkstatus*, const char * *identity*, time_t *published*)

If **networkstatus** is non-NULL, we've just received a v2 network-status for an authoritative directory with identity digest **identity** published at **published** – store it so we can serve it to others.

If **networkstatus** is NULL, remove the entry with the given identity fingerprint from the v2 cache.

References cached_dir_decref(), cached_v2_networkstatus, digestmap_get(), digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_new(), digestmap_remove(), digestmap_set(), digestmap_size(), MAX_UNTRUSTED_NETWORKSTATUSES, new_cached_dir(), cached_dir_t::published, router_get_trusted_dir_servers(), and tor_assert.

Referenced by add_networkstatus_to_cache(), and networkstatus_v2_list_clean().

4.35.5.371 void dirserv_set_router_is_running (routerinfo_t ∗ *router*, time_t *now*)

Treat a router as alive if

- It's me, and I'm not hibernating. or - We've found it reachable recently.

References or_options_t::AssumeReachable, routerinfo_t::cache_info, get_options(), signed_descriptor_t::identity_digest, routerinfo_t::is_running, routerinfo_t::last_reachable, REACHABLE_TIMEOUT, rep_hist_note_router_unreachable(), router_is_me(), and we_are_hibernating().

Referenced by dirserv_generate_networkstatus_vote_obj(), generate_v2_networkstatus_opinion(), list_server_status_v1(), and networkstatus_getinfo_by_purpose().

4.35.5.372 void dirserv_single_reachability_test (time_t *now*, routerinfo_t ∗ *router*)

Helper function for **dirserv_test_reachability()** (p. 603). Start a TLS connection to **router**, and annotate it with when we started the test.

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::cache_info, connection_or_connect(), signed_descriptor_t::identity_digest, LD_OR, routerinfo_t::nickname, routerinfo_t::or_port, routerinfo_t::testing_since, and tor_addr_from_ipv4h.

Referenced by dirserv_test_reachability(), and router_add_to_routerlist().

4.35.5.373 void dirserv_test_reachability (time_t *now*)

Auth dir server only: load balance such that we only try a few connections per call.

The load balancing is such that if we get called once every ten seconds, we will cycle through all the tests in 1280 seconds (a bit over 20 minutes).

References authdir_mode_bridge(), dirserv_single_reachability_test(), get_options(), router_get_routerlist(), router_is_me(), ROUTER_PURPOSE_BRIDGE, and routerlist_t::routers.

Referenced by run_scheduled_events().

4.35.5.374 int dirserv_would_reject_router (routerstatus_t ∗ *rs*)

Return true if there is no point in downloading the router described by **rs** because this directory would reject it.

References routerstatus_t::addr, dirserv_get_status_impl(), FP_REJECT, routerstatus_t::identity_digest, routerstatus_t::nickname, and routerstatus_t::or_port.

Referenced by update_router_descriptor_cache_downloads_v2().

4.35.5.375 void dirvote_act (or_options_t ∗ *options*, time_t *now*)

Entry point: Take whatever voting actions are pending as of **now**.

References authdir_mode_v3(), authority_cert_t::cache_info, DIGEST_LEN, dirvote_clear_votes(), dirvote_compute_consensuses(), dirvote_fetch_missing_signatures(), dirvote_fetch_missing_votes(), dirvote_perform_vote(), dirvote_publish_consensus(), dirvote_recalculate_timing(), get_my_v3_authority_cert(), hex_str(), signed_descriptor_t::identity_digest, LD_DIR, list_v3_auth_ids(), tor_free, and voting_schedule.

Referenced by run_scheduled_events().

4.35.5.376 int dirvote_add_signatures (const char * *detached_signatures_body*, const char * *source*, const char ** *msg*)

Helper: we just got the **detached_signatures_body** sent to us as signatures on the currently pending consensus. Add them to the pending consensus (if we have one); otherwise queue them until we have a consensus. Return negative on failure, nonnegative on success.

References LD_DIR, smartlist_add(), and smartlist_create().

Referenced by connection_dir_client_reached_eof(), and directory_handle_command_post().

4.35.5.377 struct pending_vote_t* dirvote_add_vote (const char * *vote_body*, const char ** *msg_out*, int * *status_out*) `[read]`

Called when we have received a networkstatus vote in **vote_body**. Parse and validate it, and on success store it as a pending vote (which we then return). Return NULL on failure. Sets ***msg_out** and ***status_out** to an HTTP response and status code. (V3 authority only)

References networkstatus_voter_info_t::address, authority_cert_get_by_digests(), authority_cert_t::cache_info, cached_dir_decref(), networkstatus_t::cert, DIGEST_LEN, format_iso_time(), get_voter(), hex_str(), signed_descriptor_t::identity_digest, networkstatus_voter_info_t::identity_digest, LD_BUG, LD_DIR, list_v3_auth_ids(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), new_cached_dir(), networkstatus_voter_info_t::nickname, networkstatus_t::published, signed_descriptor_t::signed_descriptor_body, authority_cert_t::signing_key_digest, networkstatus_voter_info_t::sigs, smartlist_create(), strcmpstart(), tor_assert, tor_free, trusted_dirs_load_certs_from_string(), trusteddirserver_get_by_v3_auth_digest(), update_consensus_router_descriptor_downloads(), networkstatus_t::valid_after, networkstatus_voter_info_t::vote_digest, networkstatus_t::voters, and voting_schedule.

Referenced by connection_dir_client_reached_eof(), directory_handle_command_post(), and dirvote_perform_vote().

4.35.5.378 microdesc_t* dirvote_create_microdescriptor (const routerinfo_t * *ri*)

Construct and return a new microdescriptor from a routerinfo **ri**.

XXX Right now, there is only one way to generate microdescriptors from router descriptors. This may change in future consensus methods. If so, we'll need an internal way to remember which method we used, and ask for a particular method.

References crypto_pk_write_public_key_to_string(), routerinfo_t::declared_family, routerinfo_t::exit_policy, LD_DIR, microdescs_parse_from_string(), routerinfo_t::onion_pkey, policy_summarize(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

4.35.5.379 ssize_t dirvote_format_microdesc_vote_line (char * *out*, size_t *out_len*, const microdesc_t * *md*)

Format the appropriate vote line to describe the microdescriptor **md** in a consensus vote document. Write it into the **out_len**-byte buffer in **out**. Return -1 on failure and the number of characters written on success.

References BASE64_DIGEST256_LEN, microdesc_t::digest, digest256_to_base64(), make_consensus_method_list(), MAX_SUPPORTED_CONSENSUS_METHOD, MIN_METHOD_FOR_MICRODESC, tor_assert, and tor_snprintf().

4.35.5.380 void dirvote_free_all (void)

Release all static storage held in **dirvote.c** (p. 612)

References dirvote_clear_pending_consensuses(), dirvote_clear_votes(), smartlist_free(), and tor_free.

Referenced by tor_free_all().

4.35.5.381 const char* dirvote_get_pending_consensus (consensus_flavor_t *flav*)

Return the body of the consensus that we're currently trying to build.

References pending_consensus_t::body, and tor_assert.

Referenced by directory_handle_command_get().

4.35.5.382 const char* dirvote_get_pending_detached_signatures (void)

Return the signatures that we know for the consensus that we're currently trying to build.

Referenced by directory_handle_command_get().

4.35.5.383 void dirvote_get_preferred_voting_intervals (vote_timing_t * *timing_out*)

Set ***timing_out** to the intervals at which we would like to vote. Note that these aren't the intervals we'll use to vote; they're the ones that we'll vote to use.

References vote_timing_t::dist_delay, get_options(), vote_timing_t::n_intervals_valid, tor_assert, or_options_t::V3AuthDistDelay, or_options_t::V3AuthNIntervalsValid, or_options_t::V3AuthVoteDelay, or_options_t::V3AuthVotingInterval, vote_timing_t::vote_delay, and vote_timing_t::vote_interval.

4.35.5.384 time_t dirvote_get_start_of_next_interval (time_t *now*, int *interval*)

Return the start of the next interval of size **interval** (in seconds) after **now**. Midnight always starts a fresh interval, and if the last interval of a day would be truncated to less than half its size, it is rolled into the previous interval.

References tor_timegm().

Referenced by dirvote_recalculate_timing().

4.35.5.385 const cached_dir_t* dirvote_get_vote (const char * *fp*, int *flags*)

Return a given vote specified by **fp**. If **by_id**, return the vote for the authority with the v3 authority identity key digest **fp**; if **by_id** is false, return the vote whose digest is **fp**. If **fp** is NULL, return our own vote. If **include_previous** is false, do not consider any votes for a consensus that's already been built. If **include_pending** is false, do not consider any votes for the consensus that's in progress. May return NULL if we have no vote for the authority in question.

References authority_cert_t::cache_info, DIGEST_LEN, get_my_v3_authority_cert(), get_voter(), networkstatus_voter_info_t::identity_digest, and signed_descriptor_t::identity_digest.

Referenced by directory_handle_command_get(), and dirvote_fetch_missing_votes().

4.35.5.386 void dirvote_recalculate_timing (or_options_t * *options*, time_t *now*)

Set voting_schedule to hold the timing for the next vote we should be doing.

References authdir_mode_v3(), dirvote_get_start_of_next_interval(), networkstatus_t::dist_seconds, format_iso_time(), networkstatus_t::fresh_until, LD_DIR, networkstatus_get_live_consensus(), or_options_t::TestingV3AuthInitialDistDelay, or_options_t::TestingV3AuthInitialVoteDelay, or_options_t::TestingV3AuthInitialVotingInterval, tor_assert, networkstatus_t::valid_after, networkstatus_t::vote_seconds, and voting_schedule.

Referenced by dirvote_act(), networkstatus_set_current_consensus(), and options_act().

4.35.5.387 void disable_control_logging (void)

Called when we are sending a log message to the controllers: suspend sending further log messages to the controllers until we're done. Used by CONN_LOG_PROTECT.

References disable_log_messages.

4.35.5.388 void dns_cancel_pending_resolve (const char * *address*)

Mark all connections waiting for **address** for close. Then cancel the resolve for **address** itself, and remove any cached results for **address** from the cache.

References edge_connection_t::_base, assert_connection_ok(), CACHE_STATE_DONE, CACHE_STATE_PENDING, circuit_detach_stream(), circuit_get_by_edge_conn(), pending_connection_t::conn, connection_edge_end(), connection_free(), escaped_safe_str(), EXIT_CONN_STATE_RESOLVEFAILED, LD_BUG, connection_t::marked_for_close, pending_connection_t::next, cached_resolve_t::pending_connections, connection_t::s, connection_t::state, cached_resolve_t::state, TO_CONN, tor_assert, tor_fragile_assert, and tor_free.

Referenced by dns_resolve().

4.35.5.389 uint32_t dns_clip_ttl (uint32_t *ttl*)

Helper: Given a TTL from a DNS response, determine what TTL to give the OP that asked us to resolve it.

References MAX_DNS_TTL, and MIN_DNS_TTL.

Referenced by client_dns_set_addressmap_impl(), connection_edge_end(), connection_edge_finished_connecting(), connection_exit_connect(), send_resolved_cell(), and send_resolved_hostname_cell().

4.35.5.390 void dns_free_all (void)

Free all storage held in the DNS cache and related structures.

References _free_cached_resolve(), CACHE_STATE_DONE, resolv_conf_fname, smartlist_free(), and tor_free.

Referenced by tor_free_all().

4.35.5.391 int dns_init (void)

Initialize the DNS subsystem; called by the OR process.

References _dns_randfn(), configure_nameservers(), get_options(), and server_mode().

Referenced by do_main_loop(), and run_scheduled_events().

4.35.5.392 void dns_launch_correctness_checks (void)

If appropriate, start testing whether our DNS servers tend to lie to us.

References dns_launch_wildcard_checks(), get_options(), launch_test_addresses(), LD_BUG, tor_evtimer_new(), and tor_libevent_get_base().

Referenced by run_scheduled_events().

4.35.5.393 int dns_reset (void)

Called when DNS-related options change (or may have changed). Returns -1 on failure, 0 on success.

References configure_nameservers(), get_options(), LD_BUG, nameservers_configured, resolv_conf_fname, resolv_conf_mtime, server_mode(), the_evdns_base, tor_free, and tor_libevent_get_base().

Referenced by do_hup(), and options_act().

4.35.5.394 void dns_reset_correctness_checks (void)

Forget what we've previously learned about our DNS servers' correctness.

References _tor_free(), dns_is_completely_invalid, dns_wildcard_notice_given, dns_wildcard_one_notice_given, dns_wildcard_response_count, dns_wildcarded_test_address_notice_given, n_wildcard_requests, smartlist_clear(), strmap_free(), and tor_free.

Referenced by dns_servers_relaunch_checks().

4.35.5.395 int dns_resolve (edge_connection_t * *exitconn*)

See if we have a cache entry for **exitconn**->address. if so, if resolve valid, put it into **exitconn**->addr and return 1. If resolve failed, free exitconn and return -1.

(For EXIT_PURPOSE_RESOLVE connections, send back a RESOLVED error cell on returning -1. For EXIT_PURPOSE_CONNECT connections, there's no need to send back an END cell, since connection_exit_begin_conn will do that for us.)

If we have a cached answer, send the answer back along **exitconn**'s circuit.

Else, if seen before and pending, add conn to the pending list, and return 0.

Else, if not seen before, add conn to pending list, hand to dns farm, and return 0.

Exitconn's on_circuit field must be set, but exitconn should not yet be linked onto the n_streams/resolving_streams list of that circuit. On success, link the connection to n_streams if it's an exit connection. On "pending", link the connection to resolving streams. Otherwise, clear its on_circuit field.

References edge_connection_t::_base, connection_t::address, connection_free(), dns_cancel_pending_resolve(), dns_resolve_impl(), EXIT_CONN_STATE_RESOLVING, EXIT_PURPOSE_RESOLVE, connection_t::marked_for_close, or_circuit_t::n_streams, edge_connection_t::next_stream, edge_connection_t::on_circuit, connection_t::purpose, or_circuit_t::resolving_streams, send_resolved_cell(), send_resolved_hostname_cell(), connection_t::state, TO_CONN, TO_OR_CIRCUIT(), tor_assert, and tor_free.

Referenced by connection_exit_begin_conn(), and connection_exit_begin_resolve().

4.35.5.396 int dns_seems_to_be_broken (void)

Return true iff our DNS servers lie to us too much to be trusted.

References dns_is_completely_invalid.

Referenced by router_dump_router_to_string().

4.35.5.397 void dns_servers_relaunch_checks (void)

Forget what we've learned about the correctness of our DNS servers, and start learning again.

References dns_reset_correctness_checks(), get_options(), server_mode(), and time_to_check_for_correct_dns.

Referenced by configure_nameservers(), and ip_address_changed().

4.35.5.398 void dnsserv_close_listener (connection_t * *conn*)

Free the evdns server port for **conn**, which must be an AP_DNS_LISTENER.

References CONN_TYPE_AP_DNS_LISTENER, connection_t::dns_server_port, tor_assert, and connection_t::type.

Referenced by connection_unregister_events().

4.35.5.399 void dnsserv_configure_listener (connection_t * *conn*)

Set up the evdns server port for the UDP socket on **conn**, which must be an AP_DNS_LISTENER

References CONN_TYPE_AP_DNS_LISTENER, connection_t::dns_server_port, evdns_server_callback(), connection_t::s, tor_assert, and connection_t::type.

Referenced by connection_create_listener().

4.35.5.400 int dnsserv_launch_request (const char ∗ *name*, int *reverse*)

Helper function: called whenever the client sends a resolve request to our controller. We need to eventually answer the request **req**. Returns 0 if the controller will be getting (or has gotten) an event in response; -1 if we couldn't launch the request.

References edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_RESOLVE_WAIT, socks_request_t::command, CONN_TYPE_AP, connection_add(), connection_ap_handshake_rewrite_and_attach(), connection_free(), edge_connection_new(), escaped_safe_str_client(), edge_connection_t::is_dns_request, LD_APP, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, edge_connection_t::socks_request, connection_t::state, TO_CONN, and tor_free.

Referenced by handle_control_resolve().

4.35.5.401 void dnsserv_reject_request (edge_connection_t ∗ *conn*)

If there is a pending request on **conn** that's waiting for an answer, send back an error and free the request.

References edge_connection_t::dns_server_request.

Referenced by connection_about_to_close_connection().

4.35.5.402 void dnsserv_resolved (edge_connection_t ∗ *conn*, int *answer_type*, size_t *answer_len*, const char ∗ *answer*, int *ttl*)

Tell the dns request waiting for an answer on **conn** that we have an answer of type **answer_type** (RESOLVE_TYPE_IPV4/IPV6/ERR), of length **answer_len**, in **answer**, with TTL **ttl**. Doesn't do any caching; that's handled elsewhere.

References socks_request_t::address, socks_request_t::command, edge_connection_t::dns_server_request, evdns_get_orig_address(), LD_APP, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, edge_connection_t::socks_request, and tor_free.

Referenced by connection_ap_handshake_socks_resolved().

4.35.5.403 document_signature_t∗ document_signature_dup (const document_signature_t ∗ *sig*)

Return a newly allocated copy of **sig**

References document_signature_t::signature, and document_signature_t::signature_len.

Referenced by networkstatus_add_detached_signatures().

4.35.5.404 void document_signature_free (document_signature_t ∗ *sig*)

Free all storage held in **sig**

References document_signature_t::signature, and tor_free.

Referenced by networkstatus_add_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), and ns_detached_signatures_free().

4.35.5.405 time_t download_status_increment_failure (download_status_t * *dls*, int *status_code*, const char * *item*, int *server*, time_t *now*)

Called when an attempt to download **dls** has failed with HTTP status **status_code**. Increment the failure count (if the code indicates a real failure) and set **dls**->next_attempt_at to an appropriate time in the future.

References find_dl_schedule_and_len(), IMPOSSIBLE_TO_DOWNLOAD, LD_DIR, download_status_t::n_download_failures, download_status_t::next_attempt_at, and tor_assert.

Referenced by dir_routerdesc_download_failed().

4.35.5.406 static INLINE int download_status_is_ready (download_status_t * *dls*, time_t *now*, int *max_failures*) `[static]`

Return true iff, as of **now**, the resource tracked by **dls** is ready to get its download reattempted.

References download_status_t::n_download_failures, and download_status_t::next_attempt_at.

Referenced by authority_certs_fetch_missing(), fetch_bridge_descriptors(), update_consensus_networkstatus_downloads(), and update_extrainfo_downloads().

4.35.5.407 static INLINE void download_status_mark_impossible (download_status_t * *dl*) `[static]`

Mark **dl** as never downloadable.

References IMPOSSIBLE_TO_DOWNLOAD, and download_status_t::n_download_failures.

Referenced by router_load_routers_from_string().

4.35.5.408 void download_status_reset (download_status_t * *dls*)

Reset **dls** so that it will be considered downloadable immediately, and/or to show that we don't need it anymore.

(We find the zeroth element of the download schedule, and set next_attempt_at to be the appropriate offset from 'now'. In most cases this means setting it to 'now', so the item will be immediately downloadable; in the case of bridge descriptors, the zeroth element is an hour from now.)

References find_dl_schedule_and_len(), get_options(), download_status_t::n_download_failures, and download_status_t::next_attempt_at.

Referenced by authority_certs_fetch_missing(), learned_bridge_descriptor(), networkstatus_set_current_consensus(), router_reset_descriptor_download_failures(), and router_set_networkstatus_v2().

4.35.5.409 void dump_cell_pool_usage (int *severity*)

Log current statistics for cell pool allocation at log level **severity**.

References _circuit_get_global_list(), CIRCUIT_IS_ORIGIN, LD_MM, cell_queue_t::n, circuit_t::n_conn_cells, circuit_t::next, or_circuit_t::p_conn_cells, and TO_OR_-

CIRCUIT().

Referenced by dumpmemusage().

4.35.5.410 void dump_distinct_digest_count (int *severity*)

Log the total count of the number of distinct router digests we've ever verified. When compared to the number of times we've verified routerdesc signatures *in toto*, this will tell us if we're doing too much multiple-verification.

References digestmap_new(), digestmap_size(), and LD_GENERAL.

Referenced by dumpstats().

4.35.5.411 void dump_dns_mem_usage (int *severity*)

Log memory information about our internal DNS cache at level 'severity'.

References dns_cache_entry_count(), and LD_MM.

Referenced by dumpmemusage().

4.35.5.412 void dump_pk_ops (int *severity*)

Log the number of times we've done each public/private-key operation.

References LD_HIST, and pk_op_counts.

Referenced by dumpstats().

4.35.5.413 void dump_routerlist_mem_usage (int *severity*)

Log information about how much memory is being used for routerlist, at log level **severity**.

References LD_DIR, routerlist_t::old_routers, routerlist, and routerlist_t::routers.

Referenced by dumpmemusage().

4.35.5.414 void dup_onion_keys (crypto_pk_env_t ∗∗ *key*, crypto_pk_env_t ∗∗ *last*)

Store a full copy of the current onion key into ∗**key**, and a full copy of the most recent onion key into ∗**last**.

References crypto_pk_copy_full(), tor_assert, tor_mutex_acquire(), and tor_mutex_release().

Referenced by cpuworker_main().

4.35.5.415 edge_connection_t∗ edge_connection_new (int *type*, int *socket_family*)

Allocate and return a new **edge_connection_t** (p. 90), initialized as by **connection_init()** (p. 420).

References CONN_TYPE_AP, CONN_TYPE_EXIT, connection_init(), edge_connection_t::socks_request, TO_CONN, and tor_assert.

Referenced by connection_ap_make_link(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_new(), dnsserv_launch_request(), and evdns_server_callback().

4.35.5.416 void enable_control_logging (void)

We're done sending a log message to the controllers: re-enable controller logging. Used by CONN_LOG_PROTECT.

References disable_log_messages, and tor_assert.

4.35.5.417 int entry_guard_register_connect_status (const char * *digest*, int *succeeded*, int *mark_relay_status*, time_t *now*)

Called when a connection to an OR with the identity digest **digest** is established (**succeeded**==1) or has failed (**succeeded**==0). If the OR is an entry, change that entry's up/down status. Return 0 normally, or -1 if we want to tear down the new connection.

If **mark_relay_status**, also call **router_set_status()** (p. 1004) on this relay.

XXX022 change succeeded and mark_relay_status into 'int flags'.

References DIGEST_LEN, and HEX_DIGEST_LEN.

Referenced by circuit_build_failed(), connection_about_to_close_connection(), connection_or_check_valid_tls_handshake(), connection_or_connect(), and connection_or_set_state_open().

4.35.5.418 void entry_guards_compute_status (void)

A new directory or router-status has arrived; update the down/listed status of the entry guards.

An entry is 'down' if the directory lists it as nonrunning. An entry is 'unlisted' if the directory doesn't include it.

Don't call this on startup; only on a fresh download. Otherwise we'll think that things are unlisted.

References digestmap_free(), digestmap_get(), digestmap_new(), digestmap_set(), entry_guard_set_status(), entry_guards_changed(), entry_is_live(), entry_nodes_should_be_added(), or_options_t::EntryNodes, get_options(), LD_CIRC, log_entry_guards(), num_live_entry_guards(), remove_dead_entry_guards(), router_get_by_digest(), and tor_assert.

Referenced by directory_info_has_arrived().

4.35.5.419 void entry_guards_free_all (void)

Release all storage held by the list of entry guards and related memory structs.

References bridge_list, clear_bridge_list(), entry_guard_free(), and smartlist_free().

Referenced by tor_free_all().

4.35.5.420 int entry_guards_parse_state (or_state_t * *state*, int *set*, char ** *msg*)

Parse **state** and learn about the entry guards it describes. If **set** is true, and there are no errors, replace the global entry_list with what we find. On success, return 0. On failure, alloc into ***msg** a string describing the error, and return -1.

References _tor_free(), entry_guard_t::bad_since, base16_decode(), crypto_rand_int(), DIGEST_LEN, digestmap_free(), digestmap_get(), digestmap_new(), digestmap_set(), entry_guard_free(), entry_guards_dirty, or_state_t::EntryGuards, escaped(), HEX_DIGEST_LEN, entry_guard_t::identity, is_legal_nickname(), config_line_t::key, entry_guard_t::last_attempted, LD_BUG, entry_guard_t::made_contact, MAX_NICKNAME_LEN, config_line_t::next, entry_guard_t::nickname, parse_iso_time(), remove_obsolete_entry_guards(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, or_state_t::TorVersion, entry_guard_t::unreachable_since, and config_line_t::value.

Referenced by or_state_set(), and or_state_validate().

4.35.5.421 void entry_guards_update_state (or_state_t * *state*)

If the entry guard info has not changed, do nothing and return. Otherwise, free the EntryGuards piece of **state** and create a new one out of the global entry_guards list, and then mark **state** dirty so it will get saved to disk.

References or_options_t::AvoidDiskWrites, base16_encode(), config_free_lines(), DIGEST_LEN, entry_guards_dirty, or_state_t::EntryGuards, format_iso_time(), get_options(), get_or_state(), HEX_DIGEST_LEN, config_line_t::key, MAX_NICKNAME_LEN, config_line_t::next, or_state_mark_dirty(), smartlist_create(), tor_snprintf(), and config_line_t::value.

Referenced by or_state_save().

4.35.5.422 int entry_list_is_constrained (or_options_t * *options*)

Return 0 if we're fine adding arbitrary routers out of the directory to our entry guard list, or return 1 if we have a list already and we'd prefer to stick to it.

References or_options_t::EntryNodes, and or_options_t::UseBridges.

Referenced by choose_random_entry(), and connection_dir_request_failed().

4.35.5.423 void entry_nodes_should_be_added (void)

Called when the value of EntryNodes changes in our configuration.

References LD_CIRC.

Referenced by entry_guards_compute_status(), and options_act().

4.35.5.424 int errno_to_orconn_end_reason (int *e*)

Given an errno from a failed ORConn connection, return a reason code appropriate for use in the controller orconn events.

References LD_OR.

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), and connection_or_connect().

4.35.5.425 uint8_t errno_to_stream_end_reason (int *e*)

Given an errno from a failed exit connection, return a reason code appropriate for use in a RELAY END cell.

Referenced by connection_edge_end_errno().

4.35.5.426 const char∗ esc_router_info (routerinfo_t ∗ *router*)

Allocate and return a new string representing the contact info and platform string for **router**, surrounded by quotes and using standard C escapes.

THIS FUNCTION IS NOT REENTRANT. Don't call it from outside the main thread. Also, each call invalidates the last-returned value, so don't try log_warn(LD_GENERAL, "%s %s", esc_router_info(a), esc_router_info(b));

If **router** is NULL, it just frees its internal memory and returns.

References routerinfo_t::contact_info, esc_for_log(), routerinfo_t::platform, tor_free, and tor_snprintf().

Referenced by authdir_wants_to_reject_router(), and tor_free_all().

4.35.5.427 const char∗ escaped_safe_str (const char ∗ *address*)

Equivalent to escaped(safe_str(address)). See reentrancy note on **escaped()** (p. 1245): don't use this outside the main thread, or twice in the same log statement.

References escaped(), and get_options().

Referenced by connection_dns_remove(), connection_edge_finished_connecting(), connection_tls_start_handshake(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve_impl(), evdns_callback(), launch_resolve(), launch_test_addresses(), and purge_expired_resolves().

4.35.5.428 const char∗ escaped_safe_str_client (const char ∗ *address*)

Equivalent to escaped(safe_str_client(address)). See reentrancy note on **escaped()** (p. 1245): don't use this outside the main thread, or twice in the same log statement.

References escaped(), and get_options().

Referenced by addressmap_rewrite(), addressmap_rewrite_reverse(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), conn_close_if_marked(), connection_ap_handshake_rewrite_and_attach(), connection_connect(), connection_exit_connect(), directory_get_from_hs_dir(), dnsserv_launch_request(), evdns_server_callback(), rend_client_introduction_acked(), rend_client_remove_intro_point(), rend_client_send_introduction(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.35.5.429 int exit_policy_is_general_exit (smartlist_t ∗ *policy*)

Return true iff **ri** is "useful as an exit node", meaning it allows exit to at least one /8 address space for at least two of ports 80, 443, and 6667.

References exit_policy_is_general_exit_helper().

Referenced by dirserv_compute_performance_thresholds().

4.35.5.430 extend_info_t* extend_info_alloc (const char * *nickname*, const char * *digest*, crypto_pk_env_t * *onion_key*, const tor_addr_t * *addr*, uint16_t *port*)

Allocate a new extend_info object based on the various arguments.

References extend_info_t::addr, crypto_pk_dup_key(), DIGEST_LEN, extend_info_t::identity_digest, extend_info_t::nickname, extend_info_t::onion_key, extend_info_t::port, and tor_addr_copy().

Referenced by circuit_extend(), circuit_get_open_circ_or_launch(), and extend_info_from_router().

4.35.5.431 extend_info_t* extend_info_dup (extend_info_t * *info*)

Allocate and return a new **extend_info_t** (p. 97) with the same contents as **info**.

References crypto_pk_dup_key(), extend_info_t::onion_key, and tor_assert.

Referenced by circuit_append_new_exit(), circuit_handle_first_hop(), onion_append_hop(), onion_extend_cpath(), onion_pick_cpath_exit(), rend_client_get_random_intro(), rend_service_launch_establish_intro(), and rend_service_update_descriptor().

4.35.5.432 void extend_info_free (extend_info_t * *info*)

Release storage held by an **extend_info_t** (p. 97) struct.

References crypto_free_pk_env(), extend_info_t::onion_key, and tor_free.

Referenced by circuit_append_new_exit(), circuit_free(), circuit_free_cpath_node(), circuit_get_open_circ_or_launch(), circuit_launch_by_router(), circuit_n_conn_done(), handle_control_extendcircuit(), onion_extend_cpath(), rend_client_get_random_intro(), rend_client_introduction_acked(), rend_intro_point_free(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.35.5.433 extend_info_t* extend_info_from_router (routerinfo_t * *r*)

Allocate and return a new **extend_info_t** (p. 97) that can be used to build a circuit to or through the router **r**.

References routerinfo_t::addr, routerinfo_t::cache_info, extend_info_alloc(), signed_descriptor_t::identity_digest, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::or_port, tor_addr_from_ipv4h, and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), circuit_launch_by_router(), handle_control_extendcircuit(), onion_extend_cpath(), onion_pick_cpath_exit(), rend_client_get_random_intro(), and rend_service_introduce().

4.35.5.434 int extrainfo_dump_to_string (char * *s*, size_t *maxlen*, extrainfo_t * *extrainfo*, crypto_pk_env_t * *ident_key*)

Write the contents of **extrainfo** to the **maxlen**-byte string **s**, signing them with **ident_key**. Return 0 on success, negative on failure.

References base16_encode(), extrainfo_t::cache_info, or_options_t::CellStatistics, DIGEST_LEN, or_options_t::DirReqStatistics, or_options_t::EntryStatistics, or_options_-

t::ExitPortStatistics, extrainfo_dump_to_string(), extrainfo_free(), extrainfo_parse_entry_from_string(), or_options_t::ExtraInfoStatistics, format_iso_time(), geoip_get_bridge_stats_extrainfo(), get_options(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, LD_BUG, LD_DIR, LD_GENERAL, load_stats_file(), extrainfo_t::nickname, signed_descriptor_t::published_on, rep_hist_get_bandwidth_lines(), router_append_dirobj_signature(), router_get_extrainfo_hash(), should_record_bridge_info(), tor_free, and tor_snprintf().

Referenced by extrainfo_dump_to_string(), and router_rebuild_descriptor().

4.35.5.435 void extrainfo_free (extrainfo_t * *extrainfo*)

Release all storage held by **extrainfo**

References extrainfo_t::cache_info, extrainfo_t::pending_sig, signed_descriptor_t::signed_descriptor_body, and tor_free.

Referenced by _extrainfo_free(), dirserv_add_extrainfo(), extrainfo_dump_to_string(), extrainfo_insert(), extrainfo_parse_entry_from_string(), router_free_all(), router_rebuild_descriptor(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

4.35.5.436 signed_descriptor_t* extrainfo_get_by_descriptor_digest (const char * *digest*)

Return the signed descriptor for the **extrainfo_t** (p. 99) in our routerlist whose extra-info-digest is **digest**. Return NULL if no such extra-info document is known.

References extrainfo_t::cache_info, routerlist_t::extra_info_map, routerlist, and tor_assert.

Referenced by connection_dirserv_add_servers_to_outbuf(), dirserv_get_routerdesc_fingerprints(), dirserv_have_any_serverdesc(), and get_signed_descriptor_by_fp().

4.35.5.437 extrainfo_t* extrainfo_parse_entry_from_string (const char * *s*, const char * *end*, int *cache_copy*, struct digest_ri_map_t * *routermap*)

Parse a single extrainfo entry from the string **s**, ending at **end**. (If **end** is NULL, parse up to the end of **s**.) If **cache_copy** is true, make a copy of the extra-info document in the cache_info fields of the result. If **routermap** is provided, use it as a map from router identity to **routerinfo_t** (p. 206) when looking up signing keys.

References directory_token_t::args, base16_decode(), routerinfo_t::cache_info, extrainfo_t::cache_info, check_signature_token(), DIGEST_LEN, digestmap_get(), dump_desc(), escaped(), extrainfo_free(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, signed_descriptor_t::is_extrainfo, is_legal_nickname(), LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, extrainfo_t::nickname, note_crypto_pk_op(), directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, parse_iso_time(), extrainfo_t::pending_sig, extrainfo_t::pending_sig_len, signed_descriptor_t::published_on, router_get_extrainfo_hash(), signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, and directory_token_t::tp.

Referenced by extrainfo_dump_to_string(), and router_parse_list_from_string().

4.35.5.438 int fascist_firewall_allows_address_dir (const tor_addr_t ∗ *addr*, uint16_t *port*)

Return true iff we think our firewall will let us make a directory connection to addr:port.

References addr_policy_permits_tor_addr().

Referenced by directory_post_to_dirservers(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.35.5.439 int fascist_firewall_allows_address_or (const tor_addr_t ∗ *addr*, uint16_t *port*)

Return true iff we think our firewall will let us make an OR connection to addr:port.

References addr_policy_permits_tor_addr().

Referenced by directory_command_should_use_begindir(), fascist_firewall_allows_or(), fetch_bridge_descriptors(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.35.5.440 int fascist_firewall_allows_or (routerinfo_t ∗ *ri*)

Return true iff we think our firewall will let us make an OR connection to **ri**.

References routerinfo_t::addr, fascist_firewall_allows_address_or(), routerinfo_t::or_port, and tor_addr_from_ipv4h.

Referenced by choose_good_entry_server(), and entry_is_live().

4.35.5.441 int fast_client_handshake (const char ∗ *handshake_state*, const char ∗ *handshake_reply_out*, char ∗ *key_out*, size_t *key_out_len*)

Implement the second half of the client side of the CREATE_FAST handshake. We sent the server **handshake_state** ("x") already, and the server told us **handshake_reply_out** (y|H(x|y)). Make sure that the hash is correct, and generate key material in **key_out**. Return 0 on success, true on failure.

NOTE: The "CREATE_FAST" handshake path is distinguishable from regular "onionskin" handshakes, and is not secure if an adversary can see or modify the messages. Therefore, it should only be used by clients, and only as the first hop of a circuit (since the first hop is already authenticated and protected by TLS).

References crypto_expand_key_material(), DIGEST_LEN, LD_PROTOCOL, and tor_free.

Referenced by circuit_finish_handshake().

4.35.5.442 int fast_server_handshake (const char ∗ *key_in*, char ∗ *handshake_reply_out*, char ∗ *key_out*, size_t *key_out_len*)

Implement the server side of the CREATE_FAST abbreviated handshake. The client has provided DIGEST_LEN key bytes in **key_in** ("x"). We generate a reply of DIGEST_LEN∗2 bytes in **key_out**, consisting of a new random "y", followed by H(x|y) to check for correctness. We set **key_out_len** bytes of key material in **key_out**. Return 0 on success, <0 on failure.

References crypto_expand_key_material(), crypto_rand(), DIGEST_LEN, and tor_free.

Referenced by command_process_create_cell().

4.35.5.443 void fetch_bridge_descriptors (time_t *now*)

For each bridge in our list for which we don't currently have a descriptor, fetch a new copy of its descriptor – either directly from the bridge or via a bridge authority.

References base16_encode(), BRIDGE_AUTHORITY, DIGEST_LEN, DIR_PURPOSE_FETCH_SERVERDESC, directory_get_from_dirserver(), download_status_failed, download_status_is_ready(), fascist_firewall_allows_address_or(), fmt_addr(), get_n_authorities(), get_options(), HEX_DIGEST_LEN, IMPOSSIBLE_TO_DOWNLOAD, launch_direct_bridge_descriptor_fetch(), LD_DIR, ROUTER_PURPOSE_BRIDGE, tor_digest_is_zero(), and or_options_t::UpdateBridgesFromAuthority.

Referenced by run_scheduled_events().

4.35.5.444 int fetch_from_buf (char ∗ *string*, size_t *string_len*, buf_t ∗ *buf*)

Remove **string_len** bytes from the front of **buf**, and store them into **string**. Return the new buffer size. **string_len** must be <= the number of bytes on the buffer.

References buf_remove_from_front(), buf_t::datalen, peek_from_buf(), and tor_assert.

Referenced by connection_fetch_from_buf(), fetch_from_buf_http(), fetch_from_buf_line(), and move_buf_to_buf().

4.35.5.445 int fetch_from_buf_http (buf_t ∗ *buf*, char ∗∗ *headers_out*, size_t *max_headerlen*, char ∗∗ *body_out*, size_t ∗ *body_used*, size_t *max_bodylen*, int *force_complete*)

There is a (possibly incomplete) http statement on **buf**, of the form "\%s\\r\\n\\r\\n\%s", headers, body. (body may contain NULs.) If a) the headers include a Content-Length field and all bytes in the body are present, or b) there's no Content-Length field and all headers are present, then:

- strdup headers into ∗**headers_out**, and NUL-terminate it.
- memdup body into ∗**body_out**, and NUL-terminate it.
- Then remove them from **buf**, and return 1.

- If headers or body is NULL, discard that part of the buf.
- If a headers or body doesn't fit in the arg, return -1. (We ensure that the headers or body don't exceed max len, _even if_ we're planning to discard them.)
- If force_complete is true, then succeed even if not all of the content has arrived.

Else, change nothing and return 0.

References buf_find_string_offset(), buf_pullup(), chunk_t::data, buf_t::datalen, chunk_t::datalen, fetch_from_buf(), buf_t::head, LD_HTTP, LD_PROTOCOL, and tor_assert.

Referenced by connection_dir_client_reached_eof(), connection_read_https_proxy_response(), and directory_handle_command().

4.35.5.446 int fetch_from_buf_line (buf_t * *buf*, char * *data_out*, size_t * *data_len*)

Try to read a single LF-terminated line from **buf**, and write it, NUL-terminated, into the ***data_len** byte buffer at **data_out**. Set ***data_len** to the number of bytes in the line, not counting the terminating NUL. Return 1 if we read a whole line, return 0 if we don't have a whole line yet, and return -1 if the line length exceeds ***data_len**.

References buf_find_offset_of_char(), fetch_from_buf(), and buf_t::head.

Referenced by connection_ap_process_natd(), and connection_control_process_inbuf().

4.35.5.447 int fetch_from_buf_socks (buf_t * *buf*, socks_request_t * *req*, int *log_sockstype*, int *safe_socks*)

There is a (possibly incomplete) socks handshake on **buf**, of one of the forms

- socks4: "socksheader username\\0"
- socks4a: "socksheader username\\0 destaddr\\0"
- socks5 phase one: "version #methods methods"
- socks5 phase two: "version command 0 addresstype..." If it's a complete and valid handshake, and destaddr fits in MAX_SOCKS_ADDR_LEN bytes, then pull the handshake off the buf, assign to **req**, and return 1.

If it's invalid or too big, return -1.

Else it's not all there yet, leave buf alone and return 0.

If you want to specify the socks reply, write it into **req->reply** and set **req->replylen**, else leave **req->replylen** alone.

If **log_sockstype** is non-zero, then do a notice-level log of whether the connection is possibly leaking DNS requests locally or not.

If **safe_socks** is true, then reject unsafe socks protocols.

If returning 0 or -1, **req->address** and **req->port** are undefined.

References socks_request_t::address, addressmap_have_mapping(), buf_clear(), buf_pullup(), buf_remove_from_front(), CHUNK_WRITE_PTR(), socks_request_t::command, control_event_client_status(), chunk_t::data, chunk_t::datalen, buf_t::datalen, escaped(), get_uint16(), get_uint32(), buf_t::head, LD_APP, LD_PROTOCOL, socks_request_t::port, socks_request_t::reply, socks_request_t::replylen, safe_str_client(), SOCKS4_NETWORK_LEN, SOCKS_COMMAND_CONNECT, SOCKS_COMMAND_RESOLVE, SOCKS_COMMAND_RESOLVE_PTR, socks_request_t::socks_version, tor_addr_from_ipv4n(), tor_addr_from_ipv6_bytes(), tor_addr_to_str(), tor_assert, tor_free, tor_inet_ntoa(), and tor_strisprint().

Referenced by connection_ap_handshake_process_socks().

4.35.5.448 int fetch_from_buf_socks_client (buf_t * *buf*, int *state*, char ** *reason*)

Inspect a reply from SOCKS server stored in **buf** according to **state**, removing the protocol data upon success. Return 0 on incomplete response, 1 on success and -1 on error, in which case **reason** is set to a descriptive message (free() when finished with it).

As a special case, 2 is returned when user/pass is required during SOCKS5 handshake and user/pass is configured.

References buf_clear(), buf_pullup(), buf_remove_from_front(), chunk_t::data, chunk_t::datalen, buf_t::datalen, buf_t::head, LD_NET, socks4_response_code_to_string(), socks5_response_code_to_string(), and tor_assert.

Referenced by connection_read_proxy_handshake().

4.35.5.449 int fetch_var_cell_from_buf (buf_t * *buf*, var_cell_t ** *out*, int *linkproto*)

Check **buf** for a variable-length cell according to the rules of link protocol version **linkproto**. If one is found, pull it off the buffer and assign a newly allocated **var_cell_t** (p. 250) to ∗**out**, and return 1. Return 0 if whatever is on the start of **buf_t** (p. 18) is not a variable-length cell. Return 1 and set ∗**out** to NULL if there seems to be the start of a variable-length cell on **buf**, but the whole thing isn't there yet.

References buf_remove_from_front(), CELL_COMMAND_IS_VAR_LENGTH, var_cell_t::circ_id, var_cell_t::command, buf_t::datalen, get_uint16(), var_cell_t::payload, peek_from_buf(), VAR_CELL_HEADER_SIZE, and var_cell_new().

Referenced by connection_fetch_var_cell_from_buf().

4.35.5.450 int firewall_is_fascist_or (void)

Return true iff the firewall options might block any address:port combination.

Referenced by choose_good_entry_server().

4.35.5.451 int flush_buf (int *s*, buf_t * *buf*, size_t *sz*, size_t * *buf_flushlen*)

Write data from **buf** to the socket **s**. Write at most **sz** bytes, decrement ∗**buf_flushlen** by the number of bytes actually written, and remove the written bytes from the buffer. Return the number of bytes written on success, -1 on failure. Return 0 if write() would block.

References chunk_t::datalen, flush_chunk(), buf_t::head, and tor_assert.

Referenced by conn_close_if_marked(), and connection_handle_write_impl().

4.35.5.452 int flush_buf_tls (tor_tls_t * *tls*, buf_t * *buf*, size_t *flushlen*, size_t * *buf_flushlen*)

As **flush_buf()** (p. 293), but writes data to a TLS connection. Can write more than **flushlen** bytes.

References chunk_t::datalen, flush_chunk_tls(), buf_t::head, and tor_assert.

Referenced by conn_close_if_marked(), and connection_handle_write_impl().

4.35.5.453 void free_cell_pool (void)

Free all storage used to hold cells (and insertion times if we measure cell statistics).

Referenced by tor_free_all().

4.35.5.454 void geoip_bridge_stats_init (time_t *now*)

Initialize bridge stats.

Referenced by run_scheduled_events().

4.35.5.455 time_t geoip_bridge_stats_write (time_t *now*)

Write bridge statistics to $DATADIR/stats/bridge-stats and return when we should next try to write statistics.

References check_private_dir(), control_event_clients_seen(), DIR_ENTRY_RECORD_USAGE_RETAIN_IPS, format_iso_time(), GEOIP_CLIENT_CONNECT, geoip_get_client_history_bridge(), geoip_remove_old_clients(), get_datadir_fname, get_datadir_fname2, parse_bridge_stats_controller(), tor_free, tor_snprintf(), and write_str_to_file().

Referenced by run_scheduled_events().

4.35.5.456 void geoip_change_dirreq_state (uint64_t *dirreq_id*, dirreq_type_t *type*, dirreq_state_t *new_state*)

Change the state of the either direct or tunneled (see **type**) directory request with **dirreq_id** to **new_state** and possibly mark it as completed. If no entry can be found for the given key parts (e.g., if this is a directory request that we are not measuring, or one that was started in the previous measurement period), or if the state cannot be advanced to **new_state**, do nothing.

References _dirreq_map_get(), DIRREQ_FLUSHING_DIR_CONN_FINISHED, DIRREQ_IS_FOR_NETWORK_STATUS, DIRREQ_OR_CONN_BUFFER_FLUSHED, get_options(), and tor_gettimeofday().

Referenced by connection_dir_finished_flushing(), connection_handle_write_impl(), connection_or_flush_from_first_active_circuit(), and relay_send_command_from_edge().

4.35.5.457 void geoip_dirreq_stats_init (time_t *now*)

Initialize directory request stats.

Referenced by run_scheduled_events().

4.35.5.458 void geoip_dirreq_stats_write (time_t *now*)

Write dirreq statistics to $DATADIR/stats/dirreq-stats.

References abort_writing_to_file(), check_private_dir(), finish_writing_to_file(), format_iso_time(), GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, geoip_get_client_history_dirreq(), geoip_get_dirreq_history(), geoip_get_mean_shares(), geoip_get_request_history(), GEOIP_REJECT_BUSY, GEOIP_REJECT_NOT_ENOUGH_SIGS, GEOIP_REJECT_NOT_FOUND, GEOIP_REJECT_NOT_MODIFIED, GEOIP_REJECT_UNAVAILABLE, geoip_remove_old_clients(), GEOIP_SUCCESS, get_datadir_fname, get_datadir_fname2, get_options(), round_uint32_to_next_multiple_of(), start_writing_to_stdio_file(), and tor_free.

Referenced by run_scheduled_events().

4.35.5.459 void geoip_entry_stats_init (time_t *now*)

Initialize entry stats.

Referenced by run_scheduled_events().

4.35.5.460 void geoip_entry_stats_write (time_t *now*)

Write entry statistics to $DATADIR/stats/entry-stats.

References abort_writing_to_file(), check_private_dir(), finish_writing_to_file(), format_iso_time(), GEOIP_CLIENT_CONNECT, geoip_get_client_history_dirreq(), geoip_remove_old_clients(), get_datadir_fname, get_datadir_fname2, get_options(), start_writing_to_stdio_file(), and tor_free.

Referenced by run_scheduled_events().

4.35.5.461 void geoip_free_all (void)

Release all storage held in this file.

References clear_geoip_db(), and tor_free.

Referenced by tor_free_all().

4.35.5.462 const char∗ geoip_get_bridge_stats_controller (time_t *now*)

Return most recent bridge statistics to be returned to controller clients, or NULL if we don't have recent bridge statistics.

References load_bridge_stats().

Referenced by getinfo_helper_events().

4.35.5.463 const char∗ geoip_get_bridge_stats_extrainfo (time_t *now*)

Return most recent bridge statistics for inclusion in extra-info descriptors, or NULL if we don't have recent bridge statistics.

References load_bridge_stats().

Referenced by extrainfo_dump_to_string().

4.35.5.464 char∗ geoip_get_client_history_bridge (time_t *now*, geoip_client_action_t *action*)

Return a newly allocated comma-separated string containing entries for all the countries from which we've seen enough clients connect as a bridge. The entry format is cc=num where num is the number of IPs we've seen connecting from that country, and cc is a lowercased country code. Returns NULL if we don't want to export geoip data yet.

References geoip_get_client_history(), GEOIP_MIN_OBSERVATION_TIME, and IP_GRANULARITY.

Referenced by geoip_bridge_stats_write().

4.35.5.465 char∗ geoip_get_client_history_dirreq (time_t *now*, geoip_client_action_t *action*)

Return a newly allocated comma-separated string containing entries for all the countries from which we've seen enough clients connect as a directory. The entry format is cc=num where num is the number of IPs we've seen connecting from that country, and cc is a lowercased country code. Returns NULL if we don't want to export geoip data yet.

References DIR_RECORD_USAGE_GRANULARITY, DIR_RECORD_USAGE_MIN_OBSERVATION_TIME, and geoip_get_client_history().

Referenced by geoip_dirreq_stats_write(), and geoip_entry_stats_write().

4.35.5.466 country_t geoip_get_country (const char ∗ *country*)

Return the index of the **country**'s entry in the GeoIP DB if it is a valid 2-letter country code, otherwise return -1.

References country_idxplus1_by_lc_code, and strmap_get_lc().

Referenced by routerset_refresh_countries().

4.35.5.467 int geoip_get_country_by_ip (uint32_t *ipaddr*)

Given an IP address in host order, return a number representing the country to which that address belongs, or -1 for unknown. The return value will always be less than **geoip_get_n_countries()** (p. 653). To decode it, call **geoip_get_country_name()** (p. 653).

References _geoip_compare_key_to_entry(), geoip_entry_t::country, and smartlist_bsearch().

Referenced by geoip_get_client_history(), geoip_note_client_seen(), getinfo_helper_geoip(), routerinfo_set_country(), and routerset_contains().

4.35.5.468 const char∗ geoip_get_country_name (country_t *num*)

Return the two-letter country code associated with the number **num**, or "??" for an unknown value.

References geoip_country_t::countrycode.

Referenced by geoip_get_client_history(), and getinfo_helper_geoip().

4.35.5.469 time_t geoip_get_history_start (void)

Return the time at which we started recording geoip data.

4.35.5.470 int geoip_get_n_countries (void)

Return the number of countries recognized by the GeoIP database.

Referenced by geoip_get_client_history(), and routerset_refresh_countries().

4.35.5.471 char∗ geoip_get_request_history (time_t *now*, geoip_client_action_t *action*)

Return a newly allocated string holding the per-country request history for **action** in a format suitable for an extra-info document, or NULL on failure.

References _c_hist_compare(), GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, GEOIP_MIN_OBSERVATION_TIME, IP_GRANULARITY, REQUEST_HIST_LEN, round_to_next_multiple_of(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), tor_asprintf(), and tor_free.

Referenced by geoip_dirreq_stats_write().

4.35.5.472 int geoip_is_loaded (void)

Return true iff we have loaded a GeoIP database.

Referenced by geoip_get_client_history(), getinfo_helper_geoip(), options_act(), and routerset_refresh_countries().

4.35.5.473 int geoip_load_file (const char ∗ *filename*, or_options_t ∗ *options*)

Clear the GeoIP database and reload it from the file **filename**. Return 0 on success, -1 on failure.

Recognized line formats are: INTIPLOW,INTIPHIGH,CC and "INTIPLOW","INTIPHIGH","CC","CC3","COUNTRY NAME" where INTIPLOW and INTIPHIGH are IPv4 addresses encoded as 4-byte unsigned integers, and CC is a country code.

It also recognizes, and skips over, blank lines and lines that start with '#' (comments).

References _geoip_compare_entries(), clear_geoip_db(), country_idxplus1_by_lc_code, geoip_country_t::countrycode, geoip_parse_entry(), LD_GENERAL, options_need_geoip_info(), refresh_all_country_info(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_sort(), and tor_free.

Referenced by options_act().

4.35.5.474 void geoip_note_client_seen (geoip_client_action_t *action*, uint32_t *addr*, time_t *now*)

Note that we've seen a client connect from the IP **addr** (host order) at time **now**. Ignored by all but bridges and directories if configured accordingly.

unresolved requests are stored at index 0.

References or_options_t::BridgeAuthoritativeDir, or_options_t::BridgeRecordUsageByCountry, or_options_t::BridgeRelay, or_options_t::DirReqStatistics, or_options_t::EntryStatistics, GEOIP_CLIENT_CONNECT, GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, geoip_determine_shares(), geoip_get_country_by_ip(), geoip_remove_old_clients(), get_options(), last_time_determined_shares, geoip_country_t::n_v2_ns_requests, geoip_country_t::n_v3_ns_requests, REQUEST_HIST_LEN, REQUEST_HIST_PERIOD, REQUEST_SHARE_INTERVAL, and smartlist_create().

Referenced by connection_or_set_state_open(), and directory_handle_command_get().

4.35.5.475 void geoip_note_ns_response (geoip_client_action_t *action*, geoip_ns_response_t *response*)

Note that we've rejected a client's request for a v2 or v3 network status, encoded in **action** for reason **reason** at time **now**.

References GEOIP_CLIENT_NETWORKSTATUS, GEOIP_CLIENT_NETWORKSTATUS_V2, get_options(), and tor_assert.

Referenced by directory_handle_command_get().

4.35.5.476 void geoip_remove_old_clients (time_t *cutoff*)

Forget about all clients that haven't connected since **cutoff**. If **cutoff** is in the future, clients won't be added to the history until this time is reached. This is useful to prevent relays that switch to bridges from reporting unbelievable numbers of clients.

References _remove_old_client_helper().

Referenced by geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), geoip_note_client_seen(), and options_act().

4.35.5.477 void geoip_start_dirreq (uint64_t *dirreq_id*, size_t *response_size*, geoip_client_action_t *action*, dirreq_type_t *type*)

Note that an either direct or tunneled (see **type**) directory request for a network status with unique ID **dirreq_id** of size **response_size** and action **action** (either v2 or v3) has started.

References _dirreq_map_put(), get_options(), and tor_gettimeofday().

Referenced by directory_handle_command_get().

4.35.5.478 smartlist_t* get_connection_array (void)

Set *****array** to an array of all connections, and *****n** to the length of the array. *****array** and *****n** must not be modified.

References smartlist_create().

Referenced by any_pending_bridge_descriptor_fetches(), choose_good_exit_server_general(), circuit_discard_optional_exit_enclaves(), connection_ap_attach_pending(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_bucket_refill(), connection_dump_buffer_mem_stats(), connection_expire_held_open(), connection_free_all(), connection_get_by_global_id(), connection_get_by_type(), connection_get_by_type_addr_port_purpose(), connection_get_by_type_purpose(), connection_get_by_type_state(), connection_get_by_type_state_rendquery(), connection_or_clear_identity_map(), control_event_stream_bandwidth_used(), control_update_global_event_mask(), cull_wedged_cpuworkers(), getinfo_helper_events(), list_pending_downloads(), rend_client_desc_trynow(), retry_listeners(), and send_control_event_string().

4.35.5.479 const char* get_dir_info_status_string (void)

Return a string describing what we're missing before we have enough directory info.

Referenced by directory_info_has_arrived().

4.35.5.480 const char* get_dirportfrontpage (void)

Return the contents of our frontpage string, or NULL if not configured.

Referenced by directory_handle_command_get().

4.35.5.481 uint32_t get_effective_bwburst (or_options_t * *options*)

Return the bandwidthburst that we are going to report to the authorities based on the config options.

References or_options_t::BandwidthBurst, and or_options_t::RelayBandwidthBurst.

Referenced by options_transition_affects_descriptor(), and router_rebuild_descriptor().

4.35.5.482 uint32_t get_effective_bwrate (or_options_t * *options*)

Return the bandwidthrate that we are going to report to the authorities based on the config options.

References or_options_t::BandwidthRate, or_options_t::MaxAdvertisedBandwidth, and or_options_t::RelayBandwidthRate.

Referenced by options_transition_affects_descriptor(), and router_rebuild_descriptor().

4.35.5.483 crypto_pk_env_t* get_identity_key (void)

Returns the current identity key; requires that the identity key has been set.

References tor_assert.

Referenced by accounting_set_wakeup_time(), connection_or_check_valid_tls_handshake(), dirserv_regenerate_directory(), do_list_fingerprint(), generate_runningrouters(), generate_v2_networkstatus_opinion(), init_keys(), router_rebuild_descriptor(), and run_scheduled_events().

4.35.5.484 microdesc_cache_t* get_microdesc_cache (void)

Return a pointer to the microdescriptor cache, loading it if necessary.

References get_datadir_fname.

4.35.5.485 authority_cert_t* get_my_v3_authority_cert (void)

Return the key certificate for this v3 (voting) authority, or NULL if we have no such certificate.

Referenced by authority_cert_get_by_sk_digest(), directory_handle_command_get(), dirvote_act(), dirvote_compute_consensuses(), dirvote_get_vote(), dirvote_perform_vote(), and init_keys().

4.35.5.486 crypto_pk_env_t* get_my_v3_authority_signing_key (void)

Return the v3 signing key for this v3 (voting) authority, or NULL if we have no such key.

Referenced by dirvote_compute_consensuses(), and dirvote_perform_vote().

4.35.5.487 authority_cert_t∗ get_my_v3_legacy_cert (void)

If we're an authority, and we're using a legacy authority identity key for emergency migration purposes, return the certificate associated with that key.

Referenced by authority_cert_get_by_sk_digest(), and dirvote_compute_consensuses().

4.35.5.488 crypto_pk_env_t∗ get_my_v3_legacy_signing_key (void)

If we're an authority, and we're using a legacy authority identity key for emergency migration purposes, return that key.

Referenced by dirvote_compute_consensuses().

4.35.5.489 int get_n_authorities (authority_type_t *type*)

Return the number of directory authorities whose type matches some bit set in **type**

Referenced by any_trusted_dir_is_v1_authority(), dirvote_compute_consensuses(), fetch_bridge_descriptors(), getinfo_helper_events(), and networkstatus_check_consensus_signature().

4.35.5.490 crypto_pk_env_t∗ get_onion_key (void)

Return the current onion key. Requires that the onion key has been loaded or generated.

References tor_assert.

Referenced by router_rebuild_descriptor().

4.35.5.491 time_t get_onion_key_set_at (void)

Return the time when the onion key was last set. This is either the time when the process launched, or the time of the most recent key rotation since the process launched.

References onionkey_set_at.

Referenced by run_scheduled_events().

4.35.5.492 or_options_t∗ get_options (void)

Return the currently configured options.

References tor_assert.

Referenced by accounting_record_bandwidth_usage(), accounting_set_wakeup_time(), add_networkstatus_to_cache(), add_trusted_dir_server(), add_wildcarded_test_address(), address_is_invalid_destination(), any_bridge_descriptors_known(), authority_certs_fetch_missing(), cell_queue_append_packed_copy(), check_descriptor_ipaddress_changed(), check_whether_dirport_reachable(), check_whether_orport_reachable(), choose_good_entry_server(), choose_good_exit_server(), choose_good_exit_server_general(), choose_good_middle_server(), choose_random_entry(), circuit_all_predicted_ports_handled(), circuit_build_needed_circs(), circuit_build_times_add_time(), circuit_build_times_get_initial_timeout(), circuit_build_times_update_state(), circuit_expire_all_dirty_circs(), circuit_expire_old_circuits_clientside(), circuit_extend(), circuit_free(), circuit_get_open_circ_or_launch(), circuit_is_acceptable(), circuit_note_clock_jumped(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), circuit_stream_is_being_handled(), circuit_testing_failed(), command_process_create_cell(), compute_retry_timeout(), configure_nameservers(), conn_close_if_marked(), connection_about_to_close_connection(), connection_ap_can_use_exit(), connection_ap_detach_retriable(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_rewrite_and_attach(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_bucket_init(), connection_bucket_refill(), connection_connect(), connection_create_listener(), connection_dir_client_reached_eof(), connection_dir_request_failed(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_handle_listener_read(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_connect_failed(), connection_or_finished_connecting(), connection_or_flush_from_first_active_circuit(), connection_or_init_conn_from_address(), connection_proxy_connect(), connection_read_proxy_handshake(), consider_hibernation(), consider_plaintext_ports(), consider_publishable_server(), consider_recording_trackhost(), control_event_bootstrap_problem(), control_event_guard_deferred(), count_loading_descriptors_progress(), cpuworkers_rotate(), decide_if_publishable_server(), dir_routerdesc_download_failed(), directories_have_accepted_server_descriptor(), directory_get_from_dirserver(), directory_handle_command_get(), directory_handle_command_post(), directory_info_has_arrived(), directory_initiate_command_rend(), directory_send_command(), dirserv_dump_directory_to_string(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_status_impl(), dirserv_load_fingerprint_file(), dirserv_orconn_tls_done(), dirserv_pick_cached_dir_obj(), dirserv_router_has_valid_address(), dirserv_set_router_is_running(), dirserv_test_reachability(), dirserv_thinks_router_is_hs_dir(), dirvote_compute_consensuses(), dirvote_get_preferred_voting_intervals(), dns_init(), dns_launch_correctness_checks(), dns_reset(), dns_servers_relaunch_checks(), do_hash_password(), do_hup(), do_list_fingerprint(), do_main_loop(), download_status_reset(), entry_guards_changed(), entry_guards_compute_status(), entry_guards_prepend_from_config(), entry_guards_update_state(), entry_is_live(), escaped_safe_str(), escaped_safe_str_client(), evdns_log_cb(), extrainfo_dump_to_string(), fetch_bridge_descriptors(), generate_runningrouters(), generate_v2_networkstatus_opinion(), geoip_change_dirreq_state(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), geoip_note_client_seen(), geoip_note_ns_response(), geoip_start_dirreq(), get_cookie_file(), get_possible_sybil_list(), getinfo_helper_accounting(), getinfo_helper_events(), getinfo_helper_misc(), global_write_bucket_low(), handle_control_attachstream(), handle_control_authenticate(), handle_control_getconf(), handle_control_protocolinfo(), hibernate_begin(), hibernate_go_dormant(), hibernate_hard_limit_reached(), hibernate_soft_limit_reached(), hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), init_cookie_authentication(), init_key_from_file(), init_keys(), init_libevent(), init_v3_authority_keys(), ip_address_changed(), is_local_addr(), is_test_address(), launch_resolve(), launch_router_descriptor_downloads(), launch_test_addresses(), learned_bridge_descriptor(), list_server_status_v1(), networkstatus_dump_bridge_status_to_file(), networkstatus_getinfo_by_purpose(), networkstatus_set_current_consensus(), networkstatus_v2_list_clean(), onion_pending_add(), options_act(), options_act_reversible(), options_save_current(), options_trial_assign(), or_state_save(), parse_reachable_addresses(), pick_entry_guards(), purpose_needs_anonymity(), refresh_all_country_info(), rend_client_refetch_v2_renddesc(), rend_consider_descriptor_republication(), rend_consider_services_upload(), rend_service_requires_uptime(), rend_services_introduce(), rep_hist_circbuilding_dormant(), rep_hist_dump_stats(), rep_hist_fill_bandwidth_history(), rep_hist_note_exit_bytes_read(), rep_hist_note_exit_bytes_written(), rep_hist_note_exit_stream_opened(), rep_hist_update_state(), rep_history_clean(), retry_all_listeners(), rotate_onion_key(), router_add_to_routerlist(), router_choose_random_node(), router_digest_is_trusted_dir_type(), router_dirport_found_reachable(), router_dump_router_to_string(), router_get_by_nickname(), router_get_my_extrainfo(), router_get_my_routerinfo(), router_new_address_suggestion(), router_orport_found_reachable(), router_parse_directory(), router_parse_runningrouters(), router_perform_bandwidth_test(), router_pick_directory_server(), router_pick_trusteddirserver(), router_rebuild_descriptor(), router_reload_consensus_networkstatus(), router_reload_v2_networkstatus(), router_set_networkstatus_v2(), router_upload_dir_desc_to_dirservers(), routerlist_add_family(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), routers_in_same_family(), routers_update_all_from_networkstatus(), routers_update_status_from_consensus_networkstatus(), run_connection_housekeeping(), run_scheduled_events(), safe_str(), safe_str_client(), second_elapsed_callback(), should_generate_v2_networkstatus(), should_use_create_fast_for_circuit(), signal_callback(), signed_desc_digest_is_recognized(), spawn_enough_cpuworkers(), tor_cleanup(), tor_init(), tor_main(), trusted_dirs_load_certs_from_string(), update_consensus_networkstatus_downloads(), update_consensus_networkstatus_fetch_time(), update_consensus_router_descriptor_downloads(), update_expected_bandwidth(), update_extrainfo_downloads(), update_networkstatus_downloads(), update_router_descriptor_cache_downloads_v2(), update_router_descriptor_downloads(), update_router_have_minimum_dir_info(), update_v2_networkstatus_cache_downloads(), upload_service_descriptor(), v3_authority_check_key_expiry(), and warn_if_last_router_excluded().

4.35.5.493 or_state_t* get_or_state (void)

Return the persistent state struct for this Tor.

References tor_assert.

Referenced by accounting_run_housekeeping(), circuit_build_times_add_time(), circuit_build_times_update_state(), do_hup(), entry_guards_changed(), entry_guards_update_state(), hibernate_begin(), hibernate_go_dormant(), init_keys(), read_bandwidth_usage(), rep_hist_update_state(), rotate_onion_key(), and tor_cleanup().

4.35.5.494 const char* get_torrc_fname (void)

Return the location for our configuration file.

References get_default_conf_file().

Referenced by getinfo_helper_misc(), options_save_current(), and parse_dir_server_line().

4.35.5.495 const char* get_version (void)

Return the current Tor version.

References tor_snprintf().

Referenced by get_platform_str(), getinfo_helper_misc(), options_init_from_torrc(), or_state_save(), and tor_init().

4.35.5.496 int getinfo_helper_accounting (control_connection_t * *conn*, const char * *question*, char ** *answer*)

Helper function: called when we get a GETINFO request for an accounting-related key on the control connection **conn**. If we can answer the request for **question**, then set ***answer** to a newly allocated string holding the result. Otherwise, set *∗***answer** to NULL.

References accounting_is_enabled(), or_options_t::AccountingMax, format_iso_time(), get_options(), hibernate_state, HIBERNATE_STATE_DORMANT, HIBERNATE_STATE_LOWBANDWIDTH, interval_end_time, interval_start_time, interval_wakeup_time, n_bytes_read_in_interval, n_bytes_written_in_interval, and tor_snprintf().

4.35.5.497 int getinfo_helper_config (control_connection_t ∗ *conn*, const char ∗ *question*, char ∗∗ *answer*)

Helper to implement GETINFO functions about configuration variables (not their values). Given a "config/names" question, set ∗**answer** to a new string describing the supported configuration variables and their types.

References CONFIG_TYPE_BOOL, CONFIG_TYPE_CSV, CONFIG_TYPE_DOUBLE, CONFIG_TYPE_FILENAME, CONFIG_TYPE_INTERVAL, CONFIG_TYPE_ISOTIME, CONFIG_TYPE_LINELIST, CONFIG_TYPE_LINELIST_S, CONFIG_TYPE_LINELIST_V, CONFIG_TYPE_MEMUNIT, CONFIG_TYPE_OBSOLETE, CONFIG_TYPE_ROUTERSET, CONFIG_TYPE_STRING, CONFIG_TYPE_UINT, config_var_t::name, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_asprintf(), tor_free, and config_var_t::type.

4.35.5.498 int getinfo_helper_entry_guards (control_connection_t ∗ *conn*, const char ∗ *question*, char ∗∗ *answer*)

If **question** is the string "entry-guards", then dump to ∗**answer** a newly allocated string describing all of the nodes in the global entry_guards list. See control-spec.txt for details. For backward compatibility, we also handle the string "helper-nodes".

References base16_encode(), DIGEST_LEN, format_iso_time(), MAX_VERBOSE_NICKNAME_LEN, router_get_by_digest(), router_get_verbose_nickname(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_free, and tor_snprintf().

4.35.5.499 int getinfo_helper_geoip (control_connection_t ∗ *control_conn*, const char ∗ *question*, char ∗∗ *answer*)

Helper used to implement GETINFO ip-to-country/... controller command.

References geoip_get_country_by_ip(), geoip_get_country_name(), geoip_is_loaded(), strcmpstart(), and tor_inet_aton().

4.35.5.500 int getinfo_helper_networkstatus (control_connection_t ∗ *conn*, const char ∗ *question*, char ∗∗ *answer*)

If **question** is a string beginning with "ns/" in a format the control interface expects for a GETINFO question, set ∗**answer** to a newly-allocated string containing networkstatus lines for the appropriate ORs. Return 0 on success, -1 on unrecognized question format.

References networkstatus_getinfo_helper_single(), networkstatus_t::routerstatus_list, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), and tor_free.

4.35.5.501 int getinfo_helper_policies (control_connection_t * *conn*, const char * *question*, char ** *answer*)

Implementation for GETINFO control command: knows the answer for questions about "exit-policy/..."

4.35.5.502 int global_write_bucket_low (connection_t * *conn*, size_t *attempt*, int *priority*)

Return 1 if the global write buckets are low enough that we shouldn't send **attempt** bytes of low-priority directory stuff out to **conn**. Else return 0.

Priority is 1 for v1 requests (directories and running-routers), and 2 for v2 requests (statuses and descriptors). But see FFFF in **directory_handle_command_get()** (p. 575) for why we don't use priority 2 yet.

There are a lot of parameters we could use here:

- global_relayed_write_bucket. Low is bad.
- global_write_bucket. Low is bad.
- bandwidthrate. Low is bad.
- bandwidthburst. Not a big factor?
- attempt. High is bad.
- total bytes queued on outbufs. High is bad. But I'm wary of using this, since a few slow-flushing queues will pump up the number without meaning what we meant to mean. What we really mean is "total directory bytes added to outbufs recently", but that's harder to quantify and harder to keep track of.

References authdir_mode(), or_options_t::BandwidthRate, connection_is_rate_limited(), get_options(), and or_options_t::RelayBandwidthRate.

Referenced by directory_handle_command_get().

4.35.5.503 void handle_signals (int *is_parent*)

Set up the signal handlers for either parent or child.

References LD_BUG, signal_callback(), tor_evsignal_new(), and tor_libevent_get_base().

Referenced by cpuworker_main(), and do_main_loop().

4.35.5.504 int has_dns_init_failed (void)

Return true iff the most recent attempt to initialize the DNS subsystem failed.

References nameserver_config_failed.

Referenced by router_dump_router_to_string(), and run_scheduled_events().

4.35.5.505 int have_lockfile (void)

Return true iff we've successfully acquired the lock file.

Referenced by init_key_from_file(), and options_act().

4.35.5.506 int hexdigest_to_digest (const char ∗ *hexdigest*, char ∗ *digest*)

If hexdigest is correctly formed, base16_decode it into digest, which must have DIGEST_LEN space in it. Return 0 on success, -1 on failure.

References base16_decode(), DIGEST_LEN, and HEX_DIGEST_LEN.

Referenced by circuit_is_acceptable(), connection_ap_fail_onehop(), and router_get_by_hexdigest().

4.35.5.507 void hibernate_begin_shutdown (void)

A wrapper around hibernate_begin, for when we get SIGINT.

References hibernate_begin(), and HIBERNATE_STATE_EXITING.

Referenced by signal_callback().

4.35.5.508 int hid_serv_acting_as_directory (void)

Return true if this node is currently acting as hidden service directory, false otherwise.

References routerinfo_t::cache_info, get_options(), signed_descriptor_t::identity_digest, routerstatus_t::is_hs_dir, LD_REND, networkstatus_get_latest_consensus(), networkstatus_vote_find_entry(), and router_get_my_routerinfo().

Referenced by hid_serv_responsible_for_desc_id(), and rend_cache_store_v2_desc_as_dir().

4.35.5.509 int hid_serv_get_responsible_directories (smartlist_t ∗ *responsible_dirs*, const char ∗ *id*)

Determine the routers that are responsible for **id** (binary) and add pointers to those routers' **routerstatus_t** (p. 218) to **responsible_dirs**. Return -1 if we're returning an empty smartlist, else return 0.

References routerstatus_t::dir_port, get_options(), routerstatus_t::is_hs_dir, LD_REND, networkstatus_get_latest_consensus(), networkstatus_vote_find_entry_idx(), routerstatus_t::nickname, REND_NUMBER_OF_CONSECUTIVE_REPLICAS, networkstatus_t::routerstatus_list, smartlist_add(), tor_assert, and or_options_t::TunnelDirConns.

Referenced by directory_get_from_hs_dir(), directory_post_to_hs_dir(), and hid_serv_responsible_for_desc_id().

4.35.5.510 int hid_serv_responsible_for_desc_id (const char ∗ *query*)

Return true if this node is responsible for storing the descriptor ID in **query** and false otherwise.

References routerinfo_t::cache_info, hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), routerstatus_t::identity_digest, signed_descriptor_t::identity_digest, rend_id_is_in_interval(), router_get_my_routerinfo(), smartlist_create(), and smartlist_free().

Referenced by rend_cache_clean_v2_descs_as_dir(), rend_cache_lookup_v2_desc_as_dir(), and rend_cache_store_v2_desc_as_dir().

4.35.5.511 int identity_key_is_set (void)

Return true iff the identity key has been set.

Referenced by accounting_set_wakeup_time(), and do_main_loop().

4.35.5.512 int inform_testing_reachability (void)

We've decided to start our reachability testing. If all is set, log this to the user. Return 1 if we did, or 0 if we chose not to log anything.

References routerinfo_t::address, control_event_server_status(), routerinfo_t::dir_port, LD_OR, routerinfo_t::or_port, router_get_my_routerinfo(), TIMEOUT_UNTIL_UNREACHABILITY_COMPLAINT, and tor_snprintf().

Referenced by circuit_send_next_onion_skin(), and options_act().

4.35.5.513 void init_cell_pool (void)

Allocate structures to hold cells.

References tor_assert.

Referenced by do_main_loop().

4.35.5.514 int init_cookie_authentication (int *enabled*)

Choose a random authentication cookie and write it to disk. Anybody who can read the cookie from disk will be considered authorized to use the control connection. Return -1 if we can't write the file, or 0 on success.

References authentication_cookie, authentication_cookie_is_set, AUTHENTICATION_COOKIE_LEN, crypto_rand(), escaped(), get_cookie_file(), get_options(), LD_FS, tor_free, and write_bytes_to_file().

Referenced by options_act().

4.35.5.515 crypto_pk_env_t∗ init_key_from_file (const char ∗ *fname*, int *generate*, int *severity*)

Try to read an RSA key from **fname**. If **fname** doesn't exist and **generate** is true, create a new RSA key and save it in **fname**. Return the read/created key, or NULL on error. Log all errors at level **severity**.

References crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_check_key(), crypto_pk_read_private_key_from_filename(), crypto_pk_write_private_key_to_filename(), file_status(), get_options(), have_lockfile(), LD_FS, LD_GENERAL, tor_assert, and try_locking().

Referenced by init_keys(), load_authority_keyset(), and rend_service_load_keys().

4.35.5.516 int init_keys (void)

Initialize all OR private keys, and the TLS context, as necessary. On OPs, this only initializes the tls context. Return 0 on success, or -1 if Tor should die.

References add_trusted_dir_server(), authdir_mode(), authdir_mode_v3(), or_options_t::AvoidDiskWrites, BRIDGE_AUTHORITY, or_options_t::BridgeAuthoritativeDir, authority_cert_t::cache_info, check_private_dir(), or_options_t::command, crypto_free_pk_env(), crypto_global_init(), crypto_new_pk_env(), crypto_pk_get_digest(), crypto_pk_get_fingerprint(), or_options_t::DataDirectory, DIGEST_LEN, or_options_t::DirPort, dirserv_add_descriptor(), dirserv_add_own_fingerprint(), dirserv_load_fingerprint_file(), file_status(), FINGERPRINT_LEN, get_datadir_fname, get_datadir_fname2, get_identity_key(), get_my_v3_authority_cert(), get_options(), get_or_state(), HIDSERV_AUTHORITY, or_options_t::HSAuthoritativeDir, init_key_from_file(), init_v3_authority_keys(), or_state_t::LastRotatedOnionKey, LD_BUG, LD_DIR, LD_FS, LD_GENERAL, MAX_NICKNAME_LEN, MAX_SSL_KEY_LIFETIME, or_options_t::Nickname, onionkey_set_at, or_state_mark_dirty(), or_options_t::ORPort, read_file_to_str(), router_get_my_descriptor(), router_get_trusteddirserver_by_digest(), router_parse_entry_from_string(), server_mode(), set_identity_key(), set_onion_key(), signed_descriptor_t::signed_descriptor_body, tor_assert, tor_free, tor_mutex_new(), tor_snprintf(), tor_tls_context_new(), trusted_dirs_load_certs_from_string(), trusted_dir_server_t::type, V1_AUTHORITY, or_options_t::V1AuthoritativeDir, V2_AUTHORITY, or_options_t::V2AuthoritativeDir, V3_AUTHORITY, trusted_dir_server_t::v3_identity_digest, or_options_t::V3AuthoritativeDir, WRA_WAS_ADDED(), and write_str_to_file().

Referenced by accounting_set_wakeup_time(), do_list_fingerprint(), do_main_loop(), ip_address_changed(), and options_act().

4.35.5.517 void ip_address_changed (int *at_interface*)

Called when our IP address seems to have changed. **at_interface** should be true if we detected a change in our interface, and false if we detected a change in our published address.

References dns_servers_relaunch_checks(), get_options(), init_keys(), mark_my_descriptor_dirty(), reset_bandwidth_test(), router_reset_reachability(), server_mode(), and stats_n_seconds_working.

Referenced by check_descriptor_ipaddress_changed(), client_check_address_changed(), options_act(), resolve_my_address(), and router_new_address_suggestion().

4.35.5.518 int is_legal_hexdigest (const char ∗ *s*)

Return true iff **s** is a legally valid hex-encoded identity-key digest.

References HEX_CHARACTERS, HEX_DIGEST_LEN, is_legal_nickname(), and tor_assert.

Referenced by is_legal_nickname_or_hexdigest(), router_rebuild_descriptor(), and routerset_parse().

4.35.5.519 int is_legal_nickname (const char ∗ *s*)

Return true iff **s** is a legally valid server nickname.

References LEGAL_NICKNAME_CHARACTERS, MAX_NICKNAME_LEN, and tor_assert.

Referenced by circuit_list_path_impl(), dirserv_load_fingerprint_file(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), is_legal_hexdigest(), is_legal_nickname_or_hexdigest(), options_validate(), parse_dir_server_line(), rend_service_introduce(), router_parse_entry_from_string(), routerset_parse(), and routerstatus_parse_entry_from_string().

4.35.5.520 int is_legal_nickname_or_hexdigest (const char * *s*)

Return true iff **s** is a legally valid server nickname or hex-encoded identity-key digest.

References is_legal_hexdigest(), and is_legal_nickname().

Referenced by add_nickname_list_to_smartlist(), check_nickname_list(), microdescs_parse_from_string(), rend_service_introduce(), router_parse_entry_from_string(), and router_rebuild_descriptor().

4.35.5.521 int is_local_addr (const tor_addr_t * *addr*)

Return true iff **addr** is judged to be on the same network as us, or on a private network.

References get_options(), tor_addr_family(), tor_addr_is_internal(), and tor_addr_to_ipv4h().

Referenced by onionskin_answer(), and write_http_response_header_impl().

4.35.5.522 int is_or_protocol_version_known (uint16_t *v*)

Return true iff **v** is a link protocol version that this Tor implementation believes it can support.

References n_or_protocol_versions, and or_protocol_versions.

Referenced by command_process_versions_cell().

4.35.5.523 void learned_bridge_descriptor (routerinfo_t * *ri*, int *from_cache*)

We just learned a descriptor for a bridge. See if that digest is in our entry guard list, and add it if not.

References add_an_entry_guard(), any_bridge_descriptors_known(), download_status_reset(), bridge_info_t::fetch_status, get_options(), routerinfo_t::is_running, LD_DIR, routerinfo_t::nickname, routerinfo_t::purpose, ROUTER_PURPOSE_BRIDGE, routerinfo_get_configured_bridge(), routerlist_retry_directory_downloads(), and tor_assert.

Referenced by routerlist_descriptors_added().

4.35.5.524 int list_server_status_v1 (smartlist_t * *routers*, char ** *router_status_out*, int *for_controller*)

Based on the routerinfo_ts in **routers**, allocate the contents of a v1-style router-status line, and store it in ***router_status_out**. Return 0 on success, -1 on failure.

If for_controller is true, include the routers with very old descriptors.

References authdir_mode_publishes_statuses(), dirserv_set_router_is_running(), get_options(), list_single_server_status(), MAX_VERBOSE_NICKNAME_LEN, router_get_verbose_nickname(), ROUTER_MAX_AGE_TO_PUBLISH, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_assert, and tor_free.

4.35.5.525 void make_circuit_active_on_conn (circuit_t ∗ *circ*, or_connection_t ∗ *conn*)

Add **circ** to the list of circuits with pending cells on **conn**. No effect if **circ** is already linked.

References or_connection_t::active_circuits, add_cell_ewma_to_conn(), circuit_t::n_cell_ewma, circuit_t::n_conn, next_circ_on_conn_p(), or_circuit_t::p_cell_ewma, or_circuit_t::p_conn, prev_circ_on_conn_p(), TO_OR_CIRCUIT(), and tor_assert.

Referenced by append_cell_to_circuit_queue(), and HT_HEAD().

4.35.5.526 void make_circuit_inactive_on_conn (circuit_t ∗ *circ*, or_connection_t ∗ *conn*)

Remove **circ** from the list of circuits with pending cells on **conn**. No effect if **circ** is already unlinked.

References or_connection_t::active_circuits, circuit_t::n_cell_ewma, circuit_t::n_conn, next_circ_on_conn_p(), or_circuit_t::p_cell_ewma, or_circuit_t::p_conn, prev_circ_on_conn_p(), remove_cell_ewma_from_conn(), TO_OR_CIRCUIT(), and tor_assert.

Referenced by connection_or_flush_from_first_active_circuit(), and HT_HEAD().

4.35.5.527 void mark_my_descriptor_dirty (void)

Call when the current descriptor is out of date.

References desc_clean_since.

Referenced by add_wildcarded_test_address(), check_descriptor_bandwidth_changed(), configure_nameservers(), ip_address_changed(), mark_my_descriptor_dirty_if_older_than(), options_act(), rotate_onion_key(), router_dirport_found_reachable(), router_orport_found_reachable(), and set_onion_key().

4.35.5.528 void mark_my_descriptor_dirty_if_older_than (time_t *when*)

Mark descriptor out of date if it's older than **when**

References desc_clean_since, and mark_my_descriptor_dirty().

Referenced by run_scheduled_events().

4.35.5.529 size_t microdesc_average_size (microdesc_cache_t ∗ *cache*)

Return the mean size of decriptors added to **cache** since it was last cleared. Used to estimate the size of large downloads.

4.35.5.530 void microdesc_cache_clear (microdesc_cache_t ∗ *cache*)

Remove every microdescriptor in **cache**.

References tor_munmap_file().

4.35.5.531 microdesc_t∗ microdesc_cache_lookup_by_digest256 (microdesc_cache_t ∗ *cache*, const char ∗ *d*)

If there is a microdescriptor in **cache** whose sha256 digest is **d**, return it. Otherwise return NULL.

References microdesc_t::digest, and DIGEST256_LEN.

4.35.5.532 int microdesc_cache_rebuild (microdesc_cache_t ∗ *cache*)

Regenerate the main cache file for **cache**, clear the journal file, and update every **microdesc_t** (p. 114) in the cache with pointers to its new location.

References microdesc_t::body, finish_writing_to_file(), LD_DIR, microdesc_t::no_save, microdesc_t::off, SAVED_IN_CACHE, microdesc_t::saved_location, smartlist_add(), smartlist_create(), smartlist_free(), start_writing_to_stdio_file(), tor_assert, tor_free, tor_mmap_file(), tor_munmap_file(), and write_str_to_file().

4.35.5.533 int microdesc_cache_reload (microdesc_cache_t ∗ *cache*)

Reload the contents of **cache** from disk. If it is empty, load it for the first time. Return 0 on success, -1 on failure.

References tor_mmap_t::data, LD_DIR, read_file_to_str(), RFTS_IGNORE_MISSING, SAVED_IN_CACHE, SAVED_IN_JOURNAL, tor_mmap_t::size, smartlist_free(), tor_free, and tor_mmap_file().

4.35.5.534 void microdesc_free (microdesc_t ∗ *md*)

Deallocate a single microdescriptor. Note: the microdescriptor MUST have previously been removed from the cache if it had ever been inserted.

References microdesc_t::body, crypto_free_pk_env(), microdesc_t::exitsummary, microdesc_t::family, microdesc_t::onion_pkey, SAVED_IN_CACHE, microdesc_t::saved_location, smartlist_free(), and tor_free.

4.35.5.535 void microdesc_free_all (void)

Free all storage held in the microdesc.c module.

References tor_free.

4.35.5.536 smartlist_t∗ microdescs_add_to_cache (microdesc_cache_t ∗ *cache*, const char ∗ *s*, const char ∗ *eos*, saved_location_t *where*, int *no_save*)

Decode the microdescriptors from the string starting at **s** and ending at **eos**, and store them in **cache**. If **no-save**, mark them as non-writable to disk. If **where** is SAVED_IN_CACHE, leave their bodies as pointers to the mmap'd cache. If where is **SAVED_NOWHERE**, do not allow annotations. Return a list of the added microdescriptors.

References microdescs_parse_from_string(), SAVED_IN_CACHE, SAVED_NOWHERE, and smartlist_free().

4.35.5.537 smartlist_t* microdescs_parse_from_string (const char * *s*, const char * *eos*, int *allow_annotations*, int *copy_body*)

Parse as many microdescriptors as are found from the string starting at **s** and ending at **eos**. If allow_annotations is set, read any annotations we recognize and ignore ones we don't. If **copy_body** is true, then strdup the bodies of the microdescriptors. Return all newly parsed microdescriptors in a newly allocated **smartlist_t** (p. 232).

References directory_token_t::args, microdesc_t::body, microdesc_t::bodylen, microdesc_t::digest, eat_whitespace_eos(), escaped(), microdesc_t::exitsummary, microdesc_t::family, find_opt_by_keyword(), find_start_of_next_microdesc(), is_legal_nickname_or_hexdigest(), directory_token_t::key, microdesc_t::last_listed, LD_DIR, memarea_clear(), memarea_drop_all(), memarea_new(), directory_token_t::n_args, microdesc_t::off, microdesc_t::onion_pkey, parse_iso_time(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), tokenize_string(), and tor_assert.

Referenced by dirvote_create_microdescriptor(), and microdescs_add_to_cache().

4.35.5.538 int move_buf_to_buf (buf_t * *buf_out*, buf_t * *buf_in*, size_t * *buf_flushlen*)

Move up to ∗**buf_flushlen** bytes from **buf_in** to **buf_out**, and modify ∗**buf_flushlen** appropriately. Return the number of bytes actually copied.

References buf_t::datalen, fetch_from_buf(), tor_assert, and write_to_buf().

Referenced by conn_close_if_marked(), and connection_read_to_buf().

4.35.5.539 int networkstatus_add_detached_signatures (networkstatus_t * *target*, ns_detached_signatures_t * *sigs*, const char ** *msg_out*)

Given a consensus vote **target** and a set of detached signatures in **sigs** that correspond to the same consensus, check whether there are any new signatures in **src_voter_list** that should be added to **target**. (A signature should be added if we have no signature for that voter in **target** yet, or if we have no verifiable signature and the new signature is verifiable.) Return the number of signatures added or changed, or -1 if the document signed by **sigs** isn't the same document as **target**.

Make sure all the digests we know match, and at least one matches.

References authority_cert_get_by_digests(), document_signature_t::bad_signature, base16_encode(), crypto_digest_algorithm_get_name(), digests_t::d, DIGEST256_LEN, DIGEST_LEN, networkstatus_t::digests, ns_detached_signatures_t::digests, document_signature_dup(), document_signature_free(), networkstatus_t::flavor, ns_detached_signatures_t::fresh_until, networkstatus_t::fresh_until, document_signature_t::good_signature, HEX_DIGEST_LEN, LD_DIR, networkstatus_check_document_signature(), networkstatus_get_flavor_name(), networkstatus_get_voter_by_id(), ns_detached_signatures_t::signatures, networkstatus_voter_info_t::sigs, smartlist_add(), smartlist_remove(), strmap_get(), tor_assert, tor_mem_is_zero(), networkstatus_t::type, ns_detached_signatures_t::valid_after, networkstatus_t::valid_after, ns_detached_signatures_t::valid_until, networkstatus_t::valid_until, and voter_get_sig_by_algorithm().

Referenced by dirvote_add_signatures_to_pending_consensus().

4.35.5.540 int networkstatus_check_consensus_signature (networkstatus_t * *consensus*, int *warn*)

Given a v3 networkstatus consensus in **consensus**, check every as-yet-unchecked signature on **consensus**. Return 1 if there is a signature from every recognized authority on it, 0 if there are enough good signatures from recognized authorities on it, -1 if we might get enough good signatures by fetching missing certificates, and -2 otherwise. Log messages at INFO or WARN: if **warn** is over 1, warn about every problem; if warn is at least 1, warn only if we can't get enough signatures; if warn is negative, log nothing at all.

References authority_cert_get_by_digests(), DIGEST_LEN, authority_cert_t::expires, get_n_authorities(), hex_str(), LD_DIR, networkstatus_check_document_signature(), networkstatus_get_voter_by_id(), router_get_trusted_dir_servers(), smartlist_add(), smartlist_create(), smartlist_free(), tor_assert, trusteddirserver_get_by_v3_auth_digest(), networkstatus_t::type, V3_AUTHORITY, and networkstatus_t::voters.

Referenced by dirvote_compute_consensuses(), dirvote_fetch_missing_signatures(), dirvote_publish_consensus(), networkstatus_note_certs_arrived(), and networkstatus_set_current_consensus().

4.35.5.541 int networkstatus_check_document_signature (const networkstatus_t * *consensus*, document_signature_t * *sig*, const authority_cert_t * *cert*)

Check whether the signature **sig** is correctly signed with the signing key in **cert**. Return -1 if **cert** doesn't match the signing key; otherwise set the good_signature or bad_signature flag on **voter**, and return 0.

References document_signature_t::alg, document_signature_t::bad_signature, authority_cert_t::cache_info, crypto_pk_get_digest(), crypto_pk_keysize(), crypto_pk_public_checksig(), digests_t::d, DIGEST256_LEN, DIGEST_LEN, networkstatus_t::digests, document_signature_t::good_signature, signed_descriptor_t::identity_digest, document_signature_t::identity_digest, LD_DIR, document_signature_t::signature, document_signature_t::signature_len, authority_cert_t::signing_key, document_signature_t::signing_key_digest, and tor_free.

Referenced by networkstatus_add_detached_signatures(), and networkstatus_check_consensus_signature().

4.35.5.542 char* networkstatus_compute_consensus (smartlist_t * *votes*, int *total_authorities*, crypto_pk_env_t * *identity_key*, crypto_pk_env_t * *signing_key*, const char * *legacy_id_key_digest*, crypto_pk_env_t * *legacy_signing_key*, consensus_flavor_t *flavor*)

Given a list of vote **networkstatus_t** (p. 122) in **votes**, our public authority **identity_key**, our private authority **signing_key**, and the number of **total_authorities** that we believe exist in our voting quorum, generate the text of a new v3 consensus vote, and return the value in a newly allocated string.

Note: this function DOES NOT check whether the votes are from recognized authorities. (dirvote_add_vote does that.)

References _compare_dir_src_ents_by_authority_id(), _compare_votes_by_authority_id(), networkstatus_voter_info_t::addr, networkstatus_voter_info_t::address, base16_encode(), compute_consensus_method(), compute_consensus_versions_list(), consensus_method_is_-

supported(), networkstatus_voter_info_t::contact, dir_src_ent_t::digest, DIGEST_LEN, networkstatus_voter_info_t::dir_port, dirvote_compute_params(), format_iso_time(), get_voter(), HEX_DIGEST_LEN, networkstatus_voter_info_t::identity_digest, INET_NTOA_BUF_LEN, dir_src_ent_t::is_legacy, LD_DIR, networkstatus_voter_info_t::legacy_id_digest, MIN_DIST_SECONDS, MIN_METHOD_FOR_PARAMS, MIN_VOTE_INTERVAL, MIN_VOTE_SECONDS, networkstatus_get_flavor_name(), networkstatus_voter_info_t::nickname, networkstatus_voter_info_t::or_port, smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), smartlist_sort_strings(), smartlist_split_string(), smartlist_string_pos(), smartlist_uniq_strings(), sort_version_list(), strmap_get_lc(), strmap_set_lc(), tor_asprintf(), tor_assert, tor_digest_is_zero(), tor_free, tor_inet_ntoa(), dir_src_ent_t::v, and networkstatus_voter_info_t::vote_digest.

Referenced by dirvote_compute_consensuses().

4.35.5.543 void networkstatus_consensus_download_failed (int *status_code*)

Called when an attempt to download a consensus fails: note that the failure occurred, and possibly retry.

References download_status_failed, and update_consensus_networkstatus_downloads().

Referenced by connection_dir_client_reached_eof(), and connection_dir_request_failed().

4.35.5.544 void networkstatus_dump_bridge_status_to_file (time_t *now*)

Write out router status entries for all our bridge descriptors.

References or_options_t::DataDirectory, get_options(), networkstatus_getinfo_by_purpose(), tor_free, tor_snprintf(), and write_str_to_file().

Referenced by run_scheduled_events().

4.35.5.545 void networkstatus_free_all (void)

Free all storage held locally in this module.

References _tor_free(), consensus_waiting_for_certs_t::body, consensus_waiting_for_certs_t::consensus, digestmap_free(), named_server_map, networkstatus_v2_free(), networkstatus_vote_free(), smartlist_free(), strmap_free(), tor_free, unnamed_server_map, and v2_download_status_map.

Referenced by tor_free_all().

4.35.5.546 int32_t networkstatus_get_bw_weight (networkstatus_t * *ns*, const char * *weight_name*, int32_t *default_val*)

Return the value of a integer bw weight parameter from the networkstatus **ns** whose name is **weight_name**. If **ns** is NULL, try loading the latest consensus ourselves. Return **default_val** if no latest consensus, or if it has no parameter called **param_name**.

References networkstatus_get_latest_consensus(), and networkstatus_t::weight_params.

Referenced by networkstatus_verify_bw_weights(), and smartlist_choose_by_bandwidth_weights().

4.35.5.547 char∗ networkstatus_get_cache_filename (const char ∗ *identity_digest*)

Helper: return a newly allocated string containing the name of the filename where we plan to cache the network status with the given identity digest.

References base16_encode(), DIGEST_LEN, get_datadir_fname2, and HEX_DIGEST_LEN.

Referenced by add_networkstatus_to_cache(), dirserv_clear_old_networkstatuses(), networkstatus_v2_list_clean(), and router_set_networkstatus_v2().

4.35.5.548 char∗ networkstatus_get_detached_signatures (smartlist_t ∗ *consensuses*)

Return a newly allocated string holding the detached-signatures document corresponding to the signatures on **consensuses**, which must contain exactly one FLAV_NS consensus, and no more than one consensus for each other flavor.

References base16_encode(), crypto_digest_algorithm_get_name(), digests_t::d, DIGEST256_LEN, DIGEST_LEN, networkstatus_t::digests, format_iso_time(), networkstatus_t::fresh_until, HEX_DIGEST256_LEN, HEX_DIGEST_LEN, LD_BUG, LD_DIR, networkstatus_format_signatures(), networkstatus_get_flavor_name(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_assert, tor_free, tor_mem_is_zero(), tor_snprintf(), networkstatus_t::valid_after, and networkstatus_t::valid_until.

Referenced by get_detached_signatures_from_pending_consensuses().

4.35.5.549 const char∗ networkstatus_get_flavor_name (consensus_flavor_t *flav*)

Return the name of the consensus flavor **flav** as used to identify the flavor in directory documents.

References tor_fragile_assert.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_compute_consensuses(), dirvote_publish_consensus(), networkstatus_add_detached_signatures(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_detached_signatures(), networkstatus_note_certs_arrived(), networkstatus_set_current_consensus(), and router_reload_consensus_networkstatus().

4.35.5.550 networkstatus_t∗ networkstatus_get_latest_consensus (void)

Return the most recent consensus that we have downloaded, or NULL if we don't have one.

Referenced by directory_get_from_dirserver(), directory_handle_command_get(), getinfo_helper_events(), hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), networkstatus_get_bw_weight(), networkstatus_get_param(), options_act(), router_add_to_routerlist(), router_pick_directory_server_impl(), routerlist_remove_old_routers(), run_scheduled_events(), signed_desc_digest_is_recognized(), update_router_have_minimum_dir_info(), and upload_service_descriptor().

4.35.5.551 networkstatus_t∗ networkstatus_get_live_consensus (time_t *now*)

Return the most recent consensus that we have downloaded, or NULL if it is no longer live.

References networkstatus_t::valid_after.

Referenced by dirvote_recalculate_timing(), routers_update_all_from_networkstatus(), update_consensus_networkstatus_downloads(), and update_consensus_networkstatus_fetch_time().

4.35.5.552 int32_t networkstatus_get_param (networkstatus_t * *ns*, const char * *param_name*, int32_t *default_val*)

Return the value of a integer parameter from the networkstatus **ns** whose name is **param_name**. If **ns** is NULL, try loading the latest consensus ourselves. Return **default_val** if no latest consensus, or if it has no parameter called **param_name**.

References networkstatus_t::net_params, and networkstatus_get_latest_consensus().

Referenced by cell_ewma_set_scale_factor(), circuit_build_times_new_consensus_params(), circuit_initial_package_window(), connection_or_init_conn_from_address(), networkstatus_verify_bw_weights(), and smartlist_choose_by_bandwidth_weights().

4.35.5.553 networkstatus_t* networkstatus_get_reasonably_live_consensus (time_t *now*)

As **networkstatus_get_live_consensus()** (p. 723), but is way more tolerant of expired consensuses.

References networkstatus_t::valid_after.

Referenced by count_loading_descriptors_progress(), update_router_descriptor_downloads(), and update_router_have_minimum_dir_info().

4.35.5.554 const char* networkstatus_get_router_digest_by_nickname (const char * *nickname*)

Return the identity digest that's mapped to officially by **nickname**.

References named_server_map, and strmap_get_lc().

Referenced by router_get_by_nickname().

4.35.5.555 const smartlist_t* networkstatus_get_v2_list (void)

Return a list of the v2 networkstatus documents.

References smartlist_create().

Referenced by networkstatus_reset_download_failures(), router_add_to_routerlist(), routerlist_remove_old_routers(), signed_desc_digest_is_recognized(), and update_router_descriptor_cache_downloads_v2().

4.35.5.556 networkstatus_voter_info_t* networkstatus_get_voter_by_id (networkstatus_t * *vote*, const char * *identity*)

Return the voter info from **vote** for the voter whose identity digest is **identity**, or NULL if no such voter is associated with **vote**.

References DIGEST_LEN, and networkstatus_t::voters.

Referenced by networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), and networkstatus_parse_vote_from_string().

4.35.5.557 char∗ networkstatus_getinfo_by_purpose (const char ∗ *purpose_string*, time_t *now*)

Alloc and return a string describing routerstatuses for the most recent info of each router we know about that is of purpose **purpose_string**. Return NULL if unrecognized purpose.

Right now this function is oriented toward listing bridges (you shouldn't use this for general-purpose routers, since those should be listed from the consensus, not from the routers list).

References authdir_mode_bridge(), dirserv_set_router_is_running(), get_options(), LD_DIR, networkstatus_getinfo_helper_single(), router_get_routerlist(), ROUTER_MAX_AGE_TO_PUBLISH, ROUTER_PURPOSE_BRIDGE, router_purpose_from_string(), ROUTER_PURPOSE_UNKNOWN, routerlist_t::routers, set_routerstatus_from_routerinfo(), smartlist_add(), and smartlist_create().

Referenced by directory_handle_command_get(), and networkstatus_dump_bridge_status_to_file().

4.35.5.558 char∗ networkstatus_getinfo_helper_single (routerstatus_t ∗ *rs*)

Generate networkstatus lines for a single **routerstatus_t** (p. 218) object, and return the result in a newly allocated string. Used only by controller interface (for now.)

References routerstatus_format_entry(), and RS_ENTRY_LEN.

Referenced by control_event_networkstatus_changed_helper(), getinfo_helper_networkstatus(), and networkstatus_getinfo_by_purpose().

4.35.5.559 int networkstatus_nickname_is_unnamed (const char ∗ *nickname*)

Return true iff **nickname** is disallowed from being the nickname of any server.

References strmap_get_lc(), and unnamed_server_map.

Referenced by router_get_by_nickname().

4.35.5.560 void networkstatus_note_certs_arrived (void)

Called when we have gotten more certificates: see whether we can now verify a pending consensus.

References consensus_waiting_for_certs_t::body, consensus_waiting_for_certs_t::consensus, networkstatus_check_consensus_signature(), networkstatus_get_flavor_name(), networkstatus_set_current_consensus(), and tor_free.

Referenced by trusted_dirs_load_certs_from_string().

4.35.5.561 ns_detached_signatures_t∗ networkstatus_parse_detached_signatures (const char ∗ *s*, const char ∗ *eos*)

Parse a detached v3 networkstatus signature document between **s** and **eos** and return the result. Return -1 on failure.

References directory_token_t::args, base16_decode(), crypto_digest_algorithm_parse_name(), digests_t::d, detached_get_digests(), detached_get_signatures(), DIGEST256_LEN, DIGEST_LEN, ns_detached_signatures_t::digests, escaped(), ns_detached_signatures_t::fresh_until, HEX_DIGEST256_LEN, HEX_DIGEST_LEN, LD_DIR, memarea_new(), directory_token_t::n_args, directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, parse_iso_time(), ns_detached_signatures_t::signatures, smartlist_add(), smartlist_create(), tokenize_string(), tor_assert, tor_free, tor_mem_is_zero(), directory_token_t::tp, ns_detached_signatures_t::valid_after, and ns_detached_signatures_t::valid_until.

4.35.5.562 int networkstatus_parse_flavor_name (const char ∗ *flavname*)

Return the consensus_flavor_t value for the flavor called **flavname**, or -1 if the flavor is not recognized.

Referenced by networkstatus_parse_vote_from_string(), and networkstatus_set_current_consensus().

4.35.5.563 networkstatus_t∗ networkstatus_parse_vote_from_string (const char ∗ *s*, const char ∗∗ *eos_out*, networkstatus_type_t *ns_type*)

Parse a v3 networkstatus vote, opinion, or consensus (depending on ns_type), from **s**, and return the result. Return NULL on failure.

References networkstatus_voter_info_t::addr, networkstatus_voter_info_t::address, document_signature_t::alg, directory_token_t::args, authority_cert_parse_from_string(), base16_decode(), authority_cert_t::cache_info, networkstatus_t::cert, check_signature_token(), networkstatus_t::client_versions, networkstatus_t::consensus_method, networkstatus_voter_info_t::contact, crypto_digest_algorithm_parse_name(), digests_t::d, DIGEST_LEN, networkstatus_t::digests, networkstatus_voter_info_t::dir_port, networkstatus_t::dist_seconds, document_signature_free(), dump_desc(), escaped(), find_opt_by_keyword(), find_start_of_next_routerstatus(), networkstatus_t::flavor, networkstatus_t::fresh_until, document_signature_t::good_signature, HEX_DIGEST_LEN, document_signature_t::identity_digest, routerstatus_t::identity_digest, signed_descriptor_t::identity_digest, networkstatus_voter_info_t::identity_digest, networkstatus_t::known_flags, LD_DIR, networkstatus_voter_info_t::legacy_id_digest, memarea_drop_all(), memarea_new(), MIN_DIST_SECONDS, MIN_VOTE_INTERVAL, MIN_VOTE_SECONDS, directory_token_t::n_args, networkstatus_t::net_params, networkstatus_get_voter_by_id(), networkstatus_parse_flavor_name(), networkstatus_vote_free(), networkstatus_voter_info_t::nickname, directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, networkstatus_voter_info_t::or_port, parse_iso_time(), networkstatus_t::published, router_get_networkstatus_v3_hashes(), networkstatus_t::routerstatus_list, routerstatus_parse_entry_from_string(), networkstatus_t::server_versions, document_signature_t::signature, document_signature_t::signature_len, authority_cert_t::signing_key, document_signature_t::signing_key_digest, networkstatus_voter_info_t::sigs, smartlist_add(), smartlist_create(), smartlist_free(), vote_routerstatus_t::status, strcmpstart(), networkstatus_t::supported_methods, token_clear(), tokenize_string(), tor_assert, tor_digest_is_zero(), tor_free, tor_inet_aton(), tor_parse_long(), directory_token_t::tp, networkstatus_t::type, networkstatus_t::valid_after, networkstatus_t::valid_until, vote_routerstatus_t::version, networkstatus_voter_info_t::vote_digest, networkstatus_t::vote_seconds, voter_get_sig_by_algorithm(), networkstatus_t::voters, and networkstatus_t::weight_params.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_add_vote(), dirvote_compute_consensuses(), format_networkstatus_vote(), and networkstatus_set_current_consensus().

4.35.5.564 void networkstatus_reset_download_failures (void)

Reset the descriptor download failure count on all networkstatus docs, so that we can retry any long-failed documents immediately.

References networkstatus_get_v2_list(), and router_get_by_descriptor_digest().

Referenced by router_reset_descriptor_download_failures().

4.35.5.565 void networkstatus_reset_warnings (void)

Forget that we've warned about anything networkstatus-related, so we will give fresh warnings if the same behavior happens again.

References have_warned_about_new_version, have_warned_about_old_version, and networkstatus_t::routerstatus_list.

Referenced by routerlist_reset_warnings().

4.35.5.566 int networkstatus_set_current_consensus (const char * *consensus*, const char * *flavor*, unsigned *flags*)

Try to replace the current cached v3 networkstatus with the one in **consensus**. If we don't have enough certificates to validate it, store it in consensus_waiting_for_certs and launch a certificate fetch.

If flags & NSSET_FROM_CACHE, this networkstatus has come from the disk cache. If flags & NSSET_WAS_WAITING_FOR_CERTS, this networkstatus was already received, but we were waiting for certificates on it. If flags & NSSET_DONT_DOWNLOAD_CERTS, do not launch certificate downloads as needed. If flags & NSSET_ACCEPT_OBSOLETE, then we should be willing to take this consensus, even if it comes from many days in the past.

Return 0 on success, <0 on failure. On failure, caller should increment the failure count as appropriate.

We return -1 for mild failures that don't need to be reported to the user, and -2 for more serious problems.

References authority_certs_fetch_missing(), consensus_waiting_for_certs_t::body, cell_ewma_set_scale_factor(), circ_times, circuit_build_times_new_consensus_params(), consensus_waiting_for_certs_t::consensus, control_event_client_status(), control_event_general_status(), cached_dir_t::digests, networkstatus_t::digests, directory_caches_dir_info(), dirserv_get_consensus(), dirserv_set_cached_consensus_networkstatus(), dirvote_recalculate_timing(), consensus_waiting_for_certs_t::dl_failed, download_status_failed, download_status_reset(), networkstatus_t::flavor, format_iso_time(), format_time_interval(), ftime_definitely_before(), get_datadir_fname, get_options(), LD_BUG, LD_DIR, LD_GENERAL, networkstatus_check_consensus_signature(), networkstatus_copy_old_consensus_info(), networkstatus_get_flavor_name(), networkstatus_parse_flavor_name(), networkstatus_parse_vote_from_string(), networkstatus_vote_free(), notify_control_networkstatus_changed(), OLD_ROUTER_DESC_MAX_AGE, cached_dir_t::published, router_dir_info_changed(), routerstatus_list_update_named_server_map(), consensus_waiting_for_certs_t::set_at, tor_free, tor_snprintf(), update_consensus_networkstatus_fetch_time(), USABLE_CONSENSUS_FLAVOR, networkstatus_t::valid_after, networkstatus_t::valid_until, and write_str_to_file().

Referenced by connection_dir_client_reached_eof(), dirvote_publish_consensus(), networkstatus_note_certs_arrived(), and router_reload_consensus_networkstatus().

4.35.5.567 routerstatus_t∗ networkstatus_v2_find_entry (networkstatus_v2_t ∗ *ns*, const char ∗ *digest*)

Return the entry in **ns** for the identity digest **digest**, or NULL if none was found.

References compare_digest_to_routerstatus_entry(), networkstatus_v2_t::entries, and smartlist_bsearch().

Referenced by router_add_to_routerlist(), and signed_desc_digest_is_recognized().

4.35.5.568 void networkstatus_v2_free (networkstatus_v2_t ∗ *ns*)

Free all storage held by the networkstatus object **ns**.

References networkstatus_v2_t::client_versions, networkstatus_v2_t::contact, crypto_free_pk_env(), networkstatus_v2_t::entries, routerstatus_free(), networkstatus_v2_t::server_versions, networkstatus_v2_t::signing_key, smartlist_free(), networkstatus_v2_t::source_address, and tor_free.

Referenced by networkstatus_free_all(), networkstatus_v2_list_clean(), networkstatus_v2_parse_from_string(), and router_set_networkstatus_v2().

4.35.5.569 networkstatus_v2_t∗ networkstatus_v2_get_by_digest (const char ∗ *digest*)

Return the network status with a given identity digest.

References DIGEST_LEN.

4.35.5.570 void networkstatus_v2_list_clean (time_t *now*)

Remove all very-old network_status_t objects from memory and from the disk cache.

References directory_caches_v2_dir_info(), dirserv_clear_old_networkstatuses(), dirserv_clear_old_v1_info(), dirserv_set_cached_networkstatus_v2(), file_status(), get_options(), networkstatus_v2_t::identity_digest, LD_DIR, MAX_NETWORKSTATUS_AGE, networkstatus_get_cache_filename(), networkstatus_v2_free(), networkstatus_v2_t::published_on, smartlist_del(), and tor_free.

Referenced by router_reload_v2_networkstatus(), and run_scheduled_events().

4.35.5.571 networkstatus_v2_t∗ networkstatus_v2_parse_from_string (const char ∗ *s*)

Given a v2 network-status object in **s**, try to parse it and return the result. Return NULL on failure. Check the signature of the network status, but do not (yet) check the signing key for authority.

References _free_duplicate_routerstatus_entry(), base16_decode(), networkstatus_v2_t::binds_names, check_signature_token(), networkstatus_v2_t::client_versions, compare_-

routerstatus_entries(), networkstatus_v2_t::contact, crypto_pk_get_digest(), DIGEST_LEN, dump_desc(), networkstatus_v2_t::entries, escaped(), find_opt_by_keyword(), find_start_of_next_routerstatus(), networkstatus_v2_t::identity_digest, crypto_pk_env_t::key, LD_BUG, LD_DIR, networkstatus_v2_t::lists_bad_directories, networkstatus_v2_t::lists_bad_exits, memarea_clear(), memarea_drop_all(), memarea_new(), networkstatus_v2_t::networkstatus_digest, networkstatus_v2_free(), note_crypto_pk_op(), parse_iso_time(), networkstatus_v2_t::published_on, networkstatus_v2_t::recommends_versions, router_get_networkstatus_v2_hash(), routerstatus_parse_entry_from_string(), networkstatus_v2_t::server_versions, networkstatus_v2_t::signing_key, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_sort(), smartlist_uniq(), networkstatus_v2_t::source_addr, networkstatus_v2_t::source_address, networkstatus_v2_t::source_dirport, strcmpstart(), token_clear(), tokenize_string(), tor_assert, tor_inet_aton(), and tor_parse_long().

Referenced by router_set_networkstatus_v2().

4.35.5.572 int networkstatus_verify_bw_weights (networkstatus_t * *ns*)

Verify the bandwidth weights of a network status document

References BW_WEIGHT_SCALE, LD_BUG, LD_DIR, networkstatus_get_bw_weight(), networkstatus_get_param(), networkstatus_t::routerstatus_list, and T.

4.35.5.573 routerstatus_t* networkstatus_vote_find_entry (networkstatus_t * *ns*, const char * *digest*)

Return the entry in **ns** for the identity digest **digest**, or NULL if none was found.

References compare_digest_to_routerstatus_entry(), networkstatus_t::routerstatus_list, and smartlist_bsearch().

Referenced by hid_serv_acting_as_directory(), router_add_to_routerlist(), router_get_consensus_status_by_nickname(), and signed_desc_digest_is_recognized().

4.35.5.574 int networkstatus_vote_find_entry_idx (networkstatus_t * *ns*, const char * *digest*, int * *found_out*)

Search the routerstatuses in **ns** for one whose identity digest is **digest**. Return value and set ***found_out** as for **smartlist_bsearch_idx()** (p. 471).

References compare_digest_to_routerstatus_entry(), networkstatus_t::routerstatus_list, and smartlist_bsearch_idx().

Referenced by hid_serv_get_responsible_directories().

4.35.5.575 void networkstatus_vote_free (networkstatus_t * *ns*)

Free all storage held in **ns**.

References authority_cert_free(), networkstatus_t::cert, networkstatus_t::client_versions, networkstatus_t::desc_digest_map, digestmap_free(), document_signature_free(), networkstatus_t::known_flags, networkstatus_t::net_params, routerstatus_free(), networkstatus_t::routerstatus_list, networkstatus_t::server_versions, smartlist_free(), networkstatus_t::supported_methods, tor_free, networkstatus_t::type, vote_routerstatus_free(), and networkstatus_t::voters.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_add_vote(), dirvote_clear_pending_consensuses(), dirvote_clear_votes(), dirvote_compute_consensuses(), dirvote_perform_vote(), format_networkstatus_vote(), networkstatus_free_all(), networkstatus_parse_vote_from_string(), and networkstatus_set_current_consensus().

4.35.5.576 cached_dir_t* new_cached_dir (char * *s*, time_t *published*)

Allocate and return a new **cached_dir_t** (p. 23) containing the string **s**, published at **published**.

References cached_dir_t::dir, cached_dir_t::dir_len, cached_dir_t::dir_z, cached_dir_t::dir_z_len, LD_BUG, cached_dir_t::published, cached_dir_t::refcnt, and tor_gzip_compress().

Referenced by dirserv_regenerate_directory(), dirserv_set_cached_consensus_networkstatus(), dirserv_set_cached_directory(), dirserv_set_cached_networkstatus_v2(), and dirvote_add_vote().

4.35.5.577 void note_crypto_pk_op (pk_op_t *operation*)

Increment the count of the number of times we've done **operation**.

References LD_BUG, and pk_op_counts.

Referenced by connection_tls_start_handshake(), dirserv_dump_directory_to_string(), extrainfo_parse_entry_from_string(), format_networkstatus_vote(), generate_runningrouters(), networkstatus_v2_parse_from_string(), onion_skin_create(), onion_skin_server_handshake(), rend_mid_establish_intro(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_intro_has_opened(), rend_service_introduce(), router_dump_router_to_string(), router_parse_directory(), router_parse_entry_from_string(), and router_parse_runningrouters().

4.35.5.578 void ns_detached_signatures_free (ns_detached_signatures_t * *s*)

Release all storage held in **s**.

References _tor_free(), ns_detached_signatures_t::digests, document_signature_free(), ns_detached_signatures_t::signatures, smartlist_free(), strmap_free(), and tor_free.

4.35.5.579 int num_rend_services (void)

Return the number of rendezvous services we have configured.

Referenced by circuit_predict_and_launch_new().

4.35.5.580 void onion_append_to_cpath (crypt_path_t ** *head_ptr*, crypt_path_t * *new_hop*)

Add **new_hop** to the end of the doubly-linked-list **head_ptr**. This function is used to extend cpath by another hop.

References crypt_path_t::next, and crypt_path_t::prev.

Referenced by onion_append_hop(), rend_client_receive_rendezvous(), and rend_service_rendezvous_has_opened().

4.35.5.581 or_circuit_t* onion_next_task (char ** *onionskin_out*)

Remove the first item from ol_list and return it, or return NULL if the list is empty.

References onion_queue_t::circ, ol_length, onion_pending_remove(), onion_queue_t::onionskin, or_circuit_t::p_conn, and tor_assert.

Referenced by process_pending_task().

4.35.5.582 int onion_pending_add (or_circuit_t * *circ*, char * *onionskin*)

Add **circ** to the end of ol_list and return 0, except if ol_list is too long, in which case do nothing and return -1.

References onion_queue_t::circ, get_options(), LD_CIRC, LD_GENERAL, onion_queue_t::next, ol_length, onion_pending_remove(), ONIONQUEUE_WAIT_CUTOFF, onion_queue_t::onionskin, TO_CIRCUIT, tor_assert, tor_free, and onion_queue_t::when_added.

Referenced by assign_onionskin_to_cpuworker().

4.35.5.583 void onion_pending_remove (or_circuit_t * *circ*)

Go through ol_list, find the **onion_queue_t** (p. 132) element which points to circ, remove and free that element. Leave circ itself alone.

References onion_queue_t::circ, LD_GENERAL, onion_queue_t::next, ol_length, onion_queue_t::onionskin, or_circuit_t::p_circ_id, and tor_free.

Referenced by _circuit_mark_for_close(), onion_next_task(), and onion_pending_add().

4.35.5.584 int onion_skin_client_handshake (crypto_dh_env_t * *handshake_state*, const char * *handshake_reply*, char * *key_out*, size_t *key_out_len*)

Finish the client side of the DH handshake. Given the 128 byte DH reply + 20 byte hash as generated by onion_skin_server_handshake and the handshake state generated by onion_skin_create, verify H(K) with the first 20 bytes of shared key material, then generate key_out_len more bytes of shared key material and store them in key_out.

After the invocation, call crypto_dh_free on handshake_state.

References crypto_dh_compute_secret(), crypto_dh_get_bytes(), DIGEST_LEN, LD_PROTOCOL, tor_assert, and tor_free.

Referenced by circuit_finish_handshake().

4.35.5.585 int onion_skin_create (crypto_pk_env_t * *dest_router_key*, crypto_dh_env_t ** *handshake_state_out*, char * *onion_skin_out*)

Given a router's 128 byte public key, stores the following in onion_skin_out:

42 bytes	OAEP padding
16 bytes	Symmetric key for encrypting blob past RSA
70 bytes	g^x part 1 (inside the RSA)

58 bytes g^x part 2 (symmetrically encrypted)

Stores the DH private key into handshake_state_out for later completion of the handshake.

The meeting point/cookies and auth are zeroed out for now.

References crypto_dh_free(), crypto_dh_get_bytes(), crypto_dh_get_public(), crypto_dh_new(), crypto_pk_keysize(), crypto_pk_public_hybrid_encrypt(), note_crypto_pk_op(), PK_PKCS1_OAEP_PADDING, and tor_assert.

Referenced by circuit_send_next_onion_skin().

4.35.5.586 int onion_skin_server_handshake (const char * *onion_skin*, crypto_pk_env_t * *private_key*, crypto_pk_env_t * *prev_private_key*, char * *handshake_reply_out*, char * *key_out*, size_t *key_out_len*)

Given an encrypted DH public key as generated by onion_skin_create, and the private key for this onion router, generate the reply (128-byte DH plus the first 20 bytes of shared key material), and store the next key_out_len bytes of key material in key_out.

References crypto_dh_compute_secret(), crypto_dh_free(), crypto_dh_get_public(), crypto_dh_new(), crypto_pk_private_hybrid_decrypt(), DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, note_crypto_pk_op(), PK_PKCS1_OAEP_PADDING, and tor_free.

Referenced by cpuworker_main().

4.35.5.587 int onionskin_answer (or_circuit_t * *circ*, uint8_t *cell_type*, const char * *payload*, const char * *keys*)

Given a response payload and keys, initialize, then send a created cell back.

References or_connection_t::_base, connection_t::addr, append_cell_to_circuit_queue(), crypt_path_t::b_crypto, crypt_path_t::b_digest, CELL_DIRECTION_IN, cell_t::circ_id, circuit_init_cpath_crypto(), circuit_set_state(), CIRCUIT_STATE_OPEN, cell_t::command, connection_or_nonopen_was_started_here(), DIGEST_LEN, crypt_path_t::f_crypto, crypt_path_t::f_digest, or_circuit_t::handshake_digest, or_circuit_t::is_first_hop, is_local_addr(), LD_BUG, LD_CIRC, crypt_path_t::magic, or_circuit_t::n_crypto, or_circuit_t::n_digest, or_circuit_t::p_circ_id, or_circuit_t::p_conn, or_circuit_t::p_crypto, or_circuit_t::p_digest, cell_t::payload, router_orport_found_reachable(), TO_CIRCUIT, and tor_free.

Referenced by command_process_create_cell(), and connection_cpu_process_inbuf().

4.35.5.588 config_line_t* option_get_assignment (or_options_t * *options*, const char * *key*)

Return a canonical list of the options assigned for key.

References get_assigned_option().

Referenced by handle_control_getconf().

4.35.5.589 const char* option_get_canonical_name (const char * *key*)

Return the canonical name of a configuration option, or NULL if no such option exists.

References config_find_option(), and config_var_t::name.

Referenced by handle_control_getconf().

4.35.5.590 int option_is_recognized (const char ∗ *key*)

Return true iff key is a valid configuration option.

References config_find_option().

Referenced by handle_control_getconf().

4.35.5.591 char∗ options_dump (or_options_t ∗ *options*, int *minimal*)

Return a string containing a possible configuration file that would give the configuration in **options**. If **minimal** is true, do not include options that are the same as Tor's defaults.

References config_dump().

Referenced by getinfo_helper_misc(), and write_configuration_file().

4.35.5.592 char∗ options_get_datadir_fname2_suffix (or_options_t ∗ *options*, const char ∗ *sub1*, const char ∗ *sub2*, const char ∗ *suffix*)

Return a newly allocated string holding a filename relative to the data directory. If **sub1** is present, it is the first path component after the data directory. If **sub2** is also present, it is the second path component after the data directory. If **suffix** is present, it is appended to the filename.

Examples: get_datadir_fname2_suffix("a", NULL, NULL) -> $DATADIR/a get_datadir_fname2_suffix("a", NULL, ".tmp") -> $DATADIR/a.tmp get_datadir_fname2_suffix("a", "b", ".tmp") -> $DATADIR/a/b/.tmp get_datadir_fname2_suffix("a", "b", NULL) -> $DATADIR/a/b

Note: Consider using the get_datadir_fname∗ macros in **or.h** (p. 739).

References or_options_t::DataDirectory, tor_assert, and tor_snprintf().

Referenced by try_locking().

4.35.5.593 void options_init (or_options_t ∗ *options*)

Set **options** to hold reasonable defaults for most options. Each option defaults to zero.

References config_init().

Referenced by options_init_from_string().

4.35.5.594 setopt_err_t options_init_from_string (const char ∗ *cf*, int *command*, const char ∗ *command_arg*, char ∗∗ *msg*)

Load the options from the configuration in **cf**, validate them for consistency and take actions based on them.

Return 0 if success, negative on error: ∗ -1 for general errors. ∗ -2 for failure to parse/validate, ∗ -3 for transition not allowed ∗ -4 for error while setting the new options

References or_options_t::_magic, or_options_t::command, or_options_t::command_arg, config_assign(), config_find_option(), config_free(), config_free_lines(), config_get_lines(), config_var_t::initvalue, config_var_t::name, options_init(), options_transition_allowed(), options_validate(), OR_OPTIONS_MAGIC, set_options(), or_options_t::TestingTorNetwork, tor_asprintf(), tor_assert, and tor_free.

Referenced by handle_control_loadconf(), and options_init_from_torrc().

4.35.5.595 int options_init_from_torrc (int *argc*, char ** *argv*)

Read a configuration file into **options**, finding the configuration file location based on the command line. After loading the file call **options_init_from_string()** (p. 395) to load the config. Return 0 if success, -1 if failure.

References config_get_commandlines(), get_version(), LD_CONFIG, list_torrc_options(), load_torrc_from_disk(), options_init_from_string(), print_usage(), and tor_free.

Referenced by do_hup(), and tor_init().

4.35.5.596 int options_need_geoip_info (or_options_t * *options*, const char ** *reason_out*)

If we need to have a GEOIP ip-to-country map to run with our configured options, return 1 and set ***reason_out** to a description of why.

References or_options_t::BridgeRecordUsageByCountry, or_options_t::BridgeRelay, or_options_t::EntryNodes, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, or_options_t::ExitNodes, and routerset_needs_geoip().

Referenced by geoip_load_file().

4.35.5.597 int options_save_current (void)

Save the current configuration file value to disk. Return 0 on success, -1 on failure.

References get_options(), get_torrc_fname(), and write_configuration_file().

Referenced by handle_control_saveconf().

4.35.5.598 setopt_err_t options_trial_assign (config_line_t * *list*, int *use_defaults*, int *clear_first*, char ** *msg*)

Try assigning **list** to the global options. You do this by duping options, assigning list to the new one, then validating it. If it's ok, then throw out the old one and stick with the new one. Else, revert to old and return failure. Return SETOPT_OK on success, or a setopt_err_t on failure.

If not success, point ***msg** to a newly allocated string describing what went wrong.

References config_assign(), config_free(), get_options(), options_dup(), options_transition_allowed(), options_validate(), and set_options().

Referenced by control_setconf_helper().

4.35.5.599 or_circuit_t∗ or_circuit_new (circid_t *p_circ_id*, or_connection_t ∗ *p_conn*)

Allocate a new **or_circuit_t** (p. 135), connected to **p_conn** as **p_circ_id**. If **p_conn** is NULL, the circuit is unattached.

References or_circuit_t::_base, cell_ewma_t::cell_count, cell_ewma_get_tick(), circuit_set_p_circid_orconn(), cell_ewma_t::heap_index, init_circuit_base(), cell_ewma_t::is_for_p_conn, cell_ewma_t::last_adjusted_tick, circuit_t::magic, MAX_RELAY_EARLY_CELLS_PER_CIRCUIT, or_circuit_t::p_cell_ewma, or_circuit_t::remaining_relay_early_cells, and TO_CIRCUIT.

Referenced by command_process_create_cell().

4.35.5.600 or_connection_t∗ or_connection_new (int *socket_family*)

Allocate and return a new **or_connection_t** (p. 139), initialized as by **connection_init()** (p. 420).

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuit_pqueue_last_recalibrated, cell_ewma_get_tick(), CONN_TYPE_OR, connection_init(), crypto_rand_int(), or_connection_t::next_circ_id, smartlist_create(), or_connection_t::timestamp_last_added_nonpadding, and TO_CONN.

Referenced by connection_new(), and connection_or_connect().

4.35.5.601 void or_handshake_state_free (or_handshake_state_t ∗ *state*)

Free all storage held by **state**.

References tor_free.

Referenced by _connection_free(), and connection_or_set_state_open().

4.35.5.602 static INLINE void or_state_mark_dirty (or_state_t ∗ *state*, time_t *when*) `[static]`

Change the next_write time of **state** to **when**, unless the state is already scheduled to be written to disk earlier than **when**.

References or_state_t::next_write.

Referenced by accounting_record_bandwidth_usage(), circuit_build_times_add_time(), circuit_build_times_update_state(), entry_guards_changed(), entry_guards_update_state(), hibernate_begin(), hibernate_go_dormant(), init_keys(), rep_hist_update_state(), rotate_onion_key(), and tor_cleanup().

4.35.5.603 int or_state_save (time_t *now*)

Write the persistent state to disk. Return 0 for success, <0 on failure.

References accounting_is_enabled(), accounting_run_housekeeping(), circ_times, circuit_build_times_update_state(), config_dump(), entry_guards_update_state(), format_local_iso_time(), get_datadir_fname, get_options(), get_version(), or_state_t::LastWritten, LD_FS, LD_GENERAL, or_state_t::next_write, rep_hist_update_state(), tor_asprintf(), tor_assert, tor_free, or_state_t::TorVersion, and write_str_to_file().

Referenced by or_state_load(), run_scheduled_events(), and tor_cleanup().

4.35.5.604 const char∗ orconn_end_reason_to_control_string (int *r*)

Convert the reason for ending an OR connection **r** into the format used in ORCONN events. Return "UNKNOWN" if the reason is unrecognized.

References LD_BUG.

Referenced by connection_about_to_close_connection(), control_event_bootstrap_problem(), and control_event_or_conn_status().

4.35.5.605 origin_circuit_t∗ origin_circuit_init (uint8_t *purpose*, int *flags*)

Create and return a new origin circuit. Initialize its purpose and build-state based on our arguments. The **flags** argument is a bitfield of CIRCLAUNCH_∗ flags.

References origin_circuit_t::_base, origin_circuit_t::build_state, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, CIRCLAUNCH_ONEHOP_TUNNEL, circuit_set_state(), CIRCUIT_STATE_OR_WAIT, cpath_build_state_t::is_internal, cpath_build_state_t::need_capacity, cpath_build_state_t::need_uptime, cpath_build_state_t::onehop_tunnel, origin_circuit_new(), circuit_t::purpose, and TO_CIRCUIT.

Referenced by circuit_establish_circuit(), and handle_control_extendcircuit().

4.35.5.606 origin_circuit_t∗ origin_circuit_new (void)

Allocate space for a new circuit, initializing with **p_circ_id** and **p_conn**. Add it to the global circuit list.

References origin_circuit_t::_base, approx_time(), circ_times, crypto_rand_int(), origin_circuit_t::global_identifier, init_circuit_base(), circuit_build_times_t::last_circ_at, circuit_t::magic, MAX_RELAY_EARLY_CELLS_PER_CIRCUIT, origin_circuit_t::next_stream_id, origin_circuit_t::remaining_relay_early_cells, and TO_CIRCUIT.

Referenced by origin_circuit_init().

4.35.5.607 hostname_type_t parse_extended_hostname (char ∗ *address*, int *allowdotexit*)

If address is of the form "y.onion" with a well-formed handle y: Put a NUL after y, lower-case it, and return ONION_HOSTNAME.

If address is of the form "y.exit" and **allowdotexit** is true: Put a NUL after y and return EXIT_HOSTNAME.

Otherwise: Return NORMAL_HOSTNAME and change nothing.

References LD_APP, REND_SERVICE_ID_LEN_BASE32, and rend_valid_service_id().

4.35.5.608 int parse_http_response (const char ∗ *headers*, int ∗ *code*, time_t ∗ *date*, compress_method_t ∗ *compression*, char ∗∗ *reason*)

Parse an HTTP response string **headers** of the form

```
 * "HTTP/1.\%d \%d\%s\r\n...".
 *
```

If it's well-formed, assign the status code to ∗**code** and return 0. Otherwise, return -1.

On success: If **date** is provided, set ∗date to the Date header in the http headers, or 0 if no such header is found. If **compression** is provided, set ∗**compression** to the compression method given in the Content-Encoding header, or 0 if no such header is found, or -1 if the value of the header is not recognized. If **reason** is provided, strdup the reason string into it.

References escaped(), LD_HTTP, parse_rfc1123_time(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcmpstart(), tor_assert, tor_free, and tor_sscanf().

Referenced by connection_dir_client_reached_eof(), and connection_read_https_proxy_response().

4.35.5.609 int parse_virtual_addr_network (const char ∗ *val*, int *validate_only*, char ∗∗ *msg*)

Read a netmask of the form 127.192.0.0/10 from "val", and check whether it's a valid set of virtual addresses to hand out in response to MAPADDRESS requests. Return 0 on success; set ∗msg (if provided) to a newly allocated string and return -1 on failure. If validate_only is false, sets the actual virtual address range to the parsed value.

References addr_mask_cmp_bits(), next_virtual_addr, parse_addr_and_port_range(), virtual_addr_netmask_bits, and virtual_addr_network.

Referenced by options_act().

4.35.5.610 int peek_buf_has_control0_command (buf_t ∗ *buf*)

Return 1 iff buf looks more like it has an (obsolete) v0 controller command on it than any valid v1 controller command.

References buf_t::datalen, get_uint16(), and peek_from_buf().

Referenced by connection_control_process_inbuf().

4.35.5.611 void policies_free_all (void)

Release all storage held by policy variables.

References addr_policy_list_free(), LD_MM, and policy_write_item().

Referenced by tor_free_all().

4.35.5.612 int policies_parse_exit_policy (config_line_t ∗ *cfg*, smartlist_t ∗∗ *dest*, int *rejectprivate*, const char ∗ *local_address*, int *add_default_policy*)

Parse the exit policy **cfg** into the linked list ∗**dest**. If cfg doesn't end in an absolute accept or reject and if **add_default_policy** is true, add the default exit policy afterwards. If **rejectprivate** is true, prepend "reject private:∗" to the policy. Return -1 if we can't parse cfg, else return 0.

References append_exit_policy_string(), exit_policy_remove_redundancies(), parse_addr_policy(), and tor_snprintf().

Referenced by router_rebuild_descriptor(), and validate_addr_policies().

4.35.5.613 int policies_parse_from_options (or_options_t * *options*)

Set all policies based on **options**, which should have been validated first by validate_addr_policies.

References or_options_t::AuthDirBadDir, or_options_t::AuthDirBadExit, or_options_t::AuthDirInvalid, or_options_t::AuthDirReject, or_options_t::DirPolicy, load_policy_from_option(), parse_reachable_addresses(), and or_options_t::SocksPolicy.

Referenced by options_act().

4.35.5.614 void policies_set_router_exitpolicy_to_reject_all (routerinfo_t * *r*)

Replace the exit policy of **r** with reject *:*.

References addr_policy_list_free(), routerinfo_t::exit_policy, router_parse_addr_policy_item_from_string(), smartlist_add(), and smartlist_create().

Referenced by connection_ap_process_end_not_open().

4.35.5.615 void policy_expand_private (smartlist_t ** *policy*)

Replace all "private" entries in ***policy** with their expanded equivalents.

References addr_policy_t::addr, addr_policy_free(), addr_policy_t::is_canonical, addr_policy_t::is_private, addr_policy_t::maskbits, private_nets, smartlist_add(), smartlist_create(), tor_addr_parse_mask_ports(), and tor_assert.

Referenced by parse_addr_policy(), and router_parse_entry_from_string().

4.35.5.616 int policy_is_reject_star (const smartlist_t * *policy*)

Return false if **policy** might permit access to some addr:port; otherwise if we are certain it rejects everything, return true.

Referenced by connection_ap_can_use_exit(), dns_resolve_impl(), and router_parse_entry_from_string().

4.35.5.617 char* policy_summarize (smartlist_t * *policy*)

Create a string representing a summary for an exit policy. The summary will either be an "accept" plus a comma-separated list of port ranges or a "reject" plus port-ranges, depending on which is shorter.

If no exits are allowed at all then NULL is returned, if no ports are blocked instead of "reject " we return "accept 1-65535" (this is an exception to the shorter-representation-wins rule).

References MAX_EXITPOLICY_SUMMARY_LEN, policy_summary_add_item(), policy_summary_create(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_assert, tor_free, and tor_snprintf().

Referenced by dirvote_create_microdescriptor(), and routerstatus_format_entry().

4.35.5.618 int policy_write_item (char ∗ *buf*, size_t *buflen*, addr_policy_t ∗ *policy*, int *format_for_desc*)

Write a single address policy to the buf_len byte buffer at buf. Return the number of characters written, or -1 on failure.

References addr_policy_t::addr, addr_policy_t::is_private, addr_policy_t::maskbits, addr_policy_t::policy_type, addr_policy_t::prt_max, addr_policy_t::prt_min, tor_addr_family(), tor_addr_to_str(), and tor_snprintf().

Referenced by exit_policy_remove_redundancies(), policies_free_all(), and router_dump_router_to_string().

4.35.5.619 int proxy_mode (or_options_t ∗ *options*)

Return true iff we are trying to be a socks proxy.

References or_options_t::DNSListenAddress, or_options_t::DNSPort, or_options_t::NatdListenAddress, or_options_t::NatdPort, or_options_t::SocksListenAddress, or_options_t::SocksPort, or_options_t::TransListenAddress, and or_options_t::TransPort.

Referenced by circuit_build_needed_circs().

4.35.5.620 int read_to_buf (int *s*, size_t *at_most*, buf_t ∗ *buf*, int ∗ *reached_eof*, int ∗ *socket_error*)

Read from socket **s**, writing onto end of **buf**. Read at most **at_most** bytes, growing the buffer as necessary. If recv() returns 0 (because of EOF), set ∗**reached_eof** to 1 and return 0. Return -1 on error; else return the number of bytes read.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), chunk_t::memlen, MIN_READ_LEN, read_to_chunk(), buf_t::tail, and tor_assert.

Referenced by connection_read_to_buf().

4.35.5.621 int read_to_buf_tls (tor_tls_t ∗ *tls*, size_t *at_most*, buf_t ∗ *buf*)

As read_to_buf, but reads from a TLS connection, and returns a TLS status value rather than the number of bytes read.

Using TLS on OR connections complicates matters in two ways.

First, a TLS stream has its own read buffer independent of the connection's read buffer. (TLS needs to read an entire frame from the network before it can decrypt any data. Thus, trying to read 1 byte from TLS can require that several KB be read from the network and decrypted. The extra data is stored in TLS's decrypt buffer.) Because the data hasn't been read by Tor (it's still inside the TLS), this means that sometimes a connection "has stuff to read" even when poll() didn't return POLLIN. The tor_tls_get_pending_bytes function is used in **connection.c** (p. 406) to detect TLS objects with non-empty internal buffers and read from them again.

Second, the TLS stream's events do not correspond directly to network events: sometimes, before a TLS stream can read, the network must be ready to write – or vice versa.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), chunk_t::memlen, MIN_READ_LEN, read_to_chunk_tls(), buf_t::tail, and tor_assert.

Referenced by connection_read_to_buf().

4.35.5.622 void refresh_all_country_info (void)

Called when we change a node set, or when we reload the geoip list: recompute all country info in all configuration node sets and in the routerlist.

References or_options_t::_ExcludeExitNodesUnion, or_options_t::EntryNodes, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, or_options_t::ExitNodes, get_options(), routerlist_refresh_countries(), and routerset_refresh_countries().

Referenced by geoip_load_file().

4.35.5.623 void relay_header_pack (char ∗ *dest*, const relay_header_t ∗ *src*)

Pack the **relay_header_t** (p. 190) host-order structure **src** into network-order in the buffer **dest**. See tor-spec.txt for details about the wire format.

References relay_header_t::command, relay_header_t::integrity, relay_header_t::length, relay_header_t::recognized, set_uint16(), and relay_header_t::stream_id.

Referenced by relay_digest_matches(), relay_send_command_from_edge(), and relay_set_digest().

4.35.5.624 void relay_header_unpack (relay_header_t ∗ *dest*, const char ∗ *src*)

Unpack the network-order buffer **src** into a host-order **relay_header_t** (p. 190) structure **dest**.

References relay_header_t::command, get_uint16(), relay_header_t::integrity, relay_header_t::length, relay_header_t::recognized, and relay_header_t::stream_id.

Referenced by circuit_extend(), connection_edge_process_relay_cell(), connection_exit_begin_conn(), connection_exit_begin_resolve(), relay_crypt(), relay_digest_matches(), relay_lookup_conn(), and relay_set_digest().

4.35.5.625 int relay_send_command_from_edge (uint16_t *stream_id*, circuit_t ∗ *circ*, uint8_t *relay_command*, const char ∗ *payload*, size_t *payload_len*, crypt_path_t ∗ *cpath_layer*)

Make a relay cell out of **relay_command** and **payload**, and send it onto the open circuit **circ**. **stream_id** is the ID on **circ** for the stream that's sending the relay cell, or 0 if it's a control cell. **cpath_layer** is NULL for OR->OP cells, or the destination hop for OP->OR cells.

If you can't send the cell, mark the circuit for close and return -1. Else return 0.

References approx_time(), CELL_DIRECTION_IN, CELL_DIRECTION_OUT, cell_t::circ_id, CIRCUIT_IS_ORIGIN, circuit_package_relay_cell(), CIRCUIT_PURPOSE_IS_ESTABLISHED_REND, or_connection_t::client_used, relay_header_t::command, cell_t::command, origin_circuit_t::cpath, DIRREQ_END_CELL_SENT, circuit_t::dirreq_id, geoip_change_dirreq_state(), LD_BUG, LD_OR, relay_header_t::length, circuit_t::n_circ_id, circuit_t::n_conn, or_circuit_t::p_circ_id, cell_t::payload, circuit_t::purpose, relay_command_to_string(), origin_circuit_t::relay_early_cells_sent, origin_circuit_t::relay_early_commands, relay_header_pack(), RELAY_HEADER_SIZE, RELAY_PAYLOAD_SIZE, origin_circuit_t::remaining_relay_early_cells, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), relay_header_t::stream_id, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and tor_free.

Referenced by circuit_consider_sending_sendme(), circuit_send_next_onion_skin(), command_process_created_cell(), command_process_destroy_cell(), connection_edge_process_relay_cell(), connection_edge_send_command(), relay_send_end_cell_from_edge(), rend_client_send_establish_rendezvous(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), and router_perform_bandwidth_test().

4.35.5.626 void release_lockfile (void)

If we have successfully acquired the lock file, release it.

References tor_lockfile_unlock().

Referenced by tor_free_all().

4.35.5.627 void remove_file_if_very_old (const char * *fname*, time_t *now*)

Given a file name check to see whether the file exists but has not been modified for a very long time. If so, remove it.

References format_local_iso_time(), and LD_GENERAL.

Referenced by router_reload_router_list_impl(), and router_reload_v2_networkstatus().

4.35.5.628 void rend_cache_clean (void)

Removes all old entries from the service descriptor cache.

References rend_cache_entry_t::parsed, rend_cache, rend_cache_entry_free(), REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, strmap_iter_done(), strmap_iter_get(), strmap_iter_init(), strmap_iter_next(), strmap_iter_next_rmv(), and rend_service_descriptor_t::timestamp.

Referenced by run_scheduled_events().

4.35.5.629 void rend_cache_clean_v2_descs_as_dir (void)

Remove all old v2 descriptors and those for which this hidden service directory is not responsible for any more.

References base32_encode(), DIGEST_LEN, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_iter_next_rmv(), hid_serv_responsible_for_desc_id(), LD_REND, rend_cache_entry_t::parsed, rend_cache_entry_free(), REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_cache_v2_dir, REND_DESC_ID_V2_LEN_BASE32, safe_str_client(), and rend_service_descriptor_t::timestamp.

Referenced by run_scheduled_events().

4.35.5.630 void rend_cache_free_all (void)

Free all storage held by the service descriptor cache.

References digestmap_free(), rend_cache, rend_cache_v2_dir, and strmap_free().

Referenced by tor_free_all().

4.35.5.631 void rend_cache_init (void)

Initializes the service descriptor cache.

References digestmap_new(), rend_cache, and rend_cache_v2_dir.

Referenced by tor_init().

4.35.5.632 int rend_cache_lookup_desc (const char * *query*, int *version*, const char ** *desc*, size_t * *desc_len*)

query is a base-32'ed service id. If it's malformed, return -1. Else look it up.

- If it is found, point *desc to it, and write its length into *desc_len, and return 1.
- If it is not found, return 0. Note: calls to rend_cache_clean or rend_cache_store may invalidate *desc.

References rend_cache_entry_t::desc, rend_cache_entry_t::len, and rend_cache_lookup_entry().

Referenced by directory_handle_command_get().

4.35.5.633 int rend_cache_lookup_entry (const char * *query*, int *version*, rend_cache_entry_t ** *e*)

If we have a cached **rend_cache_entry_t** (p. 193) for the service ID **query** with **version**, set ***e** to that entry and return 1. Else return 0. If **version** is nonnegative, only return an entry in that descriptor format version. Otherwise (if **version** is negative), return the most recent format we have.

References rend_cache, REND_SERVICE_ID_LEN_BASE32, rend_valid_service_id(), strmap_get_lc(), tor_assert, and tor_snprintf().

Referenced by rend_cache_lookup_desc(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_client_remove_intro_point(), and rend_client_send_introduction().

4.35.5.634 int rend_cache_lookup_v2_desc_as_dir (const char * *desc_id*, const char ** *desc*)

Lookup the v2 service descriptor with base32-encoded **desc_id** and copy the pointer to it to ***desc**. Return 1 on success, 0 on well-formed-but-not-found, and -1 on failure.

References base32_decode(), rend_cache_entry_t::desc, DIGEST_LEN, digestmap_get(), hid_serv_responsible_for_desc_id(), LD_REND, rend_cache_v2_dir, REND_DESC_ID_V2_LEN_BASE32, safe_str(), and tor_assert.

Referenced by directory_handle_command_get().

4.35.5.635 int rend_cache_size (void)

Return the number of entries in our rendezvous descriptor cache.

References rend_cache, and strmap_size().

4.35.5.636 int rend_cache_store (const char ∗ *desc*, size_t *desc_len*, int *published*)

Parse ∗desc, calculate its service id, and store it in the cache. If we have a newer v0 descriptor with the same ID, ignore this one. If we have an older descriptor with the same ID, replace it. If we are acting as client due to the published flag and have any v2 descriptor with the same ID, reject this one in order to not get confused with having both versions for the same service.

Return -2 if it's malformed or otherwise rejected; return -1 if we already have a v2 descriptor here; return 0 if it's the same or older than one we've already got; return 1 if it's novel.

The published flag tells us if we store the descriptor in our role as directory (1) or if we cache it as client (0).

References rend_cache_entry_t::desc, LD_BUG, LD_PROTOCOL, LD_REND, rend_cache_entry_t::len, rend_cache_entry_t::parsed, rend_service_descriptor_t::pk, rend_cache_entry_t::received, rend_cache, REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_get_service_id(), rend_parse_service_descriptor(), rend_service_descriptor_free(), REND_SERVICE_ID_LEN_BASE32, safe_str_client(), strmap_get_lc(), strmap_set_lc(), rend_service_descriptor_t::timestamp, tor_assert, tor_free, and tor_snprintf().

Referenced by connection_dir_client_reached_eof(), and directory_handle_command_post().

4.35.5.637 int rend_cache_store_v2_desc_as_client (const char ∗ *desc*, const rend_data_t ∗ *rend_query*)

Parse the v2 service descriptor in **desc**, decrypt the included list of introduction points with **descriptor_cookie** (which may also be **NULL** if decryption is not necessary), and store the descriptor to the local cache under its version and service id.

If we have a newer v2 descriptor with the same ID, ignore this one. If we have an older descriptor with the same ID, replace it. If we have any v0 descriptor with the same ID, reject this one in order to not get confused with having both versions for the same service. Return -2 if it's malformed or otherwise rejected; return -1 if we already have a v0 descriptor here; return 0 if it's the same or older than one we've already got; return 1 if it's novel.

References rend_data_t::auth_type, rend_cache_entry_t::desc, rend_data_t::descriptor_cookie, DIGEST_LEN, rend_service_descriptor_t::intro_nodes, LD_REND, rend_cache_entry_t::len, rend_cache_entry_t::parsed, rend_service_descriptor_t::pk, rend_cache_entry_t::received, rend_cache, REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_decrypt_introduction_points(), rend_get_service_id(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), REND_SERVICE_ID_LEN_BASE32, safe_str_client(), smartlist_create(), strmap_get_lc(), strmap_set_lc(), rend_service_descriptor_t::timestamp, tor_assert, tor_free, tor_mem_is_zero(), and tor_snprintf().

Referenced by connection_dir_client_reached_eof().

4.35.5.638 int rend_cache_store_v2_desc_as_dir (const char ∗ *desc*)

Parse the v2 service descriptor(s) in **desc** and store it/them to the local rend cache. Don't attempt to decrypt the included list of introduction points (as we don't have a descriptor cookie for it).

If we have a newer descriptor with the same ID, ignore this one. If we have an older descriptor with the same ID, replace it. Return -2 if we are not acting as hidden service directory; return -1 if the descriptor(s) were not parsable; return 0 if all descriptors are the same or older than those we've already got; return a positive number for the number of novel stored descriptors.

References base32_encode(), rend_cache_entry_t::desc, DIGEST_LEN, digestmap_get(), digestmap_set(), hid_serv_acting_as_directory(), hid_serv_responsible_for_desc_id(), LD_REND, rend_cache_entry_t::len, rend_cache_entry_t::parsed, rend_cache_entry_t::received, REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_cache_v2_dir, REND_DESC_ID_V2_LEN_BASE32, rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), safe_str(), safe_str_client(), strcmpstart(), rend_service_descriptor_t::timestamp, tor_assert, and tor_free.

Referenced by directory_handle_command_post().

4.35.5.639 void rend_client_desc_trynow (const char * *query*)

Find all the apconns in state AP_CONN_STATE_RENDDESC_WAIT that are waiting on **query**. If there's a working cache entry here with at least one intro point, move them to the next state.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RENDDESC_WAIT, assert_connection_ok(), CONN_TYPE_AP, connection_ap_handshake_attach_circuit(), END_STREAM_REASON_CANT_ATTACH, get_connection_array(), rend_service_descriptor_t::intro_nodes, LD_REND, connection_t::marked_for_close, rend_data_t::onion_address, rend_cache_entry_t::parsed, rend_cache_lookup_entry(), rend_cmp_service_ids(), edge_connection_t::rend_data, safe_str_client(), connection_t::state, connection_t::timestamp_created, connection_t::timestamp_lastread, connection_t::timestamp_lastwritten, TO_CONN, and TO_EDGE_CONN().

Referenced by connection_dir_client_reached_eof(), and rend_client_refetch_v2_renddesc().

4.35.5.640 extend_info_t* rend_client_get_random_intro (const rend_data_t * *rend_query*)

Return a newly allocated extend_info_t* for a randomly chosen introduction point for the named hidden service. Return NULL if all introduction points have been tried and failed.

References crypto_rand_int(), rend_intro_point_t::extend_info, extend_info_dup(), extend_info_free(), extend_info_from_router(), rend_service_descriptor_t::intro_nodes, LD_REND, extend_info_t::nickname, rend_data_t::onion_address, extend_info_t::onion_key, rend_cache_entry_t::parsed, rend_cache_lookup_entry(), rend_intro_point_free(), router_get_by_nickname(), safe_str_client(), and smartlist_del().

Referenced by circuit_get_open_circ_or_launch(), and rend_client_introduction_acked().

4.35.5.641 void rend_client_introcirc_has_opened (origin_circuit_t * *circ*)

Called when we've established a circuit to an introduction point: send the introduction request.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_INTRODUCING, connection_ap_attach_pending(), origin_circuit_t::cpath, LD_REND, circuit_t::purpose, and tor_assert.

Referenced by circuit_has_opened().

4.35.5.642 int rend_client_introduction_acked (origin_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Called when get an ACK or a NAK for a REND_INTRODUCE1 cell.

References origin_circuit_t::_base, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCLAUNCH_IS_INTERNAL, circuit_extend_to_new_exit(), circuit_get_by_rend_query_and_purpose(), circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCE_ACKED, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, escaped_safe_str_client(), extend_info_free(), LD_PROTOCOL, LD_REND, circuit_t::n_circ_id, extend_info_t::nickname, rend_data_t::onion_address, circuit_t::purpose, origin_circuit_t::remaining_relay_early_cells, rend_client_get_random_intro(), rend_client_remove_intro_point(), origin_circuit_t::rend_data, TO_CIRCUIT, and tor_assert.

Referenced by rend_process_relay_cell().

4.35.5.643 rend_service_authorization_t* rend_client_lookup_service_authorization (const char * *onion_address*)

Look up the client-side authorization for the hidden service with **onion_address**. Return NULL if no authorization is available for that address.

References strmap_get(), and tor_assert.

4.35.5.644 int rend_client_receive_rendezvous (origin_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Bob sent us a rendezvous cell; join the circuits.

References origin_circuit_t::_base, origin_circuit_t::build_state, circuit_init_cpath_crypto(), circuit_initial_package_window(), CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCWINDOW_START, connection_ap_attach_pending(), origin_circuit_t::cpath, crypto_dh_compute_secret(), crypto_dh_free(), crypt_path_t::deliver_window, crypt_path_t::dh_handshake_state, DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, LD_REND, onion_append_to_cpath(), crypt_path_t::package_window, cpath_build_state_t::pending_final_cpath, circuit_t::purpose, crypt_path_t::state, TO_CIRCUIT, and tor_assert.

Referenced by rend_process_relay_cell().

4.35.5.645 void rend_client_refetch_v2_renddesc (const rend_data_t * *rend_query*)

Unless we already have a descriptor for **rend_query** with at least one (possibly) working introduction point in it, start a connection to a hidden service directory to fetch a v2 rendezvous service descriptor.

References rend_data_t::auth_type, crypto_rand_int(), rend_data_t::descriptor_cookie, DIGEST_LEN, directory_get_from_hs_dir(), get_options(), LD_REND, rend_data_t::onion_address, rend_cache_lookup_entry(), rend_client_desc_trynow(), rend_compute_v2_desc_id(), REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS, safe_str_client(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), connection_about_to_close_connection(), and rend_client_remove_intro_point().

4.35.5.646 int rend_client_remove_intro_point (extend_info_t * *failed_intro*, const rend_data_t * *rend_query*)

Remove failed_intro from ent. If ent now has no intro points, or service is unrecognized, then launch a new renddesc fetch.

Return -1 if error, 0 if no intro points remain or service unrecognized, 1 if recognized and some intro points remain.

References AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RENDDESC_WAIT, CONN_TYPE_AP, connection_get_by_type_state_rendquery(), DIGEST_LEN, escaped_safe_str_client(), rend_intro_point_t::extend_info, extend_info_t::identity_digest, rend_service_descriptor_t::intro_nodes, LD_BUG, LD_REND, rend_data_t::onion_address, rend_cache_entry_t::parsed, rend_cache_lookup_entry(), rend_client_refetch_v2_renddesc(), rend_intro_point_free(), smartlist_del(), and connection_t::state.

Referenced by _circuit_mark_for_close(), and rend_client_introduction_acked().

4.35.5.647 void rend_client_rendcirc_has_opened (origin_circuit_t * *circ*)

Called when a rendezvous circuit is open; sends a establish rendezvous circuit as appropriate.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_ESTABLISH_REND, LD_REND, circuit_t::purpose, rend_client_send_establish_rendezvous(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), and circuit_has_opened().

4.35.5.648 int rend_client_rendezvous_acked (origin_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Called when we receive a RENDEZVOUS_ESTABLISHED cell; changes the state of the circuit to C_REND_READY.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_REND_READY, connection_ap_attach_pending(), LD_PROTOCOL, LD_REND, circuit_t::purpose, and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.35.5.649 int rend_client_send_introduction (origin_circuit_t * *introcirc*, origin_circuit_t * *rendcirc*)

Called when we're trying to connect an ap conn; sends an INTRODUCE1 cell down introcirc if possible.

References origin_circuit_t::_base, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_READY, DIGEST_LEN, escaped_safe_str_client(), extend_info_t::identity_digest, rend_service_descriptor_t::intro_nodes, LD_REND, rend_data_t::onion_address, rend_cache_entry_t::parsed, circuit_t::purpose, RELAY_PAYLOAD_SIZE, rend_cache_lookup_entry(), rend_cmp_service_ids(), origin_circuit_t::rend_data, and tor_assert.

Referenced by connection_ap_handshake_attach_circuit().

4.35.5.650 int rend_cmp_service_ids (const char ∗ *one*, const char ∗ *two*)

Return 0 if one and two are the same service ids, else -1 or 1

Referenced by circuit_get_by_rend_query_and_purpose(), circuit_is_acceptable(), connection_ap_handshake_attach_circuit(), connection_get_by_type_state_rendquery(), count_established_intro_points(), rend_client_desc_trynow(), and rend_client_send_introduction().

4.35.5.651 int rend_compute_v2_desc_id (char ∗ *desc_id_out*, const char ∗ *service_id*, const char ∗ *descriptor_cookie*, time_t *now*, uint8_t *replica*)

Compute the binary **desc_id_out** (DIGEST_LEN bytes long) for a given base32-encoded **service_id** and optional unencoded **descriptor_cookie** of length REND_DESC_COOKIE_LEN, at time **now** for replica number **replica**. **desc_id** needs to have **DIGEST_LEN** bytes free. Return 0 for success, -1 otherwise.

References base32_decode(), DIGEST_LEN, get_secret_id_part_bytes(), get_time_period(), LD_REND, rend_get_descriptor_id_bytes(), REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS, REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, safe_str(), and safe_str_client().

Referenced by rend_client_refetch_v2_renddesc().

4.35.5.652 int rend_config_services (or_options_t ∗ *options*, int *validate_only*)

Set up rend_service_list, based on the values of HiddenServiceDir and HiddenServicePort in **options**. Return 0 on success and -1 on failure. (If **validate_only** is set, parse, warn and return as normal, but don't actually change the configured services.)

References _circuit_get_global_list(), rend_service_t::auth_type, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, CIRCUIT_STATE_OPEN, rend_authorized_client_t::client_name, rend_service_t::clients, DIGEST_LEN, rend_service_t::directory, rend_service_t::intro_period_started, config_line_t::key, LD_BUG, LD_CONFIG, LD_REND, circuit_t::next, config_line_t::next, parse_port_config(), rend_service_t::ports, rend_add_service(), REND_CLIENTNAME_MAX_LEN, origin_circuit_t::rend_data, REND_LEGAL_CLIENTNAME_CHARACTERS, rend_data_t::rend_pk_digest, rend_service_free(), or_options_t::RendConfigLines, safe_str_client(), smartlist_add(), smartlist_add_all(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_sort_strings(), smartlist_split_string(), smartlist_uniq_strings(), TO_ORIGIN_CIRCUIT(), tor_assert, tor_free, and config_line_t::value.

Referenced by options_act().

4.35.5.653 void rend_consider_descriptor_republication (void)

Consider republication of v2 rendezvous service descriptors that failed previously, but without regenerating descriptor contents.

References rend_service_descriptor_t::all_uploads_performed, rend_service_t::desc, get_options(), and upload_service_descriptor().

Referenced by run_scheduled_events().

4.35.5.654 void rend_consider_services_upload (time_t *now*)

Regenerate and upload rendezvous service descriptors for all services, if necessary. If the descriptor has been dirty enough for long enough, definitely upload; else only upload when the periodic timeout has expired.

For the first upload, pick a random time between now and two periods from now, and pick it independently for each service.

References crypto_rand_int(), rend_service_t::desc_is_dirty, get_options(), rend_service_t::next_upload_time, rend_service_update_descriptor(), or_options_t::RendPostPeriod, and upload_service_descriptor().

Referenced by run_scheduled_events().

4.35.5.655 rend_data_t* rend_data_dup (const rend_data_t * *data*)

Allocate and return a new **rend_data_t** (p. 194) with the same contents as **query**.

References tor_assert.

Referenced by circuit_get_open_circ_or_launch(), connection_exit_begin_conn(), directory_initiate_command_rend(), and rend_service_relaunch_rendezvous().

4.35.5.656 static INLINE void rend_data_free (rend_data_t * *data*) [static]

Free all storage associated with **data**

References tor_free.

Referenced by _connection_free(), and circuit_free().

4.35.5.657 int rend_decrypt_introduction_points (char ** *ipos_decrypted*, size_t * *ipos_decrypted_size*, const char * *descriptor_cookie*, const char * *ipos_encrypted*, size_t *ipos_encrypted_size*)

Decrypt the encrypted introduction points in **ipos_encrypted** of length **ipos_encrypted_size** using **descriptor_cookie** and write the result to a newly allocated string that is pointed to by **ipos_decrypted** and its length to **ipos_decrypted_size**. Return 0 if decryption was successful and -1 otherwise.

References CIPHER_IV_LEN, CIPHER_KEY_LEN, crypto_cipher_decrypt(), crypto_cipher_decrypt_with_iv(), crypto_create_init_cipher(), crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_cipher_env(), crypto_free_digest_env(), crypto_new_digest_env(), LD_REND, memcmpstart(), REND_BASIC_AUTH_CLIENT_ENTRY_LEN, REND_BASIC_AUTH_CLIENT_ID_LEN, REND_BASIC_AUTH_CLIENT_MULTIPLE, REND_DESC_COOKIE_LEN, tor_assert, and tor_free.

Referenced by rend_cache_store_v2_desc_as_client().

4.35.5.658 int rend_encode_v2_descriptors (smartlist_t * *descs_out*, rend_service_descriptor_t * *desc*, time_t *now*, uint8_t *period*, rend_auth_type_t *auth_type*, crypto_pk_env_t * *client_key*, smartlist_t * *client_cookies*)

Encode a set of rend_encoded_v2_service_descriptor_t's for **desc** at time **now** using **service_key**, depending on **auth_type** a **descriptor_cookie** and a list of **client_cookies** (which are both **NULL** if no client authorization is performed), and **period** (e.g. 0 for the current period, 1 for the next period, etc.) and add them to the existing list **descs_out**; return the number of seconds that the descriptors will be found by clients, or -1 if the encoding was not successful.

References base32_encode(), base64_encode(), crypto_digest(), crypto_pk_get_digest(), crypto_pk_write_public_key_to_string(), rend_encoded_v2_service_descriptor_t::desc_id, rend_encoded_v2_service_descriptor_t::desc_str, DIGEST_LEN, format_iso_time(), get_seconds_valid(), get_secret_id_part_bytes(), get_time_period(), rend_service_descriptor_t::intro_nodes, LD_BUG, LD_REND, rend_service_descriptor_t::pk, rend_service_descriptor_t::protocols, REND_DESC_ID_V2_LEN_BASE32, rend_desc_v2_is_parsable(), rend_encode_v2_intro_points(), rend_encoded_v2_service_descriptor_free(), rend_encrypt_v2_intro_points_basic(), rend_encrypt_v2_intro_points_stealth(), rend_get_descriptor_id_bytes(), REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS, REND_SECRET_ID_PART_LEN_BASE32, REND_TIME_PERIOD_V2_DESC_VALIDITY, router_append_dirobj_signature(), smartlist_add(), smartlist_clear(), rend_service_descriptor_t::timestamp, tor_assert, tor_free, and tor_snprintf().

Referenced by upload_service_descriptor().

4.35.5.659 void rend_encoded_v2_service_descriptor_free (rend_encoded_v2_service_descriptor_t * *desc*)

Free the storage held by an encoded v2 service descriptor.

References rend_encoded_v2_service_descriptor_t::desc_str, and tor_free.

Referenced by rend_encode_v2_descriptors(), and upload_service_descriptor().

4.35.5.660 void rend_get_descriptor_id_bytes (char * *descriptor_id_out*, const char * *service_id*, const char * *secret_id_part*)

Compute the descriptor ID for **service_id** of length **REND_SERVICE_ID_LEN** and **secret_id_part** of length **DIGEST_LEN**, and write it to **descriptor_id_out** of length **DIGEST_LEN**.

References crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), DIGEST_LEN, and REND_SERVICE_ID_LEN.

Referenced by rend_compute_v2_desc_id(), rend_encode_v2_descriptors(), and rend_parse_v2_service_descriptor().

4.35.5.661 int rend_get_service_id (crypto_pk_env_t * *pk*, char * *out*)

Sets **out** to the first 10 bytes of the digest of **pk**, base32 encoded. NUL-terminates out. (We use this string to identify services in directory requests and .onion URLs.)

References base32_encode(), crypto_pk_get_digest(), DIGEST_LEN, REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, and tor_assert.

Referenced by rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_service_load_keys(), and upload_service_descriptor().

4.35.5.662 void rend_hsdir_routers_changed (void)

Called when our internal view of the directory has changed, so that we might have router descriptors of hidden service directories available that we did not have before.

Referenced by router_dir_info_changed().

4.35.5.663 int rend_id_is_in_interval (const char ∗ *a*, const char ∗ *b*, const char ∗ *c*)

Determines whether **a** is in the interval of **b** (excluded) and **c** (included) in a circular digest ring; returns 1 if this is the case, and 0 otherwise.

References DIGEST_LEN, and tor_assert.

Referenced by hid_serv_responsible_for_desc_id().

4.35.5.664 void rend_intro_point_free (rend_intro_point_t ∗ *intro*)

Free the storage held by an introduction point info.

References crypto_free_pk_env(), rend_intro_point_t::extend_info, extend_info_free(), rend_intro_point_t::intro_key, and tor_free.

Referenced by rend_client_get_random_intro(), rend_client_remove_intro_point(), rend_parse_introduction_points(), rend_service_descriptor_free(), rend_service_free(), and rend_services_introduce().

4.35.5.665 int rend_mid_establish_intro (or_circuit_t ∗ *circ*, const char ∗ *request*, size_t *request_len*)

Respond to an ESTABLISH_INTRO cell by checking the signed data and setting the circuit's purpose and service pk digest.

References or_circuit_t::_base, base32_encode(), circuit_get_intro_point(), CIRCUIT_PURPOSE_INTRO_POINT, CIRCUIT_PURPOSE_OR, crypto_digest(), crypto_free_pk_env(), crypto_pk_asn1_decode(), crypto_pk_get_digest(), crypto_pk_public_checksig_digest(), DIGEST_LEN, get_uint16(), or_circuit_t::handshake_digest, LD_BUG, LD_GENERAL, LD_PROTOCOL, LD_REND, circuit_t::n_conn, note_crypto_pk_op(), or_circuit_t::p_circ_id, circuit_t::purpose, relay_send_command_from_edge(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, or_circuit_t::rend_token, safe_str(), and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.35.5.666 int rend_mid_establish_rendezvous (or_circuit_t ∗ *circ*, const char ∗ *request*, size_t *request_len*)

Process an ESTABLISH_RENDEZVOUS cell by setting the circuit's purpose and rendezvous cookie.

References or_circuit_t::_base, base16_encode(), circuit_get_rendezvous(), CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_POINT_WAITING, LD_PROTOCOL, LD_REND, circuit_t::n_conn, or_circuit_t::p_circ_id, circuit_t::purpose, relay_send_command_from_edge(), REND_COOKIE_LEN, or_circuit_t::rend_token, and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.35.5.667 int rend_mid_introduce (or_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Process an INTRODUCE1 cell by finding the corresponding introduction circuit, and relaying the body of the INTRODUCE1 cell inside an INTRODUCE2 cell.

References or_circuit_t::_base, base32_encode(), CIPHER_KEY_LEN, circuit_get_intro_point(), CIRCUIT_PURPOSE_OR, DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, LD_REND, MAX_NICKNAME_LEN, circuit_t::n_conn, or_circuit_t::p_circ_id, PKCS1_OAEP_PADDING_OVERHEAD, circuit_t::purpose, relay_send_command_from_edge(), REND_COOKIE_LEN, REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, safe_str(), and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.35.5.668 int rend_mid_rendezvous (or_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Process a RENDEZVOUS1 cell by looking up the correct rendezvous circuit by its relaying the cell's body in a RENDEZVOUS2 cell, and connecting the two circuits.

References or_circuit_t::_base, base16_encode(), circuit_get_rendezvous(), CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_ESTABLISHED, DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, LD_REND, circuit_t::n_conn, or_circuit_t::p_circ_id, circuit_t::purpose, relay_send_command_from_edge(), REND_COOKIE_LEN, or_circuit_t::rend_splice, or_circuit_t::rend_token, and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.35.5.669 int rend_parse_client_keys (strmap_t * *parsed_clients*, const char * *ckstr*)

Parse the content of a client_key file in **ckstr** and add rend_authorized_client_t's for each parsed client to **parsed_clients**. Return the number of parsed clients as result or -1 for failure.

References directory_token_t::args, base64_decode(), rend_authorized_client_t::client_key, rend_authorized_client_t::client_name, rend_authorized_client_t::descriptor_cookie, eat_whitespace(), escaped(), find_opt_by_keyword(), directory_token_t::key, LD_CONFIG, LD_REND, memarea_clear(), memarea_drop_all(), memarea_new(), directory_token_t::n_args, REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, REND_LEGAL_CLIENTNAME_CHARACTERS, smartlist_clear(), smartlist_create(), smartlist_free(), strcmpstart(), strmap_get(), strmap_set(), strmap_size(), token_clear(), tokenize_string(), and tor_assert.

Referenced by rend_service_load_keys().

4.35.5.670 int rend_parse_introduction_points (rend_service_descriptor_t * *parsed*, const char * *intro_points_encoded*, size_t *intro_points_encoded_size*)

Parse the encoded introduction points in **intro_points_encoded** of length **intro_points_encoded_size** and write the result to the descriptor in **parsed**; return the number of successfully parsed introduction points or -1 in case of a failure.

Function may only be invoked once.

References extend_info_t::addr, directory_token_t::args, base16_encode(), base32_decode(), DIGEST_LEN, escaped(), rend_intro_point_t::extend_info, extend_info_t::identity_digest, rend_intro_point_t::intro_key, rend_service_descriptor_t::intro_nodes, directory_token_t::key, LD_REND, memarea_clear(), memarea_drop_all(), memarea_new(), memcmpstart(), extend_info_t::nickname, extend_info_t::onion_key, extend_info_t::port, rend_intro_point_free(), REND_INTRO_POINT_ID_LEN_BASE32, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_addr_family(), tor_addr_from_str(), tor_assert, and tor_parse_long().

Referenced by rend_cache_store_v2_desc_as_client().

4.35.5.671 int rend_parse_service_authorization (or_options_t * *options*, int *validate_only*)

Parse **config_line** as a client-side authorization for a hidden service and add it to the local map of hidden service authorizations. Return 0 for success and -1 for failure.

References rend_service_authorization_t::auth_type, base64_decode(), rend_service_authorization_t::descriptor_cookie, or_options_t::HidServAuth, LD_CONFIG, config_line_t::next, rend_service_authorization_t::onion_address, REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, REND_SERVICE_ADDRESS_LEN, rend_service_authorization_free(), rend_service_authorization_free_all(), rend_service_authorization_strmap_item_free(), REND_SERVICE_ID_LEN_BASE32, rend_valid_service_id(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcmpend(), strmap_free(), strmap_get(), strmap_set(), tor_free, tor_snprintf(), and config_line_t::value.

Referenced by options_act().

4.35.5.672 rend_service_descriptor_t* rend_parse_service_descriptor (const char * *str*, size_t *len*)

Parse a service descriptor at **str** (**len** bytes). On success, return a newly alloced service_descriptor_t. On failure, return NULL.

References crypto_pk_asn1_decode(), crypto_pk_keysize(), crypto_pk_public_checksig_digest(), rend_intro_point_t::extend_info, get_uint16(), get_uint32(), rend_service_descriptor_t::intro_nodes, LD_PROTOCOL, extend_info_t::nickname, note_crypto_pk_op(), rend_service_descriptor_t::pk, rend_service_descriptor_t::protocols, rend_service_descriptor_free(), smartlist_add(), smartlist_create(), rend_service_descriptor_t::timestamp, tor_assert, and rend_service_descriptor_t::version.

Referenced by rend_cache_store().

4.35.5.673 int rend_parse_v2_service_descriptor (rend_service_descriptor_t ** *parsed_out*, char * *desc_id_out*, char ** *intro_points_encrypted_out*, size_t * *intro_points_encrypted_size_out*, size_t * *encoded_size_out*, const char ** *next_out*, const char * *desc*)

Parse and validate the ASCII-encoded v2 descriptor in **desc**, write the parsed descriptor to the newly allocated ∗**parsed_out**, the binary descriptor ID of length DIGEST_LEN to **desc_id_out**, the encrypted introduction points to the newly allocated ∗**intro_points_encrypted_out**, their encrypted size to ∗**intro_points_encrypted_size_out**, the size of the encoded descriptor to ∗**encoded_size_out**, and a pointer to the possibly next descriptor to ∗**next_out**; return 0 for success (including validation) and -1 for failure.

References directory_token_t::args, BASE32_CHARS, base32_decode(), check_signature_token(), crypto_pk_get_digest(), DIGEST_LEN, escaped(), find_opt_by_keyword(), directory_token_t::key, LD_DIR, LD_REND, memarea_drop_all(), memarea_new(), directory_token_t::n_args, note_crypto_pk_op(), directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, parse_iso_time(), rend_service_descriptor_t::pk, rend_service_descriptor_t::protocols, REND_DESC_ID_V2_LEN_BASE32, REND_DESC_MAX_SIZE, rend_get_descriptor_id_bytes(), REND_SECRET_ID_PART_LEN_BASE32, rend_service_descriptor_free(), router_get_hash_impl(), smartlist_create(), smartlist_free(), smartlist_split_string(), rend_service_descriptor_t::timestamp, token_clear(), tokenize_string(), tor_assert, tor_free, tor_parse_long(), and rend_service_descriptor_t::version.

Referenced by rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), and rend_desc_v2_is_parsable().

4.35.5.674 void rend_process_relay_cell (circuit_t * *circ*, const crypt_path_t * *layer_hint*, int *command*, size_t *length*, const char * *payload*)

Called when we get a rendezvous-related relay cell on circuit **circ**. Dispatch on rendezvous relay command.

References CIRCUIT_IS_ORIGIN, origin_circuit_t::cpath, LD_APP, LD_PROTOCOL, crypt_path_t::prev, rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_rendezvous_acked(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_intro_established(), rend_service_introduce(), TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_fragile_assert.

Referenced by connection_edge_process_relay_cell().

4.35.5.675 void rend_service_authorization_free_all (void)

Release all the storage held in auth_hid_servs.

References rend_service_authorization_strmap_item_free(), and strmap_free().

Referenced by rend_parse_service_authorization(), and tor_free_all().

4.35.5.676 void rend_service_descriptor_free (rend_service_descriptor_t * *desc*)

Free the storage held by the service descriptor **desc**.

References crypto_free_pk_env(), rend_service_descriptor_t::intro_nodes, rend_service_-

descriptor_t::pk, rend_intro_point_free(), smartlist_free(), rend_service_descriptor_t::successful_uploads, and tor_free.

Referenced by rend_cache_entry_free(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_desc_v2_is_parsable(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_free(), and rend_service_update_descriptor().

4.35.5.677 void rend_service_dump_stats (int *severity*)

Log the status of introduction points for all rendezvous services at log severity **severity**.

References origin_circuit_t::_base, circuit_state_to_string(), rend_service_t::directory, rend_intro_point_t::extend_info, find_intro_circuit(), rend_service_t::intro_nodes, LD_GENERAL, extend_info_t::nickname, rend_service_t::pk_digest, safe_str_client(), and circuit_t::state.

Referenced by dumpstats().

4.35.5.678 void rend_service_free_all (void)

Release all the storage held in rend_service_list.

References rend_service_free(), and smartlist_free().

Referenced by tor_free_all().

4.35.5.679 int rend_service_intro_established (origin_circuit_t ∗ *circuit*, const char ∗ *request*, size_t *request_len*)

Called when we get an INTRO_ESTABLISHED cell; mark the circuit as a live introduction point, and note that the service descriptor is now out-of-date.

References origin_circuit_t::_base, base32_encode(), CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, rend_service_t::desc_is_dirty, LD_PROTOCOL, LD_REND, circuit_t::n_circ_id, circuit_t::purpose, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, TO_CIRCUIT, and tor_assert.

Referenced by rend_process_relay_cell().

4.35.5.680 void rend_service_intro_has_opened (origin_circuit_t ∗ *circuit*)

Called when we're done building a circuit to an introduction point: sends a RELAY_ESTABLISH_INTRO cell.

References origin_circuit_t::_base, base32_encode(), circuit_has_opened(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, count_established_intro_points(), origin_circuit_t::cpath, crypto_digest(), crypto_pk_asn1_encode(), crypto_pk_private_sign_digest(), DIGEST_LEN, crypt_path_t::handshake_digest, origin_circuit_t::intro_key, LD_BUG, LD_CIRC, LD_GENERAL, LD_REND, circuit_t::n_circ_id, note_crypto_pk_op(), NUM_INTRO_POINTS, crypt_path_t::prev, circuit_t::purpose, RELAY_PAYLOAD_SIZE, relay_send_command_from_edge(), origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(),

REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, set_uint16(), TO_CIRCUIT, and tor_assert.

Referenced by circuit_has_opened(), and rend_service_launch_establish_intro().

4.35.5.681 int rend_service_introduce (origin_circuit_t * *circuit*, const char * *request*, size_t *request_len*)

Respond to an INTRODUCE2 cell by launching a circuit to the chosen rendezvous point.

References origin_circuit_t::_base, rend_service_t::accepted_intros, extend_info_t::addr, base16_encode(), base32_encode(), origin_circuit_t::build_state, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, circuit_init_cpath_crypto(), circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_INTRO, clean_accepted_intros(), rend_service_t::clients, crypto_dh_compute_secret(), crypto_dh_free(), crypto_dh_generate_public(), crypto_dh_new(), crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), crypto_pk_asn1_decode(), crypto_pk_get_digest(), crypto_pk_keysize(), crypto_pk_private_hybrid_decrypt(), crypt_path_t::dh_handshake_state, DIGEST_LEN, digestmap_get(), digestmap_new(), digestmap_set(), escaped(), escaped_safe_str_client(), cpath_build_state_t::expiry_time, extend_info_free(), extend_info_from_router(), get_uint16(), get_uint32(), crypt_path_t::handshake_digest, extend_info_t::identity_digest, origin_circuit_t::intro_key, is_legal_nickname(), is_legal_nickname_or_hexdigest(), rend_service_t::last_cleaned_accepted_intros, LD_BUG, LD_PROTOCOL, LD_REND, crypt_path_t::magic, MAX_HEX_NICKNAME_LEN, MAX_NICKNAME_LEN, MAX_REND_FAILURES, MAX_REND_TIMEOUT, circuit_t::n_circ_id, extend_info_t::nickname, note_crypto_pk_op(), rend_data_t::onion_address, extend_info_t::onion_key, cpath_build_state_t::pending_final_cpath, PK_PKCS1_OAEP_PADDING, extend_info_t::port, circuit_t::purpose, RELAY_PAYLOAD_SIZE, rend_check_authorization(), rend_data_t::rend_cookie, REND_COOKIE_LEN, origin_circuit_t::rend_data, REND_DESC_COOKIE_LEN, rend_data_t::rend_pk_digest, REND_REPLAY_TIME_INTERVAL, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, rend_service_requires_uptime(), rep_hist_note_used_internal(), router_get_by_nickname(), rend_service_t::service_id, TO_CIRCUIT, tor_addr_from_ipv4n(), and tor_assert.

Referenced by rend_process_relay_cell().

4.35.5.682 int rend_service_load_keys (void)

Load and/or generate private keys for all hidden services, possibly including keys for client authorization. Return 0 on success, -1 on failure.

References abort_writing_to_file(), base64_encode(), check_private_dir(), rend_authorized_client_t::client_key, crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_check_key(), crypto_pk_dup_key(), crypto_pk_get_digest(), crypto_pk_write_private_key_to_string(), crypto_rand(), rend_authorized_client_t::descriptor_cookie, escaped(), finish_writing_to_file(), init_key_from_file(), LD_BUG, LD_CONFIG, LD_FS, LD_REND, read_file_to_str(), rend_authorized_client_strmap_item_free(), REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, rend_get_service_id(), rend_parse_client_keys(), RFTS_IGNORE_MISSING, start_writing_to_stdio_file(), strmap_free(), strmap_get(), strmap_size(), tor_free, tor_snprintf(), and write_str_to_file().

Referenced by options_act().

4.35.5.683 void rend_service_relaunch_rendezvous (origin_circuit_t ∗ *oldcirc*)

Called when we fail building a rendezvous circuit at some point other than the last hop: launches a new circuit to the same rendezvous point.

References origin_circuit_t::_base, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_S_CONNECT_REND, cpath_build_state_t::expiry_time, cpath_build_state_t::failure_count, LD_REND, MAX_REND_FAILURES, extend_info_t::nickname, cpath_build_state_t::pending_final_cpath, circuit_t::purpose, origin_circuit_t::rend_data, rend_data_dup(), and tor_assert.

Referenced by circuit_build_failed().

4.35.5.684 void rend_service_rendezvous_has_opened (origin_circuit_t ∗ *circuit*)

Called once a circuit to a rendezvous point is established: sends a RELAY_COMMAND_RENDEZVOUS1 cell.

References origin_circuit_t::_base, base16_encode(), base32_encode(), origin_circuit_t::build_state, circuit_initial_package_window(), CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_REND_JOINED, CIRCWINDOW_START, origin_circuit_t::cpath, crypto_dh_free(), crypto_dh_get_public(), crypt_path_t::deliver_window, crypt_path_t::dh_handshake_state, DIGEST_LEN, crypt_path_t::handshake_digest, LD_GENERAL, LD_REND, circuit_t::n_circ_id, onion_append_to_cpath(), crypt_path_t::package_window, cpath_build_state_t::pending_final_cpath, crypt_path_t::prev, circuit_t::purpose, RELAY_PAYLOAD_SIZE, relay_send_command_from_edge(), rend_data_t::rend_cookie, REND_COOKIE_LEN, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, crypt_path_t::state, TO_CIRCUIT, and tor_assert.

Referenced by circuit_has_opened().

4.35.5.685 int rend_service_set_connection_addr_port (edge_connection_t ∗ *conn*, origin_circuit_t ∗ *circ*)

Given **conn**, a rendezvous exit stream, look up the hidden service for 'circ', and look up the port and address based on conn->port. Assign the actual conn->addr and conn->port. Return -1 if failure, or 0 for success.

References edge_connection_t::_base, origin_circuit_t::_base, base32_encode(), CIRCUIT_PURPOSE_S_REND_JOINED, LD_REND, circuit_t::n_circ_id, connection_t::port, rend_service_t::ports, circuit_t::purpose, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, smartlist_add(), smartlist_create(), and tor_assert.

Referenced by connection_exit_begin_conn().

4.35.5.686 void rend_services_introduce (void)

For every service, check how many intro points it currently has, and:

- Pick new intro points as necessary.

- Launch circuits to any new intro points.

References rend_service_t::desc, rend_service_t::desc_is_dirty, DIGEST_LEN, rend_intro_point_t::extend_info, find_intro_circuit(), get_options(), extend_info_t::identity_digest, INTRO_CIRC_RETRY_PERIOD, rend_service_descriptor_t::intro_nodes, rend_service_t::intro_nodes, rend_service_t::intro_period_started, LD_REND, MAX_INTRO_CIRCS_PER_PERIOD, rend_service_t::n_intro_circuits_launched, extend_info_t::nickname, rend_service_t::pk_digest, rend_intro_point_free(), router_get_by_digest(), rend_service_t::service_id, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_del(), and tor_assert.

Referenced by circuit_build_needed_circs().

4.35.5.687 int rend_valid_service_id (const char * *query*)

Return true iff **query** is a syntactically valid service ID (as generated by rend_get_service_id).

References BASE32_CHARS, and REND_SERVICE_ID_LEN_BASE32.

Referenced by parse_extended_hostname(), rend_cache_lookup_entry(), and rend_parse_service_authorization().

4.35.5.688 int rep_hist_bandwidth_assess (void)

Find the largest sums in the past NUM_SECS_BW_SUM_IS_VALID (roughly) seconds. Find one sum for reading and one for writing. They don't have to be at the same time.

Return the smaller of these sums, divided by NUM_SECS_ROLLING_MEASURE.

References find_largest_max(), and NUM_SECS_ROLLING_MEASURE.

Referenced by check_descriptor_bandwidth_changed(), and router_rebuild_descriptor().

4.35.5.689 void rep_hist_buffer_stats_add_circ (circuit_t * *circ*, time_t *end_of_interval*)

Remember cell statistics for circuit **circ** at time **end_of_interval** and reset cell counters in case the circuit remains open in the next measurement interval.

References CIRCUIT_IS_ORIGIN, or_circuit_t::processed_cells, smartlist_add(), smartlist_create(), start_of_buffer_stats_interval, circuit_t::timestamp_created, TO_OR_CIRCUIT(), and or_circuit_t::total_cell_waiting_time.

Referenced by circuit_free(), and rep_hist_buffer_stats_write().

4.35.5.690 void rep_hist_buffer_stats_init (time_t *now*)

Initialize buffer stats.

References start_of_buffer_stats_interval.

Referenced by run_scheduled_events().

4.35.5.691 void rep_hist_buffer_stats_write (time_t *now*)

Write buffer statistics to $DATADIR/stats/buffer-stats.

References _buffer_stats_compare_entries(), _circuit_get_global_list(), abort_writing_to_file(), check_private_dir(), finish_writing_to_file(), format_iso_time(), get_datadir_fname, get_datadir_fname2, LD_HIST, circuit_t::next, rep_hist_buffer_stats_add_circ(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), start_of_buffer_stats_interval, start_writing_to_stdio_file(), tor_asprintf(), and tor_free.

Referenced by run_scheduled_events().

4.35.5.692 int rep_hist_circbuilding_dormant (time_t *now*)

Return 1 if we have no need for circuits currently, else return 0.

References any_predicted_circuits(), check_whether_dirport_reachable(), check_whether_orport_reachable(), circuit_enough_testing_circs(), get_options(), and server_mode().

Referenced by directory_too_idle_to_fetch_descriptors().

4.35.5.693 time_t rep_hist_downrate_old_runs (time_t *now*)

Helper: Discount all old MTBF data, if it is time to do so. Return the time at which we should next discount MTBF data.

References digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_new(), history_map, LD_HIST, STABILITY_ALPHA, STABILITY_INTERVAL, stability_last_downrated, or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by run_scheduled_events().

4.35.5.694 void rep_hist_dump_stats (time_t *now*, int *severity*)

Log all the reliability data we have remembered, with the chosen severity.

References base16_encode(), DIGEST_LEN, digestmap_isempty(), digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), or_history_t::downtime, get_options(), get_stability(), HEX_DIGEST_LEN, history_map, LD_HIST, or_history_t::link_history_map, or_history_t::n_conn_fail, or_history_t::n_conn_ok, link_history_t::n_extend_fail, link_history_t::n_extend_ok, routerinfo_t::nickname, rep_history_clean(), router_get_by_digest(), tor_snprintf(), update_or_history(), and or_history_t::uptime.

Referenced by dumpstats().

4.35.5.695 void rep_hist_exit_stats_init (time_t *now*)

Initialize exit port stats.

References exit_bytes_read, exit_bytes_written, EXIT_STATS_NUM_PORTS, exit_streams, and start_of_exit_stats_interval.

Referenced by run_scheduled_events().

4.35.5.696 void rep_hist_exit_stats_write (time_t *now*)

Write exit stats to $DATADIR/stats/exit-stats and reset counters.

References abort_writing_to_file(), check_private_dir(), exit_bytes_read, exit_bytes_written, EXIT_STATS_NUM_PORTS, EXIT_STATS_ROUND_UP_BYTES, EXIT_STATS_ROUND_UP_STREAMS, EXIT_STATS_THRESHOLD_RECIPROCAL, exit_streams, finish_writing_to_file(), format_iso_time(), get_datadir_fname, get_datadir_fname2, LD_HIST, round_uint32_to_next_multiple_of(), round_uint64_to_next_multiple_of(), start_of_exit_stats_interval, start_writing_to_stdio_file(), tor_assert, and tor_free.

Referenced by run_scheduled_events().

4.35.5.697 void rep_hist_free_all (void)

Free all storage held by the OR/link history caches, by the bandwidth history arrays, or by the port history.

References built_last_stability_doc_at, digestmap_free(), exit_bytes_read, exit_bytes_written, exit_streams, free_or_history(), history_map, last_stability_doc, predicted_ports_free(), and tor_free.

Referenced by tor_free_all().

4.35.5.698 char* rep_hist_get_bandwidth_lines (int *for_extrainfo*)

Allocate and return lines for representing this server's bandwidth history in its descriptor.

References format_iso_time(), bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, NUM_TOTALS, rep_hist_fill_bandwidth_history(), tor_assert, and tor_snprintf().

Referenced by extrainfo_dump_to_string().

4.35.5.699 int rep_hist_get_predicted_internal (time_t *now*, int * *need_uptime*, int * *need_capacity*)

Return 1 if we've used an internal circ recently; else return 0.

References PREDICTED_CIRCS_RELEVANCE_TIME, predicted_internal_capacity_time, predicted_internal_time, and predicted_internal_uptime_time.

Referenced by circuit_predict_and_launch_new().

4.35.5.700 smartlist_t* rep_hist_get_predicted_ports (time_t *now*)

Return a pointer to the list of port numbers that are likely to be asked for in the near future.

The caller promises not to mess with it.

References LD_CIRC, PREDICTED_CIRCS_RELEVANCE_TIME, rephist_total_alloc, smartlist_del(), tor_assert, and tor_free.

Referenced by circuit_get_unhandled_ports().

4.35.5.701 const char* rep_hist_get_router_stability_doc (time_t *now*)

Return a pointer to a NUL-terminated document describing our view of the stability of the routers we've been tracking. Return NULL on failure.

References routerinfo_t::addr, BASE64_DIGEST_LEN, built_last_stability_doc_at, routerinfo_t::cache_info, digest_to_base64(), format_iso_time(), history_map, routerinfo_t::is_hibernating, routerinfo_t::is_running, routerinfo_t::is_valid, last_stability_doc, MAX_STABILITY_DOC_BUILD_RATE, routerinfo_t::nickname, signed_descriptor_t::published_on, rep_hist_format_router_status(), rep_hist_have_measured_enough_stability(), router_get_by_digest(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_dup_ip(), tor_free, tor_snprintf(), and routerinfo_t::uptime.

Referenced by directory_handle_command_get().

4.35.5.702 double rep_hist_get_stability (const char ∗ *id*, time_t *when*)

Return an estimated MTBF for the router whose identity digest is **id**. Return 0 if the router is unknown.

References get_or_history(), and get_stability().

Referenced by dirserv_compute_performance_thresholds(), and dirserv_thinks_router_is_unreliable().

4.35.5.703 double rep_hist_get_weighted_fractional_uptime (const char ∗ *id*, time_t *when*)

Return an estimated percent-of-time-online for the router whose identity digest is **id**. Return 0 if the router is unknown.

References get_or_history(), and get_weighted_fractional_uptime().

Referenced by set_routerstatus_from_routerinfo().

4.35.5.704 long rep_hist_get_weighted_time_known (const char ∗ *id*, time_t *when*)

Return a number representing how long we've known about the router whose digest is **id**. Return 0 if the router is unknown.

Be careful: this measure increases monotonically as we know the router for longer and longer, but it doesn't increase linearly.

References get_or_history(), and get_total_weighted_time().

Referenced by dirserv_compute_performance_thresholds(), and set_routerstatus_from_routerinfo().

4.35.5.705 int rep_hist_have_measured_enough_stability (void)

Return true if we've been measuring MTBFs for long enough to pronounce on Stability.

References started_tracking_stability.

Referenced by rep_hist_get_router_stability_doc().

4.35.5.706 void rep_hist_init (void)

Initialize the static data structures for tracking history.

References bw_arrays_init(), digestmap_new(), history_map, and predicted_ports_init().

Referenced by tor_init().

4.35.5.707 int rep_hist_load_mtbf_data (time_t *now*)

Load MTBF data from disk. Returns 0 on success or recoverable error, -1 on failure.

References base16_decode(), correct_time(), DIGEST_LEN, escaped(), find_next_with(), get_datadir_fname, get_or_history(), HEX_DIGEST_LEN, LD_HIST, n_bogus_times, parse_iso_time(), parse_possibly_bad_iso_time(), read_file_to_str(), RFTS_IGNORE_MISSING, smartlist_create(), smartlist_free(), smartlist_split_string(), stability_last_downrated, or_history_t::start_of_downtime, or_history_t::start_of_run, started_tracking_stability, strcmpstart(), tor_free, tor_parse_long(), or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by options_act().

4.35.5.708 int rep_hist_load_state (or_state_t * *state*, char ** *err*)

Set bandwidth history from our saved state.

References add_obs(), bw_array_new(), or_state_t::BWHistoryReadEnds, or_state_t::BWHistoryReadInterval, or_state_t::BWHistoryReadValues, or_state_t::BWHistoryWriteEnds, or_state_t::BWHistoryWriteInterval, or_state_t::BWHistoryWriteValues, bw_array_t::cur_obs_time, LD_HIST, bw_array_t::max_total, bw_array_t::maxima, bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, NUM_SECS_ROLLING_MEASURE, NUM_TOTALS, bw_array_t::obs, tor_assert, tor_free, tor_parse_uint64(), and bw_array_t::total_obs.

Referenced by or_state_set().

4.35.5.709 void rep_hist_note_bytes_read (size_t *num_bytes*, time_t *when*)

We wrote **num_bytes** more bytes in second **when**. (like **rep_hist_note_bytes_written()** (p. 980) above)

References add_obs().

Referenced by connection_buckets_decrement().

4.35.5.710 void rep_hist_note_bytes_written (size_t *num_bytes*, time_t *when*)

We read **num_bytes** more bytes in second **when**.

Add num_bytes to the current running total for **when**.

when can go back to time, but it's safe to ignore calls earlier than the latest **when** you've heard of.

References add_obs().

Referenced by connection_buckets_decrement().

4.35.5.711 void rep_hist_note_connect_failed (const char * *id*, time_t *when*)

Remember that an attempt to connect to the OR with identity digest **id** failed at **when**.

References or_history_t::changed, get_or_history(), mark_or_down(), and or_history_t::n_conn_fail.

Referenced by connection_about_to_close_connection().

4.35.5.712 void rep_hist_note_connect_succeeded (const char * *id*, time_t *when*)

Remember that an attempt to connect to the OR with identity digest **id** succeeded at **when**.

References or_history_t::changed, get_or_history(), mark_or_up(), and or_history_t::n_conn_ok.

Referenced by connection_or_set_state_open().

4.35.5.713 void rep_hist_note_connection_died (const char * *id*, time_t *when*)

Remember that our connection to the OR with identity digest **id** had an error and stopped working at **when**.

References or_history_t::changed, get_or_history(), and mark_or_down().

Referenced by connection_about_to_close_connection().

4.35.5.714 void rep_hist_note_disconnect (const char * *id*, time_t *when*)

Remember that we intentionally closed our connection to the OR with identity digest **id** at **when**.

References or_history_t::changed, get_or_history(), and mark_or_down().

Referenced by connection_about_to_close_connection().

4.35.5.715 void rep_hist_note_exit_bytes_read (uint16_t *port*, size_t *num_bytes*)

Note that we read **num_bytes** from an exit connection to **port**.

References exit_bytes_read, get_options(), and LD_HIST.

Referenced by connection_buckets_decrement().

4.35.5.716 void rep_hist_note_exit_bytes_written (uint16_t *port*, size_t *num_bytes*)

Note that we wrote **num_bytes** to an exit connection to **port**.

References exit_bytes_written, get_options(), and LD_HIST.

Referenced by connection_buckets_decrement().

4.35.5.717 void rep_hist_note_exit_stream_opened (uint16_t *port*)

Note that we opened an exit stream to **port**.

References exit_streams, get_options(), and LD_HIST.

Referenced by connection_edge_finished_connecting().

4.35.5.718 void rep_hist_note_extend_failed (const char ∗ *from_id*, const char ∗ *to_id*)

Remember that we tried to extend from the OR with identity digest **from_id** to the OR with identity digest **to_name**, but failed.

References link_history_t::changed, get_link_history(), and link_history_t::n_extend_fail.

Referenced by circuit_rep_hist_note_result().

4.35.5.719 void rep_hist_note_extend_succeeded (const char ∗ *from_id*, const char ∗ *to_id*)

Remember that we successfully extended from the OR with identity digest **from_id** to the OR with identity digest **to_name**.

References link_history_t::changed, get_link_history(), and link_history_t::n_extend_ok.

Referenced by circuit_rep_hist_note_result().

4.35.5.720 void rep_hist_note_router_reachable (const char ∗ *id*, time_t *when*)

We have just decided that this router with identity digest **id** is reachable, meaning we will give it a "Running" flag for the next while.

References DIGEST_LEN, format_local_iso_time(), get_or_history(), hex_str(), LD_HIST, or_history_t::start_of_downtime, or_history_t::start_of_run, started_tracking_stability, tor_assert, and or_history_t::total_weighted_time.

Referenced by dirserv_orconn_tls_done().

4.35.5.721 void rep_hist_note_router_unreachable (const char ∗ *id*, time_t *when*)

We have just decided that this router is unreachable, meaning we are taking away its "Running" flag.

References DIGEST_LEN, format_local_iso_time(), get_or_history(), hex_str(), LD_HIST, or_history_t::start_of_downtime, or_history_t::start_of_run, started_tracking_stability, tor_assert, or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by dirserv_set_router_is_running(), rep_hist_record_mtbf_data(), and routerlist_remove().

4.35.5.722 void rep_hist_note_used_internal (time_t *now*, int *need_uptime*, int *need_capacity*)

Remember that we used an internal circ at time **now**.

References predicted_internal_capacity_time, predicted_internal_time, and predicted_internal_uptime_time.

Referenced by circuit_get_open_circ_or_launch(), directory_initiate_command_rend(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.35.5.723 void rep_hist_note_used_port (time_t *now*, uint16_t *port*)

Remember that **port** has been asked for as of time **now**. This is used for predicting what sorts of streams we'll make in the future and making exit circuits to anticipate that.

References add_predicted_port(), and tor_assert.

Referenced by directory_initiate_command_rend(), and rep_hist_note_used_resolve().

4.35.5.724 void rep_hist_note_used_resolve (time_t *now*)

The user asked us to do a resolve. Rather than keeping track of timings and such of resolves, we fake it for now by treating it the same way as a connection to port 80. This way we will continue to have circuits lying around if the user only uses Tor for resolves.

References rep_hist_note_used_port().

4.35.5.725 int rep_hist_record_mtbf_data (time_t *now*, int *missing_means_down*)

Write MTBF data to disk. Return 0 on success, negative on failure.

If **missing_means_down**, then if we're about to write an entry that is still considered up but isn't in our routerlist, consider it to be down.

References abort_writing_to_file(), base16_encode(), DIGEST_LEN, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), finish_writing_to_file(), format_iso_time(), get_datadir_fname, HEX_DIGEST_LEN, history_map, LD_HIST, rep_hist_note_router_unreachable(), router_get_by_digest(), stability_last_downrated, or_history_t::start_of_downtime, or_history_t::start_of_run, start_writing_to_stdio_file(), started_tracking_stability, tor_free, or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by run_scheduled_events(), and tor_cleanup().

4.35.5.726 void rep_hist_update_state (or_state_t * *state*)

Update **state** with the newest bandwidth history.

References or_options_t::AvoidDiskWrites, or_state_t::BWHistoryReadEnds, or_state_t::BWHistoryReadInterval, or_state_t::BWHistoryReadValues, or_state_t::BWHistoryWriteEnds, or_state_t::BWHistoryWriteInterval, or_state_t::BWHistoryWriteValues, get_options(), get_or_state(), bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, NUM_TOTALS, or_state_mark_dirty(), rep_hist_fill_-

bandwidth_history(), server_mode(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, tor_snprintf(), and bw_array_t::total_in_period.

Referenced by or_state_save().

4.35.5.727 void rep_history_clean (time_t *before*)

Remove history info for routers/links that haven't changed since **before**.

References authdir_mode(), link_history_t::changed, or_history_t::changed, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_iter_next_rmv(), free_or_history(), get_options(), history_map, or_history_t::link_history_map, rephist_total_alloc, STABILITY_EPSILON, or_history_t::start_of_run, tor_free, and or_history_t::total_run_weights.

Referenced by rep_hist_dump_stats(), and run_scheduled_events().

4.35.5.728 void reset_bandwidth_test (void)

Reset have_performed_bandwidth_test, so we'll start building testing circuits again so we can exercise our bandwidth.

References have_performed_bandwidth_test.

Referenced by ip_address_changed(), and run_scheduled_events().

4.35.5.729 int resolve_my_address (int *warn_severity*, or_options_t * *options*, uint32_t * *addr_out*, char ** *hostname_out*)

Based on **options->Address**, guess our public IP address and put it (in host order) into ***addr_out**. If **hostname_out** is provided, set ***hostname_out** to a new string holding the hostname we used to get the address. Return 0 if all is well, or -1 if we can't find a suitable public IP address.

References or_options_t::_PublishServerDescriptor, or_options_t::Address, or_options_t::AlternateDirAuthority, control_event_server_status(), or_options_t::DirServers, get_interface_address(), INET_NTOA_BUF_LEN, ip_address_changed(), is_internal_IP(), LD_CONFIG, LD_NET, tor_assert, tor_inet_aton(), tor_inet_ntoa(), and tor_lookup_hostname().

Referenced by add_trusted_dir_server(), check_descriptor_ipaddress_changed(), dirserv_generate_networkstatus_vote_obj(), generate_v2_networkstatus_opinion(), options_validate(), router_new_address_suggestion(), and router_pick_published_address().

4.35.5.730 int retry_all_listeners (smartlist_t * *replaced_conns*, smartlist_t * *new_conns*)

Launch listeners for each port you should have open. Only launch listeners who are not already open, and only close listeners we no longer want.

Add all old conns that should be closed to **replaced_conns**. Add all new connections to **new_conns**.

References or_options_t::ClientOnly, CONN_TYPE_AP_DNS_LISTENER, CONN_TYPE_AP_LISTENER, CONN_TYPE_AP_NATD_LISTENER, CONN_TYPE_AP_TRANS_LISTENER, CONN_TYPE_CONTROL_LISTENER, CONN_TYPE_DIR_LISTENER, CONN_TYPE_OR_LISTENER, or_options_t::ControlListenAddress, or_-

options_t::ControlPort, or_options_t::ControlSocket, or_options_t::DirListenAddress, or_options_t::DirPort, or_options_t::DNSListenAddress, or_options_t::DNSPort, get_options(), or_options_t::NatdListenAddress, or_options_t::NatdPort, or_options_t::ORListenAddress, or_options_t::ORPort, retry_listeners(), or_options_t::SocksListenAddress, or_options_t::SocksPort, or_options_t::TransListenAddress, and or_options_t::TransPort.

Referenced by options_act_reversible(), and run_scheduled_events().

4.35.5.731 void retry_bridge_descriptor_fetch_directly (const char ∗ *digest*)

Fetching the bridge descriptor from the bridge authority returned a "not found". Fall back to trying a direct fetch.

References find_bridge_by_digest(), and launch_direct_bridge_descriptor_fetch().

Referenced by dir_routerdesc_download_failed().

4.35.5.732 void rotate_onion_key (void)

Replace the previous onion key with the current onion key, and generate a new previous onion key. Immediately after calling this function, the OR should:

- schedule all previous cpuworkers to shut down _after_ processing pending work. (This will cause fresh cpuworkers to be generated.)
- generate and upload a fresh routerinfo.

References crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_write_private_key_to_filename(), file_status(), get_datadir_fname2, get_options(), get_or_state(), or_state_t::LastRotatedOnionKey, LD_BUG, LD_FS, LD_GENERAL, mark_my_descriptor_dirty(), onionkey_set_at, or_state_mark_dirty(), replace_file(), tor_free, tor_mutex_acquire(), and tor_mutex_release().

Referenced by run_scheduled_events().

4.35.5.733 was_router_added_t router_add_extrainfo_to_routerlist (extrainfo_t ∗ *ei*, const char ∗∗ *msg*, int *from_cache*, int *from_fetch*)

Insert **ei** into the routerlist, or free it. Other arguments are as for **router_add_to_routerlist()** (p. 985). Return ROUTER_ADDED_SUCCESSFULLY iff we actually inserted it, ROUTER_BAD_EI otherwise.

References extrainfo_t::cache_info, extrainfo_insert(), routerlist_t::extrainfo_store, router_get_routerlist(), routerlist, and signed_desc_append_to_journal().

Referenced by dirserv_add_extrainfo(), and router_load_extrainfo_from_string().

4.35.5.734 was_router_added_t router_add_to_routerlist (routerinfo_t ∗ *router*, const char ∗∗ *msg*, int *from_cache*, int *from_fetch*)

Add **router** to the routerlist, if we don't already have it. Replace older entries (if any) with the same key. Note: Callers should not hold their pointers to **router** if this function fails; **router** will either be inserted into the routerlist or freed. Similarly, even if this call succeeds, they should not hold their pointers to **router** after subsequent calls with other routerinfo's – they might cause the original routerinfo to get freed.

Returns the status for the operation. Might set ∗**msg** if it wants the poster of the router to know something.

If **from_cache**, this descriptor came from our disk cache. If **from_fetch**, we received it in response to a request we made. (If both are false, that means it was uploaded to us as an auth dir server or via the controller.)

This function should be called ∗after∗ routers_update_status_from_consensus_networkstatus; subsequently, you should call router_rebuild_store and routerlist_descriptors_added.

References routerinfo_t::addr, authdir_mode_handles_descs(), authdir_wants_to_reject_router(), routerinfo_t::cache_info, routerlist_t::desc_digest_map, routerlist_t::desc_store, routerstatus_t::descriptor_digest, DIGEST_LEN, directory_set_dirty(), dirserv_single_reachability_test(), get_options(), hex_str(), signed_descriptor_t::identity_digest, routerlist_t::identity_map, routerinfo_t::is_valid, routerinfo_t::last_reachable, LD_DIR, routerstatus_t::need_to_mirror, networkstatus_get_latest_consensus(), networkstatus_get_v2_list(), networkstatus_v2_find_entry(), networkstatus_v2_list, networkstatus_vote_find_entry(), routerinfo_t::nickname, OLD_ROUTER_DESC_MAX_AGE, routerinfo_t::or_port, signed_descriptor_t::published_on, routerinfo_t::purpose, router_get_routerlist(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerinfo_is_a_configured_bridge(), routerlist, routerlist_insert(), routerlist_insert_old(), routerlist_replace(), signed_desc_append_to_journal(), signed_desc_digest_is_recognized(), signed_descriptor_t::signed_descriptor_digest, routerinfo_t::testing_since, and tor_assert.

Referenced by dirserv_add_descriptor(), router_load_routers_from_string(), and router_load_single_router().

4.35.5.735 int router_addr_is_trusted_dir (uint32_t *addr*)

Return true iff **addr** is the address of one of our trusted directory authorities.

References trusted_dir_servers.

Referenced by get_possible_sybil_list().

4.35.5.736 int router_append_dirobj_signature (char ∗ *buf*, size_t *buf_len*, const char ∗ *digest*, size_t *digest_len*, crypto_pk_env_t ∗ *private_key*)

Helper: used to generate signatures for routers, directories and network-status objects. Given a digest in **digest** and a secret **private_key**, generate an PKCS1-padded signature, BASE64-encode it, surround it with —–BEGIN/END—– pairs, and write it to the **buf_len**-byte buffer at **buf**. Return 0 on success, -1 on failure.

References base64_encode(), crypto_pk_keysize(), crypto_pk_private_sign(), LD_BUG, and tor_free.

Referenced by dirserv_dump_directory_to_string(), extrainfo_dump_to_string(), format_networkstatus_vote(), generate_runningrouters(), rend_encode_v2_descriptors(), and router_dump_router_to_string().

4.35.5.737 routerinfo_t* router_choose_random_node (smartlist_t * *excludedsmartlist*, routerset_t * *excludedset*, router_crn_flags_t *flags*)

Return a random running router from the routerlist. Never pick a node whose routerinfo is in **excludedsmartlist**, or whose routerinfo matches **excludedset**, even if they are the only nodes available. If **CRN_NEED_UPTIME** is set in flags and any router has more than a minimum uptime, return one of those. If **CRN_NEED_CAPACITY** is set in flags, weight your choice by the advertised capacity of each router. If **CRN_ALLOW_INVALID** is not set in flags, consider only Valid routers. If **CRN_NEED_GUARD** is set in flags, consider only Guard routers. If **CRN_WEIGHT_AS_EXIT** is set in flags, we weight bandwidths as if picking an exit node, otherwise we weight bandwidths for picking a relay node (that is, possibly discounting exit nodes).

References or_options_t::ExcludeSingleHopRelays, get_options(), router_get_routerlist(), routerlist_t::routers, smartlist_add(), smartlist_create(), and tor_assert.

Referenced by choose_good_exit_server(), and choose_good_middle_server().

4.35.5.738 void router_clear_status_flags (routerinfo_t * *router*)

Clear all the status flags in routerinfo **router**. We put this function here because it's eerily similar to **clear_status_flags_on_sybil()** (p. 590) above. One day we should merge them.

References routerinfo_t::is_bad_directory, routerinfo_t::is_bad_exit, routerinfo_t::is_exit, routerinfo_t::is_fast, routerinfo_t::is_hs_dir, routerinfo_t::is_possible_guard, routerinfo_t::is_running, routerinfo_t::is_stable, and routerinfo_t::is_valid.

Referenced by routers_update_status_from_consensus_networkstatus().

4.35.5.739 int router_compare_to_my_exit_policy (edge_connection_t * *conn*)

OR only: Check whether my exit policy says to allow connection to conn. Return 0 if we accept; non-0 if we reject.

References edge_connection_t::_base, connection_t::addr, ADDR_POLICY_ACCEPTED, compare_tor_addr_to_addr_policy(), routerinfo_t::exit_policy, connection_t::port, router_get_my_routerinfo(), tor_addr_family(), and tor_addr_is_null().

Referenced by connection_exit_connect().

4.35.5.740 int router_differences_are_cosmetic (routerinfo_t * *r1*, routerinfo_t * *r2*)

Return true iff the only differences between r1 and r2 are such that would not cause a recent (post 0.1.1.6) dirserver to republish.

References routerinfo_t::address, routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, routerinfo_t::cache_info, cmp_addr_policies(), routerinfo_t::contact_info, crypto_pk_cmp_keys(), routerinfo_t::declared_family, routerinfo_t::dir_port, routerinfo_t::exit_policy, routerinfo_t::has_old_dnsworkers, routerinfo_t::identity_pkey, routerinfo_t::is_hibernating, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::or_port, routerinfo_t::platform, signed_descriptor_t::published_on, routerinfo_t::purpose, ROUTER_ALLOW_UPTIME_DRIFT, ROUTER_MAX_COSMETIC_TIME_DIFFERENCE, tor_assert, and routerinfo_t::uptime.

Referenced by dirserv_add_descriptor().

4.35.5.741 int router_digest_is_me (const char ∗ *digest*)

Return true iff I'm a server and **digest** is equal to my identity digest.

References DIGEST_LEN, and identitykey_digest.

Referenced by connection_or_connect(), directory_conn_is_self_reachability_test(), directory_get_from_all_authorities(), dirserv_get_networkstatus_v2(), dirserv_get_routerdescs(), get_signed_descriptor_by_fp(), lookup_cached_dir_by_fp(), router_digest_is_trusted_dir_type(), router_fingerprint_is_me(), router_is_me(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), router_set_networkstatus_v2(), smartlist_choose_by_bandwidth(), and update_v2_networkstatus_cache_downloads().

4.35.5.742 int router_digest_is_trusted_dir_type (const char ∗ *digest*, authority_type_t *type*)

Return true iff **digest** is the digest of the identity key of a trusted directory matching at least one bit of **type**. If **type** is zero, any authority is okay.

References authdir_mode(), DIGEST_LEN, get_options(), router_digest_is_me(), and trusted_dir_servers.

Referenced by router_pick_directory_server_impl().

4.35.5.743 int router_digest_version_as_new_as (const char ∗ *digest*, const char ∗ *cutoff*)

Try to find a routerinfo for **digest**. If we don't have one, return 1. If we do, ask **tor_version_as_new_as()** (p. 1020) for the answer.

References routerinfo_t::platform, router_get_by_digest(), and tor_version_as_new_as().

4.35.5.744 void router_dir_info_changed (void)

Called when our internal view of the directory has changed. This can be when the authorities change, networkstatuses change, the list of routerdescs changes, or number of running routers changes.

References rend_hsdir_routers_changed().

Referenced by add_trusted_dir_server(), clear_trusted_dir_servers(), networkstatus_set_current_consensus(), options_act(), router_set_status(), routerlist_free(), routerlist_insert(), routerlist_remove(), routerlist_replace(), and run_scheduled_events().

4.35.5.745 void router_dirport_found_reachable (void)

Annotate that we found our DirPort reachable.

References routerinfo_t::address, can_reach_dir_port, control_event_server_status(), decide_to_advertise_dirport(), routerinfo_t::dir_port, get_options(), LD_BUG, LD_DIRSERV, mark_my_descriptor_dirty(), and router_get_my_routerinfo().

Referenced by connection_dir_client_reached_eof().

4.35.5.746 int router_dump_router_to_string (char * *s*, size_t *maxlen*, routerinfo_t * *router*, crypto_pk_env_t * *ident_key*)

OR only: Given a routerinfo for this router, and an identity key to sign with, encode the routerinfo as a signed server descriptor and write the result into **s**, using at most **maxlen** bytes. Return -1 on failure, and the number of bytes used on success.

References routerinfo_t::address, or_options_t::AllowSingleHopExits, routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, base16_encode(), routerinfo_t::cache_info, or_options_t::ContactInfo, crypto_pk_cmp_keys(), crypto_pk_get_fingerprint(), crypto_pk_write_public_key_to_string(), decide_to_advertise_dirport(), routerinfo_t::declared_family, DIGEST_LEN, routerinfo_t::dir_port, dns_seems_to_be_broken(), or_options_t::DownloadExtraInfo, escaped(), routerinfo_t::exit_policy, signed_descriptor_t::extra_info_digest, FINGERPRINT_LEN, format_iso_time(), get_options(), has_dns_init_failed(), HEX_DIGEST_LEN, or_options_t::HidServDirectoryV2, routerinfo_t::identity_pkey, LD_BUG, routerinfo_t::nickname, note_crypto_pk_op(), routerinfo_t::onion_pkey, routerinfo_t::or_port, routerinfo_t::platform, policy_write_item(), signed_descriptor_t::published_on, router_append_dirobj_signature(), router_get_router_hash(), router_parse_entry_from_string(), routerinfo_free(), smartlist_join_strings(), stats_n_seconds_working, tor_assert, tor_free, tor_snprintf(), and we_are_hibernating().

Referenced by router_rebuild_descriptor().

4.35.5.747 int router_exit_policy_all_routers_reject (uint32_t *addr*, uint16_t *port*, int *need_uptime*)

Return 1 if all running sufficiently-stable routers will reject addr:port, return 0 if any might accept it.

References ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, compare_addr_to_addr_policy(), router_is_unreliable(), routerlist, and routerlist_t::routers.

Referenced by circuit_get_open_circ_or_launch().

4.35.5.748 int router_exit_policy_rejects_all (routerinfo_t * *router*)

Return true iff **router** does not permit exit streams.

References routerinfo_t::policy_is_reject_star.

Referenced by choose_good_exit_server_general(), and dirserv_compute_performance_thresholds().

4.35.5.749 int router_extrainfo_digest_is_me (const char * *digest*)

Return true iff I'm a server and **digest** is equal to my identity digest.

References extrainfo_t::cache_info, DIGEST_LEN, router_get_my_extrainfo(), and signed_descriptor_t::signed_descriptor_digest.

4.35.5.750 routerinfo_t* router_find_exact_exit_enclave (const char * *address*, uint16_t *port*)

Find a router that's up, that has this IP address, and that allows exit to this address:port, or return NULL if there isn't a good one.

References ADDR_POLICY_ACCEPTED, compare_tor_addr_to_addr_policy(), routerlist, routerlist_t::routers, tor_addr_from_ipv4h, and tor_inet_aton().

4.35.5.751 int router_fingerprint_is_me (const char * *fp*)

Return true iff **fp** is a hex fingerprint of my identity digest.

References base16_decode(), DIGEST_LEN, HEX_DIGEST_LEN, and router_digest_is_me().

4.35.5.752 void router_free_all (void)

Release all static resources held in **router.c** (p. 1117)

References authority_cert_free(), crypto_free_pk_env(), extrainfo_free(), routerinfo_free(), smartlist_free(), tor_free, and tor_mutex_free().

Referenced by tor_free_all().

4.35.5.753 uint32_t router_get_advertised_bandwidth (routerinfo_t * *router*)

Return the smaller of the router's configured BandwidthRate and its advertised capacity.

References routerinfo_t::bandwidthcapacity, and routerinfo_t::bandwidthrate.

Referenced by _compare_routerinfo_by_ip_and_bw(), dirserv_compute_performance_thresholds(), dirserv_thinks_router_is_unreliable(), and set_routerstatus_from_routerinfo().

4.35.5.754 uint32_t router_get_advertised_bandwidth_capped (routerinfo_t * *router*)

Return the smaller of the router's configured BandwidthRate and its advertised capacity, capped by max-believe-bw.

References routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, and DEFAULT_MAX_BELIEVABLE_BANDWIDTH.

Referenced by routerstatus_format_entry(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.35.5.755 signed_descriptor_t* router_get_by_descriptor_digest (const char * *digest*)

Return the router in our routerlist whose 20-byte descriptor is **digest**. Return NULL if no such router is known.

References routerlist_t::desc_digest_map, routerlist, and tor_assert.

Referenced by connection_dirserv_add_servers_to_outbuf(), count_usable_descriptors(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirserv_have_any_serverdesc(), networkstatus_reset_download_failures(), router_set_networkstatus_v2(), update_consensus_router_descriptor_downloads(), and update_router_descriptor_cache_downloads_v2().

4.35.5.756 routerinfo_t∗ router_get_by_digest (const char ∗ *digest*)

Return the router in our routerlist whose 20-byte key digest is **digest**. Return NULL if no such router is known.

References routerlist_t::identity_map, routerlist, and tor_assert.

Referenced by bridges_retry_helper(), build_state_get_exit_router(), choose_good_entry_server(), choose_good_middle_server(), circuit_find_to_cannibalize(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), command_process_create_cell(), command_process_netinfo_cell(), connection_ap_process_end_not_open(), connection_or_digest_is_known_relay(), connection_or_init_conn_from_address(), connection_or_set_state_open(), control_event_guard(), directory_get_from_hs_dir(), directory_initiate_command_routerstatus_rend(), directory_post_to_hs_dir(), dirserv_add_descriptor(), dirserv_add_extrainfo(), dirserv_get_routerdescs(), entry_guards_compute_status(), entry_is_live(), get_signed_descriptor_by_fp(), getinfo_helper_entry_guards(), handle_control_attachstream(), orconn_target_get_name(), rend_services_introduce(), rep_hist_dump_stats(), rep_hist_get_router_stability_doc(), rep_hist_record_mtbf_data(), router_digest_version_as_new_as(), router_get_by_hexdigest(), router_set_status(), router_supports_extrainfo(), routerstatus_format_entry(), smartlist_choose_by_bandwidth(), update_consensus_router_descriptor_downloads(), and update_extrainfo_downloads().

4.35.5.757 signed_descriptor_t∗ router_get_by_extrainfo_digest (const char ∗ *digest*)

Return the signed descriptor for the router in our routerlist whose 20-byte extra-info digest is **digest**. Return NULL if no such router is known.

References routerlist_t::desc_by_eid_map, routerlist, and tor_assert.

Referenced by dir_routerdesc_download_failed().

4.35.5.758 routerinfo_t∗ router_get_by_hexdigest (const char ∗ *hexdigest*)

Return the router in our routerlist whose hexadecimal key digest is **hexdigest**. Return NULL if no such router is known.

References DIGEST_LEN, HEX_DIGEST_LEN, hexdigest_to_digest(), routerinfo_t::is_named, routerinfo_t::nickname, router_get_by_digest(), routerlist, and tor_assert.

Referenced by getinfo_helper_dir(), and router_get_by_nickname().

4.35.5.759 routerinfo_t∗ router_get_by_nickname (const char ∗ *nickname*, int *warn_if_unnamed*)

Return the router in our routerlist whose (case-insensitive) nickname or (case-sensitive) hexadecimal key digest is **nickname**. Return NULL if no such router is known.

References base16_decode(), DIGEST_LEN, get_options(), HEX_DIGEST_LEN, routerlist_t::identity_map, networkstatus_get_router_digest_by_nickname(), networkstatus_nickname_is_unnamed(), router_get_by_hexdigest(), router_get_my_routerinfo(), routerlist, routerlist_t::routers, server_mode(), and tor_assert.

Referenced by add_nickname_list_to_smartlist(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), connection_ap_can_use_exit(), connection_ap_handshake_attach_circuit(), getinfo_helper_dir(), handle_control_extendcircuit(), rend_client_get_random_intro(), rend_service_introduce(), router_rebuild_descriptor(), routerlist_add_family(), and routerset_get_all_routers().

4.35.5.760 routerstatus_t* router_get_consensus_status_by_descriptor_digest (const char * *digest*)

Return the consensus view of the status of the router whose current *descriptor* digest is **digest**, or NULL if no such router is known.

References networkstatus_t::desc_digest_map, digestmap_new(), digestmap_set(), and networkstatus_t::routerstatus_list.

Referenced by router_get_dl_status_by_descriptor_digest(), and signed_descs_update_status_from_consensus_networkstatus().

4.35.5.761 routerstatus_t* router_get_consensus_status_by_id (const char * *digest*)

Return the consensus view of the status of the router whose identity digest is **digest**, or NULL if we don't know about any such router.

References compare_digest_to_routerstatus_entry(), networkstatus_t::routerstatus_list, and smartlist_bsearch().

Referenced by circuit_list_path_impl(), connection_dir_client_reached_eof(), connection_or_digest_is_known_relay(), router_get_my_share_of_directory_requests(), router_set_status(), router_supports_extrainfo(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.35.5.762 routerstatus_t* router_get_consensus_status_by_nickname (const char * *nickname*, int *warn_if_unnamed*)

Given a nickname (possibly verbose, possibly a hexadecimal digest), return the corresponding **routerstatus_t** (p. 218), or NULL if none exists. Warn the user if **warn_if_unnamed** is set, and they have specified a router by nickname, but the Named flag isn't set for that router.

References base16_decode(), DIGEST_LEN, escaped(), HEX_DIGEST_LEN, LD_GENERAL, named_server_map, networkstatus_vote_find_entry(), networkstatus_t::routerstatus_list, smartlist_create(), smartlist_free(), strmap_get_lc(), tor_fragile_assert, and unnamed_server_map.

Referenced by add_nickname_list_to_smartlist().

4.35.5.763 int router_get_dir_hash (const char * *s*, char * *digest*)

Set **digest** to the SHA-1 digest of the hash of the directory in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by dirserv_dump_directory_to_string(), and router_parse_directory().

4.35.5.764 download_status_t* router_get_dl_status_by_descriptor_digest (const char * *d*)

Given the digest of a router descriptor, return its current download status, or NULL if the digest is unrecognized.

References digestmap_get(), routerstatus_t::dl_status, router_get_consensus_status_by_descriptor_digest(), and v2_download_status_map.

Referenced by dir_routerdesc_download_failed(), and router_load_routers_from_string().

4.35.5.765 int router_get_extrainfo_hash (const char * *s*, char * *digest*)

Set **digest** to the SHA-1 digest of the hash of the extrainfo string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), and router_rebuild_descriptor().

4.35.5.766 const char* router_get_my_descriptor (void)

OR only: Return a signed server descriptor for this OR, rebuilding a fresh one if necessary. Return NULL on error.

References routerinfo_t::cache_info, LD_GENERAL, router_get_my_routerinfo(), signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_get_body(), signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by init_keys().

4.35.5.767 extrainfo_t* router_get_my_extrainfo (void)

Return the extrainfo document for this OR, or NULL if we have none. Rebuilt it (and the server descriptor) if necessary.

References get_options(), router_rebuild_descriptor(), and server_mode().

Referenced by extrainfo_insert(), get_signed_descriptor_by_fp(), router_extrainfo_digest_is_me(), and router_upload_dir_desc_to_dirservers().

4.35.5.768 routerinfo_t* router_get_my_routerinfo (void)

Return a routerinfo for this OR, rebuilding a fresh one if necessary. Return NULL on error, or if called on an OP.

References get_options(), router_rebuild_descriptor(), and server_mode().

Referenced by circuit_rep_hist_note_result(), connection_or_send_netinfo(), consider_testing_reachability(), directory_conn_is_self_reachability_test(), directory_fetches_from_authorities(), dirserv_estimate_data_size(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dns_resolve_impl(), get_signed_descriptor_by_fp(), getinfo_helper_misc(), hid_serv_acting_as_directory(), hid_serv_responsible_for_desc_id(), inform_testing_reachability(), router_compare_to_my_exit_policy(), router_dirport_found_reachable(), router_get_by_nickname(), router_get_my_descriptor(), router_get_my_share_of_directory_requests(), router_orport_found_reachable(), router_pick_trusteddirserver_impl(), router_upload_dir_desc_to_dirservers(), routerlist_insert(), routerlist_insert_old(), routerlist_replace(), run_scheduled_events(), and second_elapsed_callback().

4.35.5.769 int router_get_my_share_of_directory_requests (double ∗ *v2_share_out*, double ∗ *v3_share_out*)

Try to determine which fraction of v2 and v3 directory requests aimed at caches will be sent to us. Set ∗**v2_share_out** and ∗**v3_share_out** to the fractions of v2 and v3 protocol shares we expect to see, respectively. Return 0 on success, negative on failure.

References routerinfo_t::cache_info, signed_descriptor_t::identity_digest, routerstatus_t::is_v2_dir, PDS_ALLOW_SELF, PDS_IGNORE_FASCISTFIREWALL, router_get_consensus_status_by_id(), router_get_my_routerinfo(), router_pick_directory_server(), sl_last_total_weighted_bw, V2_AUTHORITY, V3_AUTHORITY, and routerstatus_t::version_supports_v3_dir.

Referenced by geoip_determine_shares().

4.35.5.770 int router_get_networkstatus_v2_hash (const char ∗ *s*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the network-status string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by networkstatus_v2_parse_from_string().

4.35.5.771 int router_get_networkstatus_v3_hash (const char ∗ *s*, char ∗ *digest*, digest_algorithm_t *alg*)

Set **digest** to the SHA-1 digest of the hash of the network-status string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by format_networkstatus_vote().

4.35.5.772 int router_get_networkstatus_v3_hashes (const char ∗ *s*, digests_t ∗ *digests*)

Set **digests** to all the digests of the consensus document in **s**

References router_get_hashes_impl().

Referenced by networkstatus_parse_vote_from_string().

4.35.5.773 int router_get_router_hash (const char ∗ *s*, size_t *s_len*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the first router in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by router_dump_router_to_string(), router_parse_entry_from_string(), and router_rebuild_descriptor().

4.35.5.774 routerlist_t∗ router_get_routerlist (void)

Return the current list of all known routers.

References routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, desc_store_t::description, routerlist_t::extra_info_map, routerlist_t::extrainfo_store, desc_store_t::fname_alt_base, desc_store_t::fname_base, routerlist_t::identity_map, routerlist_t::old_routers, routerlist, routerlist_t::routers, smartlist_create(), and desc_store_t::type.

Referenced by choose_good_entry_server(), directory_remove_invalid(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirserv_orconn_tls_done(), dirserv_test_reachability(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), getinfo_helper_dir(), networkstatus_getinfo_by_purpose(), onion_pick_cpath_exit(), router_add_extrainfo_to_routerlist(), router_add_to_routerlist(), router_choose_random_node(), router_load_extrainfo_from_string(), router_parse_list_from_string(), router_reload_router_list(), routerlist_refresh_countries(), routers_update_all_from_networkstatus(), signed_descriptor_get_body_impl(), and update_extrainfo_downloads().

4.35.5.775 int router_get_runningrouters_hash (const char ∗ *s*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the running-routers string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by generate_runningrouters(), and router_parse_runningrouters().

4.35.5.776 smartlist_t∗ router_get_trusted_dir_servers (void)

Return a smartlist containing a list of **trusted_dir_server_t** (p. 247) ∗ for all known trusted dirservers. Callers must not modify the list or its contents.

References smartlist_create(), and trusted_dir_servers.

Referenced by connection_dir_client_reached_eof(), connection_dir_download_networkstatus_failed(), consider_adding_dir_authorities(), directories_have_accepted_server_descriptor(), directory_get_consensus_url(), directory_get_from_all_authorities(), directory_post_to_dirservers(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), dirvote_fetch_missing_votes(), list_v3_auth_ids(), networkstatus_check_consensus_signature(), and update_v2_networkstatus_cache_downloads().

4.35.5.777 trusted_dir_server_t∗ router_get_trusteddirserver_by_digest (const char ∗ *digest*)

Return the **trusted_dir_server_t** (p. 247) for the directory authority whose identity key hashes to **digest**, or NULL if no such authority is known.

References DIGEST_LEN, and trusted_dir_servers.

Referenced by connection_dir_client_reached_eof(), dir_networkstatus_download_failed(), init_keys(), router_set_networkstatus_v2(), and update_router_descriptor_cache_downloads_v2().

4.35.5.778 void router_get_verbose_nickname (char ∗ *buf*, const routerinfo_t ∗ *router*)

Set **buf** (which must have MAX_VERBOSE_NICKNAME_LEN+1 bytes) to the verbose representation of the identity of **router**. The format is: A dollar sign. The upper-case hexadecimal encoding of the SHA1 hash of router's identity. A "=" if the router is named; a "~" if it is not. The router's nickname.

References base16_encode(), routerinfo_t::cache_info, DIGEST_LEN, HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::is_named, MAX_NICKNAME_LEN, and routerinfo_t::nickname.

Referenced by circuit_list_path_impl(), control_event_descriptors_changed(), control_event_guard(), getinfo_helper_entry_guards(), list_server_status_v1(), and orconn_target_get_name().

4.35.5.779 int router_have_minimum_dir_info (void)

Return true iff we have enough networkstatus and router information to start building circuits. Right now, this means "more than half the networkstatus documents, and at least 1/4 of expected routers."

References update_router_have_minimum_dir_info().

Referenced by add_nickname_list_to_smartlist(), circuit_get_open_circ_or_launch(), circuit_launch_by_extend_info(), directory_info_has_arrived(), getinfo_helper_events(), run_scheduled_events(), and update_extrainfo_downloads().

4.35.5.780 int router_is_me (routerinfo_t ∗ *router*)

A wrapper around **router_digest_is_me()** (p. 988).

References routerinfo_t::cache_info, signed_descriptor_t::identity_digest, and router_digest_is_me().

Referenced by choose_good_exit_server_general(), dirserv_add_descriptor(), dirserv_set_router_is_running(), dirserv_test_reachability(), router_load_single_router(), router_rebuild_descriptor(), router_set_status(), and routerlist_find_my_routerinfo().

4.35.5.781 int router_is_unreliable (routerinfo_t * *router*, int *need_uptime*, int *need_capacity*, int *need_guard*)

Return 1 if **router** is not suitable for these parameters, else 0. If **need_uptime** is non-zero, we require a minimum uptime. If **need_capacity** is non-zero, we require a minimum advertised bandwidth. If **need_guard**, we require that the router is a possible entry guard.

References routerinfo_t::is_fast, routerinfo_t::is_possible_guard, and routerinfo_t::is_stable.

Referenced by choose_good_exit_server_general(), entry_is_live(), router_add_running_routers_to_smartlist(), and router_exit_policy_all_routers_reject().

4.35.5.782 void router_load_extrainfo_from_string (const char * *s*, const char * *eos*, saved_location_t *saved_location*, smartlist_t * *requested_fingerprints*, int *descriptor_digests*)

Parse one or more extrainfos from **s** (ending immediately before **eos** if **eos** is present). Other arguments are as for **router_load_routers_from_string()** (p. 997).

References base16_encode(), DIGEST_LEN, HEX_DIGEST_LEN, LD_DIR, router_add_extrainfo_to_routerlist(), router_get_routerlist(), router_parse_list_from_string(), router_rebuild_store(), routerlist, routerlist_assert_ok(), SAVED_NOWHERE, smartlist_create(), smartlist_free(), smartlist_string_remove(), and WRA_WAS_ADDED().

Referenced by connection_dir_client_reached_eof(), and router_reload_router_list_impl().

4.35.5.783 int router_load_routers_from_string (const char * *s*, const char * *eos*, saved_location_t *saved_location*, smartlist_t * *requested_fingerprints*, int *descriptor_digests*, const char * *prepend_annotations*)

Given a string **s** containing some routerdescs, parse it and put the routers into our directory. If saved_location is SAVED_NOWHERE, the routers are in response to a query to the network: cache them by adding them to the journal.

Return the number of routers actually added.

If **requested_fingerprints** is provided, it must contain a list of uppercased fingerprints. Do not update any router whose fingerprint is not on the list; after updating a router, remove its fingerprint from the list.

If **descriptor_digests** is non-zero, then the requested_fingerprints are descriptor digests. Otherwise they are identity digests.

References base16_encode(), routerlist_t::desc_store, DIGEST_LEN, download_status_mark_impossible(), HEX_DIGEST_LEN, hex_str(), LD_DIR, LD_GENERAL, router_add_to_routerlist(), router_get_dl_status_by_descriptor_digest(), router_parse_list_from_string(), router_rebuild_store(), routerinfo_free(), routerlist, routerlist_assert_ok(), routerlist_descriptors_added(), routers_update_status_from_consensus_networkstatus(), SAVED_NOWHERE, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_string_isin(), smartlist_string_remove(), tor_free, WRA_WAS_ADDED(), and WRA_WAS_REJECTED().

Referenced by load_downloaded_routers(), and router_reload_router_list_impl().

4.35.5.784 int router_load_single_router (const char * *s*, uint8_t *purpose*, int *cache*, const char ** *msg*)

Code to parse a single router descriptor and insert it into the routerlist. Return -1 if the descriptor was ill-formed; 0 if the descriptor was well-formed but could not be added; and 1 if the descriptor was added.

If we don't add it and **msg** is not NULL, then assign to ***msg** a static string describing the reason for refusing the descriptor.

This is used only by the controller.

References routerinfo_t::cache_info, signed_descriptor_t::do_not_cache, LD_DIR, routerinfo_t::purpose, router_add_to_routerlist(), ROUTER_ANNOTATION_BUF_LEN, router_is_me(), router_parse_entry_from_string(), router_purpose_to_string(), routerinfo_free(), routerlist_descriptors_added(), routers_update_status_from_consensus_networkstatus(), smartlist_add(), smartlist_create(), smartlist_free(), tor_assert, tor_snprintf(), and WRA_WAS_ADDED().

Referenced by handle_control_postdescriptor().

4.35.5.785 void router_new_address_suggestion (const char * *suggestion*, const dir_connection_t * *d_conn*)

A directory server **d_conn** told us our IP address is **suggestion**. If this address is different from the one we think we are now, and if our computer doesn't actually know its IP address, then switch.

References dir_connection_t::_base, connection_t::addr, connection_t::address, control_event_server_status(), escaped(), get_options(), ip_address_changed(), is_internal_IP(), last_guessed_ip, LD_DIR, log_addr_has_changed(), resolve_my_address(), server_mode(), tor_addr_eq_ipv4h(), and tor_inet_aton().

Referenced by connection_dir_client_reached_eof().

4.35.5.786 int router_nickname_is_in_list (routerinfo_t * *router*, const char * *list*)

Return 1 iff any member of the (possibly NULL) comma-separated list **list** is an acceptable nickname or hexdigest for **router**. Else return 0.

References router_nickname_matches(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_assert, and tor_free.

Referenced by routers_in_same_family().

4.35.5.787 void router_orport_found_reachable (void)

Annotate that we found our ORPort reachable.

References routerinfo_t::address, can_reach_or_port, control_event_server_status(), get_options(), LD_BUG, LD_OR, mark_my_descriptor_dirty(), routerinfo_t::or_port, and router_get_my_routerinfo().

Referenced by onionskin_answer().

4.35.5.788 addr_policy_t* router_parse_addr_policy_item_from_string (const char * *s*, int *assume_action*)

Parse the addr policy in the string **s** and return it. If assume_action is nonnegative, then insert its action (ADDR_POLICY_ACCEPT or ADDR_POLICY_REJECT) for items that specify no action.

References eat_whitespace(), directory_token_t::error, escaped(), get_next_token(), LD_DIR, memarea_drop_all(), memarea_new(), router_parse_addr_policy(), token_clear(), tor_snprintf(), tor_strlower(), and directory_token_t::tp.

Referenced by parse_addr_policy(), policies_set_router_exitpolicy_to_reject_all(), and routerset_parse().

4.35.5.789 int router_parse_directory (const char * *str*)

Read a signed directory from **str**. If it's well-formed, return 0. Otherwise, return -1. If we're a directory cache, cache it.

References directory_token_t::args, authdir_mode_v1(), check_signature_token(), crypto_free_pk_env(), DIGEST_LEN, dirserv_set_cached_directory(), dump_desc(), find_dir_signing_key(), get_options(), hex_str(), LD_DIR, memarea_clear(), memarea_drop_all(), memarea_new(), directory_token_t::n_args, note_crypto_pk_op(), parse_iso_time(), router_get_dir_hash(), smartlist_clear(), smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, and directory_token_t::tp.

4.35.5.790 routerinfo_t* router_parse_entry_from_string (const char * *s*, const char * *end*, int *cache_copy*, int *allow_annotations*, const char * *prepend_annotations*)

Helper function: reads a single router entry from ***s** ... ***end**. Mallocs a new router and returns it if all goes well, else returns NULL. If **cache_copy** is true, duplicate the contents of s through end into the signed_descriptor_body of the resulting **routerinfo_t** (p. 206).

If **end** is NULL, **s** must be properly NULL-terminated.

If **allow_annotations**, it's okay to encounter annotations in **s** before the router; if it's false, reject the router if it's annotated. If **prepend_annotations** is set, it should contain some annotations: append them to the front of the router before parsing it, and keep them around when caching the router.

Only one of allow_annotations and prepend_annotations may be set.

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::allow_single_hop_exits, signed_descriptor_t::annotations_len, directory_token_t::args, routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, base16_decode(), routerinfo_t::cache_info, routerinfo_t::caches_extra_info, check_signature_token(), routerinfo_t::contact_info, routerinfo_t::country, crypto_pk_get_digest(), routerinfo_t::declared_family, DIGEST_LEN, digestmap_new(), digestmap_set(), routerinfo_t::dir_port, dump_desc(), escaped(), routerinfo_t::exit_policy, signed_descriptor_t::extra_info_digest, find_all_exitpolicy(), find_opt_by_keyword(), routerinfo_t::has_old_dnsworkers, HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, routerinfo_t::is_hibernating, is_legal_nickname(), is_legal_nickname_or_hexdigest(), directory_token_t::key, LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, routerinfo_t::nickname, note_crypto_pk_op(), routerinfo_t::onion_pkey, routerinfo_t::or_port, parse_iso_time(), routerinfo_t::platform, policy_expand_private(), routerinfo_t::policy_is_reject_-

star, policy_is_reject_star(), signed_descriptor_t::published_on, routerinfo_t::purpose, router_add_exit_policy(), router_get_router_hash(), router_purpose_from_string(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerinfo_set_country(), signed_descriptor_t::routerlist_index, signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), token_clear(), tokenize_string(), tor_assert, tor_inet_aton(), tor_parse_long(), tor_strstrip(), tor_version_as_new_as(), routerinfo_t::uptime, and routerinfo_t::wants_to_be_hs_dir.

Referenced by init_keys(), router_dump_router_to_string(), router_load_single_router(), router_parse_list_from_string(), and routerlist_reparse_old().

4.35.5.791 int router_parse_list_from_string (const char ** *s*, const char * *eos*, smartlist_t * *dest*, saved_location_t *saved_location*, int *want_extrainfo*, int *allow_annotations*, const char * *prepend_annotations*)

Given a string *s containing a concatenated sequence of router descriptors (or extra-info documents if **is_extrainfo** is set), parses them and stores the result in **dest**. All routers are marked running and valid. Advances *s to a point immediately following the last router entry. Ignore any trailing router entries that are not complete.

If **saved_location** isn't SAVED_IN_CACHE, make a local copy of each descriptor in the signed_descriptor_body field of each **routerinfo_t** (p. 206). If it isn't SAVED_NOWHERE, remember the offset of each descriptor.

Returns 0 on success and -1 on failure.

References routerinfo_t::cache_info, extrainfo_t::cache_info, extrainfo_parse_entry_from_string(), find_start_of_next_router_or_extrainfo(), routerlist_t::identity_map, LD_DIR, routerinfo_t::nickname, routerinfo_t::purpose, router_get_routerlist(), router_parse_entry_from_string(), router_purpose_to_string(), SAVED_IN_CACHE, signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_t::saved_offset, smartlist_add(), and tor_assert.

Referenced by dirserv_add_multiple_descriptors(), router_load_extrainfo_from_string(), and router_load_routers_from_string().

4.35.5.792 int router_parse_runningrouters (const char * *str*)

Read a signed router status statement from **str**. If it's well-formed, return 0. Otherwise, return -1. If we're a directory cache, cache it.

References directory_token_t::args, authdir_mode_v1(), check_signature_token(), crypto_free_pk_env(), DIGEST_LEN, dirserv_set_cached_directory(), dump_desc(), find_dir_signing_key(), find_opt_by_keyword(), get_options(), LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, note_crypto_pk_op(), parse_iso_time(), router_get_runningrouters_hash(), smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, and directory_token_t::tp.

4.35.5.793 void router_perform_bandwidth_test (int *num_circs*, time_t *now*)

We have enough testing circuits open. Send a bunch of "drop" cells down each of them, to exercise our bandwidth.

References origin_circuit_t::_base, or_options_t::BandwidthRate, CELL_NETWORK_SIZE, circuit_get_next_by_pk_and_purpose(), CIRCUIT_PURPOSE_TESTING, CIRCUIT_STATE_OPEN, CIRCWINDOW_START, origin_circuit_t::cpath, get_options(), LD_OR, crypt_path_t::prev, relay_send_command_from_edge(), circuit_t::state, circuit_t::timestamp_dirty, and TO_CIRCUIT.

Referenced by circuit_testing_opened().

4.35.5.794 routerstatus_t∗ router_pick_directory_server (authority_type_t *type*, int *flags*)

Try to find a running dirserver that supports operations of **type**.

If there are no running dirservers in our routerlist and the **PDS_RETRY_IF_NO_SERVERS** flag is set, set all the authoritative ones as running again, and pick one.

If the **PDS_IGNORE_FASCISTFIREWALL** flag is set, then include dirservers that we can't reach.

If the **PDS_ALLOW_SELF** flag is not set, then don't include ourself (if we're a dirserver).

Don't pick an authority if any non-authority is viable; try to avoid using servers that have returned 503 recently.

References get_options(), LD_DIR, mark_all_trusteddirservers_up(), PDS_RETRY_IF_NO_SERVERS, router_pick_directory_server_impl(), and routerlist.

Referenced by directory_get_from_dirserver(), and router_get_my_share_of_directory_requests().

4.35.5.795 int router_pick_published_address (or_options_t ∗ *options*, uint32_t ∗ *addr*)

Make a current best guess at our address, either because it's configured in torrc, or because we've learned it from dirserver headers. Place the answer in ∗**addr** and return 0 on success, else return -1 if we have no guess.

References LD_CONFIG, resolve_my_address(), and router_guess_address_from_dir_headers().

Referenced by directory_fetches_from_authorities(), getinfo_helper_misc(), and router_rebuild_descriptor().

4.35.5.796 routerstatus_t∗ router_pick_trusteddirserver (authority_type_t *type*, int *flags*)

Try to find a running trusted dirserver. Flags are as for router_pick_directory_server.

References get_options(), LD_DIR, mark_all_trusteddirservers_up(), PDS_NO_EXISTING_SERVERDESC_FETCH, PDS_RETRY_IF_NO_SERVERS, router_pick_trusteddirserver_impl(), and tor_assert.

Referenced by directory_get_from_dirserver().

4.35.5.797 uint8_t router_purpose_from_string (const char ∗ *s*)

Given a string, convert it to a router purpose.

References ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_CONTROLLER, ROUTER_PURPOSE_GENERAL, and ROUTER_PURPOSE_UNKNOWN.

Referenced by handle_control_postdescriptor(), networkstatus_getinfo_by_purpose(), and router_parse_entry_from_string().

4.35.5.798 const char∗ router_purpose_to_string (uint8_t *p*)

Given a router purpose, convert it to a string. Don't call this on ROUTER_PURPOSE_UNKNOWN: The whole point of that value is that we don't know its string representation.

References ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_CONTROLLER, ROUTER_PURPOSE_GENERAL, and tor_assert.

Referenced by dirserv_add_multiple_descriptors(), load_downloaded_routers(), router_load_single_router(), and router_parse_list_from_string().

4.35.5.799 int router_rebuild_descriptor (int *force*)

If **force** is true, or our descriptor is out-of-date, rebuild a fresh routerinfo, signed server descriptor, and extra-info document for this OR. Return 0 on success, -1 on temporary error.

References routerinfo_t::addr, routerinfo_t::address, authdir_mode(), routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, base16_encode(), or_options_t::BridgeRelay, extrainfo_t::cache_info, routerinfo_t::cache_info, control_event_my_descriptor_changed(), crypto_pk_dup_key(), crypto_pk_get_digest(), routerinfo_t::declared_family, desc_clean_since, desc_needs_upload, DIGEST_LEN, routerinfo_t::dir_port, or_options_t::DirPort, escaped(), routerinfo_t::exit_policy, or_options_t::ExitPolicy, or_options_t::ExitPolicyRejectPrivate, signed_descriptor_t::extra_info_digest, extrainfo_dump_to_string(), extrainfo_free(), or_options_t::ExtraInfoStatistics, get_effective_bwburst(), get_effective_bwrate(), get_identity_key(), get_onion_key(), get_options(), get_platform_str(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, signed_descriptor_t::is_extrainfo, is_legal_hexdigest(), is_legal_nickname_or_hexdigest(), routerinfo_t::is_named, routerinfo_t::is_running, routerinfo_t::is_valid, LD_BUG, LD_CONFIG, MAX_EXTRAINFO_UPLOAD_SIZE, or_options_t::MyFamily, extrainfo_t::nickname, or_options_t::Nickname, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::or_port, or_options_t::ORPort, routerinfo_t::platform, policies_parse_exit_policy(), signed_descriptor_t::published_on, routerinfo_t::purpose, rep_hist_bandwidth_assess(), router_dump_router_to_string(), router_get_by_nickname(), router_get_extrainfo_hash(), router_get_router_hash(), router_is_me(), router_pick_published_address(), ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerinfo_incompatible_with_extrainfo(), routerinfo_set_country(), signed_descriptor_t::routerlist_index, signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_sort_strings(), smartlist_split_string(), smartlist_string_isin(), smartlist_string_remove(), smartlist_uniq_strings(), tor_assert, tor_dup_ip(), tor_free, and we_are_hibernating().

Referenced by consider_publishable_server(), router_get_my_extrainfo(), router_get_my_routerinfo(), and run_scheduled_events().

4.35.5.800 int router_reload_consensus_networkstatus (void)

Read every cached v3 consensus networkstatus from the disk.

References get_datadir_fname, get_options(), LD_FS, named_server_map, networkstatus_get_flavor_name(), networkstatus_set_current_consensus(), read_file_to_str(), RFTS_IGNORE_MISSING, routers_update_all_from_networkstatus(), tor_free, tor_snprintf(), unnamed_server_map, update_certificate_downloads(), and networkstatus_t::valid_after.

Referenced by do_main_loop().

4.35.5.801 int router_reload_router_list (void)

Load all cached router descriptors and extra-info documents from the store. Return 0 on success and -1 on failure.

References routerlist_t::desc_store, routerlist_t::extrainfo_store, router_get_routerlist(), and router_reload_router_list_impl().

Referenced by do_main_loop().

4.35.5.802 int router_reload_v2_networkstatus (void)

Repopulate our list of network_status_t objects from the list cached on disk. Return 0 on success, -1 on failure.

References base16_decode(), DIGEST_LEN, directory_caches_v2_dir_info(), get_datadir_fname, get_datadir_fname2, get_options(), HEX_DIGEST_LEN, LD_DIR, LD_FS, networkstatus_v2_list_clean(), read_file_to_str(), remove_file_if_very_old(), router_set_networkstatus_v2(), routers_update_all_from_networkstatus(), smartlist_create(), smartlist_free(), tor_free, and tor_listdir().

Referenced by do_main_loop().

4.35.5.803 void router_reset_descriptor_download_failures (void)

Reset the descriptor download failure count on all routers, so that we can retry any long-failed routers immediately.

References download_status_reset(), last_routerdesc_download_attempted, networkstatus_reset_download_failures(), routerlist, and routerlist_t::routers.

Referenced by do_hup(), routerlist_retry_directory_downloads(), and run_scheduled_events().

4.35.5.804 void router_reset_reachability (void)

Forget what we have learned about our reachability status.

References can_reach_dir_port, and can_reach_or_port.

Referenced by ip_address_changed().

4.35.5.805 void router_reset_status_download_failures (void)

Reset all internal variables used to count failed downloads of network status objects.

References mark_all_trusteddirservers_up().

Referenced by do_hup(), and routerlist_retry_directory_downloads().

4.35.5.806 void router_reset_warnings (void)

Forget that we have issued any router-related warnings, so that we'll warn again if we see the same errors.

References smartlist_clear(), and tor_free.

Referenced by do_hup().

4.35.5.807 int router_set_networkstatus_v2 (const char ∗ *s*, time_t *arrived_at*, v2_networkstatus_source_t *source*, smartlist_t ∗ *requested_fingerprints*)

Given a string **s** containing a network status that we received at **arrived_at** from **source**, try to parse it, see if we want to store it, and put it into our cache as necessary.

If **source** is NS_FROM_DIR or NS_FROM_CACHE, do not replace our own **networkstatus_t** (p. 122) (if we're an authoritative directory server).

If **source** is NS_FROM_CACHE, do not write our **networkstatus_t** (p. 122) to the cache.

If **requested_fingerprints** is provided, it must contain a list of uppercased identity fingerprints. Do not update any networkstatus whose fingerprint is not on the list; after updating a networkstatus, remove its fingerprint from the list.

Return 0 on success, -1 on failure.

Callers should make sure that **routers_update_all_from_networkstatus()** (p. 729) is invoked after this function succeeds.

Retain any routerinfo mentioned in a V2 networkstatus for at least this long.

References add_networkstatus_to_cache(), base16_encode(), control_event_general_status(), trusted_dir_server_t::description, DIGEST_LEN, directory_caches_v2_dir_info(), download_status_failed, download_status_reset(), networkstatus_v2_t::entries, format_iso_time(), format_time_interval(), get_options(), HEX_DIGEST_LEN, networkstatus_v2_t::identity_digest, signed_descriptor_t::last_listed_as_valid_until, LD_DIR, LD_GENERAL, NETWORKSTATUS_ALLOW_SKEW, networkstatus_v2_t::networkstatus_digest, networkstatus_get_cache_filename(), networkstatus_v2_free(), networkstatus_v2_parse_from_string(), networkstatus_v2_t::published_on, networkstatus_v2_t::received_on, router_digest_is_me(), router_get_by_descriptor_digest(), router_get_trusteddirserver_by_digest(), smartlist_add(), smartlist_create(), smartlist_join_strings(), smartlist_string_isin(), smartlist_string_remove(), networkstatus_v2_t::source_address, networkstatus_v2_t::source_dirport, tor_assert, tor_free, touch_file(), trusted_dir_server_t::type, V2_AUTHORITY, and trusted_dir_server_t::v2_ns_dl_status.

Referenced by connection_dir_client_reached_eof(), and router_reload_v2_networkstatus().

4.35.5.808 void router_set_status (const char ∗ *digest*, int *up*)

Mark the router with ID **digest** as running or non-running in our routerlist.

References routerinfo_t::address, control_event_networkstatus_changed_single(), DIGEST_LEN, routerstatus_t::is_running, routerinfo_t::is_running, LD_DIR, LD_NET, routerinfo_-

t::nickname, router_dir_info_changed(), router_get_by_digest(), router_get_consensus_status_by_id(), router_is_me(), tor_assert, trusted_dir_servers, and we_are_hibernating().

Referenced by connection_dir_request_failed(), and connection_or_set_state_open().

4.35.5.809 int router_supports_extrainfo (const char ∗ *identity_digest*, int *is_authority*)

Return true iff **identity_digest** is the digest of a router we believe to support extrainfo downloads. (If **is_authority** we do additional checking that's only valid for authorities.)

References routerinfo_t::caches_extra_info, routerinfo_t::platform, router_get_by_digest(), router_get_consensus_status_by_id(), tor_version_as_new_as(), and routerstatus_t::version_supports_extrainfo_upload.

Referenced by directory_post_to_dirservers(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.35.5.810 void router_upload_dir_desc_to_dirservers (int *force*)

OR only: If **force** is true, or we haven't uploaded this descriptor successfully yet, try to upload our signed descriptor to all the directory servers we know about.

References or_options_t::_PublishServerDescriptor, BRIDGE_AUTHORITY, extrainfo_t::cache_info, routerinfo_t::cache_info, desc_needs_upload, DIR_PURPOSE_UPLOAD_DIR, directory_post_to_dirservers(), get_options(), LD_GENERAL, router_get_my_extrainfo(), router_get_my_routerinfo(), ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_GENERAL, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, and tor_free.

Referenced by consider_publishable_server(), and run_scheduled_events().

4.35.5.811 void routerinfo_free (routerinfo_t ∗ *router*)

Free all storage held by **router**.

References addr_policy_list_free(), routerinfo_t::address, routerinfo_t::cache_info, routerinfo_t::contact_info, crypto_free_pk_env(), routerinfo_t::declared_family, routerinfo_t::exit_policy, routerinfo_t::identity_pkey, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::platform, signed_descriptor_t::signed_descriptor_body, smartlist_free(), and tor_free.

Referenced by dirserv_add_descriptor(), router_add_to_routerlist(), router_dump_router_to_string(), router_free_all(), router_load_routers_from_string(), router_load_single_router(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerlist_free(), routerlist_insert_old(), routerlist_remove(), routerlist_replace(), and signed_descriptor_from_routerinfo().

4.35.5.812 int routerinfo_incompatible_with_extrainfo (routerinfo_t ∗ *ri*, extrainfo_t ∗ *ei*, signed_descriptor_t ∗ *sd*, const char ∗∗ *msg*)

Check whether **ri** (a.k.a. sd) is a router compatible with the extrainfo document **ei**. If no router is compatible with **ei**, **ei** should be dropped. Return 0 for "compatible", return 1 for "reject, and inform whoever uploaded **ei**, and return -1 for "reject silently.". If **msg** is present, set ∗**msg** to a description of the incompatibility (if any).

References extrainfo_t::bad_sig, extrainfo_t::cache_info, routerinfo_t::cache_info, crypto_pk_public_checksig(), DIGEST_LEN, signed_descriptor_t::extra_info_digest, signed_descriptor_t::extrainfo_is_bogus, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, extrainfo_t::pending_sig, extrainfo_t::pending_sig_len, signed_descriptor_t::published_on, signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_digest, tor_assert, and tor_free.

Referenced by dirserv_add_extrainfo(), extrainfo_insert(), and router_rebuild_descriptor().

4.35.5.813 int routerinfo_is_a_configured_bridge (routerinfo_t * *ri*)

Return 1 if **ri** is one of our known bridges, else 0.

References routerinfo_get_configured_bridge().

Referenced by router_add_to_routerlist().

4.35.5.814 void routerinfo_set_country (routerinfo_t * *ri*)

Refresh the country code of **ri**. This function MUST be called on each router when the GeoIP database is reloaded, and on all new routers.

References routerinfo_t::addr, routerinfo_t::country, and geoip_get_country_by_ip().

Referenced by router_parse_entry_from_string(), router_rebuild_descriptor(), and routerlist_refresh_countries().

4.35.5.815 void routerlist_add_family (smartlist_t * *sl*, routerinfo_t * *router*)

Add all the family of **router** to the smartlist **sl**. This is used to make sure we don't pick siblings in a single path, or pick more than one relay from a family for our entry guard list.

References routerinfo_t::declared_family, or_options_t::EnforceDistinctSubnets, get_options(), router_get_by_nickname(), router_nickname_matches(), routerlist_add_network_family(), and smartlist_add().

Referenced by choose_good_entry_server(), choose_good_middle_server(), and choose_random_entry().

4.35.5.816 void routerlist_assert_ok (routerlist_t * *rl*)

Assert that the internal representation of **rl** is self-consistent.

References routerlist_t::desc_digest_map, routerlist_t::identity_map, routerlist_t::routers, and tor_assert.

Referenced by directory_remove_invalid(), extrainfo_insert(), router_load_extrainfo_from_string(), router_load_routers_from_string(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

4.35.5.817 routerinfo_t* routerlist_find_my_routerinfo (void)

Look through the routerlist until we find a router that has my key. Return it.

References router_is_me(), routerlist, and routerlist_t::routers.

4.35.5.818 void routerlist_free (routerlist_t * *rl*)

Free all storage held by a routerlist **rl**.

References _extrainfo_free(), routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, routerlist_t::extra_info_map, routerlist_t::extrainfo_store, routerlist_t::identity_map, desc_store_t::mmap, routerlist_t::old_routers, router_dir_info_changed(), routerinfo_free(), routerlist, routerlist_t::routers, signed_descriptor_free(), smartlist_free(), tor_free, and tor_munmap_file().

Referenced by routerlist_free_all().

4.35.5.819 void routerlist_free_all (void)

Free all memory held by the routerlist module.

References authority_cert_free(), digestmap_free(), routerlist, routerlist_free(), smartlist_free(), tor_free, trusted_dir_certs, trusted_dir_server_free(), trusted_dir_servers, and warned_nicknames.

Referenced by tor_free_all().

4.35.5.820 void routerlist_refresh_countries (void)

Set the country code of all routers in the routerlist.

References router_get_routerlist(), routerinfo_set_country(), and routerlist_t::routers.

Referenced by refresh_all_country_info().

4.35.5.821 void routerlist_remove (routerlist_t * *rl*, routerinfo_t * *ri*, int *make_old*, time_t *now*)

Remove an item **ri** from the routerlist **rl**, updating indices as needed. If **idx** is nonnegative and smartlist_get(rl->routers, idx) == ri, we don't need to do a linear search over the list to decide which to remove. We fill the gap in rl->routers with a later element in the list, if any exists. **ri** is freed.

If **make_old** is true, instead of deleting the router, we try adding it to rl->old_routers.

References desc_store_t::bytes_dropped, extrainfo_t::cache_info, routerinfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, signed_descriptor_t::extra_info_digest, routerlist_t::extra_info_map, extrainfo_free(), routerlist_t::extrainfo_store, signed_descriptor_t::identity_digest, routerlist_t::identity_map, routerlist_t::old_routers, routerinfo_t::purpose, rep_hist_note_router_unreachable(), router_dir_info_changed(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerlist_assert_ok(), signed_descriptor_t::routerlist_index, routerlist_t::routers, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_from_routerinfo(), signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_del(), tor_assert, and tor_digest_is_zero().

Referenced by directory_remove_invalid(), and routerlist_remove_old_routers().

4.35.5.822 void routerlist_remove_old_routers (void)

Deactivate any routers from the routerlist that are more than ROUTER_MAX_AGE seconds old and not recommended by any networkstatuses; remove old routers from the list of cached routers if we have too many.

References _compare_old_routers_by_identity(), routerinfo_t::cache_info, routerlist_t::desc_store, DIGEST_LEN, digestset_free(), digestset_new(), directory_caches_dir_info(), routerlist_t::extrainfo_store, get_options(), signed_descriptor_t::identity_digest, signed_descriptor_t::last_listed_as_valid_until, LD_DIR, networkstatus_get_latest_consensus(), networkstatus_get_v2_list(), networkstatus_v2_list, routerinfo_t::nickname, OLD_ROUTER_DESC_MAX_AGE, routerlist_t::old_routers, signed_descriptor_t::published_on, ROUTER_MAX_AGE, router_rebuild_store(), routerlist, signed_descriptor_t::routerlist_index, routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_cached_routers_with_id(), routerlist_t::routers, networkstatus_t::routerstatus_list, signed_descriptor_t::signed_descriptor_digest, smartlist_sort(), and trusted_dirs_remove_old_certs().

Referenced by router_rebuild_store(), router_reload_router_list_impl(), and run_scheduled_events().

4.35.5.823 void routerlist_reset_warnings (void)

Forget that we have issued any router-related warnings, so that we'll warn again if we see the same errors.

References networkstatus_reset_warnings(), smartlist_clear(), smartlist_create(), tor_free, and warned_nicknames.

Referenced by do_hup().

4.35.5.824 void routerlist_retry_directory_downloads (time_t *now*)

Clear all our timeouts for fetching v2 and v3 directory stuff, and then give it all a try again.

References router_reset_descriptor_download_failures(), router_reset_status_download_failures(), update_networkstatus_downloads(), and update_router_descriptor_downloads().

Referenced by circuit_get_open_circ_or_launch(), and learned_bridge_descriptor().

4.35.5.825 routerinfo_t∗ routerlist_sl_choose_by_bandwidth (smartlist_t ∗ *sl*, bandwidth_weight_rule_t *rule*)

Choose a random element of router list **sl**, weighted by the advertised bandwidth of each router.

References smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

Referenced by choose_good_exit_server_general(), and choose_random_entry().

4.35.5.826 int routers_in_same_family (routerinfo_t ∗ *r1*, routerinfo_t ∗ *r2*)

Return true iff r1 and r2 are in the same family, but not the same router.

References routerinfo_t::declared_family, or_options_t::EnforceDistinctSubnets, get_options(), config_line_t::next, or_options_t::NodeFamilies, router_in_nickname_smartlist(), router_nickname_is_in_list(), routers_in_same_network_family(), and config_line_t::value.

Referenced by circuit_find_to_cannibalize().

4.35.5.827 void routers_sort_by_identity (smartlist_t * *routers*)

Sort a list of **routerinfo_t** (p. 206) in ascending order of identity digest.

References _compare_routerinfo_by_id_digest(), and smartlist_sort().

Referenced by routers_update_status_from_consensus_networkstatus().

4.35.5.828 void routers_update_all_from_networkstatus (time_t *now*, int *dir_version*)

If the network-status list has changed since the last time we called this function, update the status of every routerinfo from the network-status list. If **dir_version** is 2, it's a v2 networkstatus that changed. If **dir_version** is 3, it's a v3 consensus that changed.

References networkstatus_t::client_versions, control_event_general_status(), download_status_map_update_from_v2_networkstatus(), get_options(), have_warned_about_new_version, have_warned_about_old_version, LD_GENERAL, networkstatus_get_live_consensus(), networkstatus_v2_list_has_changed, routerlist_t::old_routers, router_get_routerlist(), routerlist_t::routers, routers_update_status_from_consensus_networkstatus(), server_mode(), networkstatus_t::server_versions, signed_descs_update_status_from_consensus_networkstatus(), tor_version_is_obsolete(), VS_EMPTY, VS_NEW, VS_NEW_IN_SERIES, VS_OLD, and VS_RECOMMENDED.

Referenced by connection_dir_client_reached_eof(), router_reload_consensus_networkstatus(), and router_reload_v2_networkstatus().

4.35.5.829 void routers_update_status_from_consensus_networkstatus (smartlist_t * *routers*, int *reset_failures*)

Given a list **routers** of **routerinfo_t** (p. 206) *, update each status field according to our current consensus networkstatus. May re-order **routers**.

References authdir_mode_v2(), authdir_mode_v3(), DIGEST_LEN, get_options(), or_options_t::NamingAuthoritativeDir, router_clear_status_flags(), ROUTER_PURPOSE_GENERAL, routers_sort_by_identity(), networkstatus_t::routerstatus_list, and smartlist_create().

Referenced by router_load_routers_from_string(), router_load_single_router(), and routers_update_all_from_networkstatus().

4.35.5.830 int routerset_contains_extendinfo (const routerset_t * *set*, const extend_info_t * *ei*)

Return true iff we can tell that **ei** is a member of **set**.

References extend_info_t::addr, extend_info_t::identity_digest, extend_info_t::nickname, extend_info_t::port, and routerset_contains().

Referenced by circuit_conforms_to_options(), and warn_if_last_router_excluded().

4.35.5.831 int routerset_contains_router (const routerset_t ∗ *set*, routerinfo_t ∗ *ri*)

Return true iff **ri** is in **set**.

References routerinfo_t::addr, routerinfo_t::cache_info, routerinfo_t::country, signed_descriptor_t::identity_digest, routerinfo_t::is_named, routerinfo_t::nickname, routerinfo_t::or_port, routerset_contains(), and tor_addr_from_ipv4h.

Referenced by choose_good_exit_server_general(), choose_random_entry(), connection_ap_can_use_exit(), entry_guard_set_status(), entry_is_live(), routerset_subtract_routers(), and routersets_get_disjunction().

4.35.5.832 int routerset_contains_routerstatus (const routerset_t ∗ *set*, routerstatus_t ∗ *rs*)

Return true iff **rs** is in **set**.

References routerstatus_t::addr, routerstatus_t::identity_digest, routerstatus_t::is_named, routerstatus_t::nickname, routerstatus_t::or_port, routerset_contains(), and tor_addr_from_ipv4h.

4.35.5.833 int routerset_equal (const routerset_t ∗ *old*, const routerset_t ∗ *new*)

Helper: return true iff old and new are both NULL, or both non-NULL equal routersets.

References routerset_t::list.

Referenced by options_act().

4.35.5.834 void routerset_free (routerset_t ∗ *routerset*)

Free all storage held in **routerset**.

References addr_policy_free(), routerset_t::countries, routerset_t::country_names, digestmap_free(), routerset_t::digests, routerset_t::list, routerset_t::names, routerset_t::policies, smartlist_free(), strmap_free(), and tor_free.

Referenced by config_assign_value(), option_clear(), and or_options_free().

4.35.5.835 void routerset_get_all_routers (smartlist_t ∗ *out*, const routerset_t ∗ *routerset*, int *running_only*)

Add every known **routerinfo_t** (p. 206) that is a member of **routerset** to **out**. If **running_only**, only add the running ones.

References routerset_t::list, router_get_by_nickname(), routerset_is_list(), smartlist_add(), smartlist_create(), tor_assert, and warned_nicknames.

Referenced by entry_guards_prepend_from_config().

4.35.5.836 int routerset_is_list (const routerset_t ∗ *set*)

Return true iff **set** lists only nicknames and digests, and includes no IP ranges or countries.

Referenced by options_validate(), and routerset_get_all_routers().

4.35.5.837 int routerset_needs_geoip (const routerset_t ∗ *set*)

Return true iff we need a GeoIP IP-to-country database to make sense of **set**.

Referenced by options_need_geoip_info().

4.35.5.838 routerset_t∗ routerset_new (void)

Return a new empty routerset.

References routerset_t::country_names, digestmap_new(), routerset_t::digests, routerset_t::list, routerset_t::names, routerset_t::policies, and smartlist_create().

Referenced by config_assign_value(), and options_validate().

4.35.5.839 int routerset_parse (routerset_t ∗ *target*, const char ∗ *s*, const char ∗ *description*)

Parse the string **s** to create a set of routerset entries, and add them to **target**. In log messages, refer to the string as **description**. Return 0 on success, -1 on failure.

Three kinds of elements are allowed in routersets: nicknames, IP address patterns, and fingerprints. They may be surrounded by optional space, and must be separated by commas.

References base16_decode(), routerset_t::country_names, DIGEST_LEN, digestmap_set(), routerset_t::digests, HEX_DIGEST_LEN, is_legal_hexdigest(), is_legal_nickname(), LD_CONFIG, routerset_t::list, routerset_t::names, routerset_t::policies, router_parse_addr_policy_item_from_string(), routerset_get_countryname(), routerset_refresh_countries(), smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_split_string(), strmap_set_lc(), and tor_free.

Referenced by config_assign_value(), and routerset_union().

4.35.5.840 void routerset_refresh_countries (routerset_t ∗ *target*)

Update the routerset's **countries** bitarray_t. Called whenever the GeoIP database is reloaded.

References routerset_t::countries, routerset_t::country_names, geoip_get_country(), geoip_get_n_countries(), geoip_is_loaded(), LD_CONFIG, routerset_t::n_countries, and tor_assert.

Referenced by refresh_all_country_info(), and routerset_parse().

4.35.5.841 void routerset_subtract_routers (smartlist_t ∗ *lst*, const routerset_t ∗ *routerset*)

Remove every **routerinfo_t** (p. 206) from **lst** that is in **routerset**.

References routerset_contains_router(), and tor_assert.

4.35.5.842 char∗ routerset_to_string (const routerset_t ∗ *set*)

Return a new string that when parsed by routerset_parse_string() will yield **set**.

References smartlist_join_strings().

Referenced by entry_guards_prepend_from_config(), get_assigned_option(), and routerset_union().

4.35.5.843 void routerset_union (routerset_t ∗ *target*, const routerset_t ∗ *source*)

Add all members of the set **source** to **target**.

References routerset_t::list, routerset_parse(), routerset_to_string(), tor_assert, and tor_free.

Referenced by options_validate().

4.35.5.844 void routersets_get_disjunction (smartlist_t ∗ *target*, const smartlist_t ∗ *source*, const routerset_t ∗ *include*, const routerset_t ∗ *exclude*, int *running_only*)

Add to **target** every **routerinfo_t** (p. 206) from **source** except:

1) Don't add it if **include** is non-empty and the relay isn't in **include**; and 2) Don't add it if **exclude** is non-empty and the relay is excluded in a more specific fashion by **exclude**. 3) If **running_only**, don't add non-running routers.

References routerset_contains_router(), routerset_is_empty(), and smartlist_add().

Referenced by choose_good_exit_server_general().

4.35.5.845 int routerstatus_format_entry (char ∗ *buf*, size_t *buf_len*, routerstatus_t ∗ *rs*, const char ∗ *version*, routerstatus_format_type_t *format*)

Helper: write the router-status information in **rs** into **buf**, which has at least **buf_len** free characters. Do NUL-termination. Use the same format as in network-status documents. If **version** is non-NULL, add a "v" line for the platform. Return 0 on success, -1 on failure.

The format argument has three possible values: NS_V2 - Output an entry suitable for a V2 NS opinion document NS_V3_CONSENSUS - Output the first portion of a V3 NS consensus entry NS_V3_CONSENSUS_MICRODESC - Output the first portion of a V3 microdesc consensus entry. NS_V3_VOTE - Output a complete V3 NS vote NS_CONTROL_PORT - Output a NS document for the control port

References routerstatus_t::addr, routerstatus_t::bandwidth, base16_encode(), BASE64_DIGEST_LEN, routerinfo_t::cache_info, routerstatus_t::descriptor_digest, DIGEST_LEN, digest_to_base64(), routerstatus_t::dir_port, routerinfo_t::exit_policy, format_iso_time(), routerstatus_t::has_bandwidth, routerstatus_t::has_measured_bw, HEX_DIGEST_LEN, routerstatus_t::identity_digest, INET_NTOA_BUF_LEN, routerstatus_t::is_authority, routerstatus_t::is_bad_directory, routerstatus_t::is_bad_exit, routerstatus_t::is_exit, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerstatus_t::is_running, routerstatus_t::is_stable, routerstatus_t::is_unnamed, routerstatus_t::is_v2_dir, routerstatus_t::is_valid, LD_BUG, MAX_V_LINE_LEN, routerstatus_t::measured_bw, routerstatus_t::nickname, routerstatus_t::or_port, policy_summarize(), routerstatus_t::published_on, router_get_advertised_bandwidth_capped(), router_get_by_digest(), signed_descriptor_t::signed_descriptor_digest, tor_assert, tor_free, tor_inet_ntoa(), and tor_snprintf().

Referenced by format_networkstatus_vote(), and networkstatus_getinfo_helper_single().

4.35.5.846 void routerstatus_free (routerstatus_t ∗ *rs*)

Free all storage held by the routerstatus object **rs**.

References routerstatus_t::exitsummary, and tor_free.

Referenced by _free_duplicate_routerstatus_entry(), networkstatus_v2_free(), networkstatus_vote_free(), and routerstatus_parse_entry_from_string().

4.35.5.847 void routerstatus_get_verbose_nickname (char ∗ *buf*, const routerstatus_t ∗ *router*)

Set **buf** (which must have MAX_VERBOSE_NICKNAME_LEN+1 bytes) to the verbose representation of the identity of **router**. The format is: A dollar sign. The upper-case hexadecimal encoding of the SHA1 hash of router's identity. A "=" if the router is named; a "∼" if it is not. The router's nickname.

References base16_encode(), DIGEST_LEN, HEX_DIGEST_LEN, routerstatus_t::identity_digest, routerstatus_t::is_named, MAX_NICKNAME_LEN, and routerstatus_t::nickname.

Referenced by circuit_list_path_impl().

4.35.5.848 routerstatus_t∗ routerstatus_sl_choose_by_bandwidth (smartlist_t ∗ *sl*, bandwidth_weight_rule_t *rule*)

Choose a random element of status list **sl**, weighted by the advertised bandwidth of each status.

References smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

Referenced by router_pick_directory_server_impl().

4.35.5.849 const char∗ safe_str (const char ∗ *address*)

Make **address** – a piece of information of unspecified sensitivity – safe to log according to the settings in options->SafeLogging, and return it.

(We return "[scrubbed]" if SafeLogging is anything besides "0", and address otherwise.)

References get_options(), and tor_assert.

Referenced by command_process_relay_cell(), connection_ap_process_end_not_open(), connection_edge_finished_connecting(), connection_exit_begin_conn(), directory_handle_command(), directory_handle_command_get(), evdns_callback(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store_v2_desc_as_dir(), rend_compute_v2_desc_id(), rend_mid_establish_intro(), and rend_mid_introduce().

4.35.5.850 const char∗ safe_str_client (const char ∗ *address*)

Make **address** – a piece of information related to our operation as a client – safe to log according to the settings in options->SafeLogging, and return it.

(We return "[scrubbed]" if SafeLogging is "1", and address otherwise.)

References get_options(), and tor_assert.

Referenced by _circuit_mark_for_close(), addressmap_register(), addressmap_register_virtual_address(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), circuit_handle_first_hop(), client_dns_incr_failures(), command_process_netinfo_cell(), command_process_versions_cell(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_resolve(), connection_ap_make_link(), connection_or_check_valid_tls_handshake(), connection_tls_finish_handshake(), directory_all_unreachable(), directory_post_to_hs_dir(), dumpstats(), fetch_from_buf_socks(), handle_control_mapaddress(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_compute_v2_desc_id(), rend_config_services(), and rend_service_dump_stats().

4.35.5.851 int server_mode (or_options_t * *options*)

Return true iff we are trying to be a server.

References or_options_t::ClientOnly, or_options_t::ORListenAddress, and or_options_t::ORPort.

Referenced by circuit_extend(), circuit_note_clock_jumped(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), circuit_testing_failed(), command_process_create_cell(), conn_close_if_marked(), connection_connect(), connection_exit_begin_conn(), connection_or_check_valid_tls_handshake(), connection_or_connect(), consider_publishable_server(), cpuworkers_rotate(), decide_if_publishable_server(), directory_caches_dir_info(), directory_command_should_use_begindir(), directory_fetches_from_authorities(), directory_info_has_arrived(), dns_init(), dns_reset(), dns_servers_relaunch_checks(), do_hup(), do_list_fingerprint(), do_main_loop(), getinfo_helper_events(), init_keys(), ip_address_changed(), options_act(), options_validate(), rep_hist_circbuilding_dormant(), rep_hist_update_state(), router_get_by_nickname(), router_get_my_extrainfo(), router_get_my_routerinfo(), router_new_address_suggestion(), routers_update_all_from_networkstatus(), run_scheduled_events(), second_elapsed_callback(), should_use_create_fast_for_circuit(), signal_callback(), and update_router_descriptor_downloads().

4.35.5.852 void set_identity_key (crypto_pk_env_t * *k*)

Set the current identity key to k.

References crypto_free_pk_env(), crypto_pk_get_digest(), and identitykey_digest.

Referenced by init_keys().

4.35.5.853 int set_options (or_options_t * *new_val*, char ** *msg*)

Change the current global options to contain **new_val** instead of their current value; take action based on the new value; free the old value as necessary. Returns 0 on success, -1 on failure.

References config_free(), LD_BUG, options_act(), options_act_reversible(), and tor_assert.

Referenced by options_init_from_string(), and options_trial_assign().

4.35.5.854 void set_routerstatus_from_routerinfo (routerstatus_t * *rs*, routerinfo_t * *ri*, time_t *now*, int *naming*, int *exits_can_be_guards*, int *listbadexits*, int *listbaddirs*)

Extract status information from **ri** and from other authority functions and store it in **rs**>. If **naming**, consider setting the named flag in **rs**. If not **exits_can_be_guards**, never mark an exit as a guard. If **listbadexits**, consider setting the badexit flag.

We assume that ri->is_running has already been set, e.g. by dirserv_set_router_is_running(ri, now);

References routerinfo_t::addr, routerstatus_t::addr, BANDWIDTH_TO_GUARANTEE_GUARD, routerinfo_t::cache_info, routerstatus_t::descriptor_digest, DIGEST_LEN, routerstatus_t::dir_port, routerinfo_t::dir_port, dirserv_get_name_status(), dirserv_thinks_router_is_hs_dir(), dirserv_thinks_router_is_unreliable(), FP_NAMED, FP_UNNAMED, routerstatus_t::identity_digest, signed_descriptor_t::identity_digest, routerstatus_t::is_authority, routerinfo_t::is_bad_directory, routerstatus_t::is_bad_directory, routerinfo_t::is_bad_exit, routerstatus_t::is_bad_exit, routerinfo_t::is_exit, routerstatus_t::is_exit, routerinfo_t::is_fast, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, routerinfo_t::is_hs_dir, routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerinfo_t::is_running, routerstatus_t::is_running, routerinfo_t::is_stable, routerstatus_t::is_stable, routerstatus_t::is_unnamed, routerstatus_t::is_v2_dir, routerinfo_t::is_valid, routerstatus_t::is_valid, routerstatus_t::nickname, routerinfo_t::nickname, routerinfo_t::or_port, routerstatus_t::or_port, routerinfo_t::platform, signed_descriptor_t::published_on, routerstatus_t::published_on, rep_hist_get_weighted_fractional_uptime(), rep_hist_get_weighted_time_known(), router_get_advertised_bandwidth(), router_is_active(), signed_descriptor_t::signed_descriptor_digest, and tor_version_as_new_as().

Referenced by networkstatus_getinfo_by_purpose().

4.35.5.855 int should_delay_dir_fetches (or_options_t * *options*)

Return 1 if there's a reason we shouldn't try any directory fetches yet (e.g. we demand bridges and none are yet known). Else return 0.

References any_bridge_descriptors_known(), LD_DIR, and or_options_t::UseBridges.

Referenced by authority_certs_fetch_missing(), update_extrainfo_downloads(), update_networkstatus_downloads(), update_router_descriptor_downloads(), and update_router_have_minimum_dir_info().

4.35.5.856 int should_record_bridge_info (or_options_t * *options*)

Return 1 if we should collect geoip stats on bridge users, and include them in our extrainfo descriptor. Else return 0.

References or_options_t::BridgeRecordUsageByCountry, and or_options_t::BridgeRelay.

Referenced by extrainfo_dump_to_string(), options_act(), and run_scheduled_events().

4.35.5.857 const char* signed_descriptor_get_annotations (signed_descriptor_t * *desc*)

As **signed_descriptor_get_body()** (p. 1016), but points to the beginning of the annotations section rather than the beginning of the descriptor.

References signed_descriptor_get_body_impl().

Referenced by routerlist_reparse_old().

4.35.5.858 const char∗ signed_descriptor_get_body (signed_descriptor_t ∗ *desc*)

Return a pointer to the signed textual representation of a descriptor. The returned string is not guaranteed to be NUL-terminated: the string's length will be in desc->signed_descriptor_len.

The caller must not free the string returned.

References signed_descriptor_get_body_impl().

Referenced by connection_dirserv_add_servers_to_outbuf(), getinfo_helper_dir(), munge_extrainfo_into_routerinfo(), router_get_my_descriptor(), and router_rebuild_store().

4.35.5.859 void signed_descs_update_status_from_consensus_networkstatus (smartlist_t ∗ *descs*)

Given a list of **signed_descriptor_t** (p. 227), update their fields (mainly, when they were last listed) from the most recent consensus.

References networkstatus_t::desc_digest_map, DIGEST_LEN, digestmap_get(), router_get_consensus_status_by_descriptor_digest(), and networkstatus_t::valid_until.

Referenced by routers_update_all_from_networkstatus().

4.35.5.860 const char∗ socks4_response_code_to_string (uint8_t *code*)

Return a string corresponding to a SOCKS4 reponse code.

Referenced by fetch_from_buf_socks_client().

4.35.5.861 const char∗ socks5_response_code_to_string (uint8_t *code*)

Return a string corresponding to a SOCKS5 reponse code.

Referenced by fetch_from_buf_socks_client().

4.35.5.862 int socks_policy_permits_address (const tor_addr_t ∗ *addr*)

Return 1 if **addr** is permitted to connect to our socks port, based on **socks_policy**. Else return 0.

References addr_policy_permits_tor_addr().

Referenced by connection_handle_listener_read(), and evdns_server_callback().

4.35.5.863 void sort_version_list (smartlist_t ∗ *versions*, int *remove_duplicates*)

Sort a list of string-representations of versions in ascending order.

References _compare_tor_version_str_ptr(), _tor_free(), smartlist_sort(), and smartlist_uniq().

Referenced by compute_consensus_versions_list(), format_versions_list(), and networkstatus_compute_consensus().

4.35.5.864 const char∗ stream_end_reason_to_control_string (int *reason*)

Convert the reason for ending a stream **reason** into the format used in STREAM events. Return NULL if the reason is unrecognized.

References END_STREAM_REASON_CANT_ATTACH, END_STREAM_REASON_MASK, END_STREAM_REASON_NET_UNREACHABLE, and END_STREAM_REASON_SOCKSPROTOCOL.

Referenced by control_event_stream_status().

4.35.5.865 socks5_reply_status_t stream_end_reason_to_socks5_response (int *reason*)

Translate **reason** (as from a relay 'end' cell) into an appropriate SOCKS5 reply code.

A reason of 0 means that we're not actually expecting to send this code back to the socks client; we just call it 'succeeded' to keep things simple.

References END_STREAM_REASON_CANT_ATTACH, END_STREAM_REASON_MASK, END_STREAM_REASON_NET_UNREACHABLE, END_STREAM_REASON_SOCKSPROTOCOL, and LD_PROTOCOL.

Referenced by connection_ap_handshake_socks_reply().

4.35.5.866 const char∗ stream_end_reason_to_string (int *reason*)

Translate **reason**, which came from a relay 'end' cell, into a static const string describing why the stream is closing. **reason** is -1 if no reason was provided.

References LD_PROTOCOL.

Referenced by connection_ap_process_end_not_open(), and connection_edge_process_relay_cell().

4.35.5.867 int tls_error_to_orconn_end_reason (int *e*)

Convert a TOR_TLS_∗ error code into an END_OR_CONN_∗ reason.

Referenced by connection_about_to_close_connection().

4.35.5.868 static INLINE control_connection_t ∗ TO_CONTROL_CONN (connection_t ∗ *c*) `[static]`

Convert a connection_t∗ to an control_connection_t∗; assert if the cast is invalid.

References DOWNCAST, connection_t::magic, and tor_assert.

Referenced by _connection_free(), connection_finished_flushing(), connection_free(), connection_free_all(), connection_process_inbuf(), connection_reached_eof(), control_update_global_event_mask(), and send_control_event_string().

4.35.5.869 static INLINE dir_connection_t * TO_DIR_CONN (connection_t * *c*) `[static]`

Convert a connection_t* to a dir_connection_t*; assert if the cast is invalid.

References DOWNCAST, connection_t::magic, and tor_assert.

Referenced by _connection_free(), _connection_write_to_buf_impl(), any_pending_bridge_descriptor_fetches(), connection_about_to_close_connection(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_get_by_type_state_rendquery(), connection_process_inbuf(), connection_reached_eof(), list_pending_downloads(), and run_connection_housekeeping().

4.35.5.870 static INLINE edge_connection_t * TO_EDGE_CONN (connection_t * *c*) `[static]`

Convert a connection_t* to an edge_connection_t*; assert if the cast is invalid.

References DOWNCAST, connection_t::magic, and tor_assert.

Referenced by _connection_free(), _connection_write_to_buf_impl(), ap_stream_wants_exit_attention(), assert_connection_ok(), choose_good_exit_server_general(), circuit_discard_optional_exit_enclaves(), conn_read_callback(), conn_write_callback(), connection_about_to_close_connection(), connection_add(), connection_ap_attach_pending(), connection_ap_expire_beginning(), connection_ap_fail_onehop(), connection_finished_connecting(), connection_finished_flushing(), connection_get_by_type_state_rendquery(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init_accepted_conn(), connection_process_inbuf(), connection_reached_eof(), connection_read_to_buf(), connection_unlink(), control_event_stream_bandwidth_used(), directory_all_unreachable(), get_stream(), getinfo_helper_events(), hibernate_go_dormant(), and rend_client_desc_trynow().

4.35.5.871 static INLINE or_circuit_t * TO_OR_CIRCUIT (circuit_t * *x*) `[static]`

Convert a circuit_t* to a pointer to the enclosing **or_circuit_t** (p. 135). Asserts if the cast is impossible.

References DOWNCAST, circuit_t::magic, and tor_assert.

Referenced by _circuit_mark_for_close(), append_cell_to_circuit_queue(), assert_active_circuits_ok(), assert_circuit_ok(), circuit_consider_stop_edge_reading(), circuit_detach_stream(), circuit_dump_by_conn(), circuit_expire_old_circuits_serverside(), circuit_extend(), circuit_free(), circuit_free_all(), circuit_get_by_circid_orconn_impl(), circuit_get_by_rend_token_and_purpose(), circuit_package_relay_cell(), circuit_receive_relay_cell(), circuit_resume_edge_reading(), circuit_unlink_all_from_or_conn(), command_process_destroy_cell(), command_process_relay_cell(), connection_cpu_process_inbuf(), connection_edge_process_relay_cell(), connection_exit_begin_conn(), connection_exit_connect_dir(), connection_or_flush_from_first_active_circuit(), dns_found_answer(), dns_resolve(), dump_cell_pool_usage(), HT_HEAD(), make_circuit_active_on_conn(), make_circuit_inactive_on_conn(), next_circ_on_conn_p(), prev_circ_on_conn_p(), relay_crypt(), relay_lookup_conn(), relay_send_command_from_edge(), rend_process_relay_cell(), rep_hist_buffer_stats_add_circ(), and set_streams_blocked_on_circ().

4.35.5.872 static INLINE or_connection_t * TO_OR_CONN (connection_t * *c*) [static]

Convert a connection_t* to an or_connection_t*; assert if the cast is invalid.

References DOWNCAST, connection_t::magic, and tor_assert.

Referenced by _connection_free(), assert_connection_ok(), circuit_dump_by_conn(), conn_close_if_marked(), connection_about_to_close_connection(), connection_bucket_read_limit(), connection_bucket_refill(), connection_bucket_write_limit(), connection_buckets_decrement(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_counts_as_relayed_traffic(), connection_cpu_process_inbuf(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_free(), connection_handle_read_impl(), connection_handle_write_impl(), connection_init_accepted_conn(), connection_or_clear_identity_map(), connection_process_inbuf(), connection_reached_eof(), connection_read_to_buf(), connection_unlink(), dumpstats(), getinfo_helper_events(), and run_connection_housekeeping().

4.35.5.873 static INLINE origin_circuit_t * TO_ORIGIN_CIRCUIT (circuit_t * *x*) [static]

Convert a circuit_t* to a pointer to the enclosing **origin_circuit_t** (p. 181). Asserts if the cast is impossible.

References DOWNCAST, circuit_t::magic, and tor_assert.

Referenced by _circuit_mark_for_close(), assert_circuit_ok(), circuit_consider_stop_edge_reading(), circuit_detach_stream(), circuit_dump_by_conn(), circuit_dump_details(), circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_find_to_cannibalize(), circuit_free(), circuit_get_best(), circuit_get_by_global_id(), circuit_get_by_rend_query_and_purpose(), circuit_get_next_by_pk_and_purpose(), circuit_is_acceptable(), circuit_is_better(), circuit_n_conn_done(), circuit_package_relay_cell(), circuit_predict_and_launch_new(), circuit_resume_edge_reading(), circuit_stream_is_being_handled(), command_process_created_cell(), connection_ap_expire_beginning(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), control_event_stream_status(), count_established_intro_points(), getinfo_helper_events(), relay_crypt(), relay_lookup_conn(), relay_send_command_from_edge(), rend_config_services(), rend_process_relay_cell(), and set_streams_blocked_on_circ().

4.35.5.874 void tor_cleanup (void)

Do whatever cleanup is necessary before shutting Tor down.

References accounting_is_enabled(), accounting_record_bandwidth_usage(), authdir_mode_tests_reachability(), or_options_t::command, crypto_global_cleanup(), get_options(), get_or_state(), or_state_mark_dirty(), or_state_save(), or_options_t::PidFile, rep_hist_record_mtbf_data(), and tor_free_all().

Referenced by consider_hibernation(), hibernate_begin(), signal_callback(), and tor_main().

4.35.5.875 void tor_free_all (int *postfork*)

Free all memory that we might have allocated somewhere. If **postfork**, we are a worker process and we want to free only the parts of memory that we won't touch. If !**postfork**, Tor is shutting down and we should free everything.

Helps us find the real leaks with dmalloc and the like. Also valgrind should then report 0 reachable in its leak report (in an ideal world – in practice libevent, SSL, libc etc never quite free everything).

References addressmap_free_all(), buf_shrink_freelists(), circuit_free_all(), clear_pending_onions(), config_free_all(), connection_free_all(), dirserv_free_all(), dirvote_free_all(), dns_free_all(), entry_guards_free_all(), esc_router_info(), escaped(), free_cell_pool(), geoip_free_all(), logs_free_all(), memarea_clear_freelist(), networkstatus_free_all(), policies_free_all(), release_lockfile(), rend_cache_free_all(), rend_service_authorization_free_all(), rend_service_free_all(), rep_hist_free_all(), router_free_all(), routerlist_free_all(), smartlist_free(), timeout_event, tor_free, and tor_tls_free_all().

Referenced by cpuworker_main(), and tor_cleanup().

4.35.5.876 int tor_main (int *argc*, char ∗ *argv*[ ])

Main entry point for the Tor process. Called from **main()** (p. 1210).

4.35.5.877 int tor_version_as_new_as (const char ∗ *platform*, const char ∗ *cutoff*)

Parse the Tor version of the platform string **platform**, and compare it to the version in **cutoff**. Return 1 if the router is at least as new as the cutoff, else return 0.

References eat_whitespace(), find_whitespace(), LD_BUG, LD_DIR, strcmpstart(), tor_assert, tor_version_compare(), and tor_version_parse().

Referenced by dirserv_get_status_impl(), remove_obsolete_entry_guards(), router_digest_version_as_new_as(), router_parse_entry_from_string(), router_supports_extrainfo(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

4.35.5.878 int tor_version_compare (tor_version_t ∗ *a*, tor_version_t ∗ *b*)

Compare two tor versions; Return <0 if a < b; 0 if a ==b, >0 if a > b.

References tor_version_t::git_tag, tor_version_t::git_tag_len, tor_version_t::major, tor_version_t::micro, tor_version_t::minor, tor_version_t::patchlevel, tor_version_t::status, tor_version_t::status_tag, tor_version_t::svn_revision, and tor_assert.

Referenced by _compare_tor_version_str_ptr(), tor_version_as_new_as(), and tor_version_is_obsolete().

4.35.5.879 version_status_t tor_version_is_obsolete (const char ∗ *myversion*, const char ∗ *versionlist*)

Return VS_RECOMMENDED if **myversion** is contained in **versionlist**. Else, return VS_EMPTY if versionlist has no entries. Else, return VS_OLD if every member of **versionlist** is newer than **myversion**. Else, return VS_NEW_IN_SERIES if there is at least one member of **versionlist** in the same series (major.minor.micro) as **myversion**, but no such member is newer than **myversion.**. Else, return VS_NEW if every member of **versionlist** is older than **myversion**. Else, return VS_UNRECOMMENDED.

(versionlist is a comma-separated list of version strings, optionally prefixed with "Tor". Versions that can't be parsed are ignored.)

References LD_BUG, LD_CONFIG, smartlist_create(), smartlist_free(), smartlist_split_string(), strcmpstart(), tor_assert, tor_free, tor_version_compare(), tor_version_parse(), tor_version_same_series(), VS_EMPTY, VS_NEW, VS_NEW_IN_SERIES, VS_OLD, VS_RECOMMENDED, and VS_UNRECOMMENDED.

Referenced by getinfo_helper_events(), and routers_update_all_from_networkstatus().

4.35.5.880 int tor_version_parse (const char ∗ *s*, tor_version_t ∗ *out*)

Parse a tor version from **s**, and store the result in **out**. Return 0 on success, -1 on failure.

References base16_decode(), DIGEST_LEN, eat_whitespace(), find_whitespace(), tor_version_t::git_tag, tor_version_t::git_tag_len, HEX_DIGEST_LEN, tor_version_t::major, tor_version_t::micro, tor_version_t::minor, tor_version_t::patchlevel, tor_version_t::status, tor_version_t::status_tag, strcasecmpstart(), strcmpstart(), tor_version_t::svn_revision, and tor_assert.

Referenced by _compare_tor_version_str_ptr(), remove_obsolete_entry_guards(), tor_version_as_new_as(), and tor_version_is_obsolete().

4.35.5.881 void trusted_dirs_flush_certs_to_disk (void)

Save all v3 key certificates to the cached-certs file.

References smartlist_add(), smartlist_create(), trusted_dir_certs, and trusted_dir_servers_certs_changed.

Referenced by trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

4.35.5.882 int trusted_dirs_load_certs_from_string (const char ∗ *contents*, int *from_store*, int *flush*)

Load a bunch of new key certificates from the string **contents**. If **from_store** is true, the certificates are from the cache, and we don't need to flush them to disk. If **flush** is true, we need to flush any changed certificates to disk now. Return 0 on success, -1 if any certs fail to parse.

References trusted_dir_server_t::addr, authority_cert_t::addr, trusted_dir_server_t::addr_current_at, trusted_dir_server_t::address, already_have_cert(), authority_cert_dl_failed(), authority_cert_free(), authority_cert_parse_from_string(), authority_cert_t::cache_info, cert_list_t::certs, DIGEST_LEN, trusted_dir_server_t::dir_port, authority_cert_t::dir_port, directory_caches_dir_info(), get_cert_list(), get_options(), hex_str(), signed_descriptor_t::identity_digest, LD_DIR, networkstatus_note_certs_arrived(), trusted_dir_server_t::nickname, signed_descriptor_t::published_on, authority_cert_t::signing_key_digest, smartlist_add(), tor_dup_ip(), tor_free, trusted_dir_servers_certs_changed, trusted_dirs_flush_certs_to_disk(), and trusteddirserver_get_by_v3_auth_digest().

Referenced by connection_dir_client_reached_eof(), dirvote_add_vote(), init_keys(), and trusted_dirs_reload_certs().

4.35.5.883 int trusted_dirs_reload_certs (void)

Reload the cached v3 key certificates from the cached-certs file in the data directory. Return 0 on success, -1 on failure.

References get_datadir_fname, read_file_to_str(), RFTS_IGNORE_MISSING, tor_free, and trusted_dirs_load_certs_from_string().

Referenced by do_main_loop().

4.35.5.884 trusted_dir_server_t* trusteddirserver_get_by_v3_auth_digest (const char * *digest*)

Return the **trusted_dir_server_t** (p. 247) for the directory authority whose v3 identity key hashes to **digest**, or NULL if no such authority is known.

References DIGEST_LEN, trusted_dir_servers, and V3_AUTHORITY.

Referenced by authority_certs_fetch_missing(), dirvote_add_vote(), networkstatus_check_consensus_signature(), trusted_dirs_load_certs_from_string(), and update_consensus_router_descriptor_downloads().

4.35.5.885 int try_locking (or_options_t * *options*, int *err_if_locked*)

Try to grab the lock file described in **options**, if we do not already have it. If **err_if_locked** is true, warn if somebody else is holding the lock, and exit if we can't get it after waiting. Otherwise, return -1 if we can't get the lockfile. Return 0 on success.

References LD_GENERAL, options_get_datadir_fname2_suffix(), tor_free, tor_lockfile_lock(), and try_locking().

Referenced by init_key_from_file(), options_act(), and try_locking().

4.35.5.886 void update_certificate_downloads (time_t *now*)

Launch requests as appropriate for missing directory authority certificates.

References authority_certs_fetch_missing().

Referenced by connection_dir_download_cert_failed(), router_reload_consensus_networkstatus(), and update_networkstatus_downloads().

4.35.5.887 void update_consensus_networkstatus_fetch_time (time_t *now*)

Update the time at which we'll consider replacing the current consensus.

References CONSENSUS_MIN_SECONDS_BEFORE_CACHING, crypto_rand_int(), directory_fetches_dir_info_early(), directory_fetches_dir_info_later(), or_options_t::FetchDirInfoExtraEarly, format_local_iso_time(), networkstatus_t::fresh_until, get_options(), LD_DIR, networkstatus_get_live_consensus(), time_to_download_next_consensus, tor_assert, networkstatus_t::valid_after, and networkstatus_t::valid_until.

Referenced by networkstatus_set_current_consensus(), and options_act().

4.35.5.888 void update_consensus_router_descriptor_downloads (time_t *now*, int *is_vote*, networkstatus_t ∗ *consensus*)

For any descriptor that we want that's currently listed in **consensus**, download it as appropriate.

References authdir_mode(), DIGEST_LEN, digestmap_new(), directory_too_idle_to_fetch_descriptors(), trusted_dir_server_t::fake_status, get_options(), networkstatus_voter_info_t::identity_digest, LD_DIR, list_pending_descriptor_downloads(), router_get_by_descriptor_digest(), router_get_by_digest(), networkstatus_t::routerstatus_list, smartlist_add(), smartlist_create(), tor_assert, trusteddirserver_get_by_v3_auth_digest(), and networkstatus_t::voters.

Referenced by dirvote_add_vote(), and update_router_descriptor_downloads().

4.35.5.889 void update_extrainfo_downloads (time_t *now*)

Launch extrainfo downloads as needed.

References digestmap_free(), digestmap_get(), digestmap_new(), DIR_PURPOSE_FETCH_EXTRAINFO, download_status_is_ready(), or_options_t::DownloadExtraInfo, signed_descriptor_t::ei_dl_status, signed_descriptor_t::extra_info_digest, routerlist_t::extra_info_map, signed_descriptor_t::extrainfo_is_bogus, get_options(), signed_descriptor_t::identity_digest, initiate_descriptor_downloads(), signed_descriptor_t::is_extrainfo, LD_DIR, list_pending_descriptor_downloads(), MAX_DL_PER_REQUEST, MAX_ROUTERDESC_DOWNLOAD_FAILURES, routerlist_t::old_routers, PDS_NO_EXISTING_SERVERDESC_FETCH, PDS_RETRY_IF_NO_SERVERS, router_get_by_digest(), router_get_routerlist(), router_have_minimum_dir_info(), routerlist_t::routers, should_delay_dir_fetches(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_shuffle(), and tor_digest_is_zero().

Referenced by directory_info_has_arrived(), and run_scheduled_events().

4.35.5.890 void update_networkstatus_downloads (time_t *now*)

Launch requests for networkstatus documents and authority certificates as appropriate.

References directory_fetches_dir_info_early(), get_options(), should_delay_dir_fetches(), update_certificate_downloads(), update_consensus_networkstatus_downloads(), and update_v2_networkstatus_cache_downloads().

Referenced by do_hup(), routerlist_retry_directory_downloads(), and run_scheduled_events().

4.35.5.891 void update_router_descriptor_downloads (time_t *now*)

Launch downloads for router status as needed.

References or_options_t::Address, DIR_PURPOSE_FETCH_SERVERDESC, directory_fetches_dir_info_early(), directory_get_from_dirserver(), DUMMY_DOWNLOAD_INTERVAL, get_options(), last_routerdesc_download_attempted, networkstatus_get_reasonably_live_consensus(), PDS_RETRY_IF_NO_SERVERS, ROUTER_PURPOSE_GENERAL, server_mode(), should_delay_dir_fetches(), update_consensus_router_descriptor_downloads(), and update_router_descriptor_cache_downloads_v2().

Referenced by directory_info_has_arrived(), routerlist_retry_directory_downloads(), and run_scheduled_events().

4.35.5.892 void v3_authority_check_key_expiry (void)

If we're a v3 authority, check whether we have a certificate that's likely to expire soon. Warn if we do, but not too often.

References authdir_mode_v3(), authority_cert_t::expires, get_options(), and LD_DIR.

Referenced by run_scheduled_events().

4.35.5.893 int validate_addr_policies (or_options_t * *options*, char ** *msg*)

Config helper: If there's any problem with the policy configuration options in **options**, return -1 and set **msg** to a newly allocated description of the error. Else return 0.

References addr_policy_list_free(), or_options_t::AuthDirBadDir, or_options_t::AuthDirBadExit, or_options_t::AuthDirInvalid, or_options_t::AuthDirReject, or_options_t::BridgeRelay, or_options_t::DirPolicy, or_options_t::ExitPolicy, or_options_t::ExitPolicyRejectPrivate, parse_addr_policy(), policies_parse_exit_policy(), or_options_t::ReachableAddresses, or_options_t::ReachableDirAddresses, or_options_t::ReachableORAddresses, and or_options_t::SocksPolicy.

4.35.5.894 void var_cell_free (var_cell_t * *cell*)

Release all space held by **cell**.

References tor_free.

Referenced by connection_or_process_cells_from_inbuf(), and connection_or_send_versions().

4.35.5.895 var_cell_t* var_cell_new (uint16_t *payload_len*)

Allocate and return a new **var_cell_t** (p. 250) with **payload_len** bytes of payload space.

References var_cell_t::circ_id, var_cell_t::command, and var_cell_t::payload_len.

Referenced by connection_or_send_versions(), and fetch_var_cell_from_buf().

4.35.5.896 void var_cell_pack_header (const var_cell_t * *cell*, char * *hdr_out*)

Write the header of **cell** into the first VAR_CELL_HEADER_SIZE bytes of **hdr_out**.

References var_cell_t::circ_id, var_cell_t::command, var_cell_t::payload_len, and set_uint16().

Referenced by connection_or_write_var_cell_to_buf().

4.35.5.897 int vote_routerstatus_find_microdesc_hash (char * *digest256_out*, const vote_routerstatus_t * *vrs*, int *method*, digest_algorithm_t *alg*)

If **vrs** has a hash made for the consensus method **method** with the digest algorithm **alg**, decode it and copy it into **digest256_out** and return 0. Otherwise return -1.

References BASE64_DIGEST256_LEN, crypto_digest_algorithm_get_name(), digest256_from_base64(), vote_routerstatus_t::microdesc, and tor_snprintf().

4.35.5.898 document_signature_t∗ voter_get_sig_by_algorithm (const networkstatus_voter_info_t ∗ *voter*, digest_algorithm_t *alg*)

Return the signature made by **voter** using the algorithm **alg**, or NULL if none is found.

References networkstatus_voter_info_t::sigs.

Referenced by networkstatus_add_detached_signatures(), and networkstatus_parse_vote_from_string().

4.35.5.899 int we_are_hibernating (void)

Return true iff we are currently hibernating.

References hibernate_state, and HIBERNATE_STATE_LIVE.

Referenced by check_descriptor_bandwidth_changed(), check_whether_dirport_reachable(), command_process_create_cell(), connection_exit_begin_conn(), decide_to_advertise_dirport(), directory_info_has_arrived(), dirserv_set_router_is_running(), router_dump_router_to_string(), router_rebuild_descriptor(), router_set_status(), run_connection_housekeeping(), run_scheduled_events(), and second_elapsed_callback().

4.35.5.900 static INLINE int WRA_WAS_ADDED (was_router_added_t *s*) `[static]`

Return true iff the descriptor was added. It might still be necessary to check whether the descriptor generator should be notified.

Referenced by directory_handle_command_post(), dirserv_add_descriptor(), dirserv_add_multiple_descriptors(), init_keys(), router_load_extrainfo_from_string(), router_load_routers_from_string(), and router_load_single_router().

4.35.5.901 static INLINE int WRA_WAS_OUTDATED (was_router_added_t *s*) `[static]`

Return true iff the descriptor was not added because it was either:

- not in the consensus
- neither in the consensus nor in any networkstatus document
- it was outdated.

Referenced by directory_handle_command_post().

4.35.5.902 static INLINE int WRA_WAS_REJECTED (was_router_added_t *s*) `[static]`

Return true iff the descriptor rejected because it was malformed.

Referenced by dirserv_add_descriptor(), and router_load_routers_from_string().

4.35.5.903 int write_to_buf (const char ∗ *string*, size_t *string_len*, buf_t ∗ *buf*)

Append **string_len** bytes from **string** to the end of **buf**.

Return the new length of the buffer on success, -1 on failure.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), CHUNK_WRITE_PTR(), chunk_t::datalen, buf_t::datalen, buf_t::tail, and tor_assert.

Referenced by _connection_write_to_buf_impl(), and move_buf_to_buf().

4.35.5.904 int write_to_buf_zlib (buf_t ∗ *buf*, tor_zlib_state_t ∗ *state*, const char ∗ *data*, size_t *data_len*, int *done*)

Compress on uncompress the **data_len** bytes in **data** using the zlib state **state**, appending the result to **buf**. If **done** is true, flush the data in the state and finish the compression/uncompression. Return -1 on failure, 0 on success.

References buf_add_chunk_with_capacity(), CHUNK_REMAINING_CAPACITY(), CHUNK_WRITE_PTR(), chunk_t::datalen, buf_t::datalen, buf_t::tail, and tor_zlib_process().

Referenced by _connection_write_to_buf_impl().

4.35.6 Variable Documentation

4.35.6.1 circuit_build_times_t circ_times

Global list of circuit build times

Referenced by circuit_expire_building(), circuit_expire_old_circuits_clientside(), circuit_predict_and_launch_new(), connection_or_process_cells_from_inbuf(), connection_or_set_state_open(), connection_tls_finish_handshake(), networkstatus_set_current_consensus(), or_state_save(), or_state_set(), and origin_circuit_new().

4.35.6.2 int has_completed_circuit

We set this to 1 when we've opened a circuit, so we can print a log entry to inform the user that Tor is working.

Referenced by circuit_note_clock_jumped(), circuit_send_next_onion_skin(), directory_info_has_arrived(), getinfo_helper_events(), options_act(), purpose_needs_anonymity(), run_scheduled_events(), and second_elapsed_callback().

4.35.6.3 uint64_t stats_n_create_cells_processed

How many CELL_CREATE cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.35.6.4 uint64_t stats_n_created_cells_processed

How many CELL_CREATED cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.35.6.5 uint64_t stats_n_data_bytes_packaged

How many bytes of data have we put in relay_data cells have we built, ever? This would be RELAY_PAYLOAD_SIZE∗stats_n_data_cells_packaged if every relay cell we ever sent were completely full of data.

Referenced by dumpstats().

4.35.6.6 uint64_t stats_n_data_bytes_received

How many bytes of data have we received relay_data cells, ever? This would be RELAY_-PAYLOAD_SIZE∗stats_n_data_cells_packaged if every relay cell we ever received were completely full of data.

Referenced by connection_edge_process_relay_cell(), and dumpstats().

4.35.6.7 uint64_t stats_n_data_cells_packaged

How many relay_data cells have we built, ever?

Referenced by dumpstats().

4.35.6.8 uint64_t stats_n_data_cells_received

How many relay_data cells have we received, ever?

Referenced by connection_edge_process_relay_cell(), and dumpstats().

4.35.6.9 uint64_t stats_n_destroy_cells_processed

How many CELL_DESTROY cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.35.6.10 uint64_t stats_n_padding_cells_processed

How many CELL_PADDING cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.35.6.11 uint64_t stats_n_relay_cells_delivered

Stats: how many relay cells have been delivered to streams at this hop?

Referenced by dumpstats().

4.35.6.12 uint64_t stats_n_relay_cells_processed

How many CELL_RELAY cells have we received, ever?

Referenced by command_process_cell(), and dumpstats().

4.35.6.13 uint64_t stats_n_relay_cells_relayed

Stats: how many relay cells have originated at this hop, or have been relayed onward (not recognized at this hop)?

Referenced by dumpstats().

4.36 policies.c File Reference

Code to parse and use address policies and exit policies.

`#include "or.h"`

`#include "ht.h"`

Data Structures

- struct **policy_summary_item_t**
- struct **policy_map_ent_t**

Defines

- #define **REJECT**(arg) STMT_BEGIN *msg = tor_strdup(arg); goto err; STMT_END
- #define **DEFAULT_EXIT_POLICY**
- #define **AT**(x) ((**policy_summary_item_t***)smartlist_get(summary, x))
- #define **REJECT_CUTOFF_COUNT** (1<<25)

Functions

- void **policy_expand_private** (**smartlist_t** **policy)
- static int **parse_addr_policy** (**config_line_t** *cfg, **smartlist_t** **dest, int assume_action)
- static int **parse_reachable_addresses** (void)
- int **firewall_is_fascist_or** (void)
- static int **addr_policy_permits_tor_addr** (const **tor_addr_t** *addr, uint16_t port, **smartlist_t** *policy)
- static int **addr_policy_permits_address** (uint32_t addr, uint16_t port, **smartlist_t** *policy)
- int **fascist_firewall_allows_address_or** (const **tor_addr_t** *addr, uint16_t port)
- int **fascist_firewall_allows_or** (**routerinfo_t** *ri)
- int **fascist_firewall_allows_address_dir** (const **tor_addr_t** *addr, uint16_t port)
- int **dir_policy_permits_address** (const **tor_addr_t** *addr)
- int **socks_policy_permits_address** (const **tor_addr_t** *addr)
- int **authdir_policy_permits_address** (uint32_t addr, uint16_t port)
- int **authdir_policy_valid_address** (uint32_t addr, uint16_t port)
- int **authdir_policy_baddir_address** (uint32_t addr, uint16_t port)
- int **authdir_policy_badexit_address** (uint32_t addr, uint16_t port)
- int **validate_addr_policies** (**or_options_t** *options, char **msg)
- static int **load_policy_from_option** (**config_line_t** *config, **smartlist_t** **policy, int assume_action)
- int **policies_parse_from_options** (**or_options_t** *options)
- static int **cmp_single_addr_policy** (**addr_policy_t** *a, **addr_policy_t** *b)
- int **cmp_addr_policies** (**smartlist_t** *a, **smartlist_t** *b)
- static **HT_HEAD** (policy_map, **policy_map_ent_t**)
- static unsigned int **policy_hash** (**policy_map_ent_t** *ent)
- **HT_PROTOTYPE** (HT_GENERATE(policy_map, HT_GENERATE(**policy_map_ent_t**, HT_GENERATE(node, HT_GENERATE(policy_hash, HT_GENERATE(policy_eq)

- **addr_policy_result_t compare_addr_to_addr_policy** (uint32_t addr, uint16_t port, const **smartlist_t** ∗policy)
- static **addr_policy_result_t compare_known_tor_addr_to_addr_policy** (const **tor_addr_t** ∗addr, uint16_t port, const **smartlist_t** ∗policy)
- static **addr_policy_result_t compare_known_tor_addr_to_addr_policy_noport** (const **tor_addr_t** ∗addr, const **smartlist_t** ∗policy)
- static **addr_policy_result_t compare_unknown_tor_addr_to_addr_policy** (uint16_t port, const **smartlist_t** ∗policy)
- **addr_policy_result_t compare_tor_addr_to_addr_policy** (const **tor_addr_t** ∗addr, uint16_t port, const **smartlist_t** ∗policy)
- static int **addr_policy_covers** (**addr_policy_t** ∗a, **addr_policy_t** ∗b)
- static int **addr_policy_intersects** (**addr_policy_t** ∗a, **addr_policy_t** ∗b)
- static void **append_exit_policy_string** (**smartlist_t** ∗∗policy, const char ∗more)
- static void **exit_policy_remove_redundancies** (**smartlist_t** ∗dest)
- int **policies_parse_exit_policy** (**config_line_t** ∗cfg, **smartlist_t** ∗∗dest, int rejectprivate, const char ∗local_address, int add_default_policy)
- void **policies_set_router_exitpolicy_to_reject_all** (**routerinfo_t** ∗r)
- static int **exit_policy_is_general_exit_helper** (**smartlist_t** ∗policy, int port)
- int **exit_policy_is_general_exit** (**smartlist_t** ∗policy)
- int **policy_is_reject_star** (const **smartlist_t** ∗policy)
- int **policy_write_item** (char ∗buf, size_t buflen, **addr_policy_t** ∗policy, int format_for_desc)
- static **smartlist_t** ∗ **policy_summary_create** (void)
- static **policy_summary_item_t** ∗ **policy_summary_item_split** (**policy_summary_item_t** ∗old, uint16_t new_starts)
- static int **policy_summary_split** (**smartlist_t** ∗summary, uint16_t prt_min, uint16_t prt_max)
- static void **policy_summary_accept** (**smartlist_t** ∗summary, uint16_t prt_min, uint16_t prt_max)
- static void **policy_summary_reject** (**smartlist_t** ∗summary, **maskbits_t** maskbits, uint16_t prt_min, uint16_t prt_max)
- static void **policy_summary_add_item** (**smartlist_t** ∗summary, **addr_policy_t** ∗p)
- char ∗ **policy_summarize** (**smartlist_t** ∗policy)
- int **getinfo_helper_policies** (**control_connection_t** ∗conn, const char ∗question, char ∗∗answer)
- void **addr_policy_list_free** (**smartlist_t** ∗lst)
- void **addr_policy_free** (**addr_policy_t** ∗p)
- void **policies_free_all** (void)

Variables

- static **smartlist_t** ∗ **socks_policy** = NULL
- static **smartlist_t** ∗ **dir_policy** = NULL
- static **smartlist_t** ∗ **authdir_reject_policy** = NULL
- static **smartlist_t** ∗ **authdir_invalid_policy** = NULL
- static **smartlist_t** ∗ **authdir_baddir_policy** = NULL
- static **smartlist_t** ∗ **authdir_badexit_policy** = NULL
- static **smartlist_t** ∗ **reachable_or_addr_policy** = NULL
- static **smartlist_t** ∗ **reachable_dir_addr_policy** = NULL
- static const char ∗ **private_nets** [ ]

4.36.1 Detailed Description

Code to parse and use address policies and exit policies.

4.36.2 Define Documentation

4.36.2.1 #define DEFAULT_EXIT_POLICY

Value:

```
"reject *:25,reject *:119,reject *:135-139,reject *:445,"         \
  "reject *:563,reject *:1214,reject *:4661-4666,"                \
  "reject *:6346-6429,reject *:6699,reject *:6881-6999,accept *:*"
```

4.36.3 Function Documentation

4.36.3.1 static int addr_policy_covers (addr_policy_t ∗ *a*, addr_policy_t ∗ *b*) `[static]`

Return true iff the address policy **a** covers every case that would be covered by **b**, so that a,b is redundant.

References addr_policy_t::addr, addr_policy_t::maskbits, addr_policy_t::prt_max, addr_policy_t::prt_min, and tor_addr_compare_masked().

Referenced by exit_policy_remove_redundancies().

4.36.3.2 void addr_policy_free (addr_policy_t ∗ *p*)

Release all storage held by **p**.

References addr_policy_t::is_canonical, addr_policy_t::refcnt, tor_assert, and tor_free.

Referenced by addr_policy_list_free(), exit_policy_remove_redundancies(), load_policy_from_option(), policy_expand_private(), router_add_exit_policy(), and routerset_free().

4.36.3.3 static int addr_policy_intersects (addr_policy_t ∗ *a*, addr_policy_t ∗ *b*) `[static]`

Return true iff the address policies **a** and **b** intersect, that is, there exists an address/port that is covered by **a** that is also covered by **b**.

References addr_policy_t::addr, addr_policy_t::maskbits, addr_policy_t::prt_max, addr_policy_t::prt_min, and tor_addr_compare_masked().

Referenced by exit_policy_remove_redundancies().

4.36.3.4 void addr_policy_list_free (smartlist_t ∗ *lst*)

Release all storage held by **p**.

References addr_policy_free(), and smartlist_free().

Referenced by load_policy_from_option(), parse_addr_policy(), parse_reachable_addresses(), policies_free_all(), policies_set_router_exitpolicy_to_reject_all(), routerinfo_free(), and validate_addr_policies().

4.36.3.5 static int addr_policy_permits_address (uint32_t *addr*, uint16_t *port*, smartlist_t ∗ *policy*) `[static]`

Return true iff **policy** (possibly NULL) will allow a connection to **addr**:**port**. **addr** is an IPv4 address given in host order.

References addr_policy_permits_tor_addr(), and tor_addr_from_ipv4h.

Referenced by authdir_policy_baddir_address(), authdir_policy_badexit_address(), authdir_policy_permits_address(), and authdir_policy_valid_address().

4.36.3.6 static int addr_policy_permits_tor_addr (const tor_addr_t ∗ *addr*, uint16_t *port*, smartlist_t ∗ *policy*) `[static]`

Return true iff **policy** (possibly NULL) will allow a connection to **addr**:**port**.

References ADDR_POLICY_ACCEPTED, ADDR_POLICY_PROBABLY_ACCEPTED, ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, compare_tor_addr_to_addr_policy(), and LD_BUG.

Referenced by addr_policy_permits_address(), dir_policy_permits_address(), fascist_firewall_allows_address_dir(), fascist_firewall_allows_address_or(), and socks_policy_permits_address().

4.36.3.7 static void append_exit_policy_string (smartlist_t ∗∗ *policy*, const char ∗ *more*) `[static]`

Add the exit policy described by **more** to **policy**.

References config_line_t::key, LD_BUG, config_line_t::next, parse_addr_policy(), and config_line_t::value.

Referenced by policies_parse_exit_policy().

4.36.3.8 int authdir_policy_baddir_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** should be marked as a bad dir, based on **authdir_baddir_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.36.3.9 int authdir_policy_badexit_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** should be marked as a bad exit, based on **authdir_badexit_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.36.3.10 int authdir_policy_permits_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** is permitted to publish to our directory, based on **authdir_reject_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.36.3.11 int authdir_policy_valid_address (uint32_t *addr*, uint16_t *port*)

Return 1 if **addr**:**port** is considered valid in our directory, based on **authdir_invalid_policy**. Else return 0.

References addr_policy_permits_address().

Referenced by dirserv_get_status_impl().

4.36.3.12 int cmp_addr_policies (smartlist_t * *a*, smartlist_t * *b*)

Like **cmp_single_addr_policy()** (p. 1033) above, but looks at the whole set of policies in each case.

References cmp_single_addr_policy().

Referenced by router_differences_are_cosmetic().

4.36.3.13 static int cmp_single_addr_policy (addr_policy_t * *a*, addr_policy_t * *b*) `[static]`

Compare two provided address policy items, and return -1, 0, or 1 if the first is less than, equal to, or greater than the second.

References addr_policy_t::addr, addr_policy_t::is_private, addr_policy_t::maskbits, addr_policy_t::policy_type, addr_policy_t::prt_max, addr_policy_t::prt_min, and tor_addr_compare().

Referenced by cmp_addr_policies(), and HT_PROTOTYPE().

4.36.3.14 addr_policy_result_t compare_addr_to_addr_policy (uint32_t *addr*, uint16_t *port*, const smartlist_t * *policy*)

As compare_tor_addr_to_addr_policy, but instead of a **tor_addr_t** (p. 237), takes in host order.

References compare_tor_addr_to_addr_policy(), and tor_addr_from_ipv4h.

Referenced by circuit_stream_is_being_handled(), connection_ap_can_use_exit(), router_exit_policy_all_routers_reject(), and router_handles_some_port().

4.36.3.15 static addr_policy_result_t compare_known_tor_addr_to_addr_policy (const tor_addr_t * *addr*, uint16_t *port*, const smartlist_t * *policy*) `[static]`

Helper for compare_tor_addr_to_addr_policy. Implements the case where addr and port are both known.

References tor_addr_t::addr, ADDR_POLICY_ACCEPTED, ADDR_POLICY_REJECTED, and tor_addr_compare_masked().

Referenced by compare_tor_addr_to_addr_policy().

4.36.3.16 static addr_policy_result_t compare_known_tor_addr_to_addr_policy_noport (const tor_addr_t * *addr*, const smartlist_t * *policy*) `[static]`

Helper for compare_tor_addr_to_addr_policy. Implements the case where addr is known but port is not.

References tor_addr_t::addr, ADDR_POLICY_ACCEPTED, ADDR_POLICY_PROBABLY_ACCEPTED, ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, and tor_addr_compare_masked().

Referenced by compare_tor_addr_to_addr_policy().

4.36.3.17 addr_policy_result_t compare_tor_addr_to_addr_policy (const tor_addr_t * *addr*, uint16_t *port*, const smartlist_t * *policy*)

Decide whether a given addr:port is definitely accepted, definitely rejected, probably accepted, or probably rejected by a given policy. If **addr** is 0, we don't know the IP of the target address. If **port** is 0, we don't know the port of the target address. (At least one of **addr** and **port** must be provided. If you want to know whether a policy would definitely reject an unknown address:port, use **policy_is_reject_star()** (p. 957).)

We could do better by assuming that some ranges never match typical addresses (127.0.0.1, and so on). But we'll try this for now.

References ADDR_POLICY_ACCEPTED, compare_known_tor_addr_to_addr_policy(), compare_known_tor_addr_to_addr_policy_noport(), compare_unknown_tor_addr_to_addr_policy(), tor_addr_is_null(), and tor_assert.

Referenced by addr_policy_permits_tor_addr(), compare_addr_to_addr_policy(), router_compare_to_my_exit_policy(), router_find_exact_exit_enclave(), and routerset_contains().

4.36.3.18 static addr_policy_result_t compare_unknown_tor_addr_to_addr_policy (uint16_t *port*, const smartlist_t * *policy*) `[static]`

Helper for compare_tor_addr_to_addr_policy. Implements the case where port is known but address is not.

References ADDR_POLICY_ACCEPTED, ADDR_POLICY_PROBABLY_ACCEPTED, ADDR_POLICY_PROBABLY_REJECTED, and ADDR_POLICY_REJECTED.

Referenced by compare_tor_addr_to_addr_policy().

4.36.3.19 int dir_policy_permits_address (const tor_addr_t * *addr*)

Return 1 if **addr** is permitted to connect to our dir port, based on **dir_policy**. Else return 0.

References addr_policy_permits_tor_addr().

Referenced by connection_handle_listener_read().

4.36.3.20 int exit_policy_is_general_exit (smartlist_t * *policy*)

Return true iff **ri** is "useful as an exit node", meaning it allows exit to at least one /8 address space for at least two of ports 80, 443, and 6667.

References exit_policy_is_general_exit_helper().

Referenced by dirserv_compute_performance_thresholds().

4.36.3.21 static int exit_policy_is_general_exit_helper (smartlist_t * *policy*, int *port*) `[static]`

Return 1 if there is at least one /8 subnet in **policy** that allows exiting to **port**. Otherwise, return 0.

References tor_addr_from_ipv4h, tor_addr_is_internal(), tor_addr_to_ipv4h(), and tor_assert.

Referenced by exit_policy_is_general_exit().

4.36.3.22 static void exit_policy_remove_redundancies (smartlist_t * *dest*) `[static]`

Detect and excise "dead code" from the policy ***dest**.

References addr_policy_covers(), addr_policy_free(), addr_policy_intersects(), LD_CONFIG, addr_policy_t::maskbits, addr_policy_t::policy_type, policy_write_item(), addr_policy_t::prt_max, addr_policy_t::prt_min, smartlist_del(), smartlist_del_keeporder(), and tor_assert.

Referenced by policies_parse_exit_policy().

4.36.3.23 int fascist_firewall_allows_address_dir (const tor_addr_t * *addr*, uint16_t *port*)

Return true iff we think our firewall will let us make a directory connection to addr:port.

References addr_policy_permits_tor_addr().

Referenced by directory_post_to_dirservers(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.36.3.24 int fascist_firewall_allows_address_or (const tor_addr_t * *addr*, uint16_t *port*)

Return true iff we think our firewall will let us make an OR connection to addr:port.

References addr_policy_permits_tor_addr().

Referenced by directory_command_should_use_begindir(), fascist_firewall_allows_or(), fetch_bridge_descriptors(), router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.36.3.25 int fascist_firewall_allows_or (routerinfo_t * *ri*)

Return true iff we think our firewall will let us make an OR connection to **ri**.

References routerinfo_t::addr, fascist_firewall_allows_address_or(), routerinfo_t::or_port, and tor_addr_from_ipv4h.

Referenced by choose_good_entry_server(), and entry_is_live().

4.36.3.26 int firewall_is_fascist_or (void)

Return true iff the firewall options might block any address:port combination.

Referenced by choose_good_entry_server().

4.36.3.27 int getinfo_helper_policies (control_connection_t * *conn*, const char * *question*, char ** *answer*)

Implementation for GETINFO control command: knows the answer for questions about "exit-policy/..."

4.36.3.28 HT_PROTOTYPE (HT_GENERATE(*policy_map*, HT_GENERATE(*policy_map_ent_t*, HT_GENERATE(*node*, HT_GENERATE(*policy_hash*, HT_GENERATE(*policy_eq*)

Given a pointer to an **addr_policy_t** (p. 9), return a copy of the pointer to the "canonical" copy of that **addr_policy_t** (p. 9); the canonical copy is a single reference-counted object.

References cmp_single_addr_policy(), and tor_assert.

4.36.3.29 static int load_policy_from_option (config_line_t * *config*, smartlist_t ** *policy*, int *assume_action*) `[static]`

Parse **string** in the same way that the exit policy is parsed, and put the processed version in ***policy**. Ignore port specifiers.

References addr_policy_free(), addr_policy_list_free(), addr_policy_t::is_canonical, parse_addr_policy(), addr_policy_t::prt_max, and addr_policy_t::prt_min.

Referenced by policies_parse_from_options().

4.36.3.30 static int parse_addr_policy (config_line_t * *cfg*, smartlist_t ** *dest*, int *assume_action*) `[static]`

Given a linked list of config lines containing "allow" and "deny" tokens, parse them and append the result to **dest**. Return -1 if any tokens are malformed (and don't append any), else return 0.

If **assume_action** is nonnegative, then insert its action (ADDR_POLICY_ACCEPT or ADDR_POLICY_REJECT) for items that specify no action.

References addr_policy_list_free(), LD_CONFIG, config_line_t::next, policy_expand_private(), router_parse_addr_policy_item_from_string(), smartlist_add(), smartlist_add_all(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, and config_line_t::value.

Referenced by append_exit_policy_string(), load_policy_from_option(), parse_reachable_addresses(), policies_parse_exit_policy(), and validate_addr_policies().

4.36.3.31 static int parse_reachable_addresses (void) [static]

Helper: parse the Reachable(Dir|OR)?Addresses fields into reachable_(or|dir)_addr_policy. The options should already have been validated by validate_addr_policies.

References addr_policy_list_free(), get_options(), LD_CONFIG, parse_addr_policy(), or_options_t::ReachableAddresses, or_options_t::ReachableDirAddresses, and or_options_t::ReachableORAddresses.

Referenced by policies_parse_from_options().

4.36.3.32 void policies_free_all (void)

Release all storage held by policy variables.

References addr_policy_list_free(), LD_MM, and policy_write_item().

Referenced by tor_free_all().

4.36.3.33 int policies_parse_exit_policy (config_line_t ∗ *cfg*, smartlist_t ∗∗ *dest*, int *rejectprivate*, const char ∗ *local_address*, int *add_default_policy*)

Parse the exit policy **cfg** into the linked list ∗**dest**. If cfg doesn't end in an absolute accept or reject and if **add_default_policy** is true, add the default exit policy afterwards. If **rejectprivate** is true, prepend "reject private:∗" to the policy. Return -1 if we can't parse cfg, else return 0.

References append_exit_policy_string(), exit_policy_remove_redundancies(), parse_addr_policy(), and tor_snprintf().

Referenced by router_rebuild_descriptor(), and validate_addr_policies().

4.36.3.34 int policies_parse_from_options (or_options_t ∗ *options*)

Set all policies based on **options**, which should have been validated first by validate_addr_policies.

References or_options_t::AuthDirBadDir, or_options_t::AuthDirBadExit, or_options_t::AuthDirInvalid, or_options_t::AuthDirReject, or_options_t::DirPolicy, load_policy_from_option(), parse_reachable_addresses(), and or_options_t::SocksPolicy.

Referenced by options_act().

4.36.3.35 void policies_set_router_exitpolicy_to_reject_all (routerinfo_t ∗ *r*)

Replace the exit policy of **r** with reject ∗:∗.

References addr_policy_list_free(), routerinfo_t::exit_policy, router_parse_addr_policy_item_from_string(), smartlist_add(), and smartlist_create().

Referenced by connection_ap_process_end_not_open().

4.36.3.36 void policy_expand_private (smartlist_t ∗∗ *policy*)

Replace all "private" entries in ∗**policy** with their expanded equivalents.

References addr_policy_t::addr, addr_policy_free(), addr_policy_t::is_canonical, addr_policy_t::is_private, addr_policy_t::maskbits, private_nets, smartlist_add(), smartlist_create(), tor_addr_parse_mask_ports(), and tor_assert.

Referenced by parse_addr_policy(), and router_parse_entry_from_string().

4.36.3.37 static unsigned int policy_hash (policy_map_ent_t ∗ *ent*) `[static]`

Return a hashcode for **ent**

References addr_policy_t::addr, addr_policy_t::is_private, addr_policy_t::maskbits, addr_policy_t::policy_type, addr_policy_t::prt_max, addr_policy_t::prt_min, and tor_addr_hash().

4.36.3.38 int policy_is_reject_star (const smartlist_t ∗ *policy*)

Return false if **policy** might permit access to some addr:port; otherwise if we are certain it rejects everything, return true.

Referenced by connection_ap_can_use_exit(), dns_resolve_impl(), and router_parse_entry_from_string().

4.36.3.39 char∗ policy_summarize (smartlist_t ∗ *policy*)

Create a string representing a summary for an exit policy. The summary will either be an "accept" plus a comma-separated list of port ranges or a "reject" plus port-ranges, depending on which is shorter.

If no exits are allowed at all then NULL is returned, if no ports are blocked instead of "reject " we return "accept 1-65535" (this is an exception to the shorter-representation-wins rule).

References MAX_EXITPOLICY_SUMMARY_LEN, policy_summary_add_item(), policy_summary_create(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_assert, tor_free, and tor_snprintf().

Referenced by dirvote_create_microdescriptor(), and routerstatus_format_entry().

4.36.3.40 static void policy_summary_accept (smartlist_t ∗ *summary*, uint16_t *prt_min*, uint16_t *prt_max*) `[static]`

Mark port ranges as accepted if they are below the reject_count

References policy_summary_split(), and tor_assert.

Referenced by policy_summary_add_item().

4.36.3.41 static void policy_summary_add_item (smartlist_t ∗ *summary*, addr_policy_t ∗ *p*) `[static]`

Add a single exit policy item to our summary: If it is an accept ignore it unless it is for all IP addresses ("∗"), i.e. it's prefixlen/maskbits is 0, else call **policy_summary_accept()** (p. 1038). If it's a reject ignore it if it is about one of the private networks, else call **policy_summary_reject()** (p. 1039).

References addr_policy_t::addr, addr_policy_t::maskbits, policy_summary_accept(), policy_summary_reject(), addr_policy_t::policy_type, private_nets, addr_policy_t::prt_max, addr_policy_t::prt_min, tor_addr_compare(), tor_addr_parse_mask_ports(), and tor_assert.

Referenced by policy_summarize().

4.36.3.42 static smartlist_t∗ policy_summary_create (void) [static]

Create a new exit policy summary, initially only with a single port 1-64k item

References policy_summary_item_t::accepted, policy_summary_item_t::prt_max, policy_summary_item_t::prt_min, policy_summary_item_t::reject_count, smartlist_add(), and smartlist_create().

Referenced by policy_summarize().

4.36.3.43 static policy_summary_item_t∗ policy_summary_item_split (policy_summary_item_t ∗ *old*, uint16_t *new_starts*) [static]

Split the summary item in **item** at the port **new_starts**. The current item is changed to end at new-starts - 1, the new item copies reject_count and accepted from the old item, starts at new_starts and ends at the port where the original item previously ended.

References policy_summary_item_t::accepted, policy_summary_item_t::prt_max, policy_summary_item_t::prt_min, policy_summary_item_t::reject_count, and tor_assert.

Referenced by policy_summary_split().

4.36.3.44 static void policy_summary_reject (smartlist_t ∗ *summary*, maskbits_t *maskbits*, uint16_t *prt_min*, uint16_t *prt_max*) [static]

Count the number of addresses in a network with prefixlen maskbits against the given portrange.

References policy_summary_split(), and tor_assert.

Referenced by policy_summary_add_item().

4.36.3.45 static int policy_summary_split (smartlist_t ∗ *summary*, uint16_t *prt_min*, uint16_t *prt_max*) [static]

Split an exit policy summary so that prt_min and prt_max fall at exactly the start and end of an item respectively.

References policy_summary_item_split(), and smartlist_insert().

Referenced by policy_summary_accept(), and policy_summary_reject().

4.36.3.46 int policy_write_item (char ∗ *buf*, size_t *buflen*, addr_policy_t ∗ *policy*, int *format_for_desc*)

Write a single address policy to the buf_len byte buffer at buf. Return the number of characters written, or -1 on failure.

References addr_policy_t::addr, addr_policy_t::is_private, addr_policy_t::maskbits, addr_policy_t::policy_type, addr_policy_t::prt_max, addr_policy_t::prt_min, tor_addr_family(), tor_addr_to_str(), and tor_snprintf().

Referenced by exit_policy_remove_redundancies(), policies_free_all(), and router_dump_router_to_string().

4.36.3.47 int socks_policy_permits_address (const tor_addr_t * *addr*)

Return 1 if **addr** is permitted to connect to our socks port, based on **socks_policy**. Else return 0.

References addr_policy_permits_tor_addr().

Referenced by connection_handle_listener_read(), and evdns_server_callback().

4.36.3.48 int validate_addr_policies (or_options_t * *options*, char ** *msg*)

Config helper: If there's any problem with the policy configuration options in **options**, return -1 and set **msg** to a newly allocated description of the error. Else return 0.

References addr_policy_list_free(), or_options_t::AuthDirBadDir, or_options_t::AuthDirBadExit, or_options_t::AuthDirInvalid, or_options_t::AuthDirReject, or_options_t::BridgeRelay, or_options_t::DirPolicy, or_options_t::ExitPolicy, or_options_t::ExitPolicyRejectPrivate, parse_addr_policy(), policies_parse_exit_policy(), or_options_t::ReachableAddresses, or_options_t::ReachableDirAddresses, or_options_t::ReachableORAddresses, and or_options_t::SocksPolicy.

4.36.4 Variable Documentation

4.36.4.1 smartlist_t* authdir_baddir_policy = NULL `[static]`

Policy that addresses for incoming router descriptors must **not** match in order to not be marked as BadDirectory.

4.36.4.2 smartlist_t* authdir_badexit_policy = NULL `[static]`

Policy that addresses for incoming router descriptors must **not** match in order to not be marked as BadExit.

4.36.4.3 smartlist_t* authdir_invalid_policy = NULL `[static]`

Policy that addresses for incoming router descriptors must match in order to be marked as valid in our networkstatus.

4.36.4.4 smartlist_t* authdir_reject_policy = NULL `[static]`

Policy that addresses for incoming router descriptors must match in order to be published by us.

4.36.4.5 smartlist_t* dir_policy = NULL `[static]`

Policy that addresses for incoming directory connections must match.

4.36.4.6 const char∗ private_nets[ ] [static]

Initial value:

```
{
 "0.0.0.0/8", "169.254.0.0/16",
 "127.0.0.0/8", "192.168.0.0/16", "10.0.0.0/8", "172.16.0.0/12",

 NULL }
```

Private networks. This list is used in two places, once to expand the "private" keyword when parsing our own exit policy, secondly to ignore just such networks when building exit policy summaries. It is important that all authorities agree on that list when creating summaries, so don't just change this without a proper migration plan and a proposal and stuff.

Referenced by policy_expand_private(), and policy_summary_add_item().

4.36.4.7 smartlist_t∗ reachable_dir_addr_policy = NULL [static]

Parsed **addr_policy_t** (p. 9) describing which addresses we believe we can connect to directories at.

4.36.4.8 smartlist_t∗ reachable_or_addr_policy = NULL [static]

Parsed **addr_policy_t** (p. 9) describing which addresses we believe we can start circuits at.

4.36.4.9 smartlist_t∗ socks_policy = NULL [static]

Policy that addresses for incoming SOCKS connections must match.

4.37 reasons.c File Reference

Convert circuit, stream, and orconn error reasons to and/or from strings and errno values.

`#include "or.h"`

Defines

- #define **E_CASE**(s) case s
- #define **S_CASE**(s) case s

Functions

- const char ∗ **stream_end_reason_to_control_string** (int reason)
- const char ∗ **stream_end_reason_to_string** (int reason)
- socks5_reply_status_t **stream_end_reason_to_socks5_response** (int reason)
- uint8_t **errno_to_stream_end_reason** (int e)
- const char ∗ **orconn_end_reason_to_control_string** (int r)
- int **tls_error_to_orconn_end_reason** (int e)
- int **errno_to_orconn_end_reason** (int e)
- const char ∗ **circuit_end_reason_to_control_string** (int reason)
- const char ∗ **socks4_response_code_to_string** (uint8_t code)
- const char ∗ **socks5_response_code_to_string** (uint8_t code)
- const char ∗ **bandwidth_weight_rule_to_string** (**bandwidth_weight_rule_t** rule)

4.37.1 Detailed Description

Convert circuit, stream, and orconn error reasons to and/or from strings and errno values.

4.37.2 Function Documentation

4.37.2.1 const char∗ bandwidth_weight_rule_to_string (bandwidth_weight_rule_t *rule*)

Return a string corresponding to a bandwidht_weight_rule_t

Referenced by smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.37.2.2 const char∗ circuit_end_reason_to_control_string (int *reason*)

Convert a numeric reason for destroying a circuit into a string for a CIRCUIT event.

References END_CIRC_AT_ORIGIN, END_CIRC_REASON_FLAG_REMOTE, END_CIRC_REASON_NOPATH, and LD_BUG.

Referenced by control_event_circuit_status().

4.37.2.3 int errno_to_orconn_end_reason (int *e*)

Given an errno from a failed ORConn connection, return a reason code appropriate for use in the controller orconn events.

References LD_OR.

Referenced by connection_handle_read_impl(), connection_handle_write_impl(), and connection_or_connect().

4.37.2.4 uint8_t errno_to_stream_end_reason (int *e*)

Given an errno from a failed exit connection, return a reason code appropriate for use in a RELAY END cell.

Referenced by connection_edge_end_errno().

4.37.2.5 const char∗ orconn_end_reason_to_control_string (int *r*)

Convert the reason for ending an OR connection **r** into the format used in ORCONN events. Return "UNKNOWN" if the reason is unrecognized.

References LD_BUG.

Referenced by connection_about_to_close_connection(), control_event_bootstrap_problem(), and control_event_or_conn_status().

4.37.2.6 const char∗ socks4_response_code_to_string (uint8_t *code*)

Return a string corresponding to a SOCKS4 reponse code.

Referenced by fetch_from_buf_socks_client().

4.37.2.7 const char∗ socks5_response_code_to_string (uint8_t *code*)

Return a string corresponding to a SOCKS5 reponse code.

Referenced by fetch_from_buf_socks_client().

4.37.2.8 const char∗ stream_end_reason_to_control_string (int *reason*)

Convert the reason for ending a stream **reason** into the format used in STREAM events. Return NULL if the reason is unrecognized.

References END_STREAM_REASON_CANT_ATTACH, END_STREAM_REASON_MASK, END_STREAM_REASON_NET_UNREACHABLE, and END_STREAM_REASON_SOCKSPROTOCOL.

Referenced by control_event_stream_status().

4.37.2.9 socks5_reply_status_t stream_end_reason_to_socks5_response (int *reason*)

Translate **reason** (as from a relay 'end' cell) into an appropriate SOCKS5 reply code.

A reason of 0 means that we're not actually expecting to send this code back to the socks client; we just call it 'succeeded' to keep things simple.

References END_STREAM_REASON_CANT_ATTACH, END_STREAM_REASON_MASK, END_STREAM_REASON_NET_UNREACHABLE, END_STREAM_REASON_SOCKSPROTOCOL, and LD_PROTOCOL.

Referenced by connection_ap_handshake_socks_reply().

4.37.2.10 const char* stream_end_reason_to_string (int *reason*)

Translate **reason**, which came from a relay 'end' cell, into a static const string describing why the stream is closing. **reason** is -1 if no reason was provided.

References LD_PROTOCOL.

Referenced by connection_ap_process_end_not_open(), and connection_edge_process_relay_cell().

4.37.2.11 int tls_error_to_orconn_end_reason (int *e*)

Convert a TOR_TLS_* error code into an END_OR_CONN_* reason.

Referenced by connection_about_to_close_connection().

4.38 relay.c File Reference

Handle relay cell encryption/decryption, plus packaging and receiving from circuits, plus queuing on circuits.

`#include <math.h>`

`#include "or.h"`

`#include "mempool.h"`

Defines

- #define **MAX_RESOLVE_FAILURES** 3
- #define **CELL_QUEUE_HIGHWATER_SIZE** 256
- #define **CELL_QUEUE_LOWWATER_SIZE** 64
- #define **assert_active_circuits_ok_paranoid**(conn)
- #define **SECONDS_IN_A_DAY** 86400L
- #define **EWMA_TICK_LEN** 10
- #define **EWMA_DEFAULT_HALFLIFE** 0.0
- #define **EPSILON** 0.00001
- #define **LOG_ONEHALF** -0.69314718055994529

Functions

- static int **relay_crypt** (**circuit_t** ∗circ, **cell_t** ∗cell, **cell_direction_t** cell_direction, **crypt_path_t** ∗∗layer_hint, char ∗recognized)
- static **edge_connection_t** ∗ **relay_lookup_conn** (**circuit_t** ∗circ, **cell_t** ∗cell, **cell_direction_t** cell_direction, **crypt_path_t** ∗layer_hint)
- static int **connection_edge_process_relay_cell** (**cell_t** ∗cell, **circuit_t** ∗circ, **edge_connection_t** ∗conn, **crypt_path_t** ∗layer_hint)
- static void **circuit_consider_sending_sendme** (**circuit_t** ∗circ, **crypt_path_t** ∗layer_hint)
- static void **circuit_resume_edge_reading** (**circuit_t** ∗circ, **crypt_path_t** ∗layer_hint)
- static int **circuit_resume_edge_reading_helper** (**edge_connection_t** ∗conn, **circuit_t** ∗circ, **crypt_path_t** ∗layer_hint)
- static int **circuit_consider_stop_edge_reading** (**circuit_t** ∗circ, **crypt_path_t** ∗layer_hint)
- static void **tor_gettimeofday_cached** (struct timeval ∗tv)
- void **tor_gettimeofday_cache_clear** (void)
- static void **relay_set_digest** (**crypto_digest_env_t** ∗digest, **cell_t** ∗cell)
- static int **relay_digest_matches** (**crypto_digest_env_t** ∗digest, **cell_t** ∗cell)
- static int **relay_crypt_one_payload** (**crypto_cipher_env_t** ∗cipher, char ∗in, int encrypt_mode)
- int **circuit_receive_relay_cell** (**cell_t** ∗cell, **circuit_t** ∗circ, **cell_direction_t** cell_direction)
- static int **circuit_package_relay_cell** (**cell_t** ∗cell, **circuit_t** ∗circ, **cell_direction_t** cell_direction, **crypt_path_t** ∗layer_hint)
- void **relay_header_pack** (char ∗dest, const **relay_header_t** ∗src)
- void **relay_header_unpack** (**relay_header_t** ∗dest, const char ∗src)
- static const char ∗ **relay_command_to_string** (uint8_t command)

- int **relay_send_command_from_edge** (uint16_t stream_id, **circuit_t** *circ, uint8_t relay_command, const char *payload, size_t payload_len, **crypt_path_t** *cpath_layer)
- int **connection_edge_send_command** (**edge_connection_t** *fromconn, uint8_t relay_command, const char *payload, size_t payload_len)
- static int **edge_reason_is_retriable** (int reason)
- static int **connection_ap_process_end_not_open** (**relay_header_t** *rh, **cell_t** *cell, **origin_circuit_t** *circ, **edge_connection_t** *conn, **crypt_path_t** *layer_hint)
- static void **remap_event_helper** (**edge_connection_t** *conn, uint32_t new_addr)
- static int **connection_edge_process_relay_cell_not_open** (**relay_header_t** *rh, **cell_t** *cell, **circuit_t** *circ, **edge_connection_t** *conn, **crypt_path_t** *layer_hint)
- int **connection_edge_package_raw_inbuf** (**edge_connection_t** *conn, int package_partial)
- void **connection_edge_consider_sending_sendme** (**edge_connection_t** *conn)
- void **init_cell_pool** (void)
- void **free_cell_pool** (void)
- void **clean_cell_pool** (void)
- static INLINE void **packed_cell_free_unchecked** (**packed_cell_t** *cell)
- static INLINE **packed_cell_t** * **packed_cell_alloc** (void)
- void **dump_cell_pool_usage** (int severity)
- static INLINE **packed_cell_t** * **packed_cell_copy** (const **cell_t** *cell)
- void **cell_queue_append** (**cell_queue_t** *queue, **packed_cell_t** *cell)
- void **cell_queue_append_packed_copy** (**cell_queue_t** *queue, const **cell_t** *cell)
- void **cell_queue_clear** (**cell_queue_t** *queue)
- static INLINE **packed_cell_t** * **cell_queue_pop** (**cell_queue_t** *queue)
- static INLINE **circuit_t** ** **next_circ_on_conn_p** (**circuit_t** *circ, **or_connection_t** *conn)
- static INLINE **circuit_t** ** **prev_circ_on_conn_p** (**circuit_t** *circ, **or_connection_t** *conn)
- static int **compare_cell_ewma_counts** (const void *p1, const void *p2)
- static **circuit_t** * **cell_ewma_to_circuit** (**cell_ewma_t** *ewma)
- static unsigned **cell_ewma_tick_from_timeval** (const struct timeval *now, double *remainder_out)
- unsigned **cell_ewma_get_tick** (void)
- void **cell_ewma_set_scale_factor** (**or_options_t** *options, **networkstatus_t** *consensus)
- static INLINE double **get_scale_factor** (unsigned from_tick, unsigned to_tick)
- static void **scale_single_cell_ewma** (**cell_ewma_t** *ewma, unsigned cur_tick)
- static void **scale_active_circuits** (**or_connection_t** *conn, unsigned cur_tick)
- static void **add_cell_ewma_to_conn** (**or_connection_t** *conn, **cell_ewma_t** *ewma)
- static void **remove_cell_ewma_from_conn** (**or_connection_t** *conn, **cell_ewma_t** *ewma)
- static **cell_ewma_t** * **pop_first_cell_ewma_from_conn** (**or_connection_t** *conn)
- void **make_circuit_active_on_conn** (**circuit_t** *circ, **or_connection_t** *conn)
- void **make_circuit_inactive_on_conn** (**circuit_t** *circ, **or_connection_t** *conn)
- void **connection_or_unlink_all_active_circs** (**or_connection_t** *orconn)
- static void **set_streams_blocked_on_circ** (**circuit_t** *circ, **or_connection_t** *orconn, int block)
- int **connection_or_flush_from_first_active_circuit** (**or_connection_t** *conn, int max, time_t now)

- void **append_cell_to_circuit_queue** (**circuit_t** ∗circ, **or_connection_t** ∗orconn, **cell_t** ∗cell, **cell_direction_t** direction)
- int **append_address_to_payload** (char ∗payload_out, const **tor_addr_t** ∗addr)
- const char ∗ **decode_address_from_payload** (**tor_addr_t** ∗addr_out, const char ∗payload, int payload_len)
- void **assert_active_circuits_ok** (**or_connection_t** ∗orconn)

Variables

- static struct timeval **cached_time_hires** = {0, 0}
- uint64_t **stats_n_relay_cells_relayed** = 0
- uint64_t **stats_n_relay_cells_delivered** = 0
- uint64_t **stats_n_data_cells_packaged** = 0
- uint64_t **stats_n_data_bytes_packaged** = 0
- uint64_t **stats_n_data_cells_received** = 0
- uint64_t **stats_n_data_bytes_received** = 0
- static int **total_cells_allocated** = 0
- static **mp_pool_t** ∗ **cell_pool** = NULL
- static **mp_pool_t** ∗ **it_pool** = NULL
- static double **ewma_scale_factor** = 0.1
- static int **ewma_enabled** = 0

4.38.1 Detailed Description

Handle relay cell encryption/decryption, plus packaging and receiving from circuits, plus queuing on circuits.

4.38.2 Define Documentation

4.38.2.1 #define CELL_QUEUE_HIGHWATER_SIZE 256

Stop reading on edge connections when we have this many cells waiting on the appropriate queue.

Referenced by append_cell_to_circuit_queue().

4.38.2.2 #define CELL_QUEUE_LOWWATER_SIZE 64

Start reading from edge connections again when we get down to this many cells.

Referenced by connection_or_flush_from_first_active_circuit().

4.38.2.3 #define EWMA_DEFAULT_HALFLIFE 0.0

The default per-tick scale factor, if it hasn't been overridden by a consensus or a configuration setting. zero means "disabled".

Referenced by cell_ewma_set_scale_factor().

4.38.2.4 #define EWMA_TICK_LEN 10

How long does a tick last (seconds)?

Referenced by cell_ewma_get_tick(), cell_ewma_set_scale_factor(), and cell_ewma_tick_from_timeval().

4.38.2.5 #define MAX_RESOLVE_FAILURES 3

How many times will I retry a stream that fails due to DNS resolve failure or misc error?

Referenced by connection_ap_process_end_not_open().

4.38.3 Function Documentation

4.38.3.1 static void add_cell_ewma_to_conn (or_connection_t * *conn*, cell_ewma_t * *ewma*) `[static]`

Rescale **ewma** to the same scale as **conn**, and add it to **conn**'s priority queue of active circuits

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuit_pqueue_last_recalibrated, compare_cell_ewma_counts(), cell_ewma_t::heap_index, scale_single_cell_ewma(), smartlist_pqueue_add(), STRUCT_OFFSET, and tor_assert.

Referenced by connection_or_flush_from_first_active_circuit(), and make_circuit_active_on_conn().

4.38.3.2 int append_address_to_payload (char * *payload_out*, const tor_addr_t * *addr*)

Append an encoded value of **addr** to **payload_out**, which must have at least 18 bytes of free space. The encoding is, as specified in tor-spec.txt: RESOLVED_TYPE_IPV4 or RESOLVED_TYPE_IPV6 [1 byte] LENGTH [1 byte] ADDRESS [length bytes] Return the number of bytes added, or -1 on error

References tor_addr_family(), and tor_addr_to_ipv4n().

Referenced by connection_or_send_netinfo().

4.38.3.3 void append_cell_to_circuit_queue (circuit_t * *circ*, or_connection_t * *orconn*, cell_t * *cell*, cell_direction_t *direction*)

Add **cell** to the queue of **circ** writing to **orconn** transmitting in **direction**.

References or_connection_t::_base, approx_time(), buf_datalen(), CELL_DIRECTION_OUT, cell_queue_append_packed_copy(), CELL_QUEUE_HIGHWATER_SIZE, cell_t::command, connection_or_flush_from_first_active_circuit(), LD_GENERAL, or_connection_t::link_proto, make_circuit_active_on_conn(), cell_queue_t::n, circuit_t::n_conn_cells, connection_t::outbuf, or_circuit_t::p_conn_cells, set_streams_blocked_on_circ(), circuit_t::streams_blocked_on_n_conn, circuit_t::streams_blocked_on_p_conn, and TO_OR_CIRCUIT().

Referenced by circuit_deliver_create_cell(), circuit_package_relay_cell(), circuit_receive_relay_cell(), and onionskin_answer().

4.38.3.4 void assert_active_circuits_ok (or_connection_t * *orconn*)

Fail with an assert if the active circuits ring on **orconn** is corrupt.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuits, cell_ewma_t::heap_index, cell_ewma_t::is_for_p_conn, circuit_t::n_cell_ewma, circuit_t::n_conn, next_circ_on_conn_p(), or_circuit_t::p_cell_ewma, prev_circ_on_conn_p(), TO_OR_CIRCUIT(), and tor_assert.

4.38.3.5 unsigned cell_ewma_get_tick (void)

Compute and return the current cell_ewma tick.

References approx_time(), and EWMA_TICK_LEN.

Referenced by init_circuit_base(), or_circuit_new(), and or_connection_new().

4.38.3.6 void cell_ewma_set_scale_factor (or_options_t * *options*, networkstatus_t * *consensus*)

Adjust the global cell scale factor based on **options**

References or_options_t::CircuitPriorityHalflife, EWMA_DEFAULT_HALFLIFE, EWMA_TICK_LEN, LD_OR, and networkstatus_get_param().

Referenced by networkstatus_set_current_consensus(), and options_act().

4.38.3.7 static unsigned cell_ewma_tick_from_timeval (const struct timeval * *now*, double * *remainder_out*) `[static]`

Given a timeval **now**, compute the cell_ewma tick in which it occurs and the fraction of the tick that has elapsed between the start of the tick and **now**. Return the former and store the latter in ***remainder_out**.

These tick values are not meant to be shared between Tor instances, or used for other purposes.

References EWMA_TICK_LEN.

Referenced by connection_or_flush_from_first_active_circuit().

4.38.3.8 static circuit_t* cell_ewma_to_circuit (cell_ewma_t * *ewma*) `[static]`

Given a **cell_ewma_t** (p. 27), return a pointer to the circuit containing it.

References cell_ewma_t::is_for_p_conn, SUBTYPE_P, and TO_CIRCUIT.

Referenced by connection_or_flush_from_first_active_circuit().

4.38.3.9 void cell_queue_append (cell_queue_t * *queue*, packed_cell_t * *cell*)

Append **cell** to the end of **queue**.

References cell_queue_t::head, cell_queue_t::n, packed_cell_t::next, cell_queue_t::tail, and tor_assert.

Referenced by cell_queue_append_packed_copy().

4.38.3.10 void cell_queue_append_packed_copy (cell_queue_t ∗ *queue*, const cell_t ∗ *cell*)

Append a newly allocated copy of **cell** to the end of **queue**

References cell_queue_append(), insertion_time_elem_t::counter, insertion_time_queue_t::first, get_options(), insertion_time_elem_t::insertion_time, cell_queue_t::insertion_times, insertion_time_queue_t::last, insertion_time_elem_t::next, and packed_cell_copy().

Referenced by append_cell_to_circuit_queue().

4.38.3.11 void cell_queue_clear (cell_queue_t ∗ *queue*)

Remove and free every cell in **queue**.

References insertion_time_queue_t::first, cell_queue_t::head, cell_queue_t::insertion_times, cell_queue_t::n, insertion_time_elem_t::next, packed_cell_t::next, packed_cell_free_unchecked(), cell_queue_t::tail, and tor_free.

Referenced by circuit_free().

4.38.3.12 static INLINE packed_cell_t∗ cell_queue_pop (cell_queue_t ∗ *queue*) `[static]`

Extract and return the cell at the head of **queue**; return NULL if **queue** is empty.

References cell_queue_t::head, cell_queue_t::n, packed_cell_t::next, cell_queue_t::tail, and tor_assert.

Referenced by connection_or_flush_from_first_active_circuit().

4.38.3.13 static void circuit_consider_sending_sendme (circuit_t ∗ *circ*, crypt_path_t ∗ *layer_hint*) `[static]`

Check if the deliver_window for circuit **circ** (at hop **layer_hint** if it's defined) is low enough that we should send a circuit-level sendme back down the circuit. If so, send enough sendmes that the window would be overfull if we sent any more.

References CIRCWINDOW_INCREMENT, CIRCWINDOW_START, circuit_t::deliver_window, crypt_path_t::deliver_window, LD_CIRC, and relay_send_command_from_edge().

Referenced by connection_edge_process_relay_cell().

4.38.3.14 static int circuit_consider_stop_edge_reading (circuit_t ∗ *circ*, crypt_path_t ∗ *layer_hint*) `[static]`

Check if the package window for **circ** is empty (at hop **layer_hint** if it's defined).

If yes, tell edge streams to stop reading and return 1. Else return 0.

References connection_stop_reading(), edge_connection_t::cpath_layer, LD_APP, or_circuit_t::n_streams, edge_connection_t::next_stream, crypt_path_t::package_window, circuit_t::package_window, TO_CONN, TO_OR_CIRCUIT(), and TO_ORIGIN_CIRCUIT().

Referenced by circuit_resume_edge_reading_helper(), and connection_edge_package_raw_inbuf().

4.38.3.15 static int circuit_package_relay_cell (cell_t * *cell*, circuit_t * *circ*, cell_direction_t *cell_direction*, crypt_path_t * *layer_hint*) `[static]`

Package a relay cell from an edge:

- Encrypt it to the right layer
- Append it to the appropriate cell_queue on **circ**.

References append_cell_to_circuit_queue(), assert_circuit_ok(), CELL_DIRECTION_OUT, CIRCUIT_IS_ORIGIN, crypt_path_t::f_crypto, crypt_path_t::f_digest, LD_BUG, LD_OR, circuit_t::n_conn, or_circuit_t::p_conn, or_circuit_t::p_crypto, or_circuit_t::p_digest, cell_t::payload, crypt_path_t::prev, relay_crypt_one_payload(), relay_set_digest(), TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by relay_send_command_from_edge().

4.38.3.16 int circuit_receive_relay_cell (cell_t * *cell*, circuit_t * *circ*, cell_direction_t *cell_direction*)

Receive a relay cell:

- Crypt it (encrypt if headed toward the origin or if we **are** the origin; decrypt if we're headed toward the exit).
- Check if recognized (if exitward).
- If recognized and the digest checks out, then find if there's a stream that the cell is intended for, and deliver it to the right connection_edge.
- If not recognized, then we need to relay it: append it to the appropriate cell_queue on **circ**.

Return -**reason** on failure.

References or_circuit_t::_base, append_cell_to_circuit_queue(), CELL_DIRECTION_IN, CELL_DIRECTION_OUT, cell_t::circ_id, CIRCUIT_IS_ORIGIN, CIRCUIT_PURPOSE_REND_ESTABLISHED, circuit_receive_relay_cell(), cell_t::command, connection_edge_process_relay_cell(), LD_BUG, LD_OR, LD_PROTOCOL, LD_REND, circuit_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, or_circuit_t::p_circ_id, or_circuit_t::p_conn, circuit_t::purpose, relay_crypt(), relay_lookup_conn(), or_circuit_t::rend_splice, TO_CIRCUIT, TO_OR_CIRCUIT(), and tor_assert.

Referenced by circuit_receive_relay_cell(), and command_process_relay_cell().

4.38.3.17 static void circuit_resume_edge_reading (circuit_t * *circ*, crypt_path_t * *layer_hint*) `[static]`

The circuit **circ** has received a circuit-level sendme (on hop **layer_hint**, if we're the OP). Go through all the attached streams and let them resume reading and packaging, if their stream windows allow it.

References CIRCUIT_IS_ORIGIN, circuit_resume_edge_reading_helper(), LD_APP, TO_OR_CIRCUIT(), and TO_ORIGIN_CIRCUIT().

Referenced by connection_edge_process_relay_cell().

4.38.3.18 static int circuit_resume_edge_reading_helper (edge_connection_t * *conn*, circuit_t * *circ*, crypt_path_t * *layer_hint*) `[static]`

A helper function for **circuit_resume_edge_reading()** (p. 1051) above. The arguments are the same, except that **conn** is the head of a linked list of edge streams that should each be considered.

References edge_connection_t::_base, circuit_consider_stop_edge_reading(), connection_edge_package_raw_inbuf(), connection_start_reading(), edge_connection_t::cpath_layer, connection_t::marked_for_close, edge_connection_t::next_stream, edge_connection_t::package_window, and TO_CONN.

Referenced by circuit_resume_edge_reading().

4.38.3.19 void clean_cell_pool (void)

Free excess storage in cell pool.

References tor_assert.

Referenced by run_scheduled_events().

4.38.3.20 static int compare_cell_ewma_counts (const void * *p1*, const void * *p2*) `[static]`

Helper for sorting **cell_ewma_t** (p. 27) values in their priority queue.

References cell_ewma_t::cell_count.

Referenced by add_cell_ewma_to_conn(), pop_first_cell_ewma_from_conn(), and remove_cell_ewma_from_conn().

4.38.3.21 static int connection_ap_process_end_not_open (relay_header_t * *rh*, cell_t * *cell*, origin_circuit_t * *circ*, edge_connection_t * *conn*, crypt_path_t * *layer_hint*) `[static]`

Called when we receive an END cell on a stream that isn't open yet, from the client side. Arguments are as for **connection_edge_process_relay_cell()** (p. 1053).

References edge_connection_t::_base, origin_circuit_t::_base, socks_request_t::address, addressmap_rewrite(), origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, edge_connection_t::chosen_exit_name, edge_connection_t::chosen_exit_optional, edge_connection_t::chosen_exit_retries, circuit_log_path(), client_dns_clear_failures(), client_dns_incr_failures(), client_dns_set_addressmap(), or_options_t::ClientDNSRejectInternalAddresses, connection_ap_detach_retriable(), connection_edge_is_rendezvous_stream(), control_event_stream_status(), edge_reason_is_retriable(), END_STREAM_REASON_FLAG_REMOTE, get_options(), get_uint32(), extend_info_t::identity_digest, is_internal_IP(), LD_APP, relay_header_t::length, connection_t::marked_for_close, MAX_RESOLVE_FAILURES, or_options_t::MaxCircuitDirtiness, cell_t::payload, policies_set_router_exitpolicy_to_reject_all(), RELAY_HEADER_SIZE, router_get_by_digest(), safe_str(), edge_connection_t::socks_request, stream_end_reason_to_string(), circuit_t::timestamp_dirty, tor_assert, tor_free, and tor_inet_aton().

Referenced by connection_edge_process_relay_cell_not_open().

4.38.3.22 void connection_edge_consider_sending_sendme (edge_connection_t * *conn*)

Called when we've just received a relay data cell, or when we've just finished flushing all bytes to stream **conn**.

If conn->outbuf is not too full, and our deliver window is low, send back a suitable number of stream-level sendme cells.

References edge_connection_t::_base, circuit_get_by_edge_conn(), connection_edge_send_command(), connection_outbuf_too_full(), edge_connection_t::cpath_layer, edge_connection_t::deliver_window, LD_APP, connection_t::outbuf_flushlen, STREAMWINDOW_INCREMENT, STREAMWINDOW_START, and TO_CONN.

Referenced by connection_edge_finished_flushing(), and connection_edge_process_relay_cell().

4.38.3.23 int connection_edge_package_raw_inbuf (edge_connection_t * *conn*, int *package_partial*)

While conn->inbuf has an entire relay payload of bytes on it, and the appropriate package windows aren't empty, grab a cell and send it down the circuit.

Return -1 (and send a RELAY_COMMAND_END cell if necessary) if conn should be marked for close, else return 0.

References edge_connection_t::_base, buf_datalen(), CELL_PAYLOAD_SIZE, circuit_consider_stop_edge_reading(), circuit_get_by_edge_conn(), connection_edge_send_command(), connection_fetch_from_buf(), connection_stop_reading(), edge_connection_t::cpath_layer, edge_connection_t::end_reason, END_STREAM_REASON_CANT_ATTACH, connection_t::inbuf, LD_APP, LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, crypt_path_t::package_window, circuit_t::package_window, edge_connection_t::package_window, RELAY_PAYLOAD_SIZE, connection_t::s, TO_CONN, and tor_assert.

Referenced by circuit_resume_edge_reading_helper(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), and connection_edge_process_relay_cell_not_open().

4.38.3.24 static int connection_edge_process_relay_cell (cell_t * *cell*, circuit_t * *circ*, edge_connection_t * *conn*, crypt_path_t * *layer_hint*) `[static]`

An incoming relay cell has arrived on circuit **circ**. If **conn** is NULL this is a control cell, else **cell** is destined for **conn**.

If **layer_hint** is defined, then we're the origin of the circuit, and it specifies the hop that packaged **cell**.

Return -reason if you want to warn and tear down the circuit, else 0.

References edge_connection_t::_base, circuit_consider_sending_sendme(), circuit_extend(), circuit_finish_handshake(), CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_S_REND_JOINED, circuit_resume_edge_reading(), circuit_send_next_onion_skin(), circuit_set_n_circid_orconn(), circuit_truncated(), CIRCWINDOW_INCREMENT, relay_header_t::command, connection_edge_consider_sending_sendme(), connection_edge_end(), connection_edge_package_raw_inbuf(), connection_edge_process_relay_cell_not_open(), connection_exit_begin_conn(), connection_exit_begin_resolve(), connection_-

or_send_destroy(), connection_start_reading(), connection_state_is_open(), edge_connection_t::deliver_window, circuit_t::deliver_window, crypt_path_t::deliver_window, circuit_t::dirreq_id, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_REMOTE, socks_request_t::has_finished, connection_t::hold_open_until_flushed, LD_APP, LD_BUG, LD_PROTOCOL, relay_header_t::length, connection_t::marked_for_close, circuit_t::n_circ_id, circuit_t::n_conn, edge_connection_t::package_window, circuit_t::package_window, crypt_path_t::package_window, cell_t::payload, circuit_t::purpose, RELAY_HEADER_SIZE, relay_header_unpack(), RELAY_PAYLOAD_SIZE, relay_send_command_from_edge(), rend_process_relay_cell(), connection_t::s, edge_connection_t::socks_request, stats_n_data_bytes_received, stats_n_data_cells_received, stream_end_reason_to_string(), edge_connection_t::stream_id, relay_header_t::stream_id, STREAMWINDOW_INCREMENT, TO_CONN, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by circuit_receive_relay_cell().

4.38.3.25 static int connection_edge_process_relay_cell_not_open (relay_header_t * *rh*, cell_t * *cell*, circuit_t * *circ*, edge_connection_t * *conn*, crypt_path_t * *layer_hint*) [static]

An incoming relay cell has arrived from circuit **circ** to stream **conn**.

The arguments here are the same as in **connection_edge_process_relay_cell()** (p. 1053) below; this function is called from there when **conn** is defined and not in an open state.

References edge_connection_t::_base, socks_request_t::address, AP_CONN_STATE_CONNECT_WAIT, AP_CONN_STATE_OPEN, AP_CONN_STATE_RESOLVE_WAIT, edge_connection_t::chosen_exit_name, CIRCUIT_IS_ORIGIN, circuit_log_path(), client_dns_set_addressmap(), socks_request_t::command, relay_header_t::command, conn_state_to_string(), CONN_TYPE_AP, CONN_TYPE_DIR, connection_ap_handshake_socks_reply(), connection_ap_handshake_socks_resolved(), connection_ap_process_end_not_open(), connection_edge_end(), connection_edge_package_raw_inbuf(), consensus_is_waiting_for_certs(), control_event_bootstrap(), count_loading_descriptors_progress(), DIR_PURPOSE_FETCH_CERTIFICATE, DIR_PURPOSE_FETCH_CONSENSUS, DIR_PURPOSE_FETCH_SERVERDESC, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, END_STREAM_REASON_FLAG_ALREADY_SOCKS_REPLIED, END_STREAM_REASON_FLAG_REMOTE, get_options(), get_uint32(), socks_request_t::has_finished, INET_NTOA_BUF_LEN, is_internal_IP(), LD_APP, LD_PROTOCOL, relay_header_t::length, cell_t::payload, connection_t::purpose, RELAY_HEADER_SIZE, remap_event_helper(), edge_connection_t::socks_request, connection_t::state, connection_t::timestamp_lastread, TO_CONN, TO_ORIGIN_CIRCUIT(), tor_assert, tor_inet_ntoa(), and connection_t::type.

Referenced by connection_edge_process_relay_cell().

4.38.3.26 int connection_edge_send_command (edge_connection_t * *fromconn*, uint8_t *relay_command*, const char * *payload*, size_t *payload_len*)

Make a relay cell out of **relay_command** and **payload**, and send it onto the open circuit **circ**. **fromconn** is the stream that's sending the relay cell, or NULL if it's a control cell. **cpath_layer** is NULL for OR->OP cells, or the destination hop for OP->OR cells.

If you can't send the cell, mark the circuit for close and return -1. Else return 0.

References edge_connection_t::_base, CONN_TYPE_AP, edge_connection_t::cpath_-

layer, edge_connection_t::edge_has_sent_end, edge_connection_t::end_reason, LD_APP, LD_BUG, connection_t::marked_for_close, connection_t::marked_for_close_file, edge_connection_t::on_circuit, relay_send_command_from_edge(), edge_connection_t::stream_id, TO_CONN, tor_assert, and connection_t::type.

Referenced by connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_edge_consider_sending_sendme(), connection_edge_end(), connection_edge_finished_connecting(), connection_edge_package_raw_inbuf(), connection_exit_connect(), connection_exit_connect_dir(), send_resolved_cell(), and send_resolved_hostname_cell().

4.38.3.27 int connection_or_flush_from_first_active_circuit (or_connection_t * *conn*, int *max*, time_t *now*)

Pull as many cells as possible (but no more than **max**) from the queue of the first active circuit on **conn**, and write them to **conn**->outbuf. Return the number of cells written. Advance the active circuit pointer to the next active circuit in the ring.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuit_pqueue_last_recalibrated, or_connection_t::active_circuits, add_cell_ewma_to_conn(), packed_cell_t::body, cell_ewma_tick_from_timeval(), cell_ewma_to_circuit(), CELL_NETWORK_SIZE, CELL_QUEUE_LOWWATER_SIZE, cell_queue_pop(), CIRCUIT_IS_ORIGIN, insertion_time_elem_t::counter, DIRREQ_CIRC_QUEUE_FLUSHED, ewma_scale_factor, insertion_time_queue_t::first, geoip_change_dirreq_state(), get_options(), cell_queue_t::head, insertion_time_elem_t::insertion_time, cell_queue_t::insertion_times, insertion_time_queue_t::last, LD_BUG, LD_GENERAL, make_circuit_inactive_on_conn(), cell_queue_t::n, circuit_t::n_conn, circuit_t::n_conn_cells, insertion_time_elem_t::next, next_circ_on_conn_p(), or_circuit_t::p_conn_cells, packed_cell_free_unchecked(), pop_first_cell_ewma_from_conn(), or_circuit_t::processed_cells, scale_active_circuits(), set_streams_blocked_on_circ(), circuit_t::streams_blocked_on_n_conn, circuit_t::streams_blocked_on_p_conn, or_connection_t::timestamp_last_added_nonpadding, TO_CONN, TO_OR_CIRCUIT(), tor_assert, and or_circuit_t::total_cell_waiting_time.

Referenced by append_cell_to_circuit_queue(), and connection_or_flushed_some().

4.38.3.28 void connection_or_unlink_all_active_circs (or_connection_t * *orconn*)

Remove all circuits from the list of circuits with pending cells on **conn**.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuits, next_circ_on_conn_p(), prev_circ_on_conn_p(), and smartlist_clear().

Referenced by circuit_unlink_all_from_or_conn().

4.38.3.29 const char* decode_address_from_payload (tor_addr_t * *addr_out*, const char * *payload*, int *payload_len*)

Given **payload_len** bytes at **payload**, starting with an address encoded as by **append_address_to_payload()** (p. 823), try to decode the address into ***addr_out**. Return the next byte in the payload after the address on success, or NULL on failure.

References get_uint32(), tor_addr_from_ipv4n(), tor_addr_from_ipv6_bytes(), and tor_addr_make_unspec().

Referenced by command_process_netinfo_cell().

4.38.3.30 void dump_cell_pool_usage (int *severity*)

Log current statistics for cell pool allocation at log level **severity**.

References _circuit_get_global_list(), CIRCUIT_IS_ORIGIN, LD_MM, cell_queue_t::n, circuit_t::n_conn_cells, circuit_t::next, or_circuit_t::p_conn_cells, and TO_OR_CIRCUIT().

Referenced by dumpmemusage().

4.38.3.31 static int edge_reason_is_retriable (int *reason*) [static]

Return 1 if reason is something that you should retry if you get the end cell before you've connected; else return 0.

Referenced by connection_ap_process_end_not_open().

4.38.3.32 void free_cell_pool (void)

Free all storage used to hold cells (and insertion times if we measure cell statistics).

Referenced by tor_free_all().

4.38.3.33 static INLINE double get_scale_factor (unsigned *from_tick*, unsigned *to_tick*) [static]

Return the multiplier necessary to convert the value of a cell sent in 'from_tick' to one sent in 'to_tick'.

Referenced by scale_active_circuits(), and scale_single_cell_ewma().

4.38.3.34 void init_cell_pool (void)

Allocate structures to hold cells.

References tor_assert.

Referenced by do_main_loop().

4.38.3.35 void make_circuit_active_on_conn (circuit_t * *circ*, or_connection_t * *conn*)

Add **circ** to the list of circuits with pending cells on **conn**. No effect if **circ** is already linked.

References or_connection_t::active_circuits, add_cell_ewma_to_conn(), circuit_t::n_cell_ewma, circuit_t::n_conn, next_circ_on_conn_p(), or_circuit_t::p_cell_ewma, or_circuit_t::p_conn, prev_circ_on_conn_p(), TO_OR_CIRCUIT(), and tor_assert.

Referenced by append_cell_to_circuit_queue(), and HT_HEAD().

4.38.3.36 void make_circuit_inactive_on_conn (circuit_t ∗ *circ*, or_connection_t ∗ *conn*)

Remove **circ** from the list of circuits with pending cells on **conn**. No effect if **circ** is already unlinked.

References or_connection_t::active_circuits, circuit_t::n_cell_ewma, circuit_t::n_conn, next_circ_on_conn_p(), or_circuit_t::p_cell_ewma, or_circuit_t::p_conn, prev_circ_on_conn_p(), remove_cell_ewma_from_conn(), TO_OR_CIRCUIT(), and tor_assert.

Referenced by connection_or_flush_from_first_active_circuit(), and HT_HEAD().

4.38.3.37 static INLINE circuit_t∗∗ next_circ_on_conn_p (circuit_t ∗ *circ*, or_connection_t ∗ *conn*) `[static]`

Return a pointer to the "next_active_on_{n,p}_conn" pointer of **circ**, depending on whether **conn** matches n_conn or p_conn.

References circuit_t::n_conn, circuit_t::next_active_on_n_conn, or_circuit_t::next_active_on_p_conn, or_circuit_t::p_conn, TO_OR_CIRCUIT(), and tor_assert.

Referenced by assert_active_circuits_ok(), connection_or_flush_from_first_active_circuit(), connection_or_unlink_all_active_circs(), make_circuit_active_on_conn(), and make_circuit_inactive_on_conn().

4.38.3.38 static INLINE packed_cell_t∗ packed_cell_alloc (void) `[static]`

Allocate and return a new **packed_cell_t** (p. 184).

Referenced by packed_cell_copy().

4.38.3.39 static INLINE packed_cell_t∗ packed_cell_copy (const cell_t ∗ *cell*) `[static]`

Allocate a new copy of packed **cell**.

References cell_pack(), packed_cell_t::next, and packed_cell_alloc().

Referenced by cell_queue_append_packed_copy().

4.38.3.40 static INLINE void packed_cell_free_unchecked (packed_cell_t ∗ *cell*) `[static]`

Release storage held by **cell**.

Referenced by cell_queue_clear(), and connection_or_flush_from_first_active_circuit().

4.38.3.41 static cell_ewma_t∗ pop_first_cell_ewma_from_conn (or_connection_t ∗ *conn*) `[static]`

Remove and return the first **cell_ewma_t** (p. 27) from conn's priority queue of active circuits. Requires that the priority queue is nonempty.

References or_connection_t::active_circuit_pqueue, compare_cell_ewma_counts(), smartlist_pqueue_pop(), and STRUCT_OFFSET.

Referenced by connection_or_flush_from_first_active_circuit().

4.38.3.42 static INLINE circuit_t** prev_circ_on_conn_p (circuit_t * *circ*, or_connection_t * *conn*) `[static]`

Return a pointer to the "prev_active_on_{n,p}_conn" pointer of **circ**, depending on whether **conn** matches n_conn or p_conn.

References circuit_t::n_conn, or_circuit_t::p_conn, circuit_t::prev_active_on_n_conn, or_circuit_t::prev_active_on_p_conn, TO_OR_CIRCUIT(), and tor_assert.

Referenced by assert_active_circuits_ok(), connection_or_unlink_all_active_circs(), make_circuit_active_on_conn(), and make_circuit_inactive_on_conn().

4.38.3.43 static const char* relay_command_to_string (uint8_t *command*) `[static]`

Convert the relay **command** into a human-readable string.

Referenced by relay_send_command_from_edge().

4.38.3.44 static int relay_crypt (circuit_t * *circ*, cell_t * *cell*, cell_direction_t *cell_direction*, crypt_path_t ** *layer_hint*, char * *recognized*) `[static]`

Do the appropriate en/decryptions for **cell** arriving on **circ** in direction **cell_direction**.

If cell_direction == CELL_DIRECTION_IN:

- If we're at the origin (we're the OP), for hops 1..N, decrypt cell. If recognized, stop.
- Else (we're not the OP), encrypt one hop. Cell is not recognized.

If cell_direction == CELL_DIRECTION_OUT:

- decrypt one hop. Check if recognized.

If cell is recognized, set *recognized to 1, and set *layer_hint to the hop that recognized it.

Return -1 to indicate that we should mark the circuit for close, else return 0.

References crypt_path_t::b_crypto, crypt_path_t::b_digest, CELL_DIRECTION_IN, CELL_DIRECTION_OUT, CIRCUIT_IS_ORIGIN, origin_circuit_t::cpath, LD_OR, LD_PROTOCOL, crypt_path_t::next, cell_t::payload, relay_header_t::recognized, relay_crypt_one_payload(), relay_digest_matches(), relay_header_unpack(), crypt_path_t::state, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by circuit_receive_relay_cell().

4.38.3.45 static int relay_crypt_one_payload (crypto_cipher_env_t * *cipher*, char * *in*, int *encrypt_mode*) `[static]`

Apply **cipher** to CELL_PAYLOAD_SIZE bytes of **in** (in place).

If **encrypt_mode** is 1 then encrypt, else decrypt.

Return -1 if the crypto fails, else return 0.

References CELL_PAYLOAD_SIZE, crypto_cipher_crypt_inplace(), and LD_BUG.

Referenced by circuit_package_relay_cell(), and relay_crypt().

4.38.3.46 static int relay_digest_matches (crypto_digest_env_t ∗ *digest*, cell_t ∗ *cell*) [static]

Does the digest for this circuit indicate that this cell is for us?

Update digest from the payload of cell (with the integrity part set to 0). If the integrity part is valid, return 1, else restore digest and cell to their original state and return 0.

References CELL_PAYLOAD_SIZE, crypto_digest_add_bytes(), crypto_digest_assign(), crypto_digest_dup(), crypto_digest_get_digest(), crypto_free_digest_env(), relay_header_t::integrity, cell_t::payload, relay_header_pack(), and relay_header_unpack().

Referenced by relay_crypt().

4.38.3.47 void relay_header_pack (char ∗ *dest*, const relay_header_t ∗ *src*)

Pack the **relay_header_t** (p. 190) host-order structure **src** into network-order in the buffer **dest**. See tor-spec.txt for details about the wire format.

References relay_header_t::command, relay_header_t::integrity, relay_header_t::length, relay_header_t::recognized, set_uint16(), and relay_header_t::stream_id.

Referenced by relay_digest_matches(), relay_send_command_from_edge(), and relay_set_digest().

4.38.3.48 void relay_header_unpack (relay_header_t ∗ *dest*, const char ∗ *src*)

Unpack the network-order buffer **src** into a host-order **relay_header_t** (p. 190) structure **dest**.

References relay_header_t::command, get_uint16(), relay_header_t::integrity, relay_header_t::length, relay_header_t::recognized, and relay_header_t::stream_id.

Referenced by circuit_extend(), connection_edge_process_relay_cell(), connection_exit_begin_conn(), connection_exit_begin_resolve(), relay_crypt(), relay_digest_matches(), relay_lookup_conn(), and relay_set_digest().

4.38.3.49 static edge_connection_t ∗ relay_lookup_conn (circuit_t ∗ *circ*, cell_t ∗ *cell*, cell_direction_t *cell_direction*, crypt_path_t ∗ *layer_hint*) [static]

If cell's stream_id matches the stream_id of any conn that's attached to circ, return that conn, else return NULL.

References edge_connection_t::_base, CELL_DIRECTION_OUT, CIRCUIT_IS_ORIGIN, connection_edge_is_rendezvous_stream(), edge_connection_t::cpath_layer, LD_APP, connection_t::marked_for_close, edge_connection_t::next_stream, cell_t::payload, relay_header_unpack(), edge_connection_t::stream_id, relay_header_t::stream_id, TO_OR_CIRCUIT(), and TO_ORIGIN_CIRCUIT().

Referenced by circuit_receive_relay_cell().

4.38.3.50 int relay_send_command_from_edge (uint16_t *stream_id*, circuit_t ∗ *circ*, uint8_t *relay_command*, const char ∗ *payload*, size_t *payload_len*, crypt_path_t ∗ *cpath_layer*)

Make a relay cell out of **relay_command** and **payload**, and send it onto the open circuit **circ**. **stream_id** is the ID on **circ** for the stream that's sending the relay cell, or 0 if it's a control cell. **cpath_layer** is NULL for OR->OP cells, or the destination hop for OP->OR cells.

If you can't send the cell, mark the circuit for close and return -1. Else return 0.

References approx_time(), CELL_DIRECTION_IN, CELL_DIRECTION_OUT, cell_t::circ_id, CIRCUIT_IS_ORIGIN, circuit_package_relay_cell(), CIRCUIT_PURPOSE_IS_ESTABLISHED_REND, or_connection_t::client_used, relay_header_t::command, cell_t::command, origin_circuit_t::cpath, DIRREQ_END_CELL_SENT, circuit_t::dirreq_id, geoip_change_dirreq_state(), LD_BUG, LD_OR, relay_header_t::length, circuit_t::n_circ_id, circuit_t::n_conn, or_circuit_t::p_circ_id, cell_t::payload, circuit_t::purpose, relay_command_to_string(), origin_circuit_t::relay_early_cells_sent, origin_circuit_t::relay_early_commands, relay_header_pack(), RELAY_HEADER_SIZE, RELAY_PAYLOAD_SIZE, origin_circuit_t::remaining_relay_early_cells, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), relay_header_t::stream_id, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), tor_assert, and tor_free.

Referenced by circuit_consider_sending_sendme(), circuit_send_next_onion_skin(), command_process_created_cell(), command_process_destroy_cell(), connection_edge_process_relay_cell(), connection_edge_send_command(), relay_send_end_cell_from_edge(), rend_client_send_establish_rendezvous(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_intro_has_opened(), rend_service_rendezvous_has_opened(), and router_perform_bandwidth_test().

4.38.3.51 static void relay_set_digest (crypto_digest_env_t ∗ *digest*, cell_t ∗ *cell*) `[static]`

Update digest from the payload of cell. Assign integrity part to cell.

References CELL_PAYLOAD_SIZE, crypto_digest_add_bytes(), crypto_digest_get_digest(), relay_header_t::integrity, cell_t::payload, relay_header_pack(), and relay_header_unpack().

Referenced by circuit_package_relay_cell().

4.38.3.52 static void remap_event_helper (edge_connection_t ∗ *conn*, uint32_t *new_addr*) `[static]`

Helper: change the socks_request->address field on conn to the dotted-quad representation of **new_addr** (given in host order), and send an appropriate REMAP event.

References socks_request_t::address, control_event_stream_status(), REMAP_STREAM_SOURCE_EXIT, edge_connection_t::socks_request, and tor_inet_ntoa().

Referenced by connection_edge_process_relay_cell_not_open().

4.38.3.53 static void remove_cell_ewma_from_conn (or_connection_t ∗ *conn*, cell_ewma_t ∗ *ewma*) `[static]`

Remove **ewma** from **conn**'s priority queue of active circuits

References or_connection_t::active_circuit_pqueue, compare_cell_ewma_counts(), cell_ewma_t::heap_index, smartlist_pqueue_remove(), STRUCT_OFFSET, and tor_assert.

Referenced by make_circuit_inactive_on_conn().

4.38.3.54 static void scale_active_circuits (or_connection_t * *conn*, unsigned *cur_tick*) [static]

Adjust the cell count of every active circuit on **conn** so that they are scaled with respect to **cur_tick**

Ordinarily it isn't okay to change the value of an element in a heap, but it's okay here, since we are preserving the order.

References or_connection_t::active_circuit_pqueue, or_connection_t::active_circuit_pqueue_last_recalibrated, get_scale_factor(), and tor_assert.

Referenced by connection_or_flush_from_first_active_circuit().

4.38.3.55 static void scale_single_cell_ewma (cell_ewma_t * *ewma*, unsigned *cur_tick*) [static]

Adjust the cell count of **ewma** so that it is scaled with respect to **cur_tick**

References cell_ewma_t::cell_count, get_scale_factor(), and cell_ewma_t::last_adjusted_tick.

Referenced by add_cell_ewma_to_conn().

4.38.3.56 static void set_streams_blocked_on_circ (circuit_t * *circ*, or_connection_t * *orconn*, int *block*) [static]

Block (if **block** is true) or unblock (if **block** is false) every edge connection that is using **circ** to write to **orconn**, and start or stop reading as appropriate.

References CIRCUIT_IS_ORIGIN, connection_is_reading(), connection_start_reading(), connection_stop_reading(), edge_connection_t::edge_blocked_on_circ, circuit_t::n_conn, or_circuit_t::n_streams, edge_connection_t::next_stream, origin_circuit_t::p_streams, connection_t::read_event, circuit_t::streams_blocked_on_n_conn, circuit_t::streams_blocked_on_p_conn, TO_CONN, TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_assert.

Referenced by append_cell_to_circuit_queue(), and connection_or_flush_from_first_active_circuit().

4.38.4 Variable Documentation

4.38.4.1 struct timeval cached_time_hires = {0, 0} [static]

Cache the current hi-res time; the cache gets reset when libevent calls us.

4.38.4.2 mp_pool_t* cell_pool = NULL [static]

A memory pool to allocate **packed_cell_t** (p. 184) objects.

4.38.4.3 double ewma_scale_factor = 0.1 [static]

The per-tick scale factor to be used when computing cell-count EWMA values. (A cell sent N ticks before the start of the current tick has value ewma_scale_factor ** N.)

Referenced by connection_or_flush_from_first_active_circuit().

4.38.4.4 mp_pool_t* it_pool = NULL [static]

Memory pool to allocate **insertion_time_elem_t** (p. 104) objects used for cell statistics.

4.38.4.5 uint64_t stats_n_data_bytes_packaged = 0

How many bytes of data have we put in relay_data cells have we built, ever? This would be RELAY_PAYLOAD_SIZE*stats_n_data_cells_packaged if every relay cell we ever sent were completely full of data.

Referenced by dumpstats().

4.38.4.6 uint64_t stats_n_data_bytes_received = 0

How many bytes of data have we received relay_data cells, ever? This would be RELAY_PAYLOAD_SIZE*stats_n_data_cells_packaged if every relay cell we ever received were completely full of data.

Referenced by connection_edge_process_relay_cell(), and dumpstats().

4.38.4.7 uint64_t stats_n_data_cells_packaged = 0

How many relay_data cells have we built, ever?

Referenced by dumpstats().

4.38.4.8 uint64_t stats_n_data_cells_received = 0

How many relay_data cells have we received, ever?

Referenced by connection_edge_process_relay_cell(), and dumpstats().

4.38.4.9 uint64_t stats_n_relay_cells_delivered = 0

Stats: how many relay cells have been delivered to streams at this hop?

Referenced by dumpstats().

4.38.4.10 uint64_t stats_n_relay_cells_relayed = 0

Stats: how many relay cells have originated at this hop, or have been relayed onward (not recognized at this hop)?

Referenced by dumpstats().

4.38.4.11 int total_cells_allocated = 0 [static]

The total number of cells we have allocated from the memory pool.

4.39 rendclient.c File Reference

Client code to access location-hidden services.

`#include "or.h"`

Defines

- #define **REND_HID_SERV_DIR_REQUERY_PERIOD** (15 * 60)

Functions

- void **rend_client_introcirc_has_opened** (**origin_circuit_t** *circ)
- static int **rend_client_send_establish_rendezvous** (**origin_circuit_t** *circ)
- int **rend_client_send_introduction** (**origin_circuit_t** *introcirc, **origin_circuit_t** *rendcirc)
- void **rend_client_rendcirc_has_opened** (**origin_circuit_t** *circ)
- int **rend_client_introduction_acked** (**origin_circuit_t** *circ, const char *request, size_t request_len)
- static time_t **lookup_last_hid_serv_request** (**routerstatus_t** *hs_dir, const char *desc_id_base32, time_t now, int set)
- static void **directory_clean_last_hid_serv_requests** (void)
- static int **directory_get_from_hs_dir** (const char *desc_id, const **rend_data_t** *rend_query)
- void **rend_client_refetch_v2_renddesc** (const **rend_data_t** *rend_query)
- int **rend_client_remove_intro_point** (**extend_info_t** *failed_intro, const **rend_data_t** *rend_query)
- int **rend_client_rendezvous_acked** (**origin_circuit_t** *circ, const char *request, size_t request_len)
- int **rend_client_receive_rendezvous** (**origin_circuit_t** *circ, const char *request, size_t request_len)
- void **rend_client_desc_trynow** (const char *query)
- **extend_info_t** * **rend_client_get_random_intro** (const **rend_data_t** *rend_query)
- **rend_service_authorization_t** * **rend_client_lookup_service_authorization** (const char *onion_address)
- static void **rend_service_authorization_free** (**rend_service_authorization_t** *auth)
- static void **rend_service_authorization_strmap_item_free** (void *service_auth)
- void **rend_service_authorization_free_all** (void)
- int **rend_parse_service_authorization** (**or_options_t** *options, int validate_only)

Variables

- static strmap_t * **last_hid_serv_requests** = NULL
- static strmap_t * **auth_hid_servs** = NULL

4.39.1 Detailed Description

Client code to access location-hidden services.

4.39.2 Define Documentation

4.39.2.1 #define REND_HID_SERV_DIR_REQUERY_PERIOD (15 * 60)

The period for which a hidden service directory cannot be queried for the same descriptor ID again.

Referenced by directory_clean_last_hid_serv_requests(), and directory_get_from_hs_dir().

4.39.3 Function Documentation

4.39.3.1 static void directory_clean_last_hid_serv_requests (void) `[static]`

Clean the history of request times to hidden service directories, so that it does not contain requests older than REND_HID_SERV_DIR_REQUERY_PERIOD seconds any more.

References REND_HID_SERV_DIR_REQUERY_PERIOD, strmap_iter_done(), strmap_iter_get(), strmap_iter_init(), strmap_iter_next(), strmap_iter_next_rmv(), and tor_free.

Referenced by directory_get_from_hs_dir().

4.39.3.2 static int directory_get_from_hs_dir (const char * *desc_id*, const rend_data_t * *rend_query*) `[static]`

Determine the responsible hidden service directories for **desc_id** and fetch the descriptor belonging to that ID from one of them. Only send a request to hidden service directories that we did not try within the last REND_HID_SERV_DIR_REQUERY_PERIOD seconds; on success, return 1, in the case that no hidden service directory is left to ask for the descriptor, return 0, and in case of a failure -1. **query** is only passed for pretty log statements.

References rend_data_t::auth_type, base32_encode(), base64_encode(), rend_data_t::descriptor_cookie, DIGEST_LEN, DIR_PURPOSE_FETCH_RENDDESC_V2, directory_clean_last_hid_serv_requests(), directory_initiate_command_routerstatus_rend(), escaped_safe_str_client(), hid_serv_get_responsible_directories(), LD_BUG, LD_REND, lookup_last_hid_serv_request(), rend_data_t::onion_address, REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, REND_DESC_ID_V2_LEN_BASE32, REND_HID_SERV_DIR_REQUERY_PERIOD, router_get_by_digest(), ROUTER_PURPOSE_GENERAL, smartlist_choose(), smartlist_create(), smartlist_free(), and tor_assert.

Referenced by rend_client_refetch_v2_renddesc().

4.39.3.3 static time_t lookup_last_hid_serv_request (routerstatus_t * *hs_dir*, const char * *desc_id_base32*, time_t *now*, int *set*) `[static]`

Look up the last request time to hidden service directory **hs_dir** for descriptor ID **desc_id_base32**. If **set** is non-zero, assign the current time **now** and return that. Otherwise, return the most recent request time, or 0 if no such request has been sent before.

References base32_encode(), DIGEST_LEN, routerstatus_t::identity_digest, REND_DESC_ID_V2_LEN_BASE32, strmap_get_lc(), strmap_set(), tor_free, and tor_snprintf().

Referenced by directory_get_from_hs_dir().

4.39.3.4 void rend_client_desc_trynow (const char * *query*)

Find all the apconns in state AP_CONN_STATE_RENDDESC_WAIT that are waiting on **query**. If there's a working cache entry here with at least one intro point, move them to the next state.

References edge_connection_t::_base, AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RENDDESC_WAIT, assert_connection_ok(), CONN_TYPE_AP, connection_ap_handshake_attach_circuit(), END_STREAM_REASON_CANT_ATTACH, get_connection_array(), rend_service_descriptor_t::intro_nodes, LD_REND, connection_t::marked_for_close, rend_data_t::onion_address, rend_cache_entry_t::parsed, rend_cache_lookup_entry(), rend_cmp_service_ids(), edge_connection_t::rend_data, safe_str_client(), connection_t::state, connection_t::timestamp_created, connection_t::timestamp_lastread, connection_t::timestamp_lastwritten, TO_CONN, and TO_EDGE_CONN().

Referenced by connection_dir_client_reached_eof(), and rend_client_refetch_v2_renddesc().

4.39.3.5 extend_info_t* rend_client_get_random_intro (const rend_data_t * *rend_query*)

Return a newly allocated extend_info_t* for a randomly chosen introduction point for the named hidden service. Return NULL if all introduction points have been tried and failed.

References crypto_rand_int(), rend_intro_point_t::extend_info, extend_info_dup(), extend_info_free(), extend_info_from_router(), rend_service_descriptor_t::intro_nodes, LD_REND, extend_info_t::nickname, rend_data_t::onion_address, extend_info_t::onion_key, rend_cache_entry_t::parsed, rend_cache_lookup_entry(), rend_intro_point_free(), router_get_by_nickname(), safe_str_client(), and smartlist_del().

Referenced by circuit_get_open_circ_or_launch(), and rend_client_introduction_acked().

4.39.3.6 void rend_client_introcirc_has_opened (origin_circuit_t * *circ*)

Called when we've established a circuit to an introduction point: send the introduction request.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_INTRODUCING, connection_ap_attach_pending(), origin_circuit_t::cpath, LD_REND, circuit_t::purpose, and tor_assert.

Referenced by circuit_has_opened().

4.39.3.7 int rend_client_introduction_acked (origin_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Called when get an ACK or a NAK for a REND_INTRODUCE1 cell.

References origin_circuit_t::_base, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCLAUNCH_IS_INTERNAL, circuit_extend_to_new_exit(), circuit_get_by_rend_query_and_purpose(), circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, CIRCUIT_PURPOSE_C_INTRODUCE_ACKED, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, escaped_safe_str_client(), extend_info_free(), LD_PROTOCOL, LD_REND, circuit_t::n_circ_id, extend_info_t::nickname, rend_data_t::onion_address, circuit_t::purpose, origin_circuit_t::remaining_relay_early_cells, rend_client_get_random_intro(), rend_client_remove_intro_point(), origin_circuit_t::rend_data, TO_CIRCUIT, and tor_assert.

Referenced by rend_process_relay_cell().

4.39.3.8 rend_service_authorization_t∗ rend_client_lookup_service_authorization (const char ∗ *onion_address*)

Look up the client-side authorization for the hidden service with **onion_address**. Return NULL if no authorization is available for that address.

References strmap_get(), and tor_assert.

4.39.3.9 int rend_client_receive_rendezvous (origin_circuit_t ∗ *circ*, const char ∗ *request*, size_t *request_len*)

Bob sent us a rendezvous cell; join the circuits.

References origin_circuit_t::_base, origin_circuit_t::build_state, circuit_init_cpath_crypto(), circuit_initial_package_window(), CIRCUIT_PURPOSE_C_REND_JOINED, CIRCUIT_PURPOSE_C_REND_READY, CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED, CIRCWINDOW_START, connection_ap_attach_pending(), origin_circuit_t::cpath, crypto_dh_compute_secret(), crypto_dh_free(), crypt_path_t::deliver_window, crypt_path_t::dh_handshake_state, DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, LD_REND, onion_append_to_cpath(), crypt_path_t::package_window, cpath_build_state_t::pending_final_cpath, circuit_t::purpose, crypt_path_t::state, TO_CIRCUIT, and tor_assert.

Referenced by rend_process_relay_cell().

4.39.3.10 void rend_client_refetch_v2_renddesc (const rend_data_t ∗ *rend_query*)

Unless we already have a descriptor for **rend_query** with at least one (possibly) working introduction point in it, start a connection to a hidden service directory to fetch a v2 rendezvous service descriptor.

References rend_data_t::auth_type, crypto_rand_int(), rend_data_t::descriptor_cookie, DIGEST_LEN, directory_get_from_hs_dir(), get_options(), LD_REND, rend_data_t::onion_address, rend_cache_lookup_entry(), rend_client_desc_trynow(), rend_compute_v2_desc_id(), REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS, safe_str_client(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), connection_about_to_close_connection(), and rend_client_remove_intro_point().

4.39.3.11 int rend_client_remove_intro_point (extend_info_t ∗ *failed_intro*, const rend_data_t ∗ *rend_query*)

Remove failed_intro from ent. If ent now has no intro points, or service is unrecognized, then launch a new renddesc fetch.

Return -1 if error, 0 if no intro points remain or service unrecognized, 1 if recognized and some intro points remain.

References AP_CONN_STATE_CIRCUIT_WAIT, AP_CONN_STATE_RENDDESC_WAIT, CONN_TYPE_AP, connection_get_by_type_state_rendquery(), DIGEST_LEN, escaped_safe_str_client(), rend_intro_point_t::extend_info, extend_info_t::identity_digest, rend_service_descriptor_t::intro_nodes, LD_BUG, LD_REND, rend_data_t::onion_address, rend_cache_entry_t::parsed, rend_cache_lookup_entry(), rend_client_refetch_v2_renddesc(), rend_intro_point_free(), smartlist_del(), and connection_t::state.

Referenced by _circuit_mark_for_close(), and rend_client_introduction_acked().

4.39.3.12 void rend_client_rendcirc_has_opened (origin_circuit_t * *circ*)

Called when a rendezvous circuit is open; sends a establish rendezvous circuit as appropriate.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_ESTABLISH_REND, LD_REND, circuit_t::purpose, rend_client_send_establish_rendezvous(), and tor_assert.

Referenced by circuit_get_open_circ_or_launch(), and circuit_has_opened().

4.39.3.13 int rend_client_rendezvous_acked (origin_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Called when we receive a RENDEZVOUS_ESTABLISHED cell; changes the state of the circuit to C_REND_READY.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_ESTABLISH_REND, CIRCUIT_PURPOSE_C_REND_READY, connection_ap_attach_pending(), LD_PROTOCOL, LD_REND, circuit_t::purpose, and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.39.3.14 static int rend_client_send_establish_rendezvous (origin_circuit_t * *circ*) `[static]`

Send the establish-rendezvous cell along a rendezvous circuit. if it fails, mark the circ for close and return -1. else return 0.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_C_ESTABLISH_REND, origin_circuit_t::cpath, crypto_rand(), LD_BUG, LD_GENERAL, LD_REND, crypt_path_t::prev, circuit_t::purpose, relay_send_command_from_edge(), rend_data_t::rend_cookie, REND_COOKIE_LEN, origin_circuit_t::rend_data, TO_CIRCUIT, and tor_assert.

Referenced by rend_client_rendcirc_has_opened().

4.39.3.15 int rend_client_send_introduction (origin_circuit_t * *introcirc*, origin_circuit_t * *rendcirc*)

Called when we're trying to connect an ap conn; sends an INTRODUCE1 cell down introcirc if possible.

References origin_circuit_t::_base, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCUIT_PURPOSE_C_INTRODUCING, CIRCUIT_PURPOSE_C_REND_READY, DIGEST_LEN, escaped_safe_str_client(), extend_info_t::identity_digest, rend_service_descriptor_t::intro_nodes, LD_REND, rend_data_t::onion_address, rend_cache_entry_t::parsed, circuit_t::purpose, RELAY_PAYLOAD_SIZE, rend_cache_lookup_entry(), rend_cmp_service_ids(), origin_circuit_t::rend_data, and tor_assert.

Referenced by connection_ap_handshake_attach_circuit().

4.39.3.16 int rend_parse_service_authorization (or_options_t * *options*, int *validate_only*)

Parse **config_line** as a client-side authorization for a hidden service and add it to the local map of hidden service authorizations. Return 0 for success and -1 for failure.

References rend_service_authorization_t::auth_type, base64_decode(), rend_service_authorization_t::descriptor_cookie, or_options_t::HidServAuth, LD_CONFIG, config_line_t::next, rend_service_authorization_t::onion_address, REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, REND_SERVICE_ADDRESS_LEN, rend_service_authorization_free(), rend_service_authorization_free_all(), rend_service_authorization_strmap_item_free(), REND_SERVICE_ID_LEN_BASE32, rend_valid_service_id(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_split_string(), strcmpend(), strmap_free(), strmap_get(), strmap_set(), tor_free, tor_snprintf(), and config_line_t::value.

Referenced by options_act().

4.39.3.17 static void rend_service_authorization_free (rend_service_authorization_t * *auth*) [static]

Helper: Free storage held by **rend_service_authorization_t** (p. 198).

References tor_free.

Referenced by rend_parse_service_authorization(), and rend_service_authorization_strmap_item_free().

4.39.3.18 void rend_service_authorization_free_all (void)

Release all the storage held in auth_hid_servs.

References rend_service_authorization_strmap_item_free(), and strmap_free().

Referenced by rend_parse_service_authorization(), and tor_free_all().

4.39.3.19 static void rend_service_authorization_strmap_item_free (void * *service_auth*) [static]

Helper for strmap_free.

References rend_service_authorization_free().

Referenced by rend_parse_service_authorization(), and rend_service_authorization_free_all().

4.39.4 Variable Documentation

4.39.4.1 strmap_t* auth_hid_servs = NULL [static]

Client-side authorizations for hidden services; map of onion address to rend_service_authorization_t*.

4.39.4.2 strmap_t* last_hid_serv_requests = NULL [static]

Contains the last request times to hidden service directories for certain queries; keys are strings consisting of base32-encoded hidden service directory identities and base32-encoded descriptor IDs; values are pointers to timestamps of the last requests.

4.40 rendcommon.c File Reference

Rendezvous implementation: shared code between introducers, services, clients, and rendezvous points.

`#include "or.h"`

Defines

- #define **REND_DESC_COOKIE_LEN** 16
- #define **REND_REPLICA_LEN** 1
- #define **REND_CACHE_MAX_AGE** (2∗24∗60∗60)
- #define **REND_CACHE_MAX_SKEW** (24∗60∗60)

Functions

- int **rend_cmp_service_ids** (const char ∗one, const char ∗two)
- void **rend_service_descriptor_free** (**rend_service_descriptor_t** ∗desc)
- void **rend_get_descriptor_id_bytes** (char ∗descriptor_id_out, const char ∗service_id, const char ∗secret_id_part)
- static void **get_secret_id_part_bytes** (char ∗secret_id_part, uint32_t time_period, const char ∗descriptor_cookie, uint8_t replica)
- static uint32_t **get_time_period** (time_t now, uint8_t deviation, const char ∗service_id)
- static uint32_t **get_seconds_valid** (time_t now, const char ∗service_id)
- int **rend_compute_v2_desc_id** (char ∗desc_id_out, const char ∗service_id, const char ∗descriptor_cookie, time_t now, uint8_t replica)
- static int **rend_encode_v2_intro_points** (char ∗∗encoded, **rend_service_descriptor_t** ∗desc)
- static int **rend_encrypt_v2_intro_points_basic** (char ∗∗encrypted_out, size_t ∗encrypted_len_out, const char ∗encoded, **smartlist_t** ∗client_cookies)
- static int **rend_encrypt_v2_intro_points_stealth** (char ∗∗encrypted_out, size_t ∗encrypted_len_out, const char ∗encoded, const char ∗descriptor_cookie)
- static int **rend_desc_v2_is_parsable** (**rend_encoded_v2_service_descriptor_t** ∗desc)
- void **rend_encoded_v2_service_descriptor_free** (**rend_encoded_v2_service_descriptor_t** ∗desc)
- void **rend_intro_point_free** (**rend_intro_point_t** ∗intro)
- int **rend_encode_v2_descriptors** (**smartlist_t** ∗descs_out, **rend_service_descriptor_t** ∗desc, time_t now, uint8_t period, **rend_auth_type_t** auth_type, **crypto_pk_env_t** ∗client_key, **smartlist_t** ∗client_cookies)
- **rend_service_descriptor_t** ∗ **rend_parse_service_descriptor** (const char ∗str, size_t len)
- int **rend_get_service_id** (**crypto_pk_env_t** ∗pk, char ∗out)
- void **rend_cache_init** (void)
- static void **rend_cache_entry_free** (**rend_cache_entry_t** ∗e)
- static void **_rend_cache_entry_free** (void ∗p)
- void **rend_cache_free_all** (void)
- void **rend_cache_clean** (void)
- void **rend_cache_clean_v2_descs_as_dir** (void)
- int **rend_id_is_in_interval** (const char ∗a, const char ∗b, const char ∗c)

- int **rend_valid_service_id** (const char *query)
- int **rend_cache_lookup_entry** (const char *query, int version, **rend_cache_entry_t** **e)
- int **rend_cache_lookup_desc** (const char *query, int version, const char **desc, size_t *desc_len)
- int **rend_cache_lookup_v2_desc_as_dir** (const char *desc_id, const char **desc)
- int **rend_cache_store** (const char *desc, size_t desc_len, int published)
- int **rend_cache_store_v2_desc_as_dir** (const char *desc)
- int **rend_cache_store_v2_desc_as_client** (const char *desc, const **rend_data_t** *rend_query)
- void **rend_process_relay_cell** (**circuit_t** *circ, const **crypt_path_t** *layer_hint, int command, size_t length, const char *payload)
- int **rend_cache_size** (void)
- **rend_data_t** * **rend_data_dup** (const **rend_data_t** *data)

Variables

- static strmap_t * **rend_cache** = NULL
- static digestmap_t * **rend_cache_v2_dir** = NULL

4.40.1 Detailed Description

Rendezvous implementation: shared code between introducers, services, clients, and rendezvous points.

4.40.2 Define Documentation

4.40.2.1 #define REND_CACHE_MAX_AGE (2*24*60*60)

How old do we let hidden service descriptors get before discarding them as too old?

Referenced by rend_cache_clean(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), and rend_cache_store_v2_desc_as_dir().

4.40.2.2 #define REND_CACHE_MAX_SKEW (24*60*60)

How wrong do we assume our clock may be when checking whether hidden services are too old or too new?

Referenced by rend_cache_clean(), rend_cache_clean_v2_descs_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), and rend_cache_store_v2_desc_as_dir().

4.40.2.3 #define REND_DESC_COOKIE_LEN 16

Length of the descriptor cookie that is used for versioned hidden service descriptors.

4.40.2.4 #define REND_REPLICA_LEN 1

Length of the replica number that is used to determine the secret ID part of versioned hidden service descriptors.

Referenced by get_secret_id_part_bytes().

4.40.3 Function Documentation

4.40.3.1 static uint32_t get_seconds_valid (time_t *now*, const char * *service_id*) `[static]`

Compute the time in seconds that a descriptor that is generated **now** for **service_id** will be valid.

References REND_TIME_PERIOD_V2_DESC_VALIDITY.

Referenced by rend_encode_v2_descriptors().

4.40.3.2 static void get_secret_id_part_bytes (char * *secret_id_part*, uint32_t *time_period*, const char * *descriptor_cookie*, uint8_t *replica*) `[static]`

Compute the secret ID part for time_period, a **descriptor_cookie** of length **REND_DESC_COOKIE_LEN** which may also be **NULL** if no descriptor_cookie shall be used, and **replica**, and write it to **secret_id_part** of length DIGEST_LEN.

References crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), DIGEST_LEN, REND_DESC_COOKIE_LEN, and REND_REPLICA_LEN.

Referenced by rend_compute_v2_desc_id(), and rend_encode_v2_descriptors().

4.40.3.3 static uint32_t get_time_period (time_t *now*, uint8_t *deviation*, const char * *service_id*) `[static]`

Return the time period for time **now** plus a potentially intended **deviation** of one or more periods, based on the first byte of **service_id**.

References REND_TIME_PERIOD_V2_DESC_VALIDITY.

Referenced by rend_compute_v2_desc_id(), and rend_encode_v2_descriptors().

4.40.3.4 void rend_cache_clean (void)

Removes all old entries from the service descriptor cache.

References rend_cache_entry_t::parsed, rend_cache, rend_cache_entry_free(), REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, strmap_iter_done(), strmap_iter_get(), strmap_iter_init(), strmap_iter_next(), strmap_iter_next_rmv(), and rend_service_descriptor_t::timestamp.

Referenced by run_scheduled_events().

4.40.3.5 void rend_cache_clean_v2_descs_as_dir (void)

Remove all old v2 descriptors and those for which this hidden service directory is not responsible for any more.

References base32_encode(), DIGEST_LEN, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_iter_next_rmv(), hid_serv_responsible_for_desc_id(), LD_REND, rend_cache_entry_t::parsed, rend_cache_entry_free(), REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_cache_v2_dir, REND_DESC_ID_V2_LEN_BASE32, safe_str_client(), and rend_service_descriptor_t::timestamp.

Referenced by run_scheduled_events().

4.40.3.6 static void rend_cache_entry_free (rend_cache_entry_t * *e*) `[static]`

Helper: free storage held by a single service descriptor cache entry.

References rend_cache_entry_t::desc, rend_cache_entry_t::parsed, rend_service_descriptor_free(), and tor_free.

Referenced by rend_cache_clean(), and rend_cache_clean_v2_descs_as_dir().

4.40.3.7 void rend_cache_free_all (void)

Free all storage held by the service descriptor cache.

References digestmap_free(), rend_cache, rend_cache_v2_dir, and strmap_free().

Referenced by tor_free_all().

4.40.3.8 void rend_cache_init (void)

Initializes the service descriptor cache.

References digestmap_new(), rend_cache, and rend_cache_v2_dir.

Referenced by tor_init().

4.40.3.9 int rend_cache_lookup_desc (const char * *query*, int *version*, const char ** *desc*, size_t * *desc_len*)

query is a base-32'ed service id. If it's malformed, return -1. Else look it up.

- If it is found, point *desc to it, and write its length into *desc_len, and return 1.
- If it is not found, return 0. Note: calls to rend_cache_clean or rend_cache_store may invalidate *desc.

References rend_cache_entry_t::desc, rend_cache_entry_t::len, and rend_cache_lookup_entry().

Referenced by directory_handle_command_get().

4.40.3.10 int rend_cache_lookup_entry (const char * *query*, int *version*, rend_cache_entry_t ** *e*)

If we have a cached **rend_cache_entry_t** (p. 193) for the service ID **query** with **version**, set *__e__ to that entry and return 1. Else return 0. If **version** is nonnegative, only return an entry in that descriptor format version. Otherwise (if **version** is negative), return the most recent format we have.

References rend_cache, REND_SERVICE_ID_LEN_BASE32, rend_valid_service_id(), strmap_get_lc(), tor_assert, and tor_snprintf().

Referenced by rend_cache_lookup_desc(), rend_client_desc_trynow(), rend_client_get_random_intro(), rend_client_refetch_v2_renddesc(), rend_client_remove_intro_point(), and rend_client_send_introduction().

4.40.3.11 int rend_cache_lookup_v2_desc_as_dir (const char * *desc_id*, const char ** *desc*)

Lookup the v2 service descriptor with base32-encoded **desc_id** and copy the pointer to it to *__desc__. Return 1 on success, 0 on well-formed-but-not-found, and -1 on failure.

References base32_decode(), rend_cache_entry_t::desc, DIGEST_LEN, digestmap_get(), hid_serv_responsible_for_desc_id(), LD_REND, rend_cache_v2_dir, REND_DESC_ID_V2_LEN_BASE32, safe_str(), and tor_assert.

Referenced by directory_handle_command_get().

4.40.3.12 int rend_cache_size (void)

Return the number of entries in our rendezvous descriptor cache.

References rend_cache, and strmap_size().

4.40.3.13 int rend_cache_store (const char * *desc*, size_t *desc_len*, int *published*)

Parse *desc, calculate its service id, and store it in the cache. If we have a newer v0 descriptor with the same ID, ignore this one. If we have an older descriptor with the same ID, replace it. If we are acting as client due to the published flag and have any v2 descriptor with the same ID, reject this one in order to not get confused with having both versions for the same service.

Return -2 if it's malformed or otherwise rejected; return -1 if we already have a v2 descriptor here; return 0 if it's the same or older than one we've already got; return 1 if it's novel.

The published flag tells us if we store the descriptor in our role as directory (1) or if we cache it as client (0).

References rend_cache_entry_t::desc, LD_BUG, LD_PROTOCOL, LD_REND, rend_cache_entry_t::len, rend_cache_entry_t::parsed, rend_service_descriptor_t::pk, rend_cache_entry_t::received, rend_cache, REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_get_service_id(), rend_parse_service_descriptor(), rend_service_descriptor_free(), REND_SERVICE_ID_LEN_BASE32, safe_str_client(), strmap_get_lc(), strmap_set_lc(), rend_service_descriptor_t::timestamp, tor_assert, tor_free, and tor_snprintf().

Referenced by connection_dir_client_reached_eof(), and directory_handle_command_post().

4.40.3.14 int rend_cache_store_v2_desc_as_client (const char ∗ *desc*, const rend_data_t ∗ *rend_query*)

Parse the v2 service descriptor in **desc**, decrypt the included list of introduction points with **descriptor_cookie** (which may also be **NULL** if decryption is not necessary), and store the descriptor to the local cache under its version and service id.

If we have a newer v2 descriptor with the same ID, ignore this one. If we have an older descriptor with the same ID, replace it. If we have any v0 descriptor with the same ID, reject this one in order to not get confused with having both versions for the same service. Return -2 if it's malformed or otherwise rejected; return -1 if we already have a v0 descriptor here; return 0 if it's the same or older than one we've already got; return 1 if it's novel.

References rend_data_t::auth_type, rend_cache_entry_t::desc, rend_data_t::descriptor_cookie, DIGEST_LEN, rend_service_descriptor_t::intro_nodes, LD_REND, rend_cache_entry_t::len, rend_cache_entry_t::parsed, rend_service_descriptor_t::pk, rend_cache_entry_t::received, rend_cache, REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_decrypt_introduction_points(), rend_get_service_id(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), REND_SERVICE_ID_LEN_BASE32, safe_str_client(), smartlist_create(), strmap_get_lc(), strmap_set_lc(), rend_service_descriptor_t::timestamp, tor_assert, tor_free, tor_mem_is_zero(), and tor_snprintf().

Referenced by connection_dir_client_reached_eof().

4.40.3.15 int rend_cache_store_v2_desc_as_dir (const char ∗ *desc*)

Parse the v2 service descriptor(s) in **desc** and store it/them to the local rend cache. Don't attempt to decrypt the included list of introduction points (as we don't have a descriptor cookie for it).

If we have a newer descriptor with the same ID, ignore this one. If we have an older descriptor with the same ID, replace it. Return -2 if we are not acting as hidden service directory; return -1 if the descriptor(s) were not parsable; return 0 if all descriptors are the same or older than those we've already got; return a positive number for the number of novel stored descriptors.

References base32_encode(), rend_cache_entry_t::desc, DIGEST_LEN, digestmap_get(), digestmap_set(), hid_serv_acting_as_directory(), hid_serv_responsible_for_desc_id(), LD_REND, rend_cache_entry_t::len, rend_cache_entry_t::parsed, rend_cache_entry_t::received, REND_CACHE_MAX_AGE, REND_CACHE_MAX_SKEW, rend_cache_v2_dir, REND_DESC_ID_V2_LEN_BASE32, rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), safe_str(), safe_str_client(), strcmpstart(), rend_service_descriptor_t::timestamp, tor_assert, and tor_free.

Referenced by directory_handle_command_post().

4.40.3.16 int rend_cmp_service_ids (const char ∗ *one*, const char ∗ *two*)

Return 0 if one and two are the same service ids, else -1 or 1

Referenced by circuit_get_by_rend_query_and_purpose(), circuit_is_acceptable(), connection_ap_handshake_attach_circuit(), connection_get_by_type_state_rendquery(), count_established_intro_points(), rend_client_desc_trynow(), and rend_client_send_introduction().

4.40.3.17 int rend_compute_v2_desc_id (char * *desc_id_out*, const char * *service_id*, const char * *descriptor_cookie*, time_t *now*, uint8_t *replica*)

Compute the binary **desc_id_out** (DIGEST_LEN bytes long) for a given base32-encoded **service_id** and optional unencoded **descriptor_cookie** of length REND_DESC_COOKIE_LEN, at time **now** for replica number **replica**. **desc_id** needs to have **DIGEST_LEN** bytes free. Return 0 for success, -1 otherwise.

References base32_decode(), DIGEST_LEN, get_secret_id_part_bytes(), get_time_period(), LD_REND, rend_get_descriptor_id_bytes(), REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS, REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, safe_str(), and safe_str_client().

Referenced by rend_client_refetch_v2_renddesc().

4.40.3.18 rend_data_t* rend_data_dup (const rend_data_t * *data*)

Allocate and return a new **rend_data_t** (p. 194) with the same contents as **query**.

References tor_assert.

Referenced by circuit_get_open_circ_or_launch(), connection_exit_begin_conn(), directory_initiate_command_rend(), and rend_service_relaunch_rendezvous().

4.40.3.19 static int rend_desc_v2_is_parsable (rend_encoded_v2_service_descriptor_t * *desc*) [static]

Attempt to parse the given **desc_str** and return true if this succeeds, false otherwise.

References rend_encoded_v2_service_descriptor_t::desc_str, DIGEST_LEN, rend_parse_v2_service_descriptor(), rend_service_descriptor_free(), and tor_free.

Referenced by rend_encode_v2_descriptors().

4.40.3.20 int rend_encode_v2_descriptors (smartlist_t * *descs_out*, rend_service_descriptor_t * *desc*, time_t *now*, uint8_t *period*, rend_auth_type_t *auth_type*, crypto_pk_env_t * *client_key*, smartlist_t * *client_cookies*)

Encode a set of rend_encoded_v2_service_descriptor_t's for **desc** at time **now** using **service_key**, depending on **auth_type** a **descriptor_cookie** and a list of **client_cookies** (which are both **NULL** if no client authorization is performed), and **period** (e.g. 0 for the current period, 1 for the next period, etc.) and add them to the existing list **descs_out**; return the number of seconds that the descriptors will be found by clients, or -1 if the encoding was not successful.

References base32_encode(), base64_encode(), crypto_digest(), crypto_pk_get_digest(), crypto_pk_write_public_key_to_string(), rend_encoded_v2_service_descriptor_t::desc_id, rend_encoded_v2_service_descriptor_t::desc_str, DIGEST_LEN, format_iso_time(), get_seconds_valid(), get_secret_id_part_bytes(), get_time_period(), rend_service_descriptor_t::intro_nodes, LD_BUG, LD_REND, rend_service_descriptor_t::pk, rend_service_descriptor_t::protocols, REND_DESC_ID_V2_LEN_BASE32, rend_desc_v2_is_parsable(), rend_encode_v2_intro_points(), rend_encoded_v2_service_descriptor_free(), rend_encrypt_v2_intro_points_basic(), rend_encrypt_v2_intro_points_stealth(), rend_get_descriptor_id_bytes(), REND_NUMBER_OF_NON_CONSECUTIVE_REPLICAS,

REND_SECRET_ID_PART_LEN_BASE32, REND_TIME_PERIOD_V2_DESC_VALIDITY, router_append_dirobj_signature(), smartlist_add(), smartlist_clear(), rend_service_descriptor_t::timestamp, tor_assert, tor_free, and tor_snprintf().

Referenced by upload_service_descriptor().

4.40.3.21 static int rend_encode_v2_intro_points (char ** *encoded*, rend_service_descriptor_t * *desc*) [static]

Encode the introduction points in **desc** and write the result to a newly allocated string pointed to by **encoded**. Return 0 for success, -1 otherwise.

References extend_info_t::addr, base32_encode(), crypto_pk_write_public_key_to_string(), DIGEST_LEN, rend_intro_point_t::extend_info, extend_info_t::identity_digest, rend_intro_point_t::intro_key, rend_service_descriptor_t::intro_nodes, LD_REND, extend_info_t::onion_key, extend_info_t::port, REND_INTRO_POINT_ID_LEN_BASE32, tor_dup_addr(), tor_free, and tor_snprintf().

Referenced by rend_encode_v2_descriptors().

4.40.3.22 void rend_encoded_v2_service_descriptor_free (rend_encoded_v2_service_descriptor_t * *desc*)

Free the storage held by an encoded v2 service descriptor.

References rend_encoded_v2_service_descriptor_t::desc_str, and tor_free.

Referenced by rend_encode_v2_descriptors(), and upload_service_descriptor().

4.40.3.23 static int rend_encrypt_v2_intro_points_basic (char ** *encrypted_out*, size_t * *encrypted_len_out*, const char * *encoded*, smartlist_t * *client_cookies*) [static]

Encrypt the encoded introduction points in **encoded** using authorization type 'basic' with **client_cookies** and write the result to a newly allocated string pointed to by **encrypted_out** of length **encrypted_len_out**. Return 0 for success, -1 otherwise.

References CIPHER_IV_LEN, CIPHER_KEY_LEN, crypto_cipher_encrypt(), crypto_cipher_encrypt_with_iv(), crypto_create_init_cipher(), crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_cipher_env(), crypto_free_digest_env(), crypto_new_digest_env(), crypto_rand(), LD_REND, REND_BASIC_AUTH_CLIENT_ENTRY_LEN, REND_BASIC_AUTH_CLIENT_ID_LEN, REND_BASIC_AUTH_CLIENT_MULTIPLE, REND_DESC_COOKIE_LEN, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_sort_digests(), tor_assert, and tor_free.

Referenced by rend_encode_v2_descriptors().

4.40.3.24 static int rend_encrypt_v2_intro_points_stealth (char ** *encrypted_out*, size_t * *encrypted_len_out*, const char * *encoded*, const char * *descriptor_cookie*) [static]

Encrypt the encoded introduction points in **encoded** using authorization type 'stealth' with **descriptor_cookie** of length REND_DESC_COOKIE_LEN and write the result to a newly allocated string pointed to by **encrypted_out** of length **encrypted_len_out**. Return 0 for success, -1 otherwise.

References CIPHER_IV_LEN, crypto_cipher_encrypt_with_iv(), crypto_create_init_cipher(), crypto_free_cipher_env(), LD_REND, tor_assert, and tor_free.

Referenced by rend_encode_v2_descriptors().

4.40.3.25 void rend_get_descriptor_id_bytes (char ∗ *descriptor_id_out*, const char ∗ *service_id*, const char ∗ *secret_id_part*)

Compute the descriptor ID for **service_id** of length **REND_SERVICE_ID_LEN** and **secret_id_part** of length **DIGEST_LEN**, and write it to **descriptor_id_out** of length **DIGEST_LEN**.

References crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), DIGEST_LEN, and REND_SERVICE_ID_LEN.

Referenced by rend_compute_v2_desc_id(), rend_encode_v2_descriptors(), and rend_parse_v2_service_descriptor().

4.40.3.26 int rend_get_service_id (crypto_pk_env_t ∗ *pk*, char ∗ *out*)

Sets **out** to the first 10 bytes of the digest of **pk**, base32 encoded. NUL-terminates out. (We use this string to identify services in directory requests and .onion URLs.)

References base32_encode(), crypto_pk_get_digest(), DIGEST_LEN, REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, and tor_assert.

Referenced by rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_service_load_keys(), and upload_service_descriptor().

4.40.3.27 int rend_id_is_in_interval (const char ∗ *a*, const char ∗ *b*, const char ∗ *c*)

Determines whether **a** is in the interval of **b** (excluded) and **c** (included) in a circular digest ring; returns 1 if this is the case, and 0 otherwise.

References DIGEST_LEN, and tor_assert.

Referenced by hid_serv_responsible_for_desc_id().

4.40.3.28 void rend_intro_point_free (rend_intro_point_t ∗ *intro*)

Free the storage held by an introduction point info.

References crypto_free_pk_env(), rend_intro_point_t::extend_info, extend_info_free(), rend_intro_point_t::intro_key, and tor_free.

Referenced by rend_client_get_random_intro(), rend_client_remove_intro_point(), rend_parse_introduction_points(), rend_service_descriptor_free(), rend_service_free(), and rend_services_introduce().

4.40.3.29 rend_service_descriptor_t∗ rend_parse_service_descriptor (const char ∗ *str*, size_t *len*)

Parse a service descriptor at **str** (**len** bytes). On success, return a newly alloced service_descriptor_t. On failure, return NULL.

References crypto_pk_asn1_decode(), crypto_pk_keysize(), crypto_pk_public_checksig_digest(), rend_intro_point_t::extend_info, get_uint16(), get_uint32(), rend_service_descriptor_t::intro_nodes, LD_PROTOCOL, extend_info_t::nickname, note_crypto_pk_op(), rend_service_descriptor_t::pk, rend_service_descriptor_t::protocols, rend_service_descriptor_free(), smartlist_add(), smartlist_create(), rend_service_descriptor_t::timestamp, tor_assert, and rend_service_descriptor_t::version.

Referenced by rend_cache_store().

4.40.3.30 void rend_process_relay_cell (circuit_t ∗ *circ*, const crypt_path_t ∗ *layer_hint*, int *command*, size_t *length*, const char ∗ *payload*)

Called when we get a rendezvous-related relay cell on circuit **circ**. Dispatch on rendezvous relay command.

References CIRCUIT_IS_ORIGIN, origin_circuit_t::cpath, LD_APP, LD_PROTOCOL, crypt_path_t::prev, rend_client_introduction_acked(), rend_client_receive_rendezvous(), rend_client_rendezvous_acked(), rend_mid_establish_intro(), rend_mid_establish_rendezvous(), rend_mid_introduce(), rend_mid_rendezvous(), rend_service_intro_established(), rend_service_introduce(), TO_OR_CIRCUIT(), TO_ORIGIN_CIRCUIT(), and tor_fragile_assert.

Referenced by connection_edge_process_relay_cell().

4.40.3.31 void rend_service_descriptor_free (rend_service_descriptor_t ∗ *desc*)

Free the storage held by the service descriptor **desc**.

References crypto_free_pk_env(), rend_service_descriptor_t::intro_nodes, rend_service_descriptor_t::pk, rend_intro_point_free(), smartlist_free(), rend_service_descriptor_t::successful_uploads, and tor_free.

Referenced by rend_cache_entry_free(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_desc_v2_is_parsable(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_free(), and rend_service_update_descriptor().

4.40.3.32 int rend_valid_service_id (const char ∗ *query*)

Return true iff **query** is a syntactically valid service ID (as generated by rend_get_service_id).

References BASE32_CHARS, and REND_SERVICE_ID_LEN_BASE32.

Referenced by parse_extended_hostname(), rend_cache_lookup_entry(), and rend_parse_service_authorization().

4.40.4 Variable Documentation

4.40.4.1 strmap_t∗ rend_cache = NULL [static]

Map from service id (as generated by rend_get_service_id) to **rend_cache_entry_t** (p. 193).

Referenced by rend_cache_clean(), rend_cache_free_all(), rend_cache_init(), rend_cache_lookup_entry(), rend_cache_size(), rend_cache_store(), and rend_cache_store_v2_desc_as_client().

4.40.4.2 digestmap_t∗ rend_cache_v2_dir = NULL `[static]`

Map from descriptor id to **rend_cache_entry_t** (p. 193); only for hidden service directories.

Referenced by rend_cache_clean_v2_descs_as_dir(), rend_cache_free_all(), rend_cache_init(), rend_cache_lookup_v2_desc_as_dir(), and rend_cache_store_v2_desc_as_dir().

4.41 rendmid.c File Reference

Implement introductions points and rendezvous points.

`#include "or.h"`

Functions

- int **rend_mid_establish_intro** (**or_circuit_t** ∗circ, const char ∗request, size_t request_len)
- int **rend_mid_introduce** (**or_circuit_t** ∗circ, const char ∗request, size_t request_len)
- int **rend_mid_establish_rendezvous** (**or_circuit_t** ∗circ, const char ∗request, size_t request_len)
- int **rend_mid_rendezvous** (**or_circuit_t** ∗circ, const char ∗request, size_t request_len)

4.41.1 Detailed Description

Implement introductions points and rendezvous points.

4.41.2 Function Documentation

4.41.2.1 int rend_mid_establish_intro (or_circuit_t ∗ *circ*, const char ∗ *request*, size_t *request_len*)

Respond to an ESTABLISH_INTRO cell by checking the signed data and setting the circuit's purpose and service pk digest.

References or_circuit_t::_base, base32_encode(), circuit_get_intro_point(), CIRCUIT_PURPOSE_INTRO_POINT, CIRCUIT_PURPOSE_OR, crypto_digest(), crypto_free_pk_env(), crypto_pk_asn1_decode(), crypto_pk_get_digest(), crypto_pk_public_checksig_digest(), DIGEST_LEN, get_uint16(), or_circuit_t::handshake_digest, LD_BUG, LD_GENERAL, LD_PROTOCOL, LD_REND, circuit_t::n_conn, note_crypto_pk_op(), or_circuit_t::p_circ_id, circuit_t::purpose, relay_send_command_from_edge(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, or_circuit_t::rend_token, safe_str(), and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.41.2.2 int rend_mid_establish_rendezvous (or_circuit_t ∗ *circ*, const char ∗ *request*, size_t *request_len*)

Process an ESTABLISH_RENDEZVOUS cell by setting the circuit's purpose and rendezvous cookie.

References or_circuit_t::_base, base16_encode(), circuit_get_rendezvous(), CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_POINT_WAITING, LD_PROTOCOL, LD_REND, circuit_t::n_conn, or_circuit_t::p_circ_id, circuit_t::purpose, relay_send_command_from_edge(), REND_COOKIE_LEN, or_circuit_t::rend_token, and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.41.2.3 int rend_mid_introduce (or_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Process an INTRODUCE1 cell by finding the corresponding introduction circuit, and relaying the body of the INTRODUCE1 cell inside an INTRODUCE2 cell.

References or_circuit_t::_base, base32_encode(), CIPHER_KEY_LEN, circuit_get_intro_point(), CIRCUIT_PURPOSE_OR, DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, LD_REND, MAX_NICKNAME_LEN, circuit_t::n_conn, or_circuit_t::p_circ_id, PKCS1_OAEP_PADDING_OVERHEAD, circuit_t::purpose, relay_send_command_from_edge(), REND_COOKIE_LEN, REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, safe_str(), and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.41.2.4 int rend_mid_rendezvous (or_circuit_t * *circ*, const char * *request*, size_t *request_len*)

Process a RENDEZVOUS1 cell by looking up the correct rendezvous circuit by its relaying the cell's body in a RENDEZVOUS2 cell, and connecting the two circuits.

References or_circuit_t::_base, base16_encode(), circuit_get_rendezvous(), CIRCUIT_PURPOSE_OR, CIRCUIT_PURPOSE_REND_ESTABLISHED, DIGEST_LEN, LD_GENERAL, LD_PROTOCOL, LD_REND, circuit_t::n_conn, or_circuit_t::p_circ_id, circuit_t::purpose, relay_send_command_from_edge(), REND_COOKIE_LEN, or_circuit_t::rend_splice, or_circuit_t::rend_token, and TO_CIRCUIT.

Referenced by rend_process_relay_cell().

4.42 rendservice.c File Reference

The hidden-service side of rendezvous functionality.

`#include "or.h"`

Data Structures

- struct **rend_service_port_config_t**
- struct **rend_service_t**

Defines

- #define **NUM_INTRO_POINTS** 3
- #define **INTRO_CIRC_RETRY_PERIOD** (60*5)
- #define **MAX_INTRO_CIRCS_PER_PERIOD** 10
- #define **MAX_REND_FAILURES** 30
- #define **MAX_REND_TIMEOUT** 30
- #define **NUM_INTRO_POINTS_INIT** (NUM_INTRO_POINTS + 2)

Functions

- static **origin_circuit_t** * **find_intro_circuit** (**rend_intro_point_t** *intro, const char *pk_digest)
- int **num_rend_services** (void)
- static void **rend_authorized_client_free** (**rend_authorized_client_t** *client)
- static void **rend_authorized_client_strmap_item_free** (void *authorized_client)
- static void **rend_service_free** (**rend_service_t** *service)
- void **rend_service_free_all** (void)
- static void **rend_add_service** (**rend_service_t** *service)
- static **rend_service_port_config_t** * **parse_port_config** (const char *string)
- int **rend_config_services** (**or_options_t** *options, int validate_only)
- static void **rend_service_update_descriptor** (**rend_service_t** *service)
- int **rend_service_load_keys** (void)
- static **rend_service_t** * **rend_service_get_by_pk_digest** (const char *digest)
- static int **rend_service_requires_uptime** (**rend_service_t** *service)
- static int **rend_check_authorization** (**rend_service_t** *service, const char *descriptor_cookie)
- static void **clean_accepted_intros** (**rend_service_t** *service, time_t now)
- int **rend_service_introduce** (**origin_circuit_t** *circuit, const char *request, size_t request_len)
- void **rend_service_relaunch_rendezvous** (**origin_circuit_t** *oldcirc)
- static int **rend_service_launch_establish_intro** (**rend_service_t** *service, **rend_intro_point_t** *intro)
- static int **count_established_intro_points** (const char *query)
- void **rend_service_intro_has_opened** (**origin_circuit_t** *circuit)
- int **rend_service_intro_established** (**origin_circuit_t** *circuit, const char *request, size_t request_len)
- void **rend_service_rendezvous_has_opened** (**origin_circuit_t** *circuit)

- static void **directory_post_to_hs_dir** (**rend_service_descriptor_t** ∗renddesc, **smartlist_t** ∗descs, const char ∗service_id, int seconds_valid)
- static void **upload_service_descriptor** (**rend_service_t** ∗service)
- void **rend_services_introduce** (void)
- void **rend_consider_services_upload** (time_t now)
- void **rend_hsdir_routers_changed** (void)
- void **rend_consider_descriptor_republication** (void)
- void **rend_service_dump_stats** (int severity)
- int **rend_service_set_connection_addr_port** (**edge_connection_t** ∗conn, **origin_circuit_t** ∗circ)

Variables

- static **smartlist_t** ∗ **rend_service_list** = NULL
- static int **consider_republishing_rend_descriptors** = 1

4.42.1 Detailed Description

The hidden-service side of rendezvous functionality.

4.42.2 Define Documentation

4.42.2.1 #define INTRO_CIRC_RETRY_PERIOD (60∗5)

If we can't build our intro circuits, don't retry for this long.

Referenced by rend_services_introduce().

4.42.2.2 #define MAX_INTRO_CIRCS_PER_PERIOD 10

Don't try to build more than this many circuits before giving up for a while.

Referenced by rend_services_introduce().

4.42.2.3 #define MAX_REND_FAILURES 30

How many times will a hidden service operator attempt to connect to a requested rendezvous point before giving up?

Referenced by rend_service_introduce(), and rend_service_relaunch_rendezvous().

4.42.2.4 #define MAX_REND_TIMEOUT 30

How many seconds should we spend trying to connect to a requested rendezvous point before giving up?

Referenced by rend_service_introduce().

4.42.2.5 #define NUM_INTRO_POINTS 3

Try to maintain this many intro points per service if possible.

Referenced by rend_service_intro_has_opened().

4.42.3 Function Documentation

4.42.3.1 static void clean_accepted_intros (rend_service_t * *service*, time_t *now*) `[static]`

Remove elements from **service**'s replay cache that are old enough to be noticed by timestamp checking.

References rend_service_t::accepted_intros, rend_service_t::last_cleaned_accepted_intros, REND_REPLAY_TIME_INTERVAL, and tor_free.

Referenced by rend_service_introduce().

4.42.3.2 static int count_established_intro_points (const char * *query*) `[static]`

Return the number of introduction points that are or have been established for the given service address in **query**.

References _circuit_get_global_list(), CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, CIRCUIT_STATE_OPEN, circuit_t::marked_for_close, circuit_t::next, rend_data_t::onion_address, circuit_t::purpose, rend_cmp_service_ids(), origin_circuit_t::rend_data, circuit_t::state, and TO_ORIGIN_CIRCUIT().

Referenced by rend_service_intro_has_opened().

4.42.3.3 static void directory_post_to_hs_dir (rend_service_descriptor_t * *renddesc*, smartlist_t * *descs*, const char * *service_id*, int *seconds_valid*) `[static]`

Determine the responsible hidden service directories for the rend_encoded_v2_service_descriptor_t's in **descs** and upload them; **service_id** and **seconds_valid** are only passed for logging purposes.

References routerstatus_t::addr, rend_service_descriptor_t::all_uploads_performed, base32_encode(), rend_encoded_v2_service_descriptor_t::desc_id, rend_encoded_v2_service_descriptor_t::desc_str, DIGEST_LEN, DIR_PURPOSE_UPLOAD_RENDDESC_V2, directory_initiate_command_routerstatus(), hid_serv_get_responsible_directories(), routerstatus_t::identity_digest, LD_REND, routerstatus_t::nickname, routerstatus_t::or_port, REND_DESC_ID_V2_LEN_BASE32, router_get_by_digest(), ROUTER_PURPOSE_GENERAL, safe_str_client(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_digest_isin(), smartlist_free(), rend_service_descriptor_t::successful_uploads, tor_dup_ip(), and tor_free.

Referenced by upload_service_descriptor().

4.42.3.4 static origin_circuit_t * find_intro_circuit (rend_intro_point_t * *intro*, const char * *pk_ digest*) [static]

Return the (possibly non-open) introduction circuit ending at **intro** for the service whose public key is **pk_digest**. (**desc_version** is ignored). Return NULL if no such service is found.

References origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, circuit_get_next_by_pk_and_purpose(), CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, DIGEST_LEN, rend_intro_point_t::extend_info, extend_info_t::identity_digest, origin_circuit_t::rend_data, and tor_assert.

Referenced by rend_service_dump_stats(), rend_service_update_descriptor(), and rend_services_introduce().

4.42.3.5 int num_rend_services (void)

Return the number of rendezvous services we have configured.

Referenced by circuit_predict_and_launch_new().

4.42.3.6 static rend_service_port_config_t* parse_port_config (const char * *string*) [static]

Parses a real-port to virtual-port mapping and returns a new **rend_service_port_config_t** (p. 201).

The format is: VirtualPort (IP|RealPort|IP:RealPort)?

IP defaults to 127.0.0.1; RealPort defaults to VirtualPort.

References escaped(), LD_CONFIG, rend_service_port_config_t::real_addr, rend_service_port_config_t::real_port, smartlist_create(), smartlist_free(), smartlist_split_string(), tor_addr_copy(), tor_addr_from_ipv4h, tor_addr_port_parse(), tor_free, tor_parse_long(), and rend_service_port_config_t::virtual_port.

Referenced by rend_config_services().

4.42.3.7 static void rend_add_service (rend_service_t * *service*) [static]

Validate **service** and add it to rend_service_list if possible.

References rend_service_t::auth_type, rend_service_t::clients, rend_service_t::directory, fmt_addr(), rend_service_t::intro_nodes, LD_CONFIG, LD_REND, rend_service_t::ports, rend_service_port_config_t::real_addr, rend_service_port_config_t::real_port, rend_service_free(), smartlist_add(), smartlist_create(), and rend_service_port_config_t::virtual_port.

Referenced by rend_config_services().

4.42.3.8 static void rend_authorized_client_free (rend_authorized_client_t * *client*) [static]

Helper: free storage held by a single service authorized client entry.

References rend_authorized_client_t::client_key, rend_authorized_client_t::client_name, crypto_free_pk_env(), and tor_free.

Referenced by rend_authorized_client_strmap_item_free(), and rend_service_free().

4.42.3.9 static void rend_authorized_client_strmap_item_free (void ∗ *authorized_client*) [static]

Helper for strmap_free.

References rend_authorized_client_free().

Referenced by rend_service_load_keys().

4.42.3.10 static int rend_check_authorization (rend_service_t ∗ *service*, const char ∗ *descriptor_cookie*) [static]

Check client authorization of a given **descriptor_cookie** for **service**. Return 1 for success and 0 for failure.

References rend_service_t::clients, LD_BUG, REND_DESC_COOKIE_LEN, and tor_assert.

Referenced by rend_service_introduce().

4.42.3.11 int rend_config_services (or_options_t ∗ *options*, int *validate_only*)

Set up rend_service_list, based on the values of HiddenServiceDir and HiddenServicePort in **options**. Return 0 on success and -1 on failure. (If **validate_only** is set, parse, warn and return as normal, but don't actually change the configured services.)

References _circuit_get_global_list(), rend_service_t::auth_type, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, CIRCUIT_STATE_OPEN, rend_authorized_client_t::client_name, rend_service_t::clients, DIGEST_LEN, rend_service_t::directory, rend_service_t::intro_period_started, config_line_t::key, LD_BUG, LD_CONFIG, LD_REND, circuit_t::next, config_line_t::next, parse_port_config(), rend_service_t::ports, rend_add_service(), REND_CLIENTNAME_MAX_LEN, origin_circuit_t::rend_data, REND_LEGAL_CLIENTNAME_CHARACTERS, rend_data_t::rend_pk_digest, rend_service_free(), or_options_t::RendConfigLines, safe_str_client(), smartlist_add(), smartlist_add_all(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_sort_strings(), smartlist_split_string(), smartlist_uniq_strings(), TO_ORIGIN_CIRCUIT(), tor_assert, tor_free, and config_line_t::value.

Referenced by options_act().

4.42.3.12 void rend_consider_descriptor_republication (void)

Consider republication of v2 rendezvous service descriptors that failed previously, but without regenerating descriptor contents.

References rend_service_descriptor_t::all_uploads_performed, rend_service_t::desc, get_options(), and upload_service_descriptor().

Referenced by run_scheduled_events().

4.42.3.13 void rend_consider_services_upload (time_t *now*)

Regenerate and upload rendezvous service descriptors for all services, if necessary. If the descriptor has been dirty enough for long enough, definitely upload; else only upload when the periodic timeout has expired.

For the first upload, pick a random time between now and two periods from now, and pick it independently for each service.

References crypto_rand_int(), rend_service_t::desc_is_dirty, get_options(), rend_service_t::next_upload_time, rend_service_update_descriptor(), or_options_t::RendPostPeriod, and upload_service_descriptor().

Referenced by run_scheduled_events().

4.42.3.14 void rend_hsdir_routers_changed (void)

Called when our internal view of the directory has changed, so that we might have router descriptors of hidden service directories available that we did not have before.

Referenced by router_dir_info_changed().

4.42.3.15 void rend_service_dump_stats (int *severity*)

Log the status of introduction points for all rendezvous services at log severity **severity**.

References origin_circuit_t::_base, circuit_state_to_string(), rend_service_t::directory, rend_intro_point_t::extend_info, find_intro_circuit(), rend_service_t::intro_nodes, LD_GENERAL, extend_info_t::nickname, rend_service_t::pk_digest, safe_str_client(), and circuit_t::state.

Referenced by dumpstats().

4.42.3.16 static void rend_service_free (rend_service_t * *service*) `[static]`

Release the storage held by **service**.

References _tor_free(), rend_service_t::accepted_intros, rend_service_t::clients, crypto_free_pk_env(), rend_service_t::desc, digestmap_free(), rend_service_t::directory, rend_service_t::intro_nodes, rend_service_t::ports, rend_service_t::private_key, rend_authorized_client_free(), rend_intro_point_free(), rend_service_descriptor_free(), smartlist_free(), and tor_free.

Referenced by rend_add_service(), rend_config_services(), and rend_service_free_all().

4.42.3.17 void rend_service_free_all (void)

Release all the storage held in rend_service_list.

References rend_service_free(), and smartlist_free().

Referenced by tor_free_all().

4.42.3.18 static rend_service_t* rend_service_get_by_pk_digest (const char * *digest*) `[static]`

Return the service whose public key has a digest of **digest**, or NULL if no such service exists.

References DIGEST_LEN.

Referenced by rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_rendezvous_has_opened(), and rend_service_set_connection_addr_port().

4.42.3.19 int rend_service_intro_established (origin_circuit_t ∗ *circuit*, const char ∗ *request*, size_t *request_len*)

Called when we get an INTRO_ESTABLISHED cell; mark the circuit as a live introduction point, and note that the service descriptor is now out-of-date.

References origin_circuit_t::_base, base32_encode(), CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_PURPOSE_S_INTRO, rend_service_t::desc_is_dirty, LD_PROTOCOL, LD_REND, circuit_t::n_circ_id, circuit_t::purpose, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, TO_CIRCUIT, and tor_assert.

Referenced by rend_process_relay_cell().

4.42.3.20 void rend_service_intro_has_opened (origin_circuit_t ∗ *circuit*)

Called when we're done building a circuit to an introduction point: sends a RELAY_ESTABLISH_INTRO cell.

References origin_circuit_t::_base, base32_encode(), circuit_has_opened(), CIRCUIT_PURPOSE_C_GENERAL, CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, count_established_intro_points(), origin_circuit_t::cpath, crypto_digest(), crypto_pk_asn1_encode(), crypto_pk_private_sign_digest(), DIGEST_LEN, crypt_path_t::handshake_digest, origin_circuit_t::intro_key, LD_BUG, LD_CIRC, LD_GENERAL, LD_REND, circuit_t::n_circ_id, note_crypto_pk_op(), NUM_INTRO_POINTS, crypt_path_t::prev, circuit_t::purpose, RELAY_PAYLOAD_SIZE, relay_send_command_from_edge(), origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, set_uint16(), TO_CIRCUIT, and tor_assert.

Referenced by circuit_has_opened(), and rend_service_launch_establish_intro().

4.42.3.21 int rend_service_introduce (origin_circuit_t ∗ *circuit*, const char ∗ *request*, size_t *request_len*)

Respond to an INTRODUCE2 cell by launching a circuit to the chosen rendezvous point.

References origin_circuit_t::_base, rend_service_t::accepted_intros, extend_info_t::addr, base16_encode(), base32_encode(), origin_circuit_t::build_state, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, CIRCLAUNCH_NEED_UPTIME, circuit_init_cpath_crypto(), circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_INTRO, clean_accepted_intros(), rend_service_t::clients, crypto_dh_compute_secret(), crypto_dh_free(), crypto_dh_generate_public(), crypto_dh_new(), crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_digest_env(), crypto_new_digest_env(), crypto_pk_asn1_decode(), crypto_pk_get_digest(), crypto_pk_keysize(), crypto_pk_private_hybrid_decrypt(), crypt_path_t::dh_handshake_state, DIGEST_LEN, digestmap_get(), digestmap_new(), digestmap_set(), escaped(), escaped_safe_str_client(), cpath_build_state_t::expiry_time, extend_info_free(), extend_info_from_router(), get_uint16(), get_uint32(), crypt_path_t::handshake_digest, extend_info_t::identity_digest, origin_circuit_t::intro_key, is_legal_nickname(), is_legal_nickname_or_hexdigest(), rend_service_t::last_cleaned_accepted_intros, LD_BUG, LD_PROTOCOL, LD_REND, crypt_path_t::magic, MAX_HEX_NICKNAME_LEN, MAX_NICKNAME_LEN, MAX_REND_FAILURES, MAX_REND_TIMEOUT, circuit_t::n_circ_id, extend_info_t::nickname, note_crypto_pk_op(), rend_data_t::onion_address, extend_info_t::onion_key, cpath_build_state_t::pending_final_cpath, PK_PKCS1_OAEP_PADDING, extend_info_t::port, circuit_t::purpose, RELAY_PAYLOAD_SIZE, rend_check_authorization(), rend_data_t::rend_cookie, REND_COOKIE_LEN, origin_circuit_t::rend_data, REND_DESC_COOKIE_LEN, rend_data_t::rend_pk_digest, REND_REPLAY_TIME_INTERVAL, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, rend_service_requires_uptime(), rep_hist_note_used_internal(), router_get_by_nickname(), rend_service_t::service_id, TO_CIRCUIT, tor_addr_from_ipv4n(), and tor_assert.

Referenced by rend_process_relay_cell().

4.42.3.22 static int rend_service_launch_establish_intro (rend_service_t * *service*, rend_intro_point_t * *intro*) [static]

Launch a circuit to serve as an introduction point for the service **service** at the introduction point **nickname**

References origin_circuit_t::_base, base16_encode(), origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_UPTIME, circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_S_ESTABLISH_INTRO, CIRCUIT_STATE_OPEN, crypto_pk_dup_key(), DIGEST_LEN, escaped_safe_str_client(), rend_intro_point_t::extend_info, extend_info_dup(), extend_info_free(), HEX_DIGEST_LEN, extend_info_t::identity_digest, rend_intro_point_t::intro_key, origin_circuit_t::intro_key, LD_REND, rend_service_t::n_intro_circuits_launched, extend_info_t::nickname, rend_data_t::onion_address, rend_service_t::pk_digest, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_intro_has_opened(), rep_hist_note_used_internal(), rend_service_t::service_id, and circuit_t::state.

4.42.3.23 int rend_service_load_keys (void)

Load and/or generate private keys for all hidden services, possibly including keys for client authorization. Return 0 on success, -1 on failure.

References abort_writing_to_file(), base64_encode(), check_private_dir(), rend_authorized_client_t::client_key, crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_check_key(), crypto_pk_dup_key(), crypto_pk_get_digest(), crypto_pk_write_private_key_to_string(), crypto_rand(), rend_authorized_client_t::descriptor_cookie, escaped(), finish_writing_to_file(), init_key_from_file(), LD_BUG, LD_CONFIG, LD_FS, LD_REND, read_file_to_str(), rend_authorized_client_strmap_item_free(), REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, rend_get_service_id(), rend_parse_client_keys(), RFTS_IGNORE_MISSING, start_writing_to_stdio_file(), strmap_free(), strmap_get(), strmap_size(), tor_free, tor_snprintf(), and write_str_to_file().

Referenced by options_act().

4.42.3.24 void rend_service_relaunch_rendezvous (origin_circuit_t * *oldcirc*)

Called when we fail building a rendezvous circuit at some point other than the last hop: launches a new circuit to the same rendezvous point.

References origin_circuit_t::_base, origin_circuit_t::build_state, cpath_build_state_t::chosen_exit, CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, circuit_launch_by_extend_info(), CIRCUIT_PURPOSE_S_CONNECT_REND, cpath_build_state_t::expiry_time, cpath_build_state_t::failure_count, LD_REND, MAX_REND_FAILURES, extend_info_t::nickname, cpath_build_state_t::pending_final_cpath, circuit_t::purpose, origin_circuit_t::rend_data, rend_data_dup(), and tor_assert.

Referenced by circuit_build_failed().

4.42.3.25 void rend_service_rendezvous_has_opened (origin_circuit_t * *circuit*)

Called once a circuit to a rendezvous point is established: sends a RELAY_COMMAND_RENDEZVOUS1 cell.

References origin_circuit_t::_base, base16_encode(), base32_encode(), origin_circuit_t::build_state, circuit_initial_package_window(), CIRCUIT_PURPOSE_S_CONNECT_REND, CIRCUIT_PURPOSE_S_REND_JOINED, CIRCWINDOW_START, origin_circuit_t::cpath, crypto_dh_free(), crypto_dh_get_public(), crypt_path_t::deliver_window, crypt_path_t::dh_handshake_state, DIGEST_LEN, crypt_path_t::handshake_digest, LD_GENERAL, LD_REND, circuit_t::n_circ_id, onion_append_to_cpath(), crypt_path_t::package_window, cpath_build_state_t::pending_final_cpath, crypt_path_t::prev, circuit_t::purpose, RELAY_PAYLOAD_SIZE, relay_send_command_from_edge(), rend_data_t::rend_cookie, REND_COOKIE_LEN, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, crypt_path_t::state, TO_CIRCUIT, and tor_assert.

Referenced by circuit_has_opened().

4.42.3.26 static int rend_service_requires_uptime (rend_service_t * *service*) `[static]`

Return 1 if any virtual port in **service** wants a circuit to have good uptime. Else return 0.

References get_options(), rend_service_t::ports, smartlist_string_num_isin(), and rend_service_port_config_t::virtual_port.

Referenced by rend_service_introduce().

4.42.3.27 int rend_service_set_connection_addr_port (edge_connection_t * *conn*, origin_circuit_t * *circ*)

Given **conn**, a rendezvous exit stream, look up the hidden service for 'circ', and look up the port and address based on conn->port. Assign the actual conn->addr and conn->port. Return -1 if failure, or 0 for success.

References edge_connection_t::_base, origin_circuit_t::_base, base32_encode(), CIRCUIT_PURPOSE_S_REND_JOINED, LD_REND, circuit_t::n_circ_id, connection_t::port, rend_service_t::ports, circuit_t::purpose, origin_circuit_t::rend_data, rend_data_t::rend_pk_digest, rend_service_get_by_pk_digest(), REND_SERVICE_ID_LEN, REND_SERVICE_ID_LEN_BASE32, smartlist_add(), smartlist_create(), and tor_assert.

Referenced by connection_exit_begin_conn().

4.42.3.28 static void rend_service_update_descriptor (rend_service_t * *service*) [static]

Replace the old value of **service**->desc with one that reflects the other fields in service.

References origin_circuit_t::_base, CIRCUIT_PURPOSE_S_INTRO, crypto_pk_dup_key(), rend_service_t::desc, rend_intro_point_t::extend_info, extend_info_dup(), find_intro_circuit(), rend_intro_point_t::intro_key, rend_service_t::intro_nodes, rend_service_descriptor_t::intro_nodes, rend_service_descriptor_t::pk, rend_service_t::pk_digest, rend_service_t::private_key, rend_service_descriptor_t::protocols, circuit_t::purpose, rend_service_descriptor_free(), smartlist_add(), smartlist_create(), and rend_service_descriptor_t::timestamp.

Referenced by rend_consider_services_upload().

4.42.3.29 void rend_services_introduce (void)

For every service, check how many intro points it currently has, and:

- Pick new intro points as necessary.
- Launch circuits to any new intro points.

References rend_service_t::desc, rend_service_t::desc_is_dirty, DIGEST_LEN, rend_intro_point_t::extend_info, find_intro_circuit(), get_options(), extend_info_t::identity_digest, INTRO_CIRC_RETRY_PERIOD, rend_service_descriptor_t::intro_nodes, rend_service_t::intro_nodes, rend_service_t::intro_period_started, LD_REND, MAX_INTRO_CIRCS_PER_PERIOD, rend_service_t::n_intro_circuits_launched, extend_info_t::nickname, rend_service_t::pk_digest, rend_intro_point_free(), router_get_by_digest(), rend_service_t::service_id, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_del(), and tor_assert.

Referenced by circuit_build_needed_circs().

4.42.3.30 static void upload_service_descriptor (rend_service_t * *service*) [static]

Encode and sign an up-to-date service descriptor for **service**, and upload it/them to the responsible hidden service directories.

References rend_service_t::auth_type, rend_authorized_client_t::client_key, rend_service_t::clients, rend_service_t::desc, rend_service_t::desc_is_dirty, rend_authorized_client_t::descriptor_cookie, directory_post_to_hs_dir(), get_options(), LD_BUG, LD_REND, networkstatus_get_latest_consensus(), rend_service_t::next_upload_time, rend_service_descriptor_t::pk, rend_encode_v2_descriptors(), rend_encoded_v2_service_descriptor_free(), rend_get_service_id(), REND_SERVICE_ID_LEN_BASE32, REND_TIME_PERIOD_OVERLAPPING_V2_DESCS, or_options_t::RendPostPeriod, networkstatus_t::routerstatus_list, smartlist_add(), smartlist_clear(), smartlist_create(), and smartlist_free().

Referenced by rend_consider_descriptor_republication(), and rend_consider_services_upload().

4.42.4 Variable Documentation

4.42.4.1 int consider_republishing_rend_descriptors = 1 `[static]`

True if the list of available router descriptors might have changed so that we should have a look whether we can republish previously failed rendezvous service descriptors.

4.42.4.2 smartlist_t* rend_service_list = NULL `[static]`

A list of rend_service_t's for services run on this OP.

4.43 rephist.c File Reference

Basic history and "reputation" functionality to remember which servers have worked in the past, how much bandwidth we've been using, which ports we tend to want, and so on.

`#include "or.h"`

`#include "ht.h"`

Data Structures

- struct **link_history_t**
- struct **or_history_t**
- struct **bw_array_t**
- struct **circ_buffer_stats_t**

Defines

- #define **STABILITY_EPSILON** 0.0001
- #define **STABILITY_ALPHA** 0.95
- #define **STABILITY_INTERVAL** (12*60*60)
- #define **PUT**(s) STMT_BEGIN if (fputs((s),f)<0) goto err; STMT_END
- #define **PRINTF**(args) STMT_BEGIN if (fprintf args <0) goto err; STMT_END
- #define **MAX_STABILITY_DOC_BUILD_RATE** (3*60)
- #define **NUM_SECS_ROLLING_MEASURE** 10
- #define **NUM_SECS_BW_SUM_INTERVAL** (15*60)
- #define **NUM_SECS_BW_SUM_IS_VALID** (24*60*60)
- #define **NUM_TOTALS** (NUM_SECS_BW_SUM_IS_VALID/NUM_SECS_BW_SUM_INTERVAL)
- #define **EXIT_STATS_ROUND_UP_BYTES** 1024
- #define **EXIT_STATS_ROUND_UP_STREAMS** 4
- #define **EXIT_STATS_NUM_PORTS** 65536
- #define **EXIT_STATS_THRESHOLD_RECIPROCAL** 10000
- #define **PREDICTED_CIRCS_RELEVANCE_TIME** (60*60)
- #define **SHARES** 10

Functions

- static void **bw_arrays_init** (void)
- static void **predicted_ports_init** (void)
- static **or_history_t** * **get_or_history** (const char *id)
- static **link_history_t** * **get_link_history** (const char *from_id, const char *to_id)
- static void **_free_link_history** (void *val)
- static void **free_or_history** (void *_hist)
- static void **update_or_history** (**or_history_t** *hist, time_t when)
- void **rep_hist_init** (void)
- static void **mark_or_down** (**or_history_t** *hist, time_t when, int failed)
- static void **mark_or_up** (**or_history_t** *hist, time_t when)
- void **rep_hist_note_connect_failed** (const char *id, time_t when)
- void **rep_hist_note_connect_succeeded** (const char *id, time_t when)

- void **rep_hist_note_disconnect** (const char *id, time_t when)
- void **rep_hist_note_connection_died** (const char *id, time_t when)
- void **rep_hist_note_router_reachable** (const char *id, time_t when)
- void **rep_hist_note_router_unreachable** (const char *id, time_t when)
- time_t **rep_hist_downrate_old_runs** (time_t now)
- static double **get_stability** (**or_history_t** *hist, time_t when)
- static long **get_total_weighted_time** (**or_history_t** *hist, time_t when)
- static double **get_weighted_fractional_uptime** (**or_history_t** *hist, time_t when)
- double **rep_hist_get_stability** (const char *id, time_t when)
- double **rep_hist_get_weighted_fractional_uptime** (const char *id, time_t when)
- long **rep_hist_get_weighted_time_known** (const char *id, time_t when)
- int **rep_hist_have_measured_enough_stability** (void)
- void **rep_hist_note_extend_succeeded** (const char *from_id, const char *to_id)
- void **rep_hist_note_extend_failed** (const char *from_id, const char *to_id)
- void **rep_hist_dump_stats** (time_t now, int severity)
- void **rep_history_clean** (time_t before)
- int **rep_hist_record_mtbf_data** (time_t now, int missing_means_down)
- static char * **rep_hist_format_router_status** (**or_history_t** *hist, time_t now)
- const char * **rep_hist_get_router_stability_doc** (time_t now)
- static int **find_next_with** (**smartlist_t** *sl, int i, const char *prefix)
- static int **parse_possibly_bad_iso_time** (const char *s, time_t *time_out)
- static INLINE time_t **correct_time** (time_t t, time_t now, time_t stored_at, time_t started_measuring)
- int **rep_hist_load_mtbf_data** (time_t now)
- static void **commit_max** (**bw_array_t** *b)
- static INLINE void **advance_obs** (**bw_array_t** *b)
- static INLINE void **add_obs** (**bw_array_t** *b, time_t when, uint64_t n)
- static **bw_array_t** * **bw_array_new** (void)
- void **rep_hist_note_bytes_written** (size_t num_bytes, time_t when)
- void **rep_hist_note_bytes_read** (size_t num_bytes, time_t when)
- void **rep_hist_exit_stats_init** (time_t now)
- void **rep_hist_exit_stats_write** (time_t now)
- void **rep_hist_note_exit_bytes_written** (uint16_t port, size_t num_bytes)
- void **rep_hist_note_exit_bytes_read** (uint16_t port, size_t num_bytes)
- void **rep_hist_note_exit_stream_opened** (uint16_t port)
- static uint64_t **find_largest_max** (**bw_array_t** *b)
- int **rep_hist_bandwidth_assess** (void)
- static size_t **rep_hist_fill_bandwidth_history** (char *buf, size_t len, **bw_array_t** *b)
- char * **rep_hist_get_bandwidth_lines** (int for_extrainfo)
- void **rep_hist_update_state** (**or_state_t** *state)
- int **rep_hist_load_state** (**or_state_t** *state, char **err)
- static void **add_predicted_port** (time_t now, uint16_t port)
- static void **predicted_ports_free** (void)
- void **rep_hist_note_used_port** (time_t now, uint16_t port)
- **smartlist_t** * **rep_hist_get_predicted_ports** (time_t now)
- void **rep_hist_note_used_resolve** (time_t now)
- void **rep_hist_note_used_internal** (time_t now, int need_uptime, int need_capacity)
- int **rep_hist_get_predicted_internal** (time_t now, int *need_uptime, int *need_capacity)

- int **any_predicted_circuits** (time_t now)
- int **rep_hist_circbuilding_dormant** (time_t now)
- void **note_crypto_pk_op** (**pk_op_t** operation)
- void **dump_pk_ops** (int severity)
- void **rep_hist_free_all** (void)
- void **rep_hist_buffer_stats_init** (time_t now)
- void **rep_hist_buffer_stats_add_circ** (**circuit_t** ∗circ, time_t end_of_interval)
- static int **_buffer_stats_compare_entries** (const void ∗∗_a, const void ∗∗_b)
- void **rep_hist_buffer_stats_write** (time_t now)

Variables

- uint64_t **rephist_total_alloc** = 0
- uint32_t **rephist_total_num** = 0
- static time_t **stability_last_downrated** = 0
- static time_t **started_tracking_stability** = 0
- static digestmap_t ∗ **history_map** = NULL
- static char ∗ **last_stability_doc** = NULL
- static time_t **built_last_stability_doc_at** = 0
- static int **n_bogus_times** = 0
- static **bw_array_t** ∗ **read_array** = NULL
- static **bw_array_t** ∗ **write_array** = NULL
- static uint64_t ∗ **exit_bytes_read** = NULL
- static uint64_t ∗ **exit_bytes_written** = NULL
- static uint32_t ∗ **exit_streams** = NULL
- static time_t **start_of_exit_stats_interval**
- static **smartlist_t** ∗ **predicted_ports_list** = NULL
- static **smartlist_t** ∗ **predicted_ports_times** = NULL
- static time_t **predicted_internal_time** = 0
- static time_t **predicted_internal_uptime_time** = 0
- static time_t **predicted_internal_capacity_time** = 0
- struct {
  unsigned long **n_signed_dir_objs**
  unsigned long **n_signed_routerdescs**
  unsigned long **n_verified_dir_objs**
  unsigned long **n_verified_routerdescs**
  unsigned long **n_onionskins_encrypted**
  unsigned long **n_onionskins_decrypted**
  unsigned long **n_tls_client_handshakes**
  unsigned long **n_tls_server_handshakes**
  unsigned long **n_rend_client_ops**
  unsigned long **n_rend_mid_ops**
  unsigned long **n_rend_server_ops**
  } **pk_op_counts**

- static time_t **start_of_buffer_stats_interval**
- **smartlist_t** ∗ **circuits_for_buffer_stats** = NULL

4.43.1 Detailed Description

Basic history and "reputation" functionality to remember which servers have worked in the past, how much bandwidth we've been using, which ports we tend to want, and so on.

4.43.2 Define Documentation

4.43.2.1 #define EXIT_STATS_NUM_PORTS 65536

Number of TCP ports

Referenced by rep_hist_exit_stats_init(), and rep_hist_exit_stats_write().

4.43.2.2 #define EXIT_STATS_ROUND_UP_BYTES 1024

To what multiple should byte numbers be rounded up?

Referenced by rep_hist_exit_stats_write().

4.43.2.3 #define EXIT_STATS_ROUND_UP_STREAMS 4

To what multiple should stream counts be rounded up?

Referenced by rep_hist_exit_stats_write().

4.43.2.4 #define EXIT_STATS_THRESHOLD_RECIPROCAL 10000

Reciprocal of threshold (= 0.01%) of total bytes that a port needs to see in order to be included in exit stats.

Referenced by rep_hist_exit_stats_write().

4.43.2.5 #define MAX_STABILITY_DOC_BUILD_RATE (3*60)

Shortest allowable time between building two stability documents.

Referenced by rep_hist_get_router_stability_doc().

4.43.2.6 #define NUM_SECS_BW_SUM_INTERVAL (15*60)

How large are the intervals for which we track and report bandwidth use?

Referenced by bw_array_new(), commit_max(), rep_hist_fill_bandwidth_history(), rep_hist_get_bandwidth_lines(), rep_hist_load_state(), and rep_hist_update_state().

4.43.2.7 #define NUM_SECS_BW_SUM_IS_VALID (24*60*60)

How far in the past do we remember and publish bandwidth use?

4.43.2.8 #define NUM_SECS_ROLLING_MEASURE 10

For how many seconds do we keep track of individual per-second bandwidth totals?

Referenced by advance_obs(), rep_hist_bandwidth_assess(), and rep_hist_load_state().

4.43.2.9 #define NUM_TOTALS (NUM_SECS_BW_SUM_IS_VALID/NUM_SECS_BW_SUM_INTERVAL)

How many bandwidth usage intervals do we remember? (derived)

Referenced by commit_max(), find_largest_max(), rep_hist_fill_bandwidth_history(), rep_hist_get_bandwidth_lines(), rep_hist_load_state(), and rep_hist_update_state().

4.43.2.10 #define PREDICTED_CIRCS_RELEVANCE_TIME (60∗60)

For this long after we've seen a request for a given port, assume that we'll want to make connections to the same port in the future.

Referenced by any_predicted_circuits(), rep_hist_get_predicted_internal(), and rep_hist_get_predicted_ports().

4.43.2.11 #define STABILITY_ALPHA 0.95

Value by which to discount all old intervals for MTBF purposes. This is compounded every STABILITY_INTERVAL.

Referenced by rep_hist_downrate_old_runs().

4.43.2.12 #define STABILITY_EPSILON 0.0001

If the total weighted run count of all runs for a router ever falls below this amount, the router can be treated as having 0 MTBF.

Referenced by get_stability(), and rep_history_clean().

4.43.2.13 #define STABILITY_INTERVAL (12∗60∗60)

Interval at which to discount all old intervals for MTBF purposes.

Referenced by rep_hist_downrate_old_runs().

4.43.3 Function Documentation

4.43.3.1 static int _buffer_stats_compare_entries (const void ∗∗ _*a*, const void ∗∗ _*b*) `[static]`

Sorting helper: return -1, 1, or 0 based on comparison of two circ_buffer_stats_t

Referenced by rep_hist_buffer_stats_write().

4.43.3.2 static void _free_link_history (void * *val*) [static]

Helper: free storage held by a single link history entry.

References rephist_total_alloc, and tor_free.

Referenced by free_or_history().

4.43.3.3 static INLINE void add_obs (bw_array_t * *b*, time_t *when*, uint64_t *n*) [static]

Add **n** bytes to the number of bytes in **b** for second **when**.

References advance_obs(), bw_array_t::cur_obs_idx, bw_array_t::cur_obs_time, bw_array_t::obs, and bw_array_t::total_in_period.

Referenced by rep_hist_load_state(), rep_hist_note_bytes_read(), and rep_hist_note_bytes_written().

4.43.3.4 static void add_predicted_port (time_t *now*, uint16_t *port*) [static]

We just got an application request for a connection with port **port**. Remember it for the future, so we can keep some circuits open that will exit to this port.

References rephist_total_alloc, and smartlist_add().

Referenced by predicted_ports_init(), and rep_hist_note_used_port().

4.43.3.5 static INLINE void advance_obs (bw_array_t * *b*) [static]

Shift the current observation time of 'b' forward by one second.

References commit_max(), bw_array_t::cur_obs_idx, bw_array_t::cur_obs_time, bw_array_t::max_total, bw_array_t::next_period, NUM_SECS_ROLLING_MEASURE, bw_array_t::obs, and bw_array_t::total_obs.

Referenced by add_obs().

4.43.3.6 int any_predicted_circuits (time_t *now*)

Any ports used lately? These are pre-seeded if we just started up or if we're running a hidden service.

References PREDICTED_CIRCS_RELEVANCE_TIME, and predicted_internal_time.

Referenced by directory_info_has_arrived(), options_act(), rep_hist_circbuilding_dormant(), and run_scheduled_events().

4.43.3.7 static bw_array_t* bw_array_new (void) [static]

Allocate, initialize, and return a new bw_array.

References bw_array_t::cur_obs_time, bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, and rephist_total_alloc.

Referenced by bw_arrays_init(), and rep_hist_load_state().

4.43.3.8 static void bw_arrays_init (void) [static]

Set up read_array and write_array.

References bw_array_new().

Referenced by rep_hist_init().

4.43.3.9 static void commit_max (bw_array_t * *b*) [static]

Shift the current period of b forward by one.

References bw_array_t::max_total, bw_array_t::maxima, bw_array_t::next_max_idx, bw_array_t::next_period, bw_array_t::num_maxes_set, NUM_SECS_BW_SUM_INTERVAL, NUM_TOTALS, bw_array_t::total_in_period, and bw_array_t::totals.

Referenced by advance_obs().

4.43.3.10 static INLINE time_t correct_time (time_t *t*, time_t *now*, time_t *stored_at*, time_t *started_measuring*) [static]

We've read a time **t** from a file stored at **stored_at**, which says we started measuring at **started_measuring**. Return a new number that's about as much before **now** as **t** was before **stored_at**.

Referenced by rep_hist_load_mtbf_data().

4.43.3.11 void dump_pk_ops (int *severity*)

Log the number of times we've done each public/private-key operation.

References LD_HIST, and pk_op_counts.

Referenced by dumpstats().

4.43.3.12 static uint64_t find_largest_max (bw_array_t * *b*) [static]

Helper: Return the largest value in b->maxima. (This is equal to the most bandwidth used in any NUM_SECS_ROLLING_MEASURE period for the last NUM_SECS_BW_SUM_IS_VALID seconds.)

References bw_array_t::maxima, and NUM_TOTALS.

Referenced by rep_hist_bandwidth_assess().

4.43.3.13 static int find_next_with (smartlist_t * *sl*, int *i*, const char * *prefix*) [static]

Helper: return the first j >= i such that !strcmpstart(sl[j], prefix) and such that no line sl[k] with i <= k < j starts with "R ". Return -1 if no such line exists.

References strcmpstart().

Referenced by rep_hist_load_mtbf_data().

4.43.3.14 static void free_or_history (void ∗ *_hist*) `[static]`

Helper: free storage held by a single OR history entry.

References _free_link_history(), digestmap_free(), or_history_t::link_history_map, rephist_total_alloc, rephist_total_num, and tor_free.

Referenced by rep_hist_free_all(), and rep_history_clean().

4.43.3.15 static link_history_t∗ get_link_history (const char ∗ *from_id*, const char ∗ *to_id*) `[static]`

Return the **link_history_t** (p. 106) for the link from the first named OR to the second, creating it if necessary. (ORs are identified by identity digest.)

References link_history_t::changed, DIGEST_LEN, digestmap_get(), digestmap_set(), get_or_history(), or_history_t::link_history_map, rephist_total_alloc, link_history_t::since, and tor_mem_is_zero().

Referenced by rep_hist_note_extend_failed(), and rep_hist_note_extend_succeeded().

4.43.3.16 static or_history_t∗ get_or_history (const char ∗ *id*) `[static]`

Return the **or_history_t** (p. 145) for the OR with identity digest **id**, creating it if necessary.

References or_history_t::changed, DIGEST_LEN, digestmap_get(), digestmap_new(), digestmap_set(), history_map, or_history_t::link_history_map, rephist_total_alloc, rephist_total_num, or_history_t::since, and tor_mem_is_zero().

Referenced by get_link_history(), rep_hist_get_stability(), rep_hist_get_weighted_fractional_uptime(), rep_hist_get_weighted_time_known(), rep_hist_load_mtbf_data(), rep_hist_note_connect_failed(), rep_hist_note_connect_succeeded(), rep_hist_note_connection_died(), rep_hist_note_disconnect(), rep_hist_note_router_reachable(), and rep_hist_note_router_unreachable().

4.43.3.17 static double get_stability (or_history_t ∗ *hist*, time_t *when*) `[static]`

Helper: Return the weighted MTBF of the router with history **hist**.

References STABILITY_EPSILON, or_history_t::start_of_run, or_history_t::total_run_weights, and or_history_t::weighted_run_length.

Referenced by rep_hist_dump_stats(), rep_hist_format_router_status(), and rep_hist_get_stability().

4.43.3.18 static long get_total_weighted_time (or_history_t ∗ *hist*, time_t *when*) `[static]`

Return the total amount of time we've been observing, with each run of time downrated by the appropriate factor.

References or_history_t::start_of_downtime, or_history_t::start_of_run, and or_history_t::total_weighted_time.

Referenced by rep_hist_get_weighted_time_known().

4.43.3.19 static double get_weighted_fractional_uptime (or_history_t * *hist*, time_t *when*) [static]

Helper: Return the weighted percent-of-time-online of the router with history **hist**.

References or_history_t::start_of_downtime, or_history_t::start_of_run, or_history_t::total_weighted_time, and or_history_t::weighted_uptime.

Referenced by rep_hist_format_router_status(), and rep_hist_get_weighted_fractional_uptime().

4.43.3.20 static void mark_or_down (or_history_t * *hist*, time_t *when*, int *failed*) [static]

Helper: note that we are no longer connected to the router with history **hist**. If **failed**, the connection failed; otherwise, it was closed correctly.

References or_history_t::down_since, or_history_t::up_since, and or_history_t::uptime.

Referenced by rep_hist_note_connect_failed(), rep_hist_note_connection_died(), and rep_hist_note_disconnect().

4.43.3.21 static void mark_or_up (or_history_t * *hist*, time_t *when*) [static]

Helper: note that we are connected to the router with history **hist**.

References or_history_t::down_since, or_history_t::downtime, and or_history_t::up_since.

Referenced by rep_hist_note_connect_succeeded().

4.43.3.22 void note_crypto_pk_op (pk_op_t *operation*)

Increment the count of the number of times we've done **operation**.

References LD_BUG, and pk_op_counts.

Referenced by connection_tls_start_handshake(), dirserv_dump_directory_to_string(), extrainfo_parse_entry_from_string(), format_networkstatus_vote(), generate_runningrouters(), networkstatus_v2_parse_from_string(), onion_skin_create(), onion_skin_server_handshake(), rend_mid_establish_intro(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_intro_has_opened(), rend_service_introduce(), router_dump_router_to_string(), router_parse_directory(), router_parse_entry_from_string(), and router_parse_runningrouters().

4.43.3.23 static int parse_possibly_bad_iso_time (const char * *s*, time_t * *time_out*) [static]

Parse the ISO-formatted time in **s** into ***time_out**, but rounds any pre-1970 date to Jan 1, 1970.

References n_bogus_times, parse_iso_time(), and tor_parse_long().

Referenced by rep_hist_load_mtbf_data().

4.43.3.24 static void predicted_ports_free (void) [static]

Free whatever memory is needed for predicting which ports will be used.

References rephist_total_alloc, smartlist_free(), and tor_free.

Referenced by rep_hist_free_all().

4.43.3.25 static void predicted_ports_init (void) [static]

Initialize whatever memory and structs are needed for predicting which ports will be used. Also seed it with port 80, so we'll build circuits on start-up.

References add_predicted_port(), and smartlist_create().

Referenced by rep_hist_init().

4.43.3.26 int rep_hist_bandwidth_assess (void)

Find the largest sums in the past NUM_SECS_BW_SUM_IS_VALID (roughly) seconds. Find one sum for reading and one for writing. They don't have to be at the same time.

Return the smaller of these sums, divided by NUM_SECS_ROLLING_MEASURE.

References find_largest_max(), and NUM_SECS_ROLLING_MEASURE.

Referenced by check_descriptor_bandwidth_changed(), and router_rebuild_descriptor().

4.43.3.27 void rep_hist_buffer_stats_add_circ (circuit_t * *circ*, time_t *end_of_interval*)

Remember cell statistics for circuit **circ** at time **end_of_interval** and reset cell counters in case the circuit remains open in the next measurement interval.

References CIRCUIT_IS_ORIGIN, or_circuit_t::processed_cells, smartlist_add(), smartlist_create(), start_of_buffer_stats_interval, circuit_t::timestamp_created, TO_OR_CIRCUIT(), and or_circuit_t::total_cell_waiting_time.

Referenced by circuit_free(), and rep_hist_buffer_stats_write().

4.43.3.28 void rep_hist_buffer_stats_init (time_t *now*)

Initialize buffer stats.

References start_of_buffer_stats_interval.

Referenced by run_scheduled_events().

4.43.3.29 void rep_hist_buffer_stats_write (time_t *now*)

Write buffer statistics to $DATADIR/stats/buffer-stats.

References _buffer_stats_compare_entries(), _circuit_get_global_list(), abort_writing_to_file(), check_private_dir(), finish_writing_to_file(), format_iso_time(), get_datadir_fname, get_datadir_fname2, LD_HIST, circuit_t::next, rep_hist_buffer_stats_add_circ(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_sort(), start_of_buffer_stats_interval, start_writing_to_stdio_file(), tor_asprintf(), and tor_free.

Referenced by run_scheduled_events().

4.43.3.30 int rep_hist_circbuilding_dormant (time_t *now*)

Return 1 if we have no need for circuits currently, else return 0.

References any_predicted_circuits(), check_whether_dirport_reachable(), check_whether_orport_reachable(), circuit_enough_testing_circs(), get_options(), and server_mode().

Referenced by directory_too_idle_to_fetch_descriptors().

4.43.3.31 time_t rep_hist_downrate_old_runs (time_t *now*)

Helper: Discount all old MTBF data, if it is time to do so. Return the time at which we should next discount MTBF data.

References digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_new(), history_map, LD_HIST, STABILITY_ALPHA, STABILITY_INTERVAL, stability_last_downrated, or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by run_scheduled_events().

4.43.3.32 void rep_hist_dump_stats (time_t *now*, int *severity*)

Log all the reliability data we have remembered, with the chosen severity.

References base16_encode(), DIGEST_LEN, digestmap_isempty(), digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), or_history_t::downtime, get_options(), get_stability(), HEX_DIGEST_LEN, history_map, LD_HIST, or_history_t::link_history_map, or_history_t::n_conn_fail, or_history_t::n_conn_ok, link_history_t::n_extend_fail, link_history_t::n_extend_ok, routerinfo_t::nickname, rep_history_clean(), router_get_by_digest(), tor_snprintf(), update_or_history(), and or_history_t::uptime.

Referenced by dumpstats().

4.43.3.33 void rep_hist_exit_stats_init (time_t *now*)

Initialize exit port stats.

References exit_bytes_read, exit_bytes_written, EXIT_STATS_NUM_PORTS, exit_streams, and start_of_exit_stats_interval.

Referenced by run_scheduled_events().

4.43.3.34 void rep_hist_exit_stats_write (time_t *now*)

Write exit stats to $DATADIR/stats/exit-stats and reset counters.

References abort_writing_to_file(), check_private_dir(), exit_bytes_read, exit_bytes_written, EXIT_STATS_NUM_PORTS, EXIT_STATS_ROUND_UP_BYTES, EXIT_STATS_ROUND_UP_STREAMS, EXIT_STATS_THRESHOLD_RECIPROCAL, exit_streams, finish_writing_to_file(), format_iso_time(), get_datadir_fname, get_datadir_fname2, LD_HIST, round_uint32_to_next_multiple_of(), round_uint64_to_next_multiple_of(), start_of_exit_stats_interval, start_writing_to_stdio_file(), tor_assert, and tor_free.

Referenced by run_scheduled_events().

4.43.3.35 static size_t rep_hist_fill_bandwidth_history (char * *buf*, size_t *len*, bw_array_t * *b*) [static]

Print the bandwidth history of b (either read_array or write_array) into the buffer pointed to by buf. The format is simply comma separated numbers, from oldest to newest.

It returns the number of bytes written.

References get_options(), bw_array_t::next_max_idx, bw_array_t::num_maxes_set, NUM_SECS_BW_SUM_INTERVAL, NUM_TOTALS, or_options_t::RelayBandwidthRate, tor_assert, tor_snprintf(), and bw_array_t::totals.

Referenced by rep_hist_get_bandwidth_lines(), and rep_hist_update_state().

4.43.3.36 static char* rep_hist_format_router_status (or_history_t * *hist*, time_t *now*) [static]

Format the current tracked status of the router in **hist** at time **now** for analysis; return it in a newly allocated string.

References format_iso_time(), get_stability(), get_weighted_fractional_uptime(), or_history_t::start_of_downtime, or_history_t::start_of_run, tor_asprintf(), or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by rep_hist_get_router_stability_doc().

4.43.3.37 void rep_hist_free_all (void)

Free all storage held by the OR/link history caches, by the bandwidth history arrays, or by the port history.

References built_last_stability_doc_at, digestmap_free(), exit_bytes_read, exit_bytes_written, exit_streams, free_or_history(), history_map, last_stability_doc, predicted_ports_free(), and tor_free.

Referenced by tor_free_all().

4.43.3.38 char* rep_hist_get_bandwidth_lines (int *for_extrainfo*)

Allocate and return lines for representing this server's bandwidth history in its descriptor.

References format_iso_time(), bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, NUM_TOTALS, rep_hist_fill_bandwidth_history(), tor_assert, and tor_snprintf().

Referenced by extrainfo_dump_to_string().

4.43.3.39 int rep_hist_get_predicted_internal (time_t *now*, int * *need_uptime*, int * *need_capacity*)

Return 1 if we've used an internal circ recently; else return 0.

References PREDICTED_CIRCS_RELEVANCE_TIME, predicted_internal_capacity_time, predicted_internal_time, and predicted_internal_uptime_time.

Referenced by circuit_predict_and_launch_new().

4.43.3.40 smartlist_t∗ rep_hist_get_predicted_ports (time_t *now*)

Return a pointer to the list of port numbers that are likely to be asked for in the near future.

The caller promises not to mess with it.

References LD_CIRC, PREDICTED_CIRCS_RELEVANCE_TIME, rephist_total_alloc, smartlist_del(), tor_assert, and tor_free.

Referenced by circuit_get_unhandled_ports().

4.43.3.41 const char∗ rep_hist_get_router_stability_doc (time_t *now*)

Return a pointer to a NUL-terminated document describing our view of the stability of the routers we've been tracking. Return NULL on failure.

References routerinfo_t::addr, BASE64_DIGEST_LEN, built_last_stability_doc_at, routerinfo_t::cache_info, digest_to_base64(), format_iso_time(), history_map, routerinfo_t::is_hibernating, routerinfo_t::is_running, routerinfo_t::is_valid, last_stability_doc, MAX_STABILITY_DOC_BUILD_RATE, routerinfo_t::nickname, signed_descriptor_t::published_on, rep_hist_format_router_status(), rep_hist_have_measured_enough_stability(), router_get_by_digest(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), tor_dup_ip(), tor_free, tor_snprintf(), and routerinfo_t::uptime.

Referenced by directory_handle_command_get().

4.43.3.42 double rep_hist_get_stability (const char ∗ *id*, time_t *when*)

Return an estimated MTBF for the router whose identity digest is **id**. Return 0 if the router is unknown.

References get_or_history(), and get_stability().

Referenced by dirserv_compute_performance_thresholds(), and dirserv_thinks_router_is_unreliable().

4.43.3.43 double rep_hist_get_weighted_fractional_uptime (const char ∗ *id*, time_t *when*)

Return an estimated percent-of-time-online for the router whose identity digest is **id**. Return 0 if the router is unknown.

References get_or_history(), and get_weighted_fractional_uptime().

Referenced by set_routerstatus_from_routerinfo().

4.43.3.44 long rep_hist_get_weighted_time_known (const char ∗ *id*, time_t *when*)

Return a number representing how long we've known about the router whose digest is **id**. Return 0 if the router is unknown.

Be careful: this measure increases monotonically as we know the router for longer and longer, but it doesn't increase linearly.

References get_or_history(), and get_total_weighted_time().

Referenced by dirserv_compute_performance_thresholds(), and set_routerstatus_from_routerinfo().

4.43.3.45 int rep_hist_have_measured_enough_stability (void)

Return true if we've been measuring MTBFs for long enough to pronounce on Stability.

References started_tracking_stability.

Referenced by rep_hist_get_router_stability_doc().

4.43.3.46 void rep_hist_init (void)

Initialize the static data structures for tracking history.

References bw_arrays_init(), digestmap_new(), history_map, and predicted_ports_init().

Referenced by tor_init().

4.43.3.47 int rep_hist_load_mtbf_data (time_t *now*)

Load MTBF data from disk. Returns 0 on success or recoverable error, -1 on failure.

References base16_decode(), correct_time(), DIGEST_LEN, escaped(), find_next_with(), get_datadir_fname, get_or_history(), HEX_DIGEST_LEN, LD_HIST, n_bogus_times, parse_iso_time(), parse_possibly_bad_iso_time(), read_file_to_str(), RFTS_IGNORE_MISSING, smartlist_create(), smartlist_free(), smartlist_split_string(), stability_last_downrated, or_history_t::start_of_downtime, or_history_t::start_of_run, started_tracking_stability, strcmpstart(), tor_free, tor_parse_long(), or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by options_act().

4.43.3.48 int rep_hist_load_state (or_state_t * *state*, char ** *err*)

Set bandwidth history from our saved state.

References add_obs(), bw_array_new(), or_state_t::BWHistoryReadEnds, or_state_t::BWHistoryReadInterval, or_state_t::BWHistoryReadValues, or_state_t::BWHistoryWriteEnds, or_state_t::BWHistoryWriteInterval, or_state_t::BWHistoryWriteValues, bw_array_t::cur_obs_time, LD_HIST, bw_array_t::max_total, bw_array_t::maxima, bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, NUM_SECS_ROLLING_MEASURE, NUM_TOTALS, bw_array_t::obs, tor_assert, tor_free, tor_parse_uint64(), and bw_array_t::total_obs.

Referenced by or_state_set().

4.43.3.49 void rep_hist_note_bytes_read (size_t *num_bytes*, time_t *when*)

We wrote **num_bytes** more bytes in second **when**. (like **rep_hist_note_bytes_written()** (p. 980) above)

References add_obs().

Referenced by connection_buckets_decrement().

4.43.3.50 void rep_hist_note_bytes_written (size_t *num_bytes*, time_t *when*)

We read **num_bytes** more bytes in second **when**.

Add num_bytes to the current running total for **when**.

when can go back to time, but it's safe to ignore calls earlier than the latest **when** you've heard of.

References add_obs().

Referenced by connection_buckets_decrement().

4.43.3.51 void rep_hist_note_connect_failed (const char * *id*, time_t *when*)

Remember that an attempt to connect to the OR with identity digest **id** failed at **when**.

References or_history_t::changed, get_or_history(), mark_or_down(), and or_history_t::n_conn_fail.

Referenced by connection_about_to_close_connection().

4.43.3.52 void rep_hist_note_connect_succeeded (const char * *id*, time_t *when*)

Remember that an attempt to connect to the OR with identity digest **id** succeeded at **when**.

References or_history_t::changed, get_or_history(), mark_or_up(), and or_history_t::n_conn_ok.

Referenced by connection_or_set_state_open().

4.43.3.53 void rep_hist_note_connection_died (const char * *id*, time_t *when*)

Remember that our connection to the OR with identity digest **id** had an error and stopped working at **when**.

References or_history_t::changed, get_or_history(), and mark_or_down().

Referenced by connection_about_to_close_connection().

4.43.3.54 void rep_hist_note_disconnect (const char * *id*, time_t *when*)

Remember that we intentionally closed our connection to the OR with identity digest **id** at **when**.

References or_history_t::changed, get_or_history(), and mark_or_down().

Referenced by connection_about_to_close_connection().

4.43.3.55 void rep_hist_note_exit_bytes_read (uint16_t *port*, size_t *num_bytes*)

Note that we read **num_bytes** from an exit connection to **port**.

References exit_bytes_read, get_options(), and LD_HIST.

Referenced by connection_buckets_decrement().

4.43.3.56 void rep_hist_note_exit_bytes_written (uint16_t *port*, size_t *num_bytes*)

Note that we wrote **num_bytes** to an exit connection to **port**.

References exit_bytes_written, get_options(), and LD_HIST.

Referenced by connection_buckets_decrement().

4.43.3.57 void rep_hist_note_exit_stream_opened (uint16_t *port*)

Note that we opened an exit stream to **port**.

References exit_streams, get_options(), and LD_HIST.

Referenced by connection_edge_finished_connecting().

4.43.3.58 void rep_hist_note_extend_failed (const char * *from_id*, const char * *to_id*)

Remember that we tried to extend from the OR with identity digest **from_id** to the OR with identity digest **to_name**, but failed.

References link_history_t::changed, get_link_history(), and link_history_t::n_extend_fail.

Referenced by circuit_rep_hist_note_result().

4.43.3.59 void rep_hist_note_extend_succeeded (const char * *from_id*, const char * *to_id*)

Remember that we successfully extended from the OR with identity digest **from_id** to the OR with identity digest **to_name**.

References link_history_t::changed, get_link_history(), and link_history_t::n_extend_ok.

Referenced by circuit_rep_hist_note_result().

4.43.3.60 void rep_hist_note_router_reachable (const char * *id*, time_t *when*)

We have just decided that this router with identity digest **id** is reachable, meaning we will give it a "Running" flag for the next while.

References DIGEST_LEN, format_local_iso_time(), get_or_history(), hex_str(), LD_HIST, or_history_t::start_of_downtime, or_history_t::start_of_run, started_tracking_stability, tor_assert, and or_history_t::total_weighted_time.

Referenced by dirserv_orconn_tls_done().

4.43.3.61 void rep_hist_note_router_unreachable (const char * *id*, time_t *when*)

We have just decided that this router is unreachable, meaning we are taking away its "Running" flag.

References DIGEST_LEN, format_local_iso_time(), get_or_history(), hex_str(), LD_HIST, or_history_t::start_of_downtime, or_history_t::start_of_run, started_tracking_-

stability, tor_assert, or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by dirserv_set_router_is_running(), rep_hist_record_mtbf_data(), and routerlist_remove().

4.43.3.62 void rep_hist_note_used_internal (time_t *now*, int *need_uptime*, int *need_capacity*)

Remember that we used an internal circ at time **now**.

References predicted_internal_capacity_time, predicted_internal_time, and predicted_internal_uptime_time.

Referenced by circuit_get_open_circ_or_launch(), directory_initiate_command_rend(), rend_service_introduce(), and rend_service_launch_establish_intro().

4.43.3.63 void rep_hist_note_used_port (time_t *now*, uint16_t *port*)

Remember that **port** has been asked for as of time **now**. This is used for predicting what sorts of streams we'll make in the future and making exit circuits to anticipate that.

References add_predicted_port(), and tor_assert.

Referenced by directory_initiate_command_rend(), and rep_hist_note_used_resolve().

4.43.3.64 void rep_hist_note_used_resolve (time_t *now*)

The user asked us to do a resolve. Rather than keeping track of timings and such of resolves, we fake it for now by treating it the same way as a connection to port 80. This way we will continue to have circuits lying around if the user only uses Tor for resolves.

References rep_hist_note_used_port().

4.43.3.65 int rep_hist_record_mtbf_data (time_t *now*, int *missing_means_down*)

Write MTBF data to disk. Return 0 on success, negative on failure.

If **missing_means_down**, then if we're about to write an entry that is still considered up but isn't in our routerlist, consider it to be down.

References abort_writing_to_file(), base16_encode(), DIGEST_LEN, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), finish_writing_to_file(), format_iso_time(), get_datadir_fname, HEX_DIGEST_LEN, history_map, LD_HIST, rep_hist_note_router_unreachable(), router_get_by_digest(), stability_last_downrated, or_history_t::start_of_downtime, or_history_t::start_of_run, start_writing_to_stdio_file(), started_tracking_stability, tor_free, or_history_t::total_run_weights, or_history_t::total_weighted_time, or_history_t::weighted_run_length, and or_history_t::weighted_uptime.

Referenced by run_scheduled_events(), and tor_cleanup().

4.43.3.66 void rep_hist_update_state (or_state_t * *state*)

Update **state** with the newest bandwidth history.

References or_options_t::AvoidDiskWrites, or_state_t::BWHistoryReadEnds, or_state_t::BWHistoryReadInterval, or_state_t::BWHistoryReadValues, or_state_t::BWHistoryWriteEnds, or_state_t::BWHistoryWriteInterval, or_state_t::BWHistoryWriteValues, get_options(), get_or_state(), bw_array_t::next_period, NUM_SECS_BW_SUM_INTERVAL, NUM_TOTALS, or_state_mark_dirty(), rep_hist_fill_bandwidth_history(), server_mode(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_free, tor_snprintf(), and bw_array_t::total_in_period.

Referenced by or_state_save().

4.43.3.67 void rep_history_clean (time_t *before*)

Remove history info for routers/links that haven't changed since **before**.

References authdir_mode(), link_history_t::changed, or_history_t::changed, digestmap_iter_done(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_iter_next_rmv(), free_or_history(), get_options(), history_map, or_history_t::link_history_map, rephist_total_alloc, STABILITY_EPSILON, or_history_t::start_of_run, tor_free, and or_history_t::total_run_weights.

Referenced by rep_hist_dump_stats(), and run_scheduled_events().

4.43.3.68 static void update_or_history (or_history_t * *hist*, time_t *when*) `[static]`

Update an **or_history_t** (p. 145) object **hist** so that its uptime/downtime count is up-to-date as of **when**.

References or_history_t::down_since, or_history_t::downtime, tor_assert, or_history_t::up_since, and or_history_t::uptime.

Referenced by rep_hist_dump_stats().

4.43.4 Variable Documentation

4.43.4.1 time_t built_last_stability_doc_at = 0 `[static]`

The last time we created a stability analysis document, or 0 if we never have created one.

Referenced by rep_hist_free_all(), and rep_hist_get_router_stability_doc().

4.43.4.2 smartlist_t* circuits_for_buffer_stats = NULL

Holds stats.

4.43.4.3 uint64_t* exit_bytes_read = NULL `[static]`

Number of bytes read in current period by exit port

Referenced by rep_hist_exit_stats_init(), rep_hist_exit_stats_write(), rep_hist_free_all(), and rep_hist_note_exit_bytes_read().

4.43.4.4 uint64_t* exit_bytes_written = NULL [static]

Number of bytes written in current period by exit port

Referenced by rep_hist_exit_stats_init(), rep_hist_exit_stats_write(), rep_hist_free_all(), and rep_hist_note_exit_bytes_written().

4.43.4.5 uint32_t* exit_streams = NULL [static]

Number of streams opened in current period by exit port

Referenced by rep_hist_exit_stats_init(), rep_hist_exit_stats_write(), rep_hist_free_all(), and rep_hist_note_exit_stream_opened().

4.43.4.6 digestmap_t* history_map = NULL [static]

Map from hex OR identity digest to **or_history_t** (p. 145).

Referenced by get_or_history(), rep_hist_downrate_old_runs(), rep_hist_dump_stats(), rep_hist_free_all(), rep_hist_get_router_stability_doc(), rep_hist_init(), rep_hist_record_mtbf_data(), and rep_history_clean().

4.43.4.7 char* last_stability_doc = NULL [static]

The last stability analysis document that we created, or NULL if we never have created one.

Referenced by rep_hist_free_all(), and rep_hist_get_router_stability_doc().

4.43.4.8 int n_bogus_times = 0 [static]

How many bad times has parse_possibly_bad_iso_time parsed?

Referenced by parse_possibly_bad_iso_time(), and rep_hist_load_mtbf_data().

4.43.4.9 unsigned long n_onionskins_decrypted

How many onionskins have we decrypted to do circuit build requests?

4.43.4.10 unsigned long n_onionskins_encrypted

How many onionskins have we encrypted to build circuits?

4.43.4.11 unsigned long n_rend_client_ops

How many PK operations have we done as a hidden service client?

4.43.4.12 unsigned long n_rend_mid_ops

How many PK operations have we done as a hidden service midpoint?

4.43.4.13 unsigned long n_rend_server_ops

How many PK operations have we done as a hidden service provider?

4.43.4.14 unsigned long n_signed_dir_objs

How many directory objects have we signed?

4.43.4.15 unsigned long n_signed_routerdescs

How many routerdescs have we signed?

4.43.4.16 unsigned long n_tls_client_handshakes

How many times have we done the TLS handshake as a client?

4.43.4.17 unsigned long n_tls_server_handshakes

How many times have we done the TLS handshake as a server?

4.43.4.18 unsigned long n_verified_dir_objs

How many directory objects have we verified?

4.43.4.19 unsigned long n_verified_routerdescs

How many routerdescs have we verified

4.43.4.20 struct { ... } pk_op_counts `[static]`

Structure to track how many times we've done each public key operation.

Referenced by dump_pk_ops(), and note_crypto_pk_op().

4.43.4.21 time_t predicted_internal_capacity_time = 0 `[static]`

The last time we needed an internal circ with good capacity.

Referenced by rep_hist_get_predicted_internal(), and rep_hist_note_used_internal().

4.43.4.22 time_t predicted_internal_time = 0 `[static]`

The last time at which we needed an internal circ.

Referenced by any_predicted_circuits(), rep_hist_get_predicted_internal(), and rep_hist_note_used_internal().

4.43.4.23 time_t predicted_internal_uptime_time = 0 [static]

The last time we needed an internal circ with good uptime.

Referenced by rep_hist_get_predicted_internal(), and rep_hist_note_used_internal().

4.43.4.24 smartlist_t* predicted_ports_list = NULL [static]

A list of port numbers that have been used recently.

4.43.4.25 smartlist_t* predicted_ports_times = NULL [static]

The corresponding most recently used time for each port.

4.43.4.26 bw_array_t* read_array = NULL [static]

Recent history of bandwidth observations for read operations.

4.43.4.27 uint64_t rephist_total_alloc = 0

Total number of bytes currently allocated in fields used by **rephist.c** (p. 1095).

Referenced by _free_link_history(), add_predicted_port(), bw_array_new(), dumpmemusage(), free_or_history(), get_link_history(), get_or_history(), predicted_ports_free(), rep_hist_get_predicted_ports(), and rep_history_clean().

4.43.4.28 uint32_t rephist_total_num = 0

Number of **or_history_t** (p. 145) objects currently allocated.

Referenced by dumpmemusage(), free_or_history(), and get_or_history().

4.43.4.29 time_t stability_last_downrated = 0 [static]

When did we last multiply all routers' weighted_run_length and total_run_weights by STABILITY_ALPHA?

Referenced by rep_hist_downrate_old_runs(), rep_hist_load_mtbf_data(), and rep_hist_record_mtbf_data().

4.43.4.30 time_t start_of_buffer_stats_interval [static]

Start of the current buffer stats interval.

Referenced by rep_hist_buffer_stats_add_circ(), rep_hist_buffer_stats_init(), and rep_hist_buffer_stats_write().

4.43.4.31 time_t start_of_exit_stats_interval [static]

When does the current exit stats period end?

Referenced by rep_hist_exit_stats_init(), and rep_hist_exit_stats_write().

4.43.4.32 time_t started_tracking_stability = 0 [static]

Referenced by rep_hist_have_measured_enough_stability(), rep_hist_load_mtbf_data(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), and rep_hist_record_mtbf_data().

4.43.4.33 bw_array_t* write_array = NULL [static]

Recent history of bandwidth observations for write operations.

4.44 router.c File Reference

OR functionality, including key maintenance, generating and uploading server descriptors, retrying OR connections.

`#include "or.h"`

Defines

- #define **ROUTER_PRIVATE**
- #define **MIN_BW_TO_ADVERTISE_DIRPORT** 51200
- #define **MAX_BANDWIDTH_CHANGE_FREQ** (20*60)
- #define **DEBUG_ROUTER_DUMP_ROUTER_TO_STRING**

Functions

- static void **set_onion_key** (**crypto_pk_env_t** *k)
- **crypto_pk_env_t** * **get_onion_key** (void)
- void **dup_onion_keys** (**crypto_pk_env_t** **key, **crypto_pk_env_t** **last)
- time_t **get_onion_key_set_at** (void)
- void **set_identity_key** (**crypto_pk_env_t** *k)
- **crypto_pk_env_t** * **get_identity_key** (void)
- int **identity_key_is_set** (void)
- **authority_cert_t** * **get_my_v3_authority_cert** (void)
- **crypto_pk_env_t** * **get_my_v3_authority_signing_key** (void)
- **authority_cert_t** * **get_my_v3_legacy_cert** (void)
- **crypto_pk_env_t** * **get_my_v3_legacy_signing_key** (void)
- void **rotate_onion_key** (void)
- **crypto_pk_env_t** * **init_key_from_file** (const char *fname, int generate, int severity)
- static int **load_authority_keyset** (int legacy, **crypto_pk_env_t** **key_out, **authority_cert_t** **cert_out)
- static int **init_v3_authority_keys** (void)
- void **v3_authority_check_key_expiry** (void)
- int **init_keys** (void)
- void **router_reset_reachability** (void)
- int **check_whether_orport_reachable** (void)
- int **check_whether_dirport_reachable** (void)
- static int **decide_to_advertise_dirport** (**or_options_t** *options, uint16_t dir_port)
- void **consider_testing_reachability** (int test_or, int test_dir)
- void **router_orport_found_reachable** (void)
- void **router_dirport_found_reachable** (void)
- void **router_perform_bandwidth_test** (int num_circs, time_t now)
- int **authdir_mode** (**or_options_t** *options)
- int **authdir_mode_v1** (**or_options_t** *options)
- int **authdir_mode_v2** (**or_options_t** *options)
- int **authdir_mode_v3** (**or_options_t** *options)
- int **authdir_mode_any_main** (**or_options_t** *options)
- int **authdir_mode_any_nonhidserv** (**or_options_t** *options)
- int **authdir_mode_handles_descs** (**or_options_t** *options, int purpose)
- int **authdir_mode_publishes_statuses** (**or_options_t** *options)

- int **authdir_mode_tests_reachability** (**or_options_t** ∗options)
- int **authdir_mode_bridge** (**or_options_t** ∗options)
- int **server_mode** (**or_options_t** ∗options)
- int **advertised_server_mode** (void)
- static void **set_server_advertised** (int s)
- int **proxy_mode** (**or_options_t** ∗options)
- static int **decide_if_publishable_server** (void)
- void **consider_publishable_server** (int force)
- void **router_upload_dir_desc_to_dirservers** (int force)
- int **router_compare_to_my_exit_policy** (**edge_connection_t** ∗conn)
- int **router_digest_is_me** (const char ∗digest)
- int **router_extrainfo_digest_is_me** (const char ∗digest)
- int **router_is_me** (**routerinfo_t** ∗router)
- int **router_fingerprint_is_me** (const char ∗fp)
- **routerinfo_t** ∗ **router_get_my_routerinfo** (void)
- const char ∗ **router_get_my_descriptor** (void)
- **extrainfo_t** ∗ **router_get_my_extrainfo** (void)
- static int **router_guess_address_from_dir_headers** (uint32_t ∗guess)
- int **router_pick_published_address** (**or_options_t** ∗options, uint32_t ∗addr)
- int **router_rebuild_descriptor** (int force)
- void **mark_my_descriptor_dirty_if_older_than** (time_t when)
- void **mark_my_descriptor_dirty** (void)
- void **check_descriptor_bandwidth_changed** (time_t now)
- static void **log_addr_has_changed** (int severity, uint32_t prev, uint32_t cur, const char ∗source)
- void **check_descriptor_ipaddress_changed** (time_t now)
- void **router_new_address_suggestion** (const char ∗suggestion, const **dir_connection_t** ∗d_conn)
- void **get_platform_str** (char ∗platform, size_t len)
- int **router_dump_router_to_string** (char ∗s, size_t maxlen, **routerinfo_t** ∗router, **crypto_pk_env_t** ∗ident_key)
- static int **load_stats_file** (const char ∗filename, const char ∗end_line, time_t now, char ∗∗out)
- int **extrainfo_dump_to_string** (char ∗s, size_t maxlen, **extrainfo_t** ∗extrainfo, **crypto_pk_env_t** ∗ident_key)
- int **is_legal_nickname** (const char ∗s)
- int **is_legal_nickname_or_hexdigest** (const char ∗s)
- int **is_legal_hexdigest** (const char ∗s)
- void **router_get_verbose_nickname** (char ∗buf, const **routerinfo_t** ∗router)
- void **routerstatus_get_verbose_nickname** (char ∗buf, const **routerstatus_t** ∗router)
- void **router_reset_warnings** (void)
- const char ∗ **router_purpose_to_string** (uint8_t p)
- uint8_t **router_purpose_from_string** (const char ∗s)
- void **router_free_all** (void)

Variables

- long **stats_n_seconds_working**
- static **tor_mutex_t** ∗ **key_lock** = NULL
- static time_t **onionkey_set_at** = 0
- static **crypto_pk_env_t** ∗ **onionkey** = NULL
- static **crypto_pk_env_t** ∗ **lastonionkey** = NULL
- static **crypto_pk_env_t** ∗ **identitykey** = NULL
- static char **identitykey_digest** [DIGEST_LEN]
- static **crypto_pk_env_t** ∗ **authority_signing_key** = NULL
- static **authority_cert_t** ∗ **authority_key_certificate** = NULL
- static **crypto_pk_env_t** ∗ **legacy_signing_key** = NULL
- static **authority_cert_t** ∗ **legacy_key_certificate** = NULL
- static int **can_reach_or_port** = 0
- static int **can_reach_dir_port** = 0
- static int **server_is_advertised** = 0
- static **routerinfo_t** ∗ **desc_routerinfo** = NULL
- static **extrainfo_t** ∗ **desc_extrainfo** = NULL
- static time_t **desc_clean_since** = 0
- static int **desc_needs_upload** = 0
- static **smartlist_t** ∗ **warned_nonexistent_family** = NULL
- static uint32_t **last_guessed_ip** = 0

4.44.1 Detailed Description

OR functionality, including key maintenance, generating and uploading server descriptors, retrying OR connections.

4.44.2 Define Documentation

4.44.2.1 #define MAX_BANDWIDTH_CHANGE_FREQ (20∗60)

How frequently will we republish our descriptor because of large (factor of 2) shifts in estimated bandwidth?

Referenced by check_descriptor_bandwidth_changed().

4.44.3 Function Documentation

4.44.3.1 int advertised_server_mode (void)

Return true iff we have published our descriptor lately.

References server_is_advertised.

Referenced by directory_caches_dir_info(), directory_fetches_from_authorities(), and run_scheduled_events().

4.44.3.2 int authdir_mode (or_options_t ∗ *options*)

Return true iff we believe ourselves to be an authoritative directory server.

References or_options_t::AuthoritativeDir.

Referenced by authdir_mode_bridge(), authdir_mode_v1(), authdir_mode_v2(), authdir_mode_v3(), decide_if_publishable_server(), decide_to_advertise_dirport(), directory_handle_command_post(), global_write_bucket_low(), init_keys(), options_validate(), rep_history_clean(), router_digest_is_trusted_dir_type(), router_rebuild_descriptor(), update_consensus_router_descriptor_downloads(), and update_router_descriptor_cache_downloads_v2().

4.44.3.3 int authdir_mode_any_main (or_options_t ∗ *options*)

Return true iff we are a v1, v2, or v3 directory authority.

References or_options_t::V1AuthoritativeDir, or_options_t::V2AuthoritativeDir, and or_options_t::V3AuthoritativeDir.

Referenced by authdir_mode_any_nonhidserv(), authdir_mode_handles_descs(), and options_validate().

4.44.3.4 int authdir_mode_any_nonhidserv (or_options_t ∗ *options*)

Return true if we believe ourselves to be any kind of authoritative directory beyond just a hidserv authority.

References authdir_mode_any_main(), and or_options_t::BridgeAuthoritativeDir.

Referenced by authdir_mode_handles_descs(), authdir_mode_publishes_statuses(), and launch_router_descriptor_downloads().

4.44.3.5 int authdir_mode_bridge (or_options_t ∗ *options*)

Return true iff we believe ourselves to be a bridge authoritative directory server.

References authdir_mode(), and or_options_t::BridgeAuthoritativeDir.

Referenced by authdir_mode_publishes_statuses(), directory_handle_command_post(), dirserv_orconn_tls_done(), dirserv_pick_cached_dir_obj(), dirserv_test_reachability(), and networkstatus_getinfo_by_purpose().

4.44.3.6 int authdir_mode_handles_descs (or_options_t ∗ *options*, int *purpose*)

Return true iff we are an authoritative directory server that is authoritative about receiving and serving descriptors of type **purpose** its dirport. Use -1 for "any purpose".

References authdir_mode_any_main(), authdir_mode_any_nonhidserv(), or_options_t::BridgeAuthoritativeDir, ROUTER_PURPOSE_BRIDGE, and ROUTER_PURPOSE_GENERAL.

Referenced by authdir_mode_tests_reachability(), directory_handle_command_post(), do_hup(), and router_add_to_routerlist().

4.44.3.7 int authdir_mode_publishes_statuses (or_options_t ∗ *options*)

Return true iff we are an authoritative directory server that publishes its own network statuses.

References authdir_mode_any_nonhidserv(), and authdir_mode_bridge().

Referenced by list_server_status_v1().

4.44.3.8 int authdir_mode_tests_reachability (or_options_t ∗ *options*)

Return true iff we are an authoritative directory server that tests reachability of the descriptors it learns about.

References authdir_mode_handles_descs().

Referenced by connection_about_to_close_connection(), connection_or_check_valid_tls_handshake(), connection_or_connect_failed(), directory_handle_command_get(), run_scheduled_events(), and tor_cleanup().

4.44.3.9 int authdir_mode_v1 (or_options_t ∗ *options*)

Return true iff we believe ourselves to be a v1 authoritative directory server.

References authdir_mode(), and or_options_t::V1AuthoritativeDir.

Referenced by dirserv_pick_cached_dir_obj(), router_parse_directory(), and router_parse_runningrouters().

4.44.3.10 int authdir_mode_v2 (or_options_t ∗ *options*)

Return true iff we believe ourselves to be a v2 authoritative directory server.

References authdir_mode(), and or_options_t::V2AuthoritativeDir.

Referenced by dirserv_get_networkstatus_v2_fingerprints(), dirserv_pick_cached_dir_obj(), routers_update_status_from_consensus_networkstatus(), should_generate_v2_networkstatus(), and update_v2_networkstatus_cache_downloads().

4.44.3.11 int authdir_mode_v3 (or_options_t ∗ *options*)

Return true iff we believe ourselves to be a v3 authoritative directory server.

References authdir_mode(), and or_options_t::V3AuthoritativeDir.

Referenced by directory_handle_command_post(), dirvote_act(), dirvote_recalculate_timing(), init_keys(), options_act(), routers_update_status_from_consensus_networkstatus(), run_scheduled_events(), update_consensus_networkstatus_downloads(), and v3_authority_check_key_expiry().

4.44.3.12 void check_descriptor_bandwidth_changed (time_t *now*)

Check whether bandwidth has changed a lot since the last time we announced bandwidth. If so, mark our descriptor dirty.

References routerinfo_t::bandwidthcapacity, LD_GENERAL, mark_my_descriptor_dirty(), MAX_BANDWIDTH_CHANGE_FREQ, rep_hist_bandwidth_assess(), and we_are_hibernating().

Referenced by run_scheduled_events().

4.44.3.13 void check_descriptor_ipaddress_changed (time_t *now*)

Check whether our own address as defined by the Address configuration has changed. This is for routers that get their address from a service like dyndns. If our address has changed, mark our descriptor dirty.

References routerinfo_t::addr, get_options(), ip_address_changed(), LD_CONFIG, log_addr_has_changed(), and resolve_my_address().

Referenced by run_scheduled_events().

4.44.3.14 int check_whether_dirport_reachable (void)

Return 1 if we don't have a dirport configured, or if it's reachable.

References or_options_t::AssumeReachable, can_reach_dir_port, or_options_t::DirPort, get_options(), and we_are_hibernating().

Referenced by consider_testing_reachability(), decide_to_advertise_dirport(), getinfo_helper_events(), rep_hist_circbuilding_dormant(), and second_elapsed_callback().

4.44.3.15 int check_whether_orport_reachable (void)

Return 1 if ORPort is known reachable; else return 0.

References or_options_t::AssumeReachable, can_reach_or_port, and get_options().

Referenced by circuit_send_next_onion_skin(), circuit_testing_failed(), circuit_testing_opened(), consider_testing_reachability(), decide_if_publishable_server(), getinfo_helper_events(), rep_hist_circbuilding_dormant(), and second_elapsed_callback().

4.44.3.16 void consider_publishable_server (int *force*)

Initiate server descriptor upload as reasonable (if server is publishable, etc). **force** is as for router_upload_dir_desc_to_dirservers.

We need to rebuild the descriptor if it's dirty even if we're not uploading, because our reachability testing ∗uses∗ our descriptor to determine what IP address and ports to test.

References decide_if_publishable_server(), get_options(), router_rebuild_descriptor(), router_upload_dir_desc_to_dirservers(), server_mode(), and set_server_advertised().

Referenced by run_scheduled_events().

4.44.3.17 void consider_testing_reachability (int *test_or*, int *test_dir*)

Some time has passed, or we just got new directory information. See if we currently believe our ORPort or DirPort to be unreachable. If so, launch a new test for it.

For ORPort, we simply try making a circuit that ends at ourselves. Success is noticed in **onionskin_answer()** (p. 324).

For DirPort, we make a connection via Tor to our DirPort and ask for our own server descriptor. Success is noticed in **connection_dir_client_reached_eof()** (p. 569).

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::cache_info, check_whether_dirport_reachable(), check_whether_orport_reachable(), CIRCLAUNCH_IS_INTERNAL, CIRCLAUNCH_NEED_CAPACITY, circuit_enough_testing_circs(), circuit_launch_by_router(), CIRCUIT_PURPOSE_TESTING, CONN_TYPE_DIR, connection_get_by_type_addr_port_purpose(), routerinfo_t::dir_port, DIR_PURPOSE_FETCH_SERVERDESC, directory_initiate_command(), signed_descriptor_t::identity_digest, LD_CIRC, routerinfo_t::or_port, router_get_my_routerinfo(), ROUTER_PURPOSE_GENERAL, and tor_addr_from_ipv4h.

Referenced by circuit_send_next_onion_skin(), circuit_testing_opened(), directory_info_has_arrived(), and run_scheduled_events().

4.44.3.18 static int decide_if_publishable_server (void) `[static]`

Decide if we're a publishable server. We are a publishable server if:

- We don't have the ClientOnly option set and
- We have the PublishServerDescriptor option set to non-empty and
- We have ORPort set and
- We believe we are reachable from the outside; or
- We are an authoritative directory server.

References or_options_t::_PublishServerDescriptor, authdir_mode(), check_whether_orport_reachable(), or_options_t::ClientOnly, get_options(), and server_mode().

Referenced by consider_publishable_server().

4.44.3.19 static int decide_to_advertise_dirport (or_options_t * *options*, uint16_t *dir_port*) `[static]`

Look at a variety of factors, and return 0 if we don't want to advertise the fact that we have a DirPort open. Else return the DirPort we want to advertise.

Log a helpful message if we change our mind about whether to publish a DirPort.

References accounting_is_enabled(), authdir_mode(), or_options_t::BandwidthRate, check_whether_dirport_reachable(), LD_DIR, or_options_t::RelayBandwidthRate, tor_assert, and we_are_hibernating().

Referenced by router_dirport_found_reachable(), and router_dump_router_to_string().

4.44.3.20 void dup_onion_keys (crypto_pk_env_t ** *key*, crypto_pk_env_t ** *last*)

Store a full copy of the current onion key into *$*$***key**, and a full copy of the most recent onion key into *$*$***last**.

References crypto_pk_copy_full(), tor_assert, tor_mutex_acquire(), and tor_mutex_release().

Referenced by cpuworker_main().

4.44.3.21 int extrainfo_dump_to_string (char * *s*, size_t *maxlen*, extrainfo_t * *extrainfo*, crypto_pk_env_t * *ident_key*)

Write the contents of **extrainfo** to the **maxlen**-byte string **s**, signing them with **ident_key**. Return 0 on success, negative on failure.

References base16_encode(), extrainfo_t::cache_info, or_options_t::CellStatistics, DIGEST_LEN, or_options_t::DirReqStatistics, or_options_t::EntryStatistics, or_options_t::ExitPortStatistics, extrainfo_dump_to_string(), extrainfo_free(), extrainfo_parse_entry_from_string(), or_options_t::ExtraInfoStatistics, format_iso_time(), geoip_get_bridge_stats_extrainfo(), get_options(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, LD_BUG, LD_DIR, LD_GENERAL, load_stats_file(), extrainfo_t::nickname, signed_descriptor_t::published_on, rep_hist_get_bandwidth_lines(), router_append_dirobj_signature(), router_get_extrainfo_hash(), should_record_bridge_info(), tor_free, and tor_snprintf().

Referenced by extrainfo_dump_to_string(), and router_rebuild_descriptor().

4.44.3.22 crypto_pk_env_t* get_identity_key (void)

Returns the current identity key; requires that the identity key has been set.

References tor_assert.

Referenced by accounting_set_wakeup_time(), connection_or_check_valid_tls_handshake(), dirserv_regenerate_directory(), do_list_fingerprint(), generate_runningrouters(), generate_v2_networkstatus_opinion(), init_keys(), router_rebuild_descriptor(), and run_scheduled_events().

4.44.3.23 authority_cert_t* get_my_v3_authority_cert (void)

Return the key certificate for this v3 (voting) authority, or NULL if we have no such certificate.

Referenced by authority_cert_get_by_sk_digest(), directory_handle_command_get(), dirvote_act(), dirvote_compute_consensuses(), dirvote_get_vote(), dirvote_perform_vote(), and init_keys().

4.44.3.24 crypto_pk_env_t* get_my_v3_authority_signing_key (void)

Return the v3 signing key for this v3 (voting) authority, or NULL if we have no such key.

Referenced by dirvote_compute_consensuses(), and dirvote_perform_vote().

4.44.3.25 authority_cert_t* get_my_v3_legacy_cert (void)

If we're an authority, and we're using a legacy authority identity key for emergency migration purposes, return the certificate associated with that key.

Referenced by authority_cert_get_by_sk_digest(), and dirvote_compute_consensuses().

4.44.3.26 crypto_pk_env_t* get_my_v3_legacy_signing_key (void)

If we're an authority, and we're using a legacy authority identity key for emergency migration purposes, return that key.

Referenced by dirvote_compute_consensuses().

4.44.3.27 crypto_pk_env_t∗ get_onion_key (void)

Return the current onion key. Requires that the onion key has been loaded or generated.

References tor_assert.

Referenced by router_rebuild_descriptor().

4.44.3.28 time_t get_onion_key_set_at (void)

Return the time when the onion key was last set. This is either the time when the process launched, or the time of the most recent key rotation since the process launched.

References onionkey_set_at.

Referenced by run_scheduled_events().

4.44.3.29 void get_platform_str (char ∗ *platform*, size_t *len*)

Set **platform** (max length **len**) to a NUL-terminated short string describing the version of Tor and the operating system we're currently running on.

References get_uname(), get_version(), and tor_snprintf().

Referenced by router_rebuild_descriptor().

4.44.3.30 int identity_key_is_set (void)

Return true iff the identity key has been set.

Referenced by accounting_set_wakeup_time(), and do_main_loop().

4.44.3.31 crypto_pk_env_t∗ init_key_from_file (const char ∗ *fname*, int *generate*, int *severity*)

Try to read an RSA key from **fname**. If **fname** doesn't exist and **generate** is true, create a new RSA key and save it in **fname**. Return the read/created key, or NULL on error. Log all errors at level **severity**.

References crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_check_key(), crypto_pk_read_private_key_from_filename(), crypto_pk_write_private_key_to_filename(), file_status(), get_options(), have_lockfile(), LD_FS, LD_GENERAL, tor_assert, and try_locking().

Referenced by init_keys(), load_authority_keyset(), and rend_service_load_keys().

4.44.3.32 int init_keys (void)

Initialize all OR private keys, and the TLS context, as necessary. On OPs, this only initializes the tls context. Return 0 on success, or -1 if Tor should die.

References add_trusted_dir_server(), authdir_mode(), authdir_mode_v3(), or_options_t::AvoidDiskWrites, BRIDGE_AUTHORITY, or_options_t::BridgeAuthoritativeDir, authority_cert_t::cache_info, check_private_dir(), or_options_t::command, crypto_free_pk_env(), crypto_global_init(), crypto_new_pk_env(), crypto_pk_get_digest(), crypto_pk_get_fingerprint(), or_options_t::DataDirectory, DIGEST_LEN, or_options_t::DirPort, dirserv_add_descriptor(), dirserv_add_own_fingerprint(), dirserv_load_fingerprint_file(), file_status(), FINGERPRINT_LEN, get_datadir_fname, get_datadir_fname2, get_identity_key(), get_my_v3_authority_cert(), get_options(), get_or_state(), HIDSERV_AUTHORITY, or_options_t::HSAuthoritativeDir, init_key_from_file(), init_v3_authority_keys(), or_state_t::LastRotatedOnionKey, LD_BUG, LD_DIR, LD_FS, LD_GENERAL, MAX_NICKNAME_LEN, MAX_SSL_KEY_LIFETIME, or_options_t::Nickname, onionkey_set_at, or_state_mark_dirty(), or_options_t::ORPort, read_file_to_str(), router_get_my_descriptor(), router_get_trusteddirserver_by_digest(), router_parse_entry_from_string(), server_mode(), set_identity_key(), set_onion_key(), signed_descriptor_t::signed_descriptor_body, tor_assert, tor_free, tor_mutex_new(), tor_snprintf(), tor_tls_context_new(), trusted_dirs_load_certs_from_string(), trusted_dir_server_t::type, V1_AUTHORITY, or_options_t::V1AuthoritativeDir, V2_AUTHORITY, or_options_t::V2AuthoritativeDir, V3_AUTHORITY, trusted_dir_server_t::v3_identity_digest, or_options_t::V3AuthoritativeDir, WRA_WAS_ADDED(), and write_str_to_file().

Referenced by accounting_set_wakeup_time(), do_list_fingerprint(), do_main_loop(), ip_address_changed(), and options_act().

4.44.3.33 static int init_v3_authority_keys (void) `[static]`

Load the v3 (voting) authority signing key and certificate, if they are present. Return -1 if anything is missing, mismatched, or unloadable; return 0 on success.

References get_options(), and load_authority_keyset().

Referenced by init_keys().

4.44.3.34 int is_legal_hexdigest (const char * *s*)

Return true iff **s** is a legally valid hex-encoded identity-key digest.

References HEX_CHARACTERS, HEX_DIGEST_LEN, is_legal_nickname(), and tor_assert.

Referenced by is_legal_nickname_or_hexdigest(), router_rebuild_descriptor(), and routerset_parse().

4.44.3.35 int is_legal_nickname (const char * *s*)

Return true iff **s** is a legally valid server nickname.

References LEGAL_NICKNAME_CHARACTERS, MAX_NICKNAME_LEN, and tor_assert.

Referenced by circuit_list_path_impl(), dirserv_load_fingerprint_file(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), is_legal_hexdigest(), is_legal_nickname_or_hexdigest(), options_validate(), parse_dir_server_line(), rend_service_introduce(), router_parse_entry_from_string(), routerset_parse(), and routerstatus_parse_entry_from_string().

4.44.3.36 int is_legal_nickname_or_hexdigest (const char * *s*)

Return true iff **s** is a legally valid server nickname or hex-encoded identity-key digest.

References is_legal_hexdigest(), and is_legal_nickname().

Referenced by add_nickname_list_to_smartlist(), check_nickname_list(), microdescs_parse_from_string(), rend_service_introduce(), router_parse_entry_from_string(), and router_rebuild_descriptor().

4.44.3.37 static int load_authority_keyset (int *legacy*, crypto_pk_env_t ** *key_out*, authority_cert_t ** *cert_out*) `[static]`

Try to load the vote-signing private key and certificate for being a v3 directory authority, and make sure they match. If **legacy**, load a legacy key/cert set for emergency key migration; otherwise load the regular key/cert set. On success, store them into ∗**key_out** and ∗**cert_out** respectively, and return 0. On failure, return -1.

References authority_cert_free(), authority_cert_parse_from_string(), crypto_free_pk_env(), crypto_pk_cmp_keys(), get_datadir_fname2, init_key_from_file(), LD_DIR, read_file_to_str(), authority_cert_t::signing_key, and tor_free.

Referenced by init_v3_authority_keys().

4.44.3.38 static int load_stats_file (const char * *filename*, const char * *end_line*, time_t *now*, char ** *out*) `[static]`

Load the contents of **filename**, find the last line starting with **end_line**, ensure that its timestamp is not more than 25 hours in the past or more than 1 hour in the future with respect to **now**, and write the file contents starting with that line to ∗**out**. Return 1 for success, 0 if the file does not exist, or -1 if the file does not contain a line matching these criteria or other failure.

References file_status(), get_datadir_fname, parse_iso_time(), read_file_to_str(), and tor_free.

Referenced by extrainfo_dump_to_string().

4.44.3.39 static void log_addr_has_changed (int *severity*, uint32_t *prev*, uint32_t *cur*, const char * *source*) `[static]`

Note at log level severity that our best guess of address has changed from **prev** to **cur**.

References INET_NTOA_BUF_LEN, LD_GENERAL, and tor_inet_ntoa().

Referenced by check_descriptor_ipaddress_changed(), and router_new_address_suggestion().

4.44.3.40 void mark_my_descriptor_dirty (void)

Call when the current descriptor is out of date.

References desc_clean_since.

Referenced by add_wildcarded_test_address(), check_descriptor_bandwidth_changed(), configure_nameservers(), ip_address_changed(), mark_my_descriptor_dirty_if_older_than(), options_act(), rotate_onion_key(), router_dirport_found_reachable(), router_orport_found_reachable(), and set_onion_key().

4.44.3.41 void mark_my_descriptor_dirty_if_older_than (time_t *when*)

Mark descriptor out of date if it's older than **when**

References desc_clean_since, and mark_my_descriptor_dirty().

Referenced by run_scheduled_events().

4.44.3.42 int proxy_mode (or_options_t * *options*)

Return true iff we are trying to be a socks proxy.

References or_options_t::DNSListenAddress, or_options_t::DNSPort, or_options_t::NatdListenAddress, or_options_t::NatdPort, or_options_t::SocksListenAddress, or_options_t::SocksPort, or_options_t::TransListenAddress, and or_options_t::TransPort.

Referenced by circuit_build_needed_circs().

4.44.3.43 void rotate_onion_key (void)

Replace the previous onion key with the current onion key, and generate a new previous onion key. Immediately after calling this function, the OR should:

- schedule all previous cpuworkers to shut down _after_ processing pending work. (This will cause fresh cpuworkers to be generated.)
- generate and upload a fresh routerinfo.

References crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_write_private_key_to_filename(), file_status(), get_datadir_fname2, get_options(), get_or_state(), or_state_t::LastRotatedOnionKey, LD_BUG, LD_FS, LD_GENERAL, mark_my_descriptor_dirty(), onionkey_set_at, or_state_mark_dirty(), replace_file(), tor_free, tor_mutex_acquire(), and tor_mutex_release().

Referenced by run_scheduled_events().

4.44.3.44 int router_compare_to_my_exit_policy (edge_connection_t * *conn*)

OR only: Check whether my exit policy says to allow connection to conn. Return 0 if we accept; non-0 if we reject.

References edge_connection_t::_base, connection_t::addr, ADDR_POLICY_ACCEPTED, compare_tor_addr_to_addr_policy(), routerinfo_t::exit_policy, connection_t::port, router_get_my_routerinfo(), tor_addr_family(), and tor_addr_is_null().

Referenced by connection_exit_connect().

4.44.3.45 int router_digest_is_me (const char * *digest*)

Return true iff I'm a server and **digest** is equal to my identity digest.

References DIGEST_LEN, and identitykey_digest.

Referenced by connection_or_connect(), directory_conn_is_self_reachability_test(), directory_get_from_all_authorities(), dirserv_get_networkstatus_v2(), dirserv_get_routerdescs(), get_signed_descriptor_by_fp(), lookup_cached_dir_by_fp(), router_digest_is_trusted_dir_type(), router_fingerprint_is_me(), router_is_me(), router_pick_directory_server_impl(), router_pick_trusteddirserver_impl(), router_set_networkstatus_v2(), smartlist_choose_by_bandwidth(), and update_v2_networkstatus_cache_downloads().

4.44.3.46 void router_dirport_found_reachable (void)

Annotate that we found our DirPort reachable.

References routerinfo_t::address, can_reach_dir_port, control_event_server_status(), decide_to_advertise_dirport(), routerinfo_t::dir_port, get_options(), LD_BUG, LD_DIRSERV, mark_my_descriptor_dirty(), and router_get_my_routerinfo().

Referenced by connection_dir_client_reached_eof().

4.44.3.47 int router_dump_router_to_string (char * *s*, size_t *maxlen*, routerinfo_t * *router*, crypto_pk_env_t * *ident_key*)

OR only: Given a routerinfo for this router, and an identity key to sign with, encode the routerinfo as a signed server descriptor and write the result into **s**, using at most **maxlen** bytes. Return -1 on failure, and the number of bytes used on success.

References routerinfo_t::address, or_options_t::AllowSingleHopExits, routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, base16_encode(), routerinfo_t::cache_info, or_options_t::ContactInfo, crypto_pk_cmp_keys(), crypto_pk_get_fingerprint(), crypto_pk_write_public_key_to_string(), decide_to_advertise_dirport(), routerinfo_t::declared_family, DIGEST_LEN, routerinfo_t::dir_port, dns_seems_to_be_broken(), or_options_t::DownloadExtraInfo, escaped(), routerinfo_t::exit_policy, signed_descriptor_t::extra_info_digest, FINGERPRINT_LEN, format_iso_time(), get_options(), has_dns_init_failed(), HEX_DIGEST_LEN, or_options_t::HidServDirectoryV2, routerinfo_t::identity_pkey, LD_BUG, routerinfo_t::nickname, note_crypto_pk_op(), routerinfo_t::onion_pkey, routerinfo_t::or_port, routerinfo_t::platform, policy_write_item(), signed_descriptor_t::published_on, router_append_dirobj_signature(), router_get_router_hash(), router_parse_entry_from_string(), routerinfo_free(), smartlist_join_strings(), stats_n_seconds_working, tor_assert, tor_free, tor_snprintf(), and we_are_hibernating().

Referenced by router_rebuild_descriptor().

4.44.3.48 int router_extrainfo_digest_is_me (const char * *digest*)

Return true iff I'm a server and **digest** is equal to my identity digest.

References extrainfo_t::cache_info, DIGEST_LEN, router_get_my_extrainfo(), and signed_descriptor_t::signed_descriptor_digest.

4.44.3.49 int router_fingerprint_is_me (const char * *fp*)

Return true iff **fp** is a hex fingerprint of my identity digest.

References base16_decode(), DIGEST_LEN, HEX_DIGEST_LEN, and router_digest_is_me().

4.44.3.50 void router_free_all (void)

Release all static resources held in **router.c** (p. 1117)

References authority_cert_free(), crypto_free_pk_env(), extrainfo_free(), routerinfo_free(), smartlist_free(), tor_free, and tor_mutex_free().

Referenced by tor_free_all().

4.44.3.51 const char∗ router_get_my_descriptor (void)

OR only: Return a signed server descriptor for this OR, rebuilding a fresh one if necessary. Return NULL on error.

References routerinfo_t::cache_info, LD_GENERAL, router_get_my_routerinfo(), signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_get_body(), signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by init_keys().

4.44.3.52 extrainfo_t∗ router_get_my_extrainfo (void)

Return the extrainfo document for this OR, or NULL if we have none. Rebuilt it (and the server descriptor) if necessary.

References get_options(), router_rebuild_descriptor(), and server_mode().

Referenced by extrainfo_insert(), get_signed_descriptor_by_fp(), router_extrainfo_digest_is_me(), and router_upload_dir_desc_to_dirservers().

4.44.3.53 routerinfo_t∗ router_get_my_routerinfo (void)

Return a routerinfo for this OR, rebuilding a fresh one if necessary. Return NULL on error, or if called on an OP.

References get_options(), router_rebuild_descriptor(), and server_mode().

Referenced by circuit_rep_hist_note_result(), connection_or_send_netinfo(), consider_testing_reachability(), directory_conn_is_self_reachability_test(), directory_fetches_from_authorities(), dirserv_estimate_data_size(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dns_resolve_impl(), get_signed_descriptor_by_fp(), getinfo_helper_misc(), hid_serv_acting_as_directory(), hid_serv_responsible_for_desc_id(), inform_testing_reachability(), router_compare_to_my_exit_policy(), router_dirport_found_reachable(), router_get_by_nickname(), router_get_my_descriptor(), router_get_my_share_of_directory_requests(), router_orport_found_reachable(), router_pick_trusteddirserver_impl(), router_upload_dir_desc_to_dirservers(), routerlist_insert(), routerlist_insert_old(), routerlist_replace(), run_scheduled_events(), and second_elapsed_callback().

4.44.3.54 void router_get_verbose_nickname (char ∗ *buf*, const routerinfo_t ∗ *router*)

Set **buf** (which must have MAX_VERBOSE_NICKNAME_LEN+1 bytes) to the verbose representation of the identity of **router**. The format is: A dollar sign. The upper-case hexadecimal encoding of the SHA1 hash of router's identity. A "=" if the router is named; a "~" if it is not. The router's nickname.

References base16_encode(), routerinfo_t::cache_info, DIGEST_LEN, HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::is_named, MAX_NICKNAME_LEN, and routerinfo_t::nickname.

Referenced by circuit_list_path_impl(), control_event_descriptors_changed(), control_event_guard(), getinfo_helper_entry_guards(), list_server_status_v1(), and orconn_target_get_name().

4.44.3.55 static int router_guess_address_from_dir_headers (uint32_t * *guess*) [static]

We failed to resolve our address locally, but we'd like to build a descriptor and publish / test reachability. If we have a guess about our address based on directory headers, answer it and return 0; else return -1.

References last_guessed_ip.

Referenced by router_pick_published_address().

4.44.3.56 int router_is_me (routerinfo_t * *router*)

A wrapper around **router_digest_is_me()** (p. 988).

References routerinfo_t::cache_info, signed_descriptor_t::identity_digest, and router_digest_is_me().

Referenced by choose_good_exit_server_general(), dirserv_add_descriptor(), dirserv_set_router_is_running(), dirserv_test_reachability(), router_load_single_router(), router_rebuild_descriptor(), router_set_status(), and routerlist_find_my_routerinfo().

4.44.3.57 void router_new_address_suggestion (const char * *suggestion*, const dir_connection_t * *d_conn*)

A directory server **d_conn** told us our IP address is **suggestion**. If this address is different from the one we think we are now, and if our computer doesn't actually know its IP address, then switch.

References dir_connection_t::_base, connection_t::addr, connection_t::address, control_event_server_status(), escaped(), get_options(), ip_address_changed(), is_internal_IP(), last_guessed_ip, LD_DIR, log_addr_has_changed(), resolve_my_address(), server_mode(), tor_addr_eq_ipv4h(), and tor_inet_aton().

Referenced by connection_dir_client_reached_eof().

4.44.3.58 void router_orport_found_reachable (void)

Annotate that we found our ORPort reachable.

References routerinfo_t::address, can_reach_or_port, control_event_server_status(), get_options(), LD_BUG, LD_OR, mark_my_descriptor_dirty(), routerinfo_t::or_port, and router_get_my_routerinfo().

Referenced by onionskin_answer().

4.44.3.59 void router_perform_bandwidth_test (int *num_circs*, time_t *now*)

We have enough testing circuits open. Send a bunch of "drop" cells down each of them, to exercise our bandwidth.

References origin_circuit_t::_base, or_options_t::BandwidthRate, CELL_NETWORK_SIZE, circuit_get_next_by_pk_and_purpose(), CIRCUIT_PURPOSE_TESTING, CIRCUIT_STATE_OPEN, CIRCWINDOW_START, origin_circuit_t::cpath, get_options(), LD_OR, crypt_path_t::prev, relay_send_command_from_edge(), circuit_t::state, circuit_t::timestamp_dirty, and TO_CIRCUIT.

Referenced by circuit_testing_opened().

4.44.3.60 int router_pick_published_address (or_options_t ∗ *options*, uint32_t ∗ *addr*)

Make a current best guess at our address, either because it's configured in torrc, or because we've learned it from dirserver headers. Place the answer in ∗**addr** and return 0 on success, else return -1 if we have no guess.

References LD_CONFIG, resolve_my_address(), and router_guess_address_from_dir_headers().

Referenced by directory_fetches_from_authorities(), getinfo_helper_misc(), and router_rebuild_descriptor().

4.44.3.61 uint8_t router_purpose_from_string (const char ∗ *s*)

Given a string, convert it to a router purpose.

References ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_CONTROLLER, ROUTER_PURPOSE_GENERAL, and ROUTER_PURPOSE_UNKNOWN.

Referenced by handle_control_postdescriptor(), networkstatus_getinfo_by_purpose(), and router_parse_entry_from_string().

4.44.3.62 const char∗ router_purpose_to_string (uint8_t *p*)

Given a router purpose, convert it to a string. Don't call this on ROUTER_PURPOSE_UNKNOWN: The whole point of that value is that we don't know its string representation.

References ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_CONTROLLER, ROUTER_PURPOSE_GENERAL, and tor_assert.

Referenced by dirserv_add_multiple_descriptors(), load_downloaded_routers(), router_load_single_router(), and router_parse_list_from_string().

4.44.3.63 int router_rebuild_descriptor (int *force*)

If **force** is true, or our descriptor is out-of-date, rebuild a fresh routerinfo, signed server descriptor, and extra-info document for this OR. Return 0 on success, -1 on temporary error.

References routerinfo_t::addr, routerinfo_t::address, authdir_mode(), routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, base16_encode(), or_options_t::BridgeRelay, extrainfo_t::cache_info, routerinfo_t::cache_info, control_event_my_descriptor_changed(), crypto_pk_dup_key(), crypto_pk_get_digest(), routerinfo_t::declared_family, desc_clean_since, desc_needs_upload, DIGEST_LEN, routerinfo_t::dir_port, or_options_t::DirPort, escaped(), routerinfo_t::exit_policy, or_options_t::ExitPolicy, or_options_t::ExitPolicyRejectPrivate, signed_descriptor_t::extra_info_digest, extrainfo_dump_to_string(), extrainfo_free(), or_options_t::ExtraInfoStatistics, get_effective_bwburst(), get_effective_bwrate(), get_identity_key(), get_onion_key(), get_options(), get_platform_str(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, signed_descriptor_t::is_extrainfo, is_legal_hexdigest(), is_legal_nickname_or_hexdigest(), routerinfo_t::is_named, routerinfo_t::is_running, routerinfo_t::is_valid, LD_BUG, LD_CONFIG, MAX_EXTRAINFO_UPLOAD_SIZE, or_options_t::MyFamily, extrainfo_t::nickname, or_options_t::Nickname, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::or_port, or_options_t::ORPort, routerinfo_t::platform, policies_parse_exit_policy(), signed_descriptor_t::published_on, routerinfo_t::purpose, rep_hist_bandwidth_assess(), router_dump_router_to_string(), router_get_by_nickname(), router_get_extrainfo_hash(), router_get_router_hash(), router_is_me(), router_pick_published_address(), ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerinfo_incompatible_with_extrainfo(), routerinfo_set_country(), signed_descriptor_t::routerlist_index, signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_sort_strings(), smartlist_split_string(), smartlist_string_isin(), smartlist_string_remove(), smartlist_uniq_strings(), tor_assert, tor_dup_ip(), tor_free, and we_are_hibernating().

Referenced by consider_publishable_server(), router_get_my_extrainfo(), router_get_my_routerinfo(), and run_scheduled_events().

4.44.3.64 void router_reset_reachability (void)

Forget what we have learned about our reachability status.

References can_reach_dir_port, and can_reach_or_port.

Referenced by ip_address_changed().

4.44.3.65 void router_reset_warnings (void)

Forget that we have issued any router-related warnings, so that we'll warn again if we see the same errors.

References smartlist_clear(), and tor_free.

Referenced by do_hup().

4.44.3.66 void router_upload_dir_desc_to_dirservers (int *force*)

OR only: If **force** is true, or we haven't uploaded this descriptor successfully yet, try to upload our signed descriptor to all the directory servers we know about.

References or_options_t::_PublishServerDescriptor, BRIDGE_AUTHORITY, extrainfo_t::cache_info, routerinfo_t::cache_info, desc_needs_upload, DIR_PURPOSE_UPLOAD_DIR, directory_post_to_dirservers(), get_options(), LD_GENERAL, router_get_my_extrainfo(), router_get_my_routerinfo(), ROUTER_PURPOSE_BRIDGE, ROUTER_PURPOSE_GENERAL, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, and tor_free.

Referenced by consider_publishable_server(), and run_scheduled_events().

4.44.3.67 void routerstatus_get_verbose_nickname (char ∗ *buf*, const routerstatus_t ∗ *router*)

Set **buf** (which must have MAX_VERBOSE_NICKNAME_LEN+1 bytes) to the verbose representation of the identity of **router**. The format is: A dollar sign. The upper-case hexadecimal encoding of the SHA1 hash of router's identity. A "=" if the router is named; a "∼" if it is not. The router's nickname.

References base16_encode(), DIGEST_LEN, HEX_DIGEST_LEN, routerstatus_t::identity_digest, routerstatus_t::is_named, MAX_NICKNAME_LEN, and routerstatus_t::nickname.

Referenced by circuit_list_path_impl().

4.44.3.68 int server_mode (or_options_t ∗ *options*)

Return true iff we are trying to be a server.

References or_options_t::ClientOnly, or_options_t::ORListenAddress, and or_options_t::ORPort.

Referenced by circuit_extend(), circuit_note_clock_jumped(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), circuit_testing_failed(), command_process_create_cell(), conn_close_if_marked(), connection_connect(), connection_exit_begin_conn(), connection_or_check_valid_tls_handshake(), connection_or_connect(), consider_publishable_server(), cpuworkers_rotate(), decide_if_publishable_server(), directory_caches_dir_info(), directory_command_should_use_begindir(), directory_fetches_from_authorities(), directory_info_has_arrived(), dns_init(), dns_reset(), dns_servers_relaunch_checks(), do_hup(), do_list_fingerprint(), do_main_loop(), getinfo_helper_events(), init_keys(), ip_address_changed(), options_act(), options_validate(), rep_hist_circbuilding_dormant(), rep_hist_update_state(), router_get_by_nickname(), router_get_my_extrainfo(), router_get_my_routerinfo(), router_new_address_suggestion(), routers_update_all_from_networkstatus(), run_scheduled_events(), second_elapsed_callback(), should_use_create_fast_for_circuit(), signal_callback(), and update_router_descriptor_downloads().

4.44.3.69 void set_identity_key (crypto_pk_env_t ∗ *k*)

Set the current identity key to k.

References crypto_free_pk_env(), crypto_pk_get_digest(), and identitykey_digest.

Referenced by init_keys().

4.44.3.70 static void set_onion_key (crypto_pk_env_t ∗ *k*) `[static]`

Replace the current onion key with **k**. Does not affect lastonionkey; to update lastonionkey correctly, call **rotate_onion_key()** (p. 985).

References crypto_free_pk_env(), mark_my_descriptor_dirty(), onionkey_set_at, tor_mutex_acquire(), and tor_mutex_release().

Referenced by init_keys().

4.44.3.71 static void set_server_advertised (int *s*) `[static]`

Called with a boolean: set whether we have recently published our descriptor.

References server_is_advertised.

Referenced by consider_publishable_server().

4.44.3.72 void v3_authority_check_key_expiry (void)

If we're a v3 authority, check whether we have a certificate that's likely to expire soon. Warn if we do, but not too often.

References authdir_mode_v3(), authority_cert_t::expires, get_options(), and LD_DIR.

Referenced by run_scheduled_events().

4.44.4 Variable Documentation

4.44.4.1 authority_cert_t* authority_key_certificate = NULL [static]

Key certificate to authenticate v3 directory material; only set for authorities.

4.44.4.2 crypto_pk_env_t* authority_signing_key = NULL [static]

Signing key used for v3 directory material; only set for authorities.

4.44.4.3 int can_reach_dir_port = 0 [static]

Whether we can reach our DirPort from the outside.

Referenced by check_whether_dirport_reachable(), router_dirport_found_reachable(), and router_reset_reachability().

4.44.4.4 int can_reach_or_port = 0 [static]

Whether we can reach our ORPort from the outside.

Referenced by check_whether_orport_reachable(), router_orport_found_reachable(), and router_reset_reachability().

4.44.4.5 time_t desc_clean_since = 0 [static]

Since when has our descriptor been "clean"? 0 if we need to regenerate it now.

Referenced by mark_my_descriptor_dirty(), mark_my_descriptor_dirty_if_older_than(), and router_rebuild_descriptor().

4.44.4.6 extrainfo_t* desc_extrainfo = NULL [static]

My extrainfo

4.44.4.7 int desc_needs_upload = 0 [static]

Boolean: do we need to regenerate the above?

Referenced by router_rebuild_descriptor(), and router_upload_dir_desc_to_dirservers().

4.44.4.8 routerinfo_t* desc_routerinfo = NULL [static]

My routerinfo.

4.44.4.9 crypto_pk_env_t* identitykey = NULL [static]

Private "identity key": used to sign directory info and TLS certificates. Never changes.

4.44.4.10 char identitykey_digest[DIGEST_LEN] [static]

Digest of identitykey.

Referenced by router_digest_is_me(), and set_identity_key().

4.44.4.11 tor_mutex_t* key_lock = NULL [static]

Private keys for this OR. There is also an SSL key managed by **tortls.c** (p. 1219).

4.44.4.12 uint32_t last_guessed_ip = 0 [static]

The most recently guessed value of our IP address, based on directory headers.

Referenced by router_guess_address_from_dir_headers(), and router_new_address_suggestion().

4.44.4.13 crypto_pk_env_t* lastonionkey = NULL [static]

Previous private onionskin decryption key: used to decode CREATE cells generated by clients that have an older version of our descriptor.

4.44.4.14 authority_cert_t* legacy_key_certificate = NULL [static]

For emergency V3 authority key migration: An extra certificate to authenticate legacy_signing_key with our obsolete identity key.

4.44.4.15 crypto_pk_env_t* legacy_signing_key = NULL [static]

For emergency V3 authority key migration: An extra signing key that we use with our old (obsolete) identity key for a while.

4.44.4.16 crypto_pk_env_t* onionkey = NULL [static]

Current private onionskin decryption key: used to decode CREATE cells.

4.44.4.17 time_t onionkey_set_at = 0 `[static]`

When was onionkey last changed?

Referenced by get_onion_key_set_at(), init_keys(), rotate_onion_key(), and set_onion_key().

4.44.4.18 int server_is_advertised = 0 `[static]`

Remember if we've advertised ourselves to the dirservers.

Referenced by advertised_server_mode(), and set_server_advertised().

4.44.4.19 long stats_n_seconds_working

How many seconds have we been running?

4.44.4.20 smartlist_t* warned_nonexistent_family = NULL `[static]`

A list of nicknames that we've warned about including in our family declaration verbatim rather than as digests.

4.45 routerlist.c File Reference

Code to maintain and access the global list of routerinfos for known servers.

`#include "or.h"`

Data Structures

- struct **cert_list_t**
- struct **duration_idx_t**
- struct **routerset_t**

Defines

- #define **SDMAP_FOREACH**(map, keyvar, valvar)
- #define **RIMAP_FOREACH**(map, keyvar, valvar) DIGESTMAP_FOREACH(rimap_to_digestmap(map), keyvar, **routerinfo_t** ∗, valvar)
- #define **EIMAP_FOREACH**(map, keyvar, valvar) DIGESTMAP_FOREACH(eimap_to_digestmap(map), keyvar, **extrainfo_t** ∗, valvar)
- #define **get_n_v2_authorities**() get_n_authorities(V2_AUTHORITY)
- #define **DEAD_CERT_LIFETIME** (2∗24∗60∗60)
- #define **OLD_CERT_LIFETIME** (7∗24∗60∗60)
- #define **MAX_CERT_DL_FAILURES** 8
- #define **RRS_FORCE** 1
- #define **RRS_DONT_REMOVE_OLD** 2
- #define **DIR_503_TIMEOUT** (60∗60)
- #define **DEFAULT_MAX_BELIEVABLE_BANDWIDTH** 10000000
- #define **should_cache_old_descriptors**() directory_caches_dir_info(get_options())
- #define **MAX_DL_PER_REQUEST** 96
- #define **MIN_DL_PER_REQUEST** 4
- #define **MIN_REQUESTS** 3
- #define **MAX_DL_TO_DELAY** 16
- #define **MAX_CLIENT_INTERVAL_WITHOUT_REQUEST** (10∗60)
- #define **DUMMY_DOWNLOAD_INTERVAL** (20∗60)
- #define **ROUTER_MAX_COSMETIC_TIME_DIFFERENCE** (12∗60∗60)
- #define **ROUTER_ALLOW_UPTIME_DRIFT** (6∗60∗60)

Functions

- static **routerstatus_t** ∗ **router_pick_directory_server_impl** (**authority_type_t** auth, int flags)
- static **routerstatus_t** ∗ **router_pick_trusteddirserver_impl** (**authority_type_t** auth, int flags, int ∗n_busy_out)
- static void **mark_all_trusteddirservers_up** (void)
- static int **router_nickname_matches** (**routerinfo_t** ∗router, const char ∗nickname)
- static void **trusted_dir_server_free** (**trusted_dir_server_t** ∗ds)
- static void **launch_router_descriptor_downloads** (**smartlist_t** ∗downloadable, **routerstatus_t** ∗source, time_t now)
- static int **signed_desc_digest_is_recognized** (**signed_descriptor_t** ∗desc)
- static void **update_router_have_minimum_dir_info** (void)

- static const char * **signed_descriptor_get_body_impl** (**signed_descriptor_t** *desc, int with_annotations)
- static void **list_pending_downloads** (digestmap_t *result, int purpose, const char *prefix)
- int **get_n_authorities** (**authority_type_t** type)
- static **cert_list_t** * **get_cert_list** (const char *id_digest)
- int **trusted_dirs_reload_certs** (void)
- static INLINE int **already_have_cert** (**authority_cert_t** *cert)
- int **trusted_dirs_load_certs_from_string** (const char *contents, int from_store, int flush)
- void **trusted_dirs_flush_certs_to_disk** (void)
- static void **trusted_dirs_remove_old_certs** (void)
- **authority_cert_t** * **authority_cert_get_newest_by_id** (const char *id_digest)
- **authority_cert_t** * **authority_cert_get_by_sk_digest** (const char *sk_digest)
- **authority_cert_t** * **authority_cert_get_by_digests** (const char *id_digest, const char *sk_digest)
- void **authority_cert_get_all** (**smartlist_t** *certs_out)
- void **authority_cert_dl_failed** (const char *id_digest, int status)
- void **authority_certs_fetch_missing** (**networkstatus_t** *status, time_t now)
- static int **router_should_rebuild_store** (**desc_store_t** *store)
- static INLINE **desc_store_t** * **desc_get_store** (**routerlist_t** *rl, **signed_descriptor_t** *sd)
- static int **signed_desc_append_to_journal** (**signed_descriptor_t** *desc, **desc_store_t** *store)
- static int **_compare_signed_descriptors_by_age** (const void **_a, const void **_b)
- static int **router_rebuild_store** (int flags, **desc_store_t** *store)
- static int **router_reload_router_list_impl** (**desc_store_t** *store)
- int **router_reload_router_list** (void)
- **smartlist_t** * **router_get_trusted_dir_servers** (void)
- **routerstatus_t** * **router_pick_directory_server** (**authority_type_t** type, int flags)
- int **router_get_my_share_of_directory_requests** (double *v2_share_out, double *v3_share_out)
- **trusted_dir_server_t** * **router_get_trusteddirserver_by_digest** (const char *digest)
- **trusted_dir_server_t** * **trusteddirserver_get_by_v3_auth_digest** (const char *digest)
- **routerstatus_t** * **router_pick_trusteddirserver** (**authority_type_t** type, int flags)
- void **router_reset_status_download_failures** (void)
- static INLINE int **routers_in_same_network_family** (**routerinfo_t** *r1, **routerinfo_t** *r2)
- static void **routerlist_add_network_family** (**smartlist_t** *sl, **routerinfo_t** *router)
- void **routerlist_add_family** (**smartlist_t** *sl, **routerinfo_t** *router)
- static INLINE int **router_in_nickname_smartlist** (**smartlist_t** *lst, **routerinfo_t** *r)
- int **routers_in_same_family** (**routerinfo_t** *r1, **routerinfo_t** *r2)
- void **add_nickname_list_to_smartlist** (**smartlist_t** *sl, const char *list, int must_be_running)
- int **router_nickname_is_in_list** (**routerinfo_t** *router, const char *list)
- static void **router_add_running_routers_to_smartlist** (**smartlist_t** *sl, int allow_invalid, int need_uptime, int need_capacity, int need_guard)

- **routerinfo_t** ∗ **routerlist_find_my_routerinfo** (void)
- **routerinfo_t** ∗ **router_find_exact_exit_enclave** (const char ∗address, uint16_t port)
- int **router_is_unreliable** (**routerinfo_t** ∗router, int need_uptime, int need_capacity, int need_guard)
- uint32_t **router_get_advertised_bandwidth** (**routerinfo_t** ∗router)
- uint32_t **router_get_advertised_bandwidth_capped** (**routerinfo_t** ∗router)
- static INLINE int32_t **kb_to_bytes** (uint32_t bw)
- static void ∗ **smartlist_choose_by_bandwidth_weights** (**smartlist_t** ∗sl, **bandwidth_weight_rule_t** rule, int statuses)
- static void ∗ **smartlist_choose_by_bandwidth** (**smartlist_t** ∗sl, **bandwidth_weight_rule_t** rule, int statuses)
- **routerinfo_t** ∗ **routerlist_sl_choose_by_bandwidth** (**smartlist_t** ∗sl, **bandwidth_weight_rule_t** rule)
- **routerstatus_t** ∗ **routerstatus_sl_choose_by_bandwidth** (**smartlist_t** ∗sl, **bandwidth_weight_rule_t** rule)
- **routerinfo_t** ∗ **router_choose_random_node** (**smartlist_t** ∗excludedsmartlist, **routerset_t** ∗excludedset, **router_crn_flags_t** flags)
- static INLINE int **hex_digest_matches** (const char ∗hexdigest, const char ∗identity_digest, const char ∗nickname, int is_named)
- static INLINE int **router_hex_digest_matches** (**routerinfo_t** ∗router, const char ∗hexdigest)
- **routerinfo_t** ∗ **router_get_by_nickname** (const char ∗nickname, int warn_if_unnamed)
- int **router_digest_version_as_new_as** (const char ∗digest, const char ∗cutoff)
- int **router_digest_is_trusted_dir_type** (const char ∗digest, **authority_type_t** type)
- int **router_addr_is_trusted_dir** (uint32_t addr)
- int **hexdigest_to_digest** (const char ∗hexdigest, char ∗digest)
- **routerinfo_t** ∗ **router_get_by_hexdigest** (const char ∗hexdigest)
- **routerinfo_t** ∗ **router_get_by_digest** (const char ∗digest)
- **signed_descriptor_t** ∗ **router_get_by_descriptor_digest** (const char ∗digest)
- **signed_descriptor_t** ∗ **router_get_by_extrainfo_digest** (const char ∗digest)
- **signed_descriptor_t** ∗ **extrainfo_get_by_descriptor_digest** (const char ∗digest)
- const char ∗ **signed_descriptor_get_body** (**signed_descriptor_t** ∗desc)
- const char ∗ **signed_descriptor_get_annotations** (**signed_descriptor_t** ∗desc)
- **routerlist_t** ∗ **router_get_routerlist** (void)
- void **routerinfo_free** (**routerinfo_t** ∗router)
- void **extrainfo_free** (**extrainfo_t** ∗extrainfo)
- static void **signed_descriptor_free** (**signed_descriptor_t** ∗sd)
- static **signed_descriptor_t** ∗ **signed_descriptor_from_routerinfo** (**routerinfo_t** ∗ri)
- static void **_extrainfo_free** (void ∗e)
- void **routerlist_free** (**routerlist_t** ∗rl)
- void **dump_routerlist_mem_usage** (int severity)
- static INLINE int **_routerlist_find_elt** (**smartlist_t** ∗sl, void ∗ri, int idx)
- static void **routerlist_insert** (**routerlist_t** ∗rl, **routerinfo_t** ∗ri)
- static int **extrainfo_insert** (**routerlist_t** ∗rl, **extrainfo_t** ∗ei)
- static void **routerlist_insert_old** (**routerlist_t** ∗rl, **routerinfo_t** ∗ri)
- void **routerlist_remove** (**routerlist_t** ∗rl, **routerinfo_t** ∗ri, int make_old, time_t now)
- static void **routerlist_remove_old** (**routerlist_t** ∗rl, **signed_descriptor_t** ∗sd, int idx)

- static void **routerlist_replace** (**routerlist_t** *rl, **routerinfo_t** *ri_old, **routerinfo_t** *ri_new)
- static **routerinfo_t** * **routerlist_reparse_old** (**routerlist_t** *rl, **signed_descriptor_t** *sd)
- void **routerlist_free_all** (void)
- void **routerlist_reset_warnings** (void)
- void **router_set_status** (const char *digest, int up)
- **was_router_added_t** **router_add_to_routerlist** (**routerinfo_t** *router, const char **msg, int from_cache, int from_fetch)
- **was_router_added_t** **router_add_extrainfo_to_routerlist** (**extrainfo_t** *ei, const char **msg, int from_cache, int from_fetch)
- static int **_compare_old_routers_by_identity** (const void **_a, const void **_b)
- static int **_compare_duration_idx** (const void *_d1, const void *_d2)
- static void **routerlist_remove_old_cached_routers_with_id** (time_t now, time_t cutoff, int lo, int hi, **digestset_t** *retain)
- void **routerlist_remove_old_routers** (void)
- static void **routerlist_descriptors_added** (**smartlist_t** *sl, int from_cache)
- int **router_load_single_router** (const char *s, uint8_t purpose, int cache, const char **msg)
- int **router_load_routers_from_string** (const char *s, const char *eos, **saved_location_t** saved_location, **smartlist_t** *requested_fingerprints, int descriptor_digests, const char *prepend_annotations)
- void **router_load_extrainfo_from_string** (const char *s, const char *eos, **saved_location_t** saved_location, **smartlist_t** *requested_fingerprints, int descriptor_digests)
- void **routerlist_retry_directory_downloads** (time_t now)
- int **router_exit_policy_all_routers_reject** (uint32_t addr, uint16_t port, int need_uptime)
- int **router_exit_policy_rejects_all** (**routerinfo_t** *router)
- **trusted_dir_server_t** * **add_trusted_dir_server** (const char *nickname, const char *address, uint16_t dir_port, uint16_t or_port, const char *digest, const char *v3_auth_digest, **authority_type_t** type)
- void **authority_cert_free** (**authority_cert_t** *cert)
- void **clear_trusted_dir_servers** (void)
- int **any_trusted_dir_is_v1_authority** (void)
- static void **list_pending_descriptor_downloads** (digestmap_t *result, int extrainfo)
- static void **initiate_descriptor_downloads** (**routerstatus_t** *source, int purpose, **smartlist_t** *digests, int lo, int hi, int pds_flags)
- static INLINE int **client_would_use_router** (**routerstatus_t** *rs, time_t now, **or_options_t** *options)
- static void **update_router_descriptor_cache_downloads_v2** (time_t now)
- void **update_consensus_router_descriptor_downloads** (time_t now, int is_vote, **networkstatus_t** *consensus)
- void **update_router_descriptor_downloads** (time_t now)
- void **update_extrainfo_downloads** (time_t now)
- int **router_have_minimum_dir_info** (void)
- void **router_dir_info_changed** (void)
- const char * **get_dir_info_status_string** (void)
- static void **count_usable_descriptors** (int *num_present, int *num_usable, const **networkstatus_t** *consensus, **or_options_t** *options, time_t now)
- int **count_loading_descriptors_progress** (void)
- void **router_reset_descriptor_download_failures** (void)

- int **router_differences_are_cosmetic** (**routerinfo_t** *r1, **routerinfo_t** *r2)
- int **routerinfo_incompatible_with_extrainfo** (**routerinfo_t** *ri, **extrainfo_t** *ei, **signed_descriptor_t** *sd, const char **msg)
- void **routerlist_assert_ok** (**routerlist_t** *rl)
- const char * **esc_router_info** (**routerinfo_t** *router)
- static int **_compare_routerinfo_by_id_digest** (const void **a, const void **b)
- void **routers_sort_by_identity** (**smartlist_t** *routers)
- **routerset_t** * **routerset_new** (void)
- static char * **routerset_get_countryname** (const char *c)
- void **routerset_refresh_countries** (**routerset_t** *target)
- int **routerset_parse** (**routerset_t** *target, const char *s, const char *description)
- void **refresh_all_country_info** (void)
- void **routerset_union** (**routerset_t** *target, const **routerset_t** *source)
- int **routerset_is_list** (const **routerset_t** *set)
- int **routerset_needs_geoip** (const **routerset_t** *set)
- static int **routerset_is_empty** (const **routerset_t** *set)
- static int **routerset_contains** (const **routerset_t** *set, const **tor_addr_t** *addr, uint16_t orport, const char *nickname, const char *id_digest, int is_named, **country_t** country)
- int **routerset_contains_extendinfo** (const **routerset_t** *set, const **extend_info_t** *ei)
- int **routerset_contains_router** (const **routerset_t** *set, **routerinfo_t** *ri)
- int **routerset_contains_routerstatus** (const **routerset_t** *set, **routerstatus_t** *rs)
- void **routerset_get_all_routers** (**smartlist_t** *out, const **routerset_t** *routerset, int running_only)
- void **routersets_get_disjunction** (**smartlist_t** *target, const **smartlist_t** *source, const **routerset_t** *include, const **routerset_t** *exclude, int running_only)
- void **routerset_subtract_routers** (**smartlist_t** *lst, const **routerset_t** *routerset)
- char * **routerset_to_string** (const **routerset_t** *set)
- int **routerset_equal** (const **routerset_t** *old, const **routerset_t** *new)
- void **routerset_free** (**routerset_t** *routerset)
- void **routerinfo_set_country** (**routerinfo_t** *ri)
- void **routerlist_refresh_countries** (void)
- int **hid_serv_get_responsible_directories** (**smartlist_t** *responsible_dirs, const char *id)
- int **hid_serv_acting_as_directory** (void)
- int **hid_serv_responsible_for_desc_id** (const char *query)

Variables

- static **smartlist_t** * **trusted_dir_servers** = NULL
- static digestmap_t * **trusted_dir_certs** = NULL
- static int **trusted_dir_servers_certs_changed** = 0
- static **routerlist_t** * **routerlist** = NULL
- static **smartlist_t** * **warned_nicknames** = NULL
- static time_t **last_routerdesc_download_attempted** = 0
- static uint64_t **sl_last_total_weighted_bw** = 0
- static uint64_t **sl_last_weighted_bw_of_me** = 0
- static int **have_min_dir_info** = 0
- static int **need_to_update_have_min_dir_info** = 1
- static char **dir_info_status** [128] = ""

4.45.1 Detailed Description

Code to maintain and access the global list of routerinfos for known servers.

4.45.2 Define Documentation

4.45.2.1 #define DEFAULT_MAX_BELIEVABLE_BANDWIDTH 10000000

Do not weight any declared bandwidth more than this much when picking routers by bandwidth.

Referenced by router_get_advertised_bandwidth_capped().

4.45.2.2 #define DIR_503_TIMEOUT (60*60)

How long do we avoid using a directory server after it's given us a 503?

Referenced by router_pick_directory_server_impl(), and router_pick_trusteddirserver_impl().

4.45.2.3 #define DUMMY_DOWNLOAD_INTERVAL (20*60)

How often should we launch a server/authority request to be sure of getting a guess for our IP?

Referenced by update_router_descriptor_downloads().

4.45.2.4 #define MAX_CERT_DL_FAILURES 8

How many times will we try to fetch a certificate before giving up?

Referenced by authority_certs_fetch_missing().

4.45.2.5 #define MAX_CLIENT_INTERVAL_WITHOUT_REQUEST (10*60)

When directory clients have only a few servers to request, they batch them until they have more, or until this amount of time has passed.

Referenced by launch_router_descriptor_downloads().

4.45.2.6 #define MAX_DL_PER_REQUEST 96

Max amount of hashes to download per request. Since squid does not like URLs >= 4096 bytes we limit it to 96. 4096 - strlen(`http://255.255.255.255/tor/server/d/.z`) == 4058 4058/41 (40 for the hash and 1 for the + that separates them) => 98 So use 96 because it's a nice number.

Referenced by launch_router_descriptor_downloads(), and update_extrainfo_downloads().

4.45.2.7 #define MAX_DL_TO_DELAY 16

If we want fewer than this many descriptors, wait until we want more, or until MAX_CLIENT_-INTERVAL_WITHOUT_REQUEST has passed.

Referenced by launch_router_descriptor_downloads().

4.45.2.8 #define MIN_DL_PER_REQUEST 4

Don't split our requests so finely that we are requesting fewer than this number per server.

Referenced by launch_router_descriptor_downloads().

4.45.2.9 #define MIN_REQUESTS 3

To prevent a single screwy cache from confusing us by selective reply, try to split our requests into at least this many requests.

Referenced by launch_router_descriptor_downloads().

4.45.2.10 #define ROUTER_ALLOW_UPTIME_DRIFT (6*60*60)

We allow uptime to vary from how much it ought to be by this much.

Referenced by router_differences_are_cosmetic().

4.45.2.11 #define ROUTER_MAX_COSMETIC_TIME_DIFFERENCE (12*60*60)

Any changes in a router descriptor's publication time larger than this are automatically non-cosmetic.

Referenced by router_differences_are_cosmetic().

4.45.2.12 #define SDMAP_FOREACH(map, keyvar, valvar)

Value:

```
DIGESTMAP_FOREACH(sdmap_to_digestmap(map), keyvar, signed_descriptor_t *, \
                  valvar)
```

4.45.3 Function Documentation

4.45.3.1 static int _compare_duration_idx (const void * _*d1*, const void * _*d2*) `[static]`

Sorting helper: compare two **duration_idx_t** (p. 89) by their duration.

References duration_idx_t::duration.

Referenced by routerlist_remove_old_cached_routers_with_id().

4.45.3.2 static int _compare_old_routers_by_identity (const void ** _*a*, const void ** _*b*) `[static]`

Sorting helper: return <0, 0, or >0 depending on whether the signed_descriptor_t* in ***a** has an identity digest preceding, equal to, or later than that of ***b**.

References DIGEST_LEN, signed_descriptor_t::identity_digest, and signed_descriptor_t::published_on.

Referenced by routerlist_remove_old_routers().

4.45.3.3 static int _compare_routerinfo_by_id_digest (const void ** *a*, const void ** *b*) [static]

Helper for sorting: compare two routerinfos by their identity digest.

References routerinfo_t::cache_info, DIGEST_LEN, and signed_descriptor_t::identity_digest.

Referenced by routers_sort_by_identity().

4.45.3.4 static int _compare_signed_descriptors_by_age (const void ** *_a*, const void ** *_b*) [static]

Sorting helper: return <0, 0, or >0 depending on whether the signed_descriptor_t* in *__a__ is older, the same age as, or newer than the signed_descriptor_t* in *__b__.

References signed_descriptor_t::published_on.

Referenced by router_rebuild_store().

4.45.3.5 static void _extrainfo_free (void * *e*) [static]

Helper: free the storage held by the **extrainfo_t** (p. 99) in **e**.

References extrainfo_free().

Referenced by routerlist_free().

4.45.3.6 static INLINE int _routerlist_find_elt (smartlist_t * *sl*, void * *ri*, int *idx*) [static]

Debugging helper: If **idx** is nonnegative, assert that **ri** is in **sl** at position **idx**. Otherwise, search **sl** for **ri**. Return the index of **ri** in **sl**, or -1 if **ri** is not in **sl**.

References tor_assert.

Referenced by routerlist_replace().

4.45.3.7 void add_nickname_list_to_smartlist (smartlist_t * *sl*, const char * *list*, int *must_be_running*)

Given a (possibly NULL) comma-and-whitespace separated list of nicknames, see which nicknames in **list** name routers in our routerlist, and add the routerinfos for those routers to **sl**. If **must_be_running**, only include routers that we think are running. Warn if any non-Named routers are specified by nickname.

References is_legal_nickname_or_hexdigest(), routerinfo_t::is_running, LD_CONFIG, router_get_by_nickname(), router_get_consensus_status_by_nickname(), router_have_minimum_dir_info(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_split_string(), smartlist_string_isin(), tor_assert, tor_free, and warned_nicknames.

4.45.3.8 trusted_dir_server_t* add_trusted_dir_server (const char * *nickname*, const char * *address*, uint16_t *dir_port*, uint16_t *or_port*, const char * *digest*, const char * *v3_auth_digest*, authority_type_t *type*)

Add to the list of authoritative directory servers one at **address**:**port**, with identity key **digest**. If **address** is NULL, add ourself. Return the new trusted directory server entry on success or NULL if we couldn't add it.

References routerstatus_t::addr, trusted_dir_server_t::addr, trusted_dir_server_t::address, trusted_dir_server_t::description, trusted_dir_server_t::digest, DIGEST_LEN, routerstatus_t::dir_port, trusted_dir_server_t::dir_port, trusted_dir_server_t::fake_status, get_options(), routerstatus_t::identity_digest, trusted_dir_server_t::is_running, LD_CONFIG, routerstatus_t::nickname, trusted_dir_server_t::nickname, routerstatus_t::or_port, trusted_dir_server_t::or_port, resolve_my_address(), router_dir_info_changed(), smartlist_add(), smartlist_create(), tor_lookup_hostname(), tor_snprintf(), trusted_dir_servers, trusted_dir_server_t::type, V3_AUTHORITY, trusted_dir_server_t::v3_identity_digest, routerstatus_t::version_supports_begindir, and routerstatus_t::version_supports_conditional_consensus.

Referenced by init_keys(), and parse_dir_server_line().

4.45.3.9 static INLINE int already_have_cert (authority_cert_t * *cert*) [static]

Helper: return true iff we already have loaded the exact cert **cert**.

References authority_cert_t::cache_info, cert_list_t::certs, DIGEST_LEN, get_cert_list(), signed_descriptor_t::identity_digest, and signed_descriptor_t::signed_descriptor_digest.

Referenced by trusted_dirs_load_certs_from_string().

4.45.3.10 int any_trusted_dir_is_v1_authority (void)

Return 1 if any trusted dir server supports v1 directories, else return 0.

References get_n_authorities(), trusted_dir_servers, and V1_AUTHORITY.

4.45.3.11 void authority_cert_dl_failed (const char * *id_digest*, int *status*)

Called when an attempt to download a certificate with the authority with ID **id_digest** fails with HTTP response code **status**: remember the failure, so we don't try again immediately.

References digestmap_get(), cert_list_t::dl_status, download_status_failed, and trusted_dir_certs.

Referenced by connection_dir_download_cert_failed(), and trusted_dirs_load_certs_from_string().

4.45.3.12 void authority_cert_free (authority_cert_t * *cert*)

Free storage held in **cert**.

References authority_cert_t::cache_info, crypto_free_pk_env(), authority_cert_t::identity_key, signed_descriptor_t::signed_descriptor_body, authority_cert_t::signing_key, and tor_free.

Referenced by authority_cert_parse_from_string(), load_authority_keyset(), networkstatus_vote_free(), router_free_all(), routerlist_free_all(), trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

4.45.3.13 void authority_cert_get_all (smartlist_t ∗ *certs_ out*)

Add every known **authority_cert_t** (p. 14) to **certs_out**.

References smartlist_add(), tor_assert, and trusted_dir_certs.

Referenced by directory_handle_command_get().

4.45.3.14 authority_cert_t∗ authority_cert_get_by_digests (const char ∗ *id_ digest*, const char ∗ *sk_ digest*)

Return the v3 authority certificate with signing key matching **sk_digest**, for the authority with identity digest **id_digest**. Return NULL if no such authority is known.

References cert_list_t::certs, DIGEST_LEN, digestmap_get(), and trusted_dir_certs.

Referenced by authority_cert_parse_from_string(), authority_certs_fetch_missing(), directory_handle_command_get(), dirvote_add_vote(), networkstatus_add_detached_signatures(), and networkstatus_check_consensus_signature().

4.45.3.15 authority_cert_t∗ authority_cert_get_by_sk_digest (const char ∗ *sk_ digest*)

Return the newest v3 authority certificate whose directory signing key has digest **sk_digest**. Return NULL if no such certificate is known.

References DIGEST_LEN, get_my_v3_authority_cert(), get_my_v3_legacy_cert(), authority_cert_t::signing_key_digest, and trusted_dir_certs.

Referenced by directory_handle_command_get().

4.45.3.16 authority_cert_t∗ authority_cert_get_newest_by_id (const char ∗ *id_ digest*)

Return the newest v3 authority certificate whose v3 authority identity key has digest **id_digest**. Return NULL if no such authority is known, or it has no certificate.

References authority_cert_t::cache_info, cert_list_t::certs, digestmap_get(), signed_descriptor_t::published_on, and trusted_dir_certs.

Referenced by directory_handle_command_get().

4.45.3.17 void authority_certs_fetch_missing (networkstatus_t ∗ *status*, time_t *now*)

Try to download any v3 authority certificates that we may be missing. If **status** is provided, try to get all the ones that were used to sign **status**. Additionally, try to have a non-expired certificate for every V3 authority in trusted_dir_servers. Don't fetch certificates we already have.

References authority_cert_get_by_digests(), cert_list_t::certs, DIGEST_LEN, digestmap_get(), digestmap_new(), DIR_PURPOSE_FETCH_CERTIFICATE, directory_caches_dir_info(), cert_list_t::dl_status, download_status_is_ready(), download_status_reset(), authority_cert_t::expires, ftime_definitely_after(), get_cert_list(), get_options(), hex_str(), LD_DIR, list_pending_downloads(), MAX_CERT_DL_FAILURES, should_delay_dir_fetches(), smartlist_add(), smartlist_create(), smartlist_digest_isin(), trusted_dir_servers, trusteddirserver_get_by_v3_auth_digest(), V3_AUTHORITY, and networkstatus_t::voters.

Referenced by networkstatus_set_current_consensus(), and update_certificate_downloads().

4.45.3.18 void clear_trusted_dir_servers (void)

Remove all members from the list of trusted dir servers.

References router_dir_info_changed(), smartlist_clear(), smartlist_create(), trusted_dir_server_free(), and trusted_dir_servers.

Referenced by consider_adding_dir_authorities().

4.45.3.19 static INLINE int client_would_use_router (routerstatus_t ∗ *rs*, time_t *now*, or_options_t ∗ *options*) [static]

Return 0 if this routerstatus is obsolete, too new, isn't running, or otherwise not a descriptor that we would make any use of even if we had it. Else return 1.

References or_options_t::FetchUselessDescriptors, routerstatus_t::is_running, OLD_ROUTER_DESC_MAX_AGE, routerstatus_t::published_on, and or_options_t::TestingEstimatedDescriptorPropagationTime.

Referenced by count_usable_descriptors().

4.45.3.20 int count_loading_descriptors_progress (void)

We just fetched a new set of descriptors. Compute how far through the "loading descriptors" bootstrapping phase we are, so we can inform the controller of our progress.

References count_usable_descriptors(), get_options(), and networkstatus_get_reasonably_live_consensus().

Referenced by connection_edge_process_relay_cell_not_open(), and load_downloaded_routers().

4.45.3.21 static void count_usable_descriptors (int ∗ *num_present*, int ∗ *num_usable*, const networkstatus_t ∗ *consensus*, or_options_t ∗ *options*, time_t *now*) [static]

Iterate over the servers listed in **consensus**, and count how many of them seem like ones we'd use, and how many of *those* we have descriptors for. Store the former in ∗**num_usable** and the latter in ∗**num_present**.

References client_would_use_router(), router_get_by_descriptor_digest(), and networkstatus_t::routerstatus_list.

Referenced by count_loading_descriptors_progress(), and update_router_have_minimum_dir_info().

4.45.3.22 static INLINE desc_store_t∗ desc_get_store (routerlist_t ∗ *rl*, signed_descriptor_t ∗ *sd*) [static]

Return the **desc_store_t** (p. 75) in **rl** that should be used to store **sd**.

References routerlist_t::desc_store, routerlist_t::extrainfo_store, and signed_descriptor_t::is_extrainfo.

Referenced by routerlist_remove_old(), and signed_descriptor_get_body_impl().

4.45.3.23 void dump_routerlist_mem_usage (int *severity*)

Log information about how much memory is being used for routerlist, at log level **severity**.

References LD_DIR, routerlist_t::old_routers, routerlist, and routerlist_t::routers.

Referenced by dumpmemusage().

4.45.3.24 const char∗ esc_router_info (routerinfo_t ∗ *router*)

Allocate and return a new string representing the contact info and platform string for **router**, surrounded by quotes and using standard C escapes.

THIS FUNCTION IS NOT REENTRANT. Don't call it from outside the main thread. Also, each call invalidates the last-returned value, so don't try log_warn(LD_GENERAL, "%s %s", esc_router_info(a), esc_router_info(b));

If **router** is NULL, it just frees its internal memory and returns.

References routerinfo_t::contact_info, esc_for_log(), routerinfo_t::platform, tor_free, and tor_snprintf().

Referenced by authdir_wants_to_reject_router(), and tor_free_all().

4.45.3.25 void extrainfo_free (extrainfo_t ∗ *extrainfo*)

Release all storage held by **extrainfo**

References extrainfo_t::cache_info, extrainfo_t::pending_sig, signed_descriptor_t::signed_descriptor_body, and tor_free.

Referenced by _extrainfo_free(), dirserv_add_extrainfo(), extrainfo_dump_to_string(), extrainfo_insert(), extrainfo_parse_entry_from_string(), router_free_all(), router_rebuild_descriptor(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

4.45.3.26 signed_descriptor_t∗ extrainfo_get_by_descriptor_digest (const char ∗ *digest*)

Return the signed descriptor for the **extrainfo_t** (p. 99) in our routerlist whose extra-info-digest is **digest**. Return NULL if no such extra-info document is known.

References extrainfo_t::cache_info, routerlist_t::extra_info_map, routerlist, and tor_assert.

Referenced by connection_dirserv_add_servers_to_outbuf(), dirserv_get_routerdesc_fingerprints(), dirserv_have_any_serverdesc(), and get_signed_descriptor_by_fp().

4.45.3.27 static int extrainfo_insert (routerlist_t ∗ *rl*, extrainfo_t ∗ *ei*) [static]

Adds the **extrainfo_t** (p. 99) **ei** to the routerlist **rl**, if there is a corresponding router in rl->routers or rl->old_routers. Return true iff we actually inserted **ei**. Free **ei** if it isn't inserted.

References desc_store_t::bytes_dropped, extrainfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::extra_info_map, extrainfo_free(), routerlist_t::extrainfo_store, signed_descriptor_t::identity_digest, routerlist_t::identity_map, router_get_my_extrainfo(), routerinfo_incompatible_with_extrainfo(), routerlist_assert_ok(), signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by router_add_extrainfo_to_routerlist().

4.45.3.28 static cert_list_t∗ get_cert_list (const char ∗ *id_digest*) [static]

Helper: Return the **cert_list_t** (p. 31) for an authority whose authority ID is **id_digest**, allocating a new list if necessary.

References cert_list_t::certs, digestmap_get(), digestmap_new(), digestmap_set(), cert_list_t::dl_status, download_status_t::schedule, and smartlist_create().

Referenced by already_have_cert(), authority_certs_fetch_missing(), and trusted_dirs_load_certs_from_string().

4.45.3.29 const char∗ get_dir_info_status_string (void)

Return a string describing what we're missing before we have enough directory info.

Referenced by directory_info_has_arrived().

4.45.3.30 int get_n_authorities (authority_type_t *type*)

Return the number of directory authorities whose type matches some bit set in **type**

Referenced by any_trusted_dir_is_v1_authority(), dirvote_compute_consensuses(), fetch_bridge_descriptors(), getinfo_helper_events(), and networkstatus_check_consensus_signature().

4.45.3.31 static INLINE int hex_digest_matches (const char ∗ *hexdigest*, const char ∗ *identity_digest*, const char ∗ *nickname*, int *is_named*) [static]

Helper: Return true iff the **identity_digest** and **nickname** combination of a router, encoded in hexadecimal, matches **hexdigest** (which is optionally prefixed with a single dollar sign). Return false if **hexdigest** is malformed, or it doesn't match.

References base16_decode(), DIGEST_LEN, HEX_DIGEST_LEN, and tor_assert.

Referenced by router_hex_digest_matches().

4.45.3.32 int hexdigest_to_digest (const char ∗ *hexdigest*, char ∗ *digest*)

If hexdigest is correctly formed, base16_decode it into digest, which must have DIGEST_LEN space in it. Return 0 on success, -1 on failure.

References base16_decode(), DIGEST_LEN, and HEX_DIGEST_LEN.

Referenced by circuit_is_acceptable(), connection_ap_fail_onehop(), and router_get_by_hexdigest().

4.45.3.33 int hid_serv_acting_as_directory (void)

Return true if this node is currently acting as hidden service directory, false otherwise.

References routerinfo_t::cache_info, get_options(), signed_descriptor_t::identity_digest, routerstatus_t::is_hs_dir, LD_REND, networkstatus_get_latest_consensus(), networkstatus_vote_find_entry(), and router_get_my_routerinfo().

Referenced by hid_serv_responsible_for_desc_id(), and rend_cache_store_v2_desc_as_dir().

4.45.3.34 int hid_serv_get_responsible_directories (smartlist_t * *responsible_dirs*, const char * *id*)

Determine the routers that are responsible for **id** (binary) and add pointers to those routers' **routerstatus_t** (p. 218) to **responsible_dirs**. Return -1 if we're returning an empty smartlist, else return 0.

References routerstatus_t::dir_port, get_options(), routerstatus_t::is_hs_dir, LD_REND, networkstatus_get_latest_consensus(), networkstatus_vote_find_entry_idx(), routerstatus_t::nickname, REND_NUMBER_OF_CONSECUTIVE_REPLICAS, networkstatus_t::routerstatus_list, smartlist_add(), tor_assert, and or_options_t::TunnelDirConns.

Referenced by directory_get_from_hs_dir(), directory_post_to_hs_dir(), and hid_serv_responsible_for_desc_id().

4.45.3.35 int hid_serv_responsible_for_desc_id (const char * *query*)

Return true if this node is responsible for storing the descriptor ID in **query** and false otherwise.

References routerinfo_t::cache_info, hid_serv_acting_as_directory(), hid_serv_get_responsible_directories(), routerstatus_t::identity_digest, signed_descriptor_t::identity_digest, rend_id_is_in_interval(), router_get_my_routerinfo(), smartlist_create(), and smartlist_free().

Referenced by rend_cache_clean_v2_descs_as_dir(), rend_cache_lookup_v2_desc_as_dir(), and rend_cache_store_v2_desc_as_dir().

4.45.3.36 static void initiate_descriptor_downloads (routerstatus_t * *source*, int *purpose*, smartlist_t * *digests*, int *lo*, int *hi*, int *pds_flags*) `[static]`

Launch downloads for all the descriptors whose digests are listed as digests[i] for lo <= i < hi. (Lo and hi may be out of range.) If **source** is given, download from **source**; otherwise, download from an appropriate random directory server.

References base16_encode(), DIGEST_LEN, directory_get_from_dirserver(), directory_initiate_command_routerstatus(), HEX_DIGEST_LEN, ROUTER_PURPOSE_GENERAL, and tor_free.

Referenced by launch_router_descriptor_downloads(), and update_extrainfo_downloads().

4.45.3.37 static INLINE int32_t kb_to_bytes (uint32_t *bw*) `[static]`

Return bw∗1000, unless bw∗1000 would overflow, in which case return INT32_MAX.

Referenced by smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.45.3.38 static void launch_router_descriptor_downloads (smartlist_t ∗ *downloadable*, routerstatus_t ∗ *source*, time_t *now*) `[static]`

Given a list of router descriptor digests in **downloadable**, decide whether to delay fetching until we have more. If we don't want to delay, launch one or more requests to the appropriate directory authorities.

References authdir_mode_any_nonhidserv(), DIR_PURPOSE_FETCH_SERVERDESC, directory_fetches_dir_info_early(), get_options(), initiate_descriptor_downloads(), last_routerdesc_download_attempted, LD_DIR, MAX_CLIENT_INTERVAL_WITHOUT_REQUEST, MAX_DL_PER_REQUEST, MAX_DL_TO_DELAY, MIN_DL_PER_REQUEST, MIN_REQUESTS, PDS_NO_EXISTING_SERVERDESC_FETCH, PDS_RETRY_IF_NO_SERVERS, and smartlist_sort_digests().

4.45.3.39 static void list_pending_descriptor_downloads (digestmap_t ∗ *result*, int *extrainfo*) `[static]`

For every router descriptor (or extra-info document if **extrainfo** is true) we are currently downloading by descriptor digest, set result[d] to (void∗)1.

References DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_SERVERDESC, and list_pending_downloads().

Referenced by update_consensus_router_descriptor_downloads(), update_extrainfo_downloads(), and update_router_descriptor_cache_downloads_v2().

4.45.3.40 static void list_pending_downloads (digestmap_t ∗ *result*, int *purpose*, const char ∗ *prefix*) `[static]`

For every current directory connection whose purpose is **purpose**, and where the resource being downloaded begins with **prefix**, split rest of the resource into base16 fingerprints, decode them, and set the corresponding elements of **result** to a nonzero value.

References CONN_TYPE_DIR, dir_split_resource_into_fingerprints(), get_connection_array(), smartlist_create(), strcmpstart(), TO_DIR_CONN(), and tor_assert.

Referenced by authority_certs_fetch_missing(), and list_pending_descriptor_downloads().

4.45.3.41 static void mark_all_trusteddirservers_up (void) `[static]`

Go through and mark the authoritative dirservers as up.

References routerlist, and routerlist_t::routers.

Referenced by router_pick_directory_server(), router_pick_trusteddirserver(), and router_reset_status_download_failures().

4.45.3.42 void refresh_all_country_info (void)

Called when we change a node set, or when we reload the geoip list: recompute all country info in all configuration node sets and in the routerlist.

References or_options_t::_ExcludeExitNodesUnion, or_options_t::EntryNodes, or_options_t::ExcludeExitNodes, or_options_t::ExcludeNodes, or_options_t::ExitNodes, get_options(), routerlist_refresh_countries(), and routerset_refresh_countries().

Referenced by geoip_load_file().

4.45.3.43 was_router_added_t router_add_extrainfo_to_routerlist (extrainfo_t * *ei*, const char ** *msg*, int *from_cache*, int *from_fetch*)

Insert **ei** into the routerlist, or free it. Other arguments are as for **router_add_to_routerlist()** (p. 985). Return ROUTER_ADDED_SUCCESSFULLY iff we actually inserted it, ROUTER_BAD_EI otherwise.

References extrainfo_t::cache_info, extrainfo_insert(), routerlist_t::extrainfo_store, router_get_routerlist(), routerlist, and signed_desc_append_to_journal().

Referenced by dirserv_add_extrainfo(), and router_load_extrainfo_from_string().

4.45.3.44 static void router_add_running_routers_to_smartlist (smartlist_t * *sl*, int *allow_invalid*, int *need_uptime*, int *need_capacity*, int *need_guard*) `[static]`

Add every suitable router from our routerlist to **sl**, so that we can pick a node for a circuit.

References router_is_unreliable(), ROUTER_PURPOSE_GENERAL, routerlist, routerlist_t::routers, and smartlist_add().

4.45.3.45 was_router_added_t router_add_to_routerlist (routerinfo_t * *router*, const char ** *msg*, int *from_cache*, int *from_fetch*)

Add **router** to the routerlist, if we don't already have it. Replace older entries (if any) with the same key. Note: Callers should not hold their pointers to **router** if this function fails; **router** will either be inserted into the routerlist or freed. Similarly, even if this call succeeds, they should not hold their pointers to **router** after subsequent calls with other routerinfo's – they might cause the original routerinfo to get freed.

Returns the status for the operation. Might set *__msg__ if it wants the poster of the router to know something.

If **from_cache**, this descriptor came from our disk cache. If **from_fetch**, we received it in response to a request we made. (If both are false, that means it was uploaded to us as an auth dir server or via the controller.)

This function should be called *after* routers_update_status_from_consensus_networkstatus; subsequently, you should call router_rebuild_store and routerlist_descriptors_added.

References routerinfo_t::addr, authdir_mode_handles_descs(), authdir_wants_to_reject_router(), routerinfo_t::cache_info, routerlist_t::desc_digest_map, routerlist_t::desc_store, routerstatus_t::descriptor_digest, DIGEST_LEN, directory_set_dirty(), dirserv_single_reachability_test(), get_options(), hex_str(), signed_descriptor_t::identity_digest, routerlist_t::identity_map, routerinfo_t::is_valid, routerinfo_t::last_reachable, LD_DIR, routerstatus_-

t::need_to_mirror, networkstatus_get_latest_consensus(), networkstatus_get_v2_list(), networkstatus_v2_find_entry(), networkstatus_v2_list, networkstatus_vote_find_entry(), routerinfo_t::nickname, OLD_ROUTER_DESC_MAX_AGE, routerinfo_t::or_port, signed_descriptor_t::published_on, routerinfo_t::purpose, router_get_routerlist(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerinfo_is_a_configured_bridge(), routerlist, routerlist_insert(), routerlist_insert_old(), routerlist_replace(), signed_desc_append_to_journal(), signed_desc_digest_is_recognized(), signed_descriptor_t::signed_descriptor_digest, routerinfo_t::testing_since, and tor_assert.

Referenced by dirserv_add_descriptor(), router_load_routers_from_string(), and router_load_single_router().

4.45.3.46 int router_addr_is_trusted_dir (uint32_t *addr*)

Return true iff **addr** is the address of one of our trusted directory authorities.

References trusted_dir_servers.

Referenced by get_possible_sybil_list().

4.45.3.47 routerinfo_t* router_choose_random_node (smartlist_t * *excludedsmartlist*, routerset_t * *excludedset*, router_crn_flags_t *flags*)

Return a random running router from the routerlist. Never pick a node whose routerinfo is in **excludedsmartlist**, or whose routerinfo matches **excludedset**, even if they are the only nodes available. If **CRN_NEED_UPTIME** is set in flags and any router has more than a minimum uptime, return one of those. If **CRN_NEED_CAPACITY** is set in flags, weight your choice by the advertised capacity of each router. If **CRN_ALLOW_INVALID** is not set in flags, consider only Valid routers. If **CRN_NEED_GUARD** is set in flags, consider only Guard routers. If **CRN_WEIGHT_AS_EXIT** is set in flags, we weight bandwidths as if picking an exit node, otherwise we weight bandwidths for picking a relay node (that is, possibly discounting exit nodes).

References or_options_t::ExcludeSingleHopRelays, get_options(), router_get_routerlist(), routerlist_t::routers, smartlist_add(), smartlist_create(), and tor_assert.

Referenced by choose_good_exit_server(), and choose_good_middle_server().

4.45.3.48 int router_differences_are_cosmetic (routerinfo_t * *r1*, routerinfo_t * *r2*)

Return true iff the only differences between r1 and r2 are such that would not cause a recent (post 0.1.1.6) dirserver to republish.

References routerinfo_t::address, routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, routerinfo_t::cache_info, cmp_addr_policies(), routerinfo_t::contact_info, crypto_pk_cmp_keys(), routerinfo_t::declared_family, routerinfo_t::dir_port, routerinfo_t::exit_policy, routerinfo_t::has_old_dnsworkers, routerinfo_t::identity_pkey, routerinfo_t::is_hibernating, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::or_port, routerinfo_t::platform, signed_descriptor_t::published_on, routerinfo_t::purpose, ROUTER_ALLOW_UPTIME_DRIFT, ROUTER_MAX_COSMETIC_TIME_DIFFERENCE, tor_assert, and routerinfo_t::uptime.

Referenced by dirserv_add_descriptor().

4.45.3.49 int router_digest_is_trusted_dir_type (const char ∗ *digest*, authority_type_t *type*)

Return true iff **digest** is the digest of the identity key of a trusted directory matching at least one bit of **type**. If **type** is zero, any authority is okay.

References authdir_mode(), DIGEST_LEN, get_options(), router_digest_is_me(), and trusted_dir_servers.

Referenced by router_pick_directory_server_impl().

4.45.3.50 int router_digest_version_as_new_as (const char ∗ *digest*, const char ∗ *cutoff*)

Try to find a routerinfo for **digest**. If we don't have one, return 1. If we do, ask **tor_version_as_new_as()** (p. 1020) for the answer.

References routerinfo_t::platform, router_get_by_digest(), and tor_version_as_new_as().

4.45.3.51 void router_dir_info_changed (void)

Called when our internal view of the directory has changed. This can be when the authorities change, networkstatuses change, the list of routerdescs changes, or number of running routers changes.

References rend_hsdir_routers_changed().

Referenced by add_trusted_dir_server(), clear_trusted_dir_servers(), networkstatus_set_current_consensus(), options_act(), router_set_status(), routerlist_free(), routerlist_insert(), routerlist_remove(), routerlist_replace(), and run_scheduled_events().

4.45.3.52 int router_exit_policy_all_routers_reject (uint32_t *addr*, uint16_t *port*, int *need_uptime*)

Return 1 if all running sufficiently-stable routers will reject addr:port, return 0 if any might accept it.

References ADDR_POLICY_PROBABLY_REJECTED, ADDR_POLICY_REJECTED, compare_addr_to_addr_policy(), router_is_unreliable(), routerlist, and routerlist_t::routers.

Referenced by circuit_get_open_circ_or_launch().

4.45.3.53 int router_exit_policy_rejects_all (routerinfo_t ∗ *router*)

Return true iff **router** does not permit exit streams.

References routerinfo_t::policy_is_reject_star.

Referenced by choose_good_exit_server_general(), and dirserv_compute_performance_thresholds().

4.45.3.54 routerinfo_t∗ router_find_exact_exit_enclave (const char ∗ *address*, uint16_t *port*)

Find a router that's up, that has this IP address, and that allows exit to this address:port, or return NULL if there isn't a good one.

References ADDR_POLICY_ACCEPTED, compare_tor_addr_to_addr_policy(), routerlist, routerlist_t::routers, tor_addr_from_ipv4h, and tor_inet_aton().

4.45.3.55 uint32_t router_get_advertised_bandwidth (routerinfo_t ∗ *router*)

Return the smaller of the router's configured BandwidthRate and its advertised capacity.

References routerinfo_t::bandwidthcapacity, and routerinfo_t::bandwidthrate.

Referenced by _compare_routerinfo_by_ip_and_bw(), dirserv_compute_performance_thresholds(), dirserv_thinks_router_is_unreliable(), and set_routerstatus_from_routerinfo().

4.45.3.56 uint32_t router_get_advertised_bandwidth_capped (routerinfo_t ∗ *router*)

Return the smaller of the router's configured BandwidthRate and its advertised capacity, capped by max-believe-bw.

References routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, and DEFAULT_MAX_BELIEVABLE_BANDWIDTH.

Referenced by routerstatus_format_entry(), smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

4.45.3.57 signed_descriptor_t∗ router_get_by_descriptor_digest (const char ∗ *digest*)

Return the router in our routerlist whose 20-byte descriptor is **digest**. Return NULL if no such router is known.

References routerlist_t::desc_digest_map, routerlist, and tor_assert.

Referenced by connection_dirserv_add_servers_to_outbuf(), count_usable_descriptors(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirserv_have_any_serverdesc(), networkstatus_reset_download_failures(), router_set_networkstatus_v2(), update_consensus_router_descriptor_downloads(), and update_router_descriptor_cache_downloads_v2().

4.45.3.58 routerinfo_t∗ router_get_by_digest (const char ∗ *digest*)

Return the router in our routerlist whose 20-byte key digest is **digest**. Return NULL if no such router is known.

References routerlist_t::identity_map, routerlist, and tor_assert.

Referenced by bridges_retry_helper(), build_state_get_exit_router(), choose_good_entry_server(), choose_good_middle_server(), circuit_find_to_cannibalize(), circuit_list_path_impl(), circuit_rep_hist_note_result(), circuit_send_next_onion_skin(), command_process_create_cell(), command_process_netinfo_cell(), connection_ap_process_end_-

not_open(), connection_or_digest_is_known_relay(), connection_or_init_conn_from_address(), connection_or_set_state_open(), control_event_guard(), directory_get_from_hs_dir(), directory_initiate_command_routerstatus_rend(), directory_post_to_hs_dir(), dirserv_add_descriptor(), dirserv_add_extrainfo(), dirserv_get_routerdescs(), entry_guards_compute_status(), entry_is_live(), get_signed_descriptor_by_fp(), getinfo_helper_entry_guards(), handle_control_attachstream(), orconn_target_get_name(), rend_services_introduce(), rep_hist_dump_stats(), rep_hist_get_router_stability_doc(), rep_hist_record_mtbf_data(), router_digest_version_as_new_as(), router_get_by_hexdigest(), router_set_status(), router_supports_extrainfo(), routerstatus_format_entry(), smartlist_choose_by_bandwidth(), update_consensus_router_descriptor_downloads(), and update_extrainfo_downloads().

4.45.3.59 signed_descriptor_t* router_get_by_extrainfo_digest (const char * *digest*)

Return the signed descriptor for the router in our routerlist whose 20-byte extra-info digest is **digest**. Return NULL if no such router is known.

References routerlist_t::desc_by_eid_map, routerlist, and tor_assert.

Referenced by dir_routerdesc_download_failed().

4.45.3.60 routerinfo_t* router_get_by_hexdigest (const char * *hexdigest*)

Return the router in our routerlist whose hexadecimal key digest is **hexdigest**. Return NULL if no such router is known.

References DIGEST_LEN, HEX_DIGEST_LEN, hexdigest_to_digest(), routerinfo_t::is_named, routerinfo_t::nickname, router_get_by_digest(), routerlist, and tor_assert.

Referenced by getinfo_helper_dir(), and router_get_by_nickname().

4.45.3.61 routerinfo_t* router_get_by_nickname (const char * *nickname*, int *warn_if_unnamed*)

Return the router in our routerlist whose (case-insensitive) nickname or (case-sensitive) hexadecimal key digest is **nickname**. Return NULL if no such router is known.

References base16_decode(), DIGEST_LEN, get_options(), HEX_DIGEST_LEN, routerlist_t::identity_map, networkstatus_get_router_digest_by_nickname(), networkstatus_nickname_is_unnamed(), router_get_by_hexdigest(), router_get_my_routerinfo(), routerlist, routerlist_t::routers, server_mode(), and tor_assert.

Referenced by add_nickname_list_to_smartlist(), circuit_discard_optional_exit_enclaves(), circuit_get_open_circ_or_launch(), connection_ap_can_use_exit(), connection_ap_handshake_attach_circuit(), getinfo_helper_dir(), handle_control_extendcircuit(), rend_client_get_random_intro(), rend_service_introduce(), router_rebuild_descriptor(), routerlist_add_family(), and routerset_get_all_routers().

4.45.3.62 int router_get_my_share_of_directory_requests (double * *v2_share_out*, double * *v3_share_out*)

Try to determine which fraction of v2 and v3 directory requests aimed at caches will be sent to us. Set ***v2_share_out** and ***v3_share_out** to the fractions of v2 and v3 protocol shares we expect to see, respectively. Return 0 on success, negative on failure.

References routerinfo_t::cache_info, signed_descriptor_t::identity_digest, routerstatus_t::is_v2_dir, PDS_ALLOW_SELF, PDS_IGNORE_FASCISTFIREWALL, router_get_consensus_status_by_id(), router_get_my_routerinfo(), router_pick_directory_server(), sl_last_total_weighted_bw, V2_AUTHORITY, V3_AUTHORITY, and routerstatus_t::version_supports_v3_dir.

Referenced by geoip_determine_shares().

4.45.3.63 routerlist_t∗ router_get_routerlist (void)

Return the current list of all known routers.

References routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, desc_store_t::description, routerlist_t::extra_info_map, routerlist_t::extrainfo_store, desc_store_t::fname_alt_base, desc_store_t::fname_base, routerlist_t::identity_map, routerlist_t::old_routers, routerlist, routerlist_t::routers, smartlist_create(), and desc_store_t::type.

Referenced by choose_good_entry_server(), directory_remove_invalid(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirserv_orconn_tls_done(), dirserv_test_reachability(), format_networkstatus_vote(), generate_v2_networkstatus_opinion(), getinfo_helper_dir(), networkstatus_getinfo_by_purpose(), onion_pick_cpath_exit(), router_add_extrainfo_to_routerlist(), router_add_to_routerlist(), router_choose_random_node(), router_load_extrainfo_from_string(), router_parse_list_from_string(), router_reload_router_list(), routerlist_refresh_countries(), routers_update_all_from_networkstatus(), signed_descriptor_get_body_impl(), and update_extrainfo_downloads().

4.45.3.64 smartlist_t∗ router_get_trusted_dir_servers (void)

Return a smartlist containing a list of **trusted_dir_server_t** (p. 247) ∗ for all known trusted dirservers. Callers must not modify the list or its contents.

References smartlist_create(), and trusted_dir_servers.

Referenced by connection_dir_client_reached_eof(), connection_dir_download_networkstatus_failed(), consider_adding_dir_authorities(), directories_have_accepted_server_descriptor(), directory_get_consensus_url(), directory_get_from_all_authorities(), directory_post_to_dirservers(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_set_cached_networkstatus_v2(), dirvote_fetch_missing_votes(), list_v3_auth_ids(), networkstatus_check_consensus_signature(), and update_v2_networkstatus_cache_downloads().

4.45.3.65 trusted_dir_server_t∗ router_get_trusteddirserver_by_digest (const char ∗ *digest*)

Return the **trusted_dir_server_t** (p. 247) for the directory authority whose identity key hashes to **digest**, or NULL if no such authority is known.

References DIGEST_LEN, and trusted_dir_servers.

Referenced by connection_dir_client_reached_eof(), dir_networkstatus_download_failed(), init_keys(), router_set_networkstatus_v2(), and update_router_descriptor_cache_downloads_v2().

4.45.3.66 int router_have_minimum_dir_info (void)

Return true iff we have enough networkstatus and router information to start building circuits. Right now, this means "more than half the networkstatus documents, and at least 1/4 of expected routers."

References update_router_have_minimum_dir_info().

Referenced by add_nickname_list_to_smartlist(), circuit_get_open_circ_or_launch(), circuit_launch_by_extend_info(), directory_info_has_arrived(), getinfo_helper_events(), run_scheduled_events(), and update_extrainfo_downloads().

4.45.3.67 static INLINE int router_hex_digest_matches (routerinfo_t ∗ *router*, const char ∗ *hexdigest*) `[static]`

Return true iff the digest of **router**'s identity key, encoded in hexadecimal, matches **hexdigest** (which is optionally prefixed with a single dollar sign). Return false if **hexdigest** is malformed, or it doesn't match.

References routerinfo_t::cache_info, hex_digest_matches(), signed_descriptor_t::identity_digest, routerinfo_t::is_named, and routerinfo_t::nickname.

Referenced by router_nickname_matches().

4.45.3.68 static INLINE int router_in_nickname_smartlist (smartlist_t ∗ *lst*, routerinfo_t ∗ *r*) `[static]`

Return true iff r is named by some nickname in **lst**.

References router_nickname_matches().

Referenced by routers_in_same_family().

4.45.3.69 int router_is_unreliable (routerinfo_t ∗ *router*, int *need_uptime*, int *need_capacity*, int *need_guard*)

Return 1 if **router** is not suitable for these parameters, else 0. If **need_uptime** is non-zero, we require a minimum uptime. If **need_capacity** is non-zero, we require a minimum advertised bandwidth. If **need_guard**, we require that the router is a possible entry guard.

References routerinfo_t::is_fast, routerinfo_t::is_possible_guard, and routerinfo_t::is_stable.

Referenced by choose_good_exit_server_general(), entry_is_live(), router_add_running_routers_to_smartlist(), and router_exit_policy_all_routers_reject().

4.45.3.70 void router_load_extrainfo_from_string (const char ∗ *s*, const char ∗ *eos*, saved_location_t *saved_location*, smartlist_t ∗ *requested_fingerprints*, int *descriptor_digests*)

Parse one or more extrainfos from **s** (ending immediately before **eos** if **eos** is present). Other arguments are as for **router_load_routers_from_string()** (p. 997).

References base16_encode(), DIGEST_LEN, HEX_DIGEST_LEN, LD_DIR, router_add_extrainfo_to_routerlist(), router_get_routerlist(), router_parse_list_from_string(), router_rebuild_store(), routerlist, routerlist_assert_ok(), SAVED_NOWHERE, smartlist_create(), smartlist_free(), smartlist_string_remove(), and WRA_WAS_ADDED().

Referenced by connection_dir_client_reached_eof(), and router_reload_router_list_impl().

4.45.3.71 int router_load_routers_from_string (const char * *s*, const char * *eos*, saved_location_t *saved_location*, smartlist_t * *requested_fingerprints*, int *descriptor_digests*, const char * *prepend_annotations*)

Given a string **s** containing some routerdescs, parse it and put the routers into our directory. If saved_location is SAVED_NOWHERE, the routers are in response to a query to the network: cache them by adding them to the journal.

Return the number of routers actually added.

If **requested_fingerprints** is provided, it must contain a list of uppercased fingerprints. Do not update any router whose fingerprint is not on the list; after updating a router, remove its fingerprint from the list.

If **descriptor_digests** is non-zero, then the requested_fingerprints are descriptor digests. Otherwise they are identity digests.

References base16_encode(), routerlist_t::desc_store, DIGEST_LEN, download_status_mark_impossible(), HEX_DIGEST_LEN, hex_str(), LD_DIR, LD_GENERAL, router_add_to_routerlist(), router_get_dl_status_by_descriptor_digest(), router_parse_list_from_string(), router_rebuild_store(), routerinfo_free(), routerlist, routerlist_assert_ok(), routerlist_descriptors_added(), routers_update_status_from_consensus_networkstatus(), SAVED_NOWHERE, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_join_strings(), smartlist_string_isin(), smartlist_string_remove(), tor_free, WRA_WAS_ADDED(), and WRA_WAS_REJECTED().

Referenced by load_downloaded_routers(), and router_reload_router_list_impl().

4.45.3.72 int router_load_single_router (const char * *s*, uint8_t *purpose*, int *cache*, const char ** *msg*)

Code to parse a single router descriptor and insert it into the routerlist. Return -1 if the descriptor was ill-formed; 0 if the descriptor was well-formed but could not be added; and 1 if the descriptor was added.

If we don't add it and **msg** is not NULL, then assign to ***msg** a static string describing the reason for refusing the descriptor.

This is used only by the controller.

References routerinfo_t::cache_info, signed_descriptor_t::do_not_cache, LD_DIR, routerinfo_t::purpose, router_add_to_routerlist(), ROUTER_ANNOTATION_BUF_LEN, router_is_me(), router_parse_entry_from_string(), router_purpose_to_string(), routerinfo_free(), routerlist_descriptors_added(), routers_update_status_from_consensus_networkstatus(), smartlist_add(), smartlist_create(), smartlist_free(), tor_assert, tor_snprintf(), and WRA_WAS_ADDED().

Referenced by handle_control_postdescriptor().

4.45.3.73 int router_nickname_is_in_list (routerinfo_t * *router*, const char * *list*)

Return 1 iff any member of the (possibly NULL) comma-separated list **list** is an acceptable nickname or hexdigest for **router**. Else return 0.

References router_nickname_matches(), smartlist_create(), smartlist_free(), smartlist_split_string(), tor_assert, and tor_free.

Referenced by routers_in_same_family().

4.45.3.74 static int router_nickname_matches (routerinfo_t * *router*, const char * *nickname*) [static]

Return true if **router**'s nickname matches **nickname** (case-insensitive), or if **router's** identity key digest matches a hexadecimal value stored in **nickname**. Return false otherwise.

References routerinfo_t::nickname, and router_hex_digest_matches().

Referenced by router_in_nickname_smartlist(), router_nickname_is_in_list(), and routerlist_add_family().

4.45.3.75 routerstatus_t* router_pick_directory_server (authority_type_t *type*, int *flags*)

Try to find a running dirserver that supports operations of **type**.

If there are no running dirservers in our routerlist and the **PDS_RETRY_IF_NO_SERVERS** flag is set, set all the authoritative ones as running again, and pick one.

If the **PDS_IGNORE_FASCISTFIREWALL** flag is set, then include dirservers that we can't reach.

If the **PDS_ALLOW_SELF** flag is not set, then don't include ourself (if we're a dirserver).

Don't pick an authority if any non-authority is viable; try to avoid using servers that have returned 503 recently.

References get_options(), LD_DIR, mark_all_trusteddirservers_up(), PDS_RETRY_IF_NO_SERVERS, router_pick_directory_server_impl(), and routerlist.

Referenced by directory_get_from_dirserver(), and router_get_my_share_of_directory_requests().

4.45.3.76 static routerstatus_t * router_pick_directory_server_impl (authority_type_t *type*, int *flags*) [static]

Pick a random running valid directory server/mirror from our routerlist. Arguments are as for **router_pick_directory_server()** (p. 1001), except that RETRY_IF_NO_SERVERS is ignored, and:

If the _PDS_PREFER_TUNNELED_DIR_CONNS flag is set, prefer directory servers that we can use with BEGINDIR.

References tor_addr_t::addr, DIR_503_TIMEOUT, EXTRAINFO_CACHE, fascist_firewall_allows_address_dir(), fascist_firewall_allows_address_or(), networkstatus_get_latest_consensus(), PDS_ALLOW_SELF, PDS_IGNORE_FASCISTFIREWALL, router_digest_is_me(), router_digest_is_trusted_dir_type(), router_supports_extrainfo(), networkstatus_t::routerstatus_list, routerstatus_sl_choose_by_bandwidth(), smartlist_add(), smartlist_choose(), smartlist_create(), smartlist_free(), tor_addr_from_ipv4h, V2_AUTHORITY, and V3_AUTHORITY.

Referenced by router_pick_directory_server().

4.45.3.77 routerstatus_t* router_pick_trusteddirserver (authority_type_t *type*, int *flags*)

Try to find a running trusted dirserver. Flags are as for router_pick_directory_server.

References get_options(), LD_DIR, mark_all_trusteddirservers_up(), PDS_NO_EXISTING_SERVERDESC_FETCH, PDS_RETRY_IF_NO_SERVERS, router_pick_trusteddirserver_impl(), and tor_assert.

Referenced by directory_get_from_dirserver().

4.45.3.78 static routerstatus_t * router_pick_trusteddirserver_impl (authority_type_t *type*, int *flags*, int * *n_busy_out*) [static]

Choose randomly from among the trusted dirservers that are up. Flags are as for **router_pick_directory_server_impl()** (p. 1161).

References tor_addr_t::addr, CONN_TYPE_DIR, connection_get_by_type_addr_port_purpose(), DIR_503_TIMEOUT, DIR_PURPOSE_FETCH_EXTRAINFO, DIR_PURPOSE_FETCH_SERVERDESC, EXTRAINFO_CACHE, fascist_firewall_allows_address_dir(), fascist_firewall_allows_address_or(), PDS_ALLOW_SELF, PDS_IGNORE_FASCISTFIREWALL, PDS_NO_EXISTING_SERVERDESC_FETCH, router_digest_is_me(), router_get_my_routerinfo(), router_supports_extrainfo(), smartlist_add(), smartlist_choose(), smartlist_create(), smartlist_free(), tor_addr_from_ipv4h, and trusted_dir_servers.

Referenced by router_pick_trusteddirserver().

4.45.3.79 static int router_rebuild_store (int *flags*, desc_store_t * *store*) [static]

If the journal of **store** is too long, or if RRS_FORCE is set in **flags**, then atomically replace the saved router store with the routers currently in our routerlist, and clear the journal. Unless RRS_DONT_REMOVE_OLD is set in **flags**, delete expired routers before rebuilding the store. Return 0 on success, -1 on failure.

References _compare_signed_descriptors_by_age(), sized_chunk_t::bytes, extrainfo_t::cache_info, desc_store_t::description, routerlist_t::extra_info_map, desc_store_t::fname_base, get_datadir_fname, get_datadir_fname_suffix, LD_BUG, LD_DIR, LD_FS, sized_chunk_t::len, desc_store_t::mmap, routerlist_t::old_routers, replace_file(), router_should_rebuild_store(), routerlist, routerlist_remove_old_routers(), routerlist_t::routers, SAVED_IN_CACHE, signed_descriptor_get_body(), signed_descriptor_get_body_impl(), smartlist_add(), smartlist_create(), smartlist_sort(), tor_free, tor_mmap_file(), tor_munmap_file(), desc_store_t::type, and write_chunks_to_file().

Referenced by router_load_extrainfo_from_string(), router_load_routers_from_string(), router_reload_router_list_impl(), and routerlist_remove_old_routers().

4.45.3.80 int router_reload_router_list (void)

Load all cached router descriptors and extra-info documents from the store. Return 0 on success and -1 on failure.

References routerlist_t::desc_store, routerlist_t::extrainfo_store, router_get_routerlist(), and router_reload_router_list_impl().

Referenced by do_main_loop().

4.45.3.81 static int router_reload_router_list_impl (desc_store_t * *store*) [static]

Helper: Reload a cache file and its associated journal, setting metadata appropriately. If **extrainfo** is true, reload the extrainfo store; else reload the router descriptor store.

References tor_mmap_t::data, file_status(), desc_store_t::fname_alt_base, desc_store_t::fname_base, get_datadir_fname, get_datadir_fname_suffix, desc_store_t::journal_len, LD_DIR, desc_store_t::mmap, read_file_to_str(), remove_file_if_very_old(), RFTS_BIN, RFTS_IGNORE_MISSING, router_load_extrainfo_from_string(), router_load_routers_from_string(), router_rebuild_store(), routerlist_remove_old_routers(), SAVED_IN_CACHE, SAVED_IN_JOURNAL, tor_mmap_t::size, desc_store_t::store_len, tor_free, tor_mmap_file(), tor_munmap_file(), and desc_store_t::type.

Referenced by router_reload_router_list().

4.45.3.82 void router_reset_descriptor_download_failures (void)

Reset the descriptor download failure count on all routers, so that we can retry any long-failed routers immediately.

References download_status_reset(), last_routerdesc_download_attempted, networkstatus_reset_download_failures(), routerlist, and routerlist_t::routers.

Referenced by do_hup(), routerlist_retry_directory_downloads(), and run_scheduled_events().

4.45.3.83 void router_reset_status_download_failures (void)

Reset all internal variables used to count failed downloads of network status objects.

References mark_all_trusteddirservers_up().

Referenced by do_hup(), and routerlist_retry_directory_downloads().

4.45.3.84 void router_set_status (const char * *digest*, int *up*)

Mark the router with ID **digest** as running or non-running in our routerlist.

References routerinfo_t::address, control_event_networkstatus_changed_single(), DIGEST_LEN, routerstatus_t::is_running, routerinfo_t::is_running, LD_DIR, LD_NET, routerinfo_t::nickname, router_dir_info_changed(), router_get_by_digest(), router_get_consensus_status_by_id(), router_is_me(), tor_assert, trusted_dir_servers, and we_are_hibernating().

Referenced by connection_dir_request_failed(), and connection_or_set_state_open().

4.45.3.85 static int router_should_rebuild_store (desc_store_t * *store*) [static]

Helper: return 1 iff the router log is so big we want to rebuild the store.

References desc_store_t::bytes_dropped, desc_store_t::journal_len, and desc_store_t::store_len.

Referenced by router_rebuild_store().

4.45.3.86 void routerinfo_free (routerinfo_t * *router*)

Free all storage held by **router**.

References addr_policy_list_free(), routerinfo_t::address, routerinfo_t::cache_info, routerinfo_t::contact_info, crypto_free_pk_env(), routerinfo_t::declared_family, routerinfo_t::exit_policy, routerinfo_t::identity_pkey, routerinfo_t::nickname, routerinfo_t::onion_pkey, routerinfo_t::platform, signed_descriptor_t::signed_descriptor_body, smartlist_free(), and tor_free.

Referenced by dirserv_add_descriptor(), router_add_to_routerlist(), router_dump_router_to_string(), router_free_all(), router_load_routers_from_string(), router_load_single_router(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerlist_free(), routerlist_insert_old(), routerlist_remove(), routerlist_replace(), and signed_descriptor_from_routerinfo().

4.45.3.87 int routerinfo_incompatible_with_extrainfo (routerinfo_t * *ri*, extrainfo_t * *ei*, signed_descriptor_t * *sd*, const char ** *msg*)

Check whether **ri** (a.k.a. sd) is a router compatible with the extrainfo document **ei**. If no router is compatible with **ei**, **ei** should be dropped. Return 0 for "compatible", return 1 for "reject, and inform whoever uploaded **ei**, and return -1 for "reject silently.". If **msg** is present, set ***msg** to a description of the incompatibility (if any).

References extrainfo_t::bad_sig, extrainfo_t::cache_info, routerinfo_t::cache_info, crypto_pk_public_checksig(), DIGEST_LEN, signed_descriptor_t::extra_info_digest, signed_descriptor_t::extrainfo_is_bogus, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, extrainfo_t::pending_sig, extrainfo_t::pending_sig_len, signed_descriptor_t::published_on, signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_digest, tor_assert, and tor_free.

Referenced by dirserv_add_extrainfo(), extrainfo_insert(), and router_rebuild_descriptor().

4.45.3.88 void routerinfo_set_country (routerinfo_t * *ri*)

Refresh the country code of **ri**. This function MUST be called on each router when the GeoIP database is reloaded, and on all new routers.

References routerinfo_t::addr, routerinfo_t::country, and geoip_get_country_by_ip().

Referenced by router_parse_entry_from_string(), router_rebuild_descriptor(), and routerlist_refresh_countries().

4.45.3.89 void routerlist_add_family (smartlist_t * *sl*, routerinfo_t * *router*)

Add all the family of **router** to the smartlist **sl**. This is used to make sure we don't pick siblings in a single path, or pick more than one relay from a family for our entry guard list.

References routerinfo_t::declared_family, or_options_t::EnforceDistinctSubnets, get_options(), router_get_by_nickname(), router_nickname_matches(), routerlist_add_network_family(), and smartlist_add().

Referenced by choose_good_entry_server(), choose_good_middle_server(), and choose_random_entry().

4.45.3.90 static void routerlist_add_network_family (smartlist_t * *sl*, routerinfo_t * *router*) [static]

Look through the routerlist and identify routers that advertise the same /16 network address as **router**. Add each of them to **sl**.

References routerlist, routerlist_t::routers, routers_in_same_network_family(), and smartlist_add().

Referenced by routerlist_add_family().

4.45.3.91 void routerlist_assert_ok (routerlist_t * *rl*)

Assert that the internal representation of **rl** is self-consistent.

References routerlist_t::desc_digest_map, routerlist_t::identity_map, routerlist_t::routers, and tor_assert.

Referenced by directory_remove_invalid(), extrainfo_insert(), router_load_extrainfo_from_string(), router_load_routers_from_string(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), and routerlist_replace().

4.45.3.92 static void routerlist_descriptors_added (smartlist_t * *sl*, int *from_cache*) [static]

We just added a new set of descriptors. Take whatever extra steps we need.

References control_event_descriptors_changed(), learned_bridge_descriptor(), ROUTER_PURPOSE_BRIDGE, and tor_assert.

Referenced by router_load_routers_from_string(), and router_load_single_router().

4.45.3.93 routerinfo_t* routerlist_find_my_routerinfo (void)

Look through the routerlist until we find a router that has my key. Return it.

References router_is_me(), routerlist, and routerlist_t::routers.

4.45.3.94 void routerlist_free (routerlist_t * *rl*)

Free all storage held by a routerlist **rl**.

References _extrainfo_free(), routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, routerlist_t::extra_info_map, routerlist_t::extrainfo_store, routerlist_t::identity_map, desc_store_t::mmap, routerlist_t::old_routers, router_dir_info_changed(), routerinfo_free(), routerlist, routerlist_t::routers, signed_descriptor_free(), smartlist_free(), tor_free, and tor_munmap_file().

Referenced by routerlist_free_all().

4.45.3.95 void routerlist_free_all (void)

Free all memory held by the routerlist module.

References authority_cert_free(), digestmap_free(), routerlist, routerlist_free(), smartlist_free(), tor_free, trusted_dir_certs, trusted_dir_server_free(), trusted_dir_servers, and warned_nicknames.

Referenced by tor_free_all().

4.45.3.96 static void routerlist_insert (routerlist_t ∗ *rl*, routerinfo_t ∗ *ri*) `[static]`

Insert an item **ri** into the routerlist **rl**, updating indices as needed. There must be no previous member of **rl** with the same identity digest as **ri**: If there is, call routerlist_replace instead.

References routerinfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, signed_descriptor_t::extra_info_digest, signed_descriptor_t::identity_digest, routerlist_t::identity_map, router_dir_info_changed(), router_get_my_routerinfo(), routerlist_assert_ok(), signed_descriptor_t::routerlist_index, routerlist_t::routers, signed_descriptor_t::signed_descriptor_digest, smartlist_add(), tor_assert, and tor_digest_is_zero().

Referenced by router_add_to_routerlist(), and routerlist_replace().

4.45.3.97 static void routerlist_insert_old (routerlist_t ∗ *rl*, routerinfo_t ∗ *ri*) `[static]`

If we're a directory cache and routerlist **rl** doesn't have a copy of router **ri** yet, add it to the list of old (not recommended but still served) descriptors. Else free it.

References routerinfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, signed_descriptor_t::extra_info_digest, routerlist_t::old_routers, routerinfo_t::purpose, router_get_my_routerinfo(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerlist_assert_ok(), signed_descriptor_t::routerlist_index, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_from_routerinfo(), smartlist_add(), tor_assert, and tor_digest_is_zero().

Referenced by router_add_to_routerlist().

4.45.3.98 void routerlist_refresh_countries (void)

Set the country code of all routers in the routerlist.

References router_get_routerlist(), routerinfo_set_country(), and routerlist_t::routers.

Referenced by refresh_all_country_info().

4.45.3.99 void routerlist_remove (routerlist_t ∗ *rl*, routerinfo_t ∗ *ri*, int *make_old*, time_t *now*)

Remove an item **ri** from the routerlist **rl**, updating indices as needed. If **idx** is nonnegative and smartlist_get(rl->routers, idx) == ri, we don't need to do a linear search over the list to decide which to remove. We fill the gap in rl->routers with a later element in the list, if any exists. **ri** is freed.

If **make_old** is true, instead of deleting the router, we try adding it to rl->old_routers.

References desc_store_t::bytes_dropped, extrainfo_t::cache_info, routerinfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, signed_descriptor_t::extra_info_digest, routerlist_t::extra_info_map, extrainfo_free(), routerlist_t::extrainfo_store, signed_descriptor_t::identity_digest, routerlist_t::identity_map, routerlist_t::old_routers, routerinfo_t::purpose, rep_hist_note_router_unreachable(), router_dir_info_changed(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerlist_assert_ok(), signed_descriptor_t::routerlist_index, routerlist_t::routers, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_from_routerinfo(), signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_del(), tor_assert, and tor_digest_is_zero().

Referenced by directory_remove_invalid(), and routerlist_remove_old_routers().

4.45.3.100 static void routerlist_remove_old (routerlist_t ∗ *rl*, signed_descriptor_t ∗ *sd*, int *idx*) [static]

Remove a **signed_descriptor_t** (p. 227) **sd** from **rl**->old_routers, and adjust **rl** as appropriate. **idx** is -1, or the index of **sd**.

References desc_store_t::bytes_dropped, extrainfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, desc_get_store(), signed_descriptor_t::extra_info_digest, routerlist_t::extra_info_map, extrainfo_free(), routerlist_t::extrainfo_store, routerlist_t::old_routers, routerlist_assert_ok(), signed_descriptor_t::routerlist_index, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_free(), signed_descriptor_t::signed_descriptor_len, smartlist_del(), tor_assert, and tor_digest_is_zero().

Referenced by routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), and routerlist_reparse_old().

4.45.3.101 static void routerlist_remove_old_cached_routers_with_id (time_t *now*, time_t *cutoff*, int *lo*, int *hi*, digestset_t ∗ *retain*) [static]

The range **lo** through **hi** inclusive of routerlist->old_routers must contain **routerinfo_t** (p. 206) with the same identity and with publication time in ascending order. Remove members from this range until there are no more than max_descriptors_per_router() remaining. Start by removing the oldest members from before **cutoff**, then remove members which were current for the lowest amount of time. The order of members of old_routers at indices **lo** or higher may be changed.

We aren't removing enough servers for being old. Sort lifespans by the duration of liveness, and remove the ones we're not already going to remove based on how long they were alive.

References _compare_duration_idx(), DIGEST_LEN, directory_caches_dir_info(), duration_idx_t::duration, get_options(), signed_descriptor_t::identity_digest, duration_idx_t::idx, signed_descriptor_t::last_listed_as_valid_until, duration_idx_t::old, routerlist_t::old_routers, signed_descriptor_t::published_on, routerlist, routerlist_remove_old(), signed_descriptor_t::signed_descriptor_digest, tor_assert, and tor_free.

Referenced by routerlist_remove_old_routers().

4.45.3.102 void routerlist_remove_old_routers (void)

Deactivate any routers from the routerlist that are more than ROUTER_MAX_AGE seconds old and not recommended by any networkstatuses; remove old routers from the list of cached routers if we have too many.

References _compare_old_routers_by_identity(), routerinfo_t::cache_info, routerlist_t::desc_store, DIGEST_LEN, digestset_free(), digestset_new(), directory_caches_dir_info(), routerlist_t::extrainfo_store, get_options(), signed_descriptor_t::identity_digest, signed_-

descriptor_t::last_listed_as_valid_until, LD_DIR, networkstatus_get_latest_consensus(), networkstatus_get_v2_list(), networkstatus_v2_list, routerinfo_t::nickname, OLD_ROUTER_DESC_MAX_AGE, routerlist_t::old_routers, signed_descriptor_t::published_on, ROUTER_MAX_AGE, router_rebuild_store(), routerlist, signed_descriptor_t::routerlist_index, routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_cached_routers_with_id(), routerlist_t::routers, networkstatus_t::routerstatus_list, signed_descriptor_t::signed_descriptor_digest, smartlist_sort(), and trusted_dirs_remove_old_certs().

Referenced by router_rebuild_store(), router_reload_router_list_impl(), and run_scheduled_events().

4.45.3.103 static routerinfo_t∗ routerlist_reparse_old (routerlist_t ∗ *rl*, signed_descriptor_t ∗ *sd*) `[static]`

Extract the descriptor **sd** from old_routerlist, and re-parse it as a fresh **routerinfo_t** (p. 206).

References signed_descriptor_t::annotations_len, routerinfo_t::cache_info, router_parse_entry_from_string(), signed_descriptor_t::routerlist_index, routerlist_remove_old(), signed_descriptor_t::signed_descriptor_body, signed_descriptor_get_annotations(), and signed_descriptor_t::signed_descriptor_len.

4.45.3.104 static void routerlist_replace (routerlist_t ∗ *rl*, routerinfo_t ∗ *ri_old*, routerinfo_t ∗ *ri_new*) `[static]`

Remove **ri_old** from the routerlist **rl**, and replace it with **ri_new**, updating all index info. If **idx** is nonnegative and smartlist_get(rl->routers, idx) == ri, we don't need to do a linear search over the list to decide which to remove. We put ri_new in the same index as ri_old, if possible. ri is freed as appropriate.

If should_cache_descriptors() is true, instead of deleting the router, we add it to rl->old_routers.

References _routerlist_find_elt(), desc_store_t::bytes_dropped, extrainfo_t::cache_info, routerinfo_t::cache_info, routerlist_t::desc_by_eid_map, routerlist_t::desc_digest_map, routerlist_t::desc_store, DIGEST_LEN, signed_descriptor_t::extra_info_digest, routerlist_t::extra_info_map, extrainfo_free(), routerlist_t::extrainfo_store, signed_descriptor_t::identity_digest, routerlist_t::identity_map, LD_BUG, routerlist_t::old_routers, routerinfo_t::purpose, router_dir_info_changed(), router_get_my_routerinfo(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerlist_assert_ok(), signed_descriptor_t::routerlist_index, routerlist_insert(), routerlist_t::routers, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_from_routerinfo(), signed_descriptor_t::signed_descriptor_len, smartlist_add(), tor_assert, and tor_digest_is_zero().

Referenced by router_add_to_routerlist().

4.45.3.105 void routerlist_reset_warnings (void)

Forget that we have issued any router-related warnings, so that we'll warn again if we see the same errors.

References networkstatus_reset_warnings(), smartlist_clear(), smartlist_create(), tor_free, and warned_nicknames.

Referenced by do_hup().

4.45.3.106 void routerlist_retry_directory_downloads (time_t *now*)

Clear all our timeouts for fetching v2 and v3 directory stuff, and then give it all a try again.

References router_reset_descriptor_download_failures(), router_reset_status_download_failures(), update_networkstatus_downloads(), and update_router_descriptor_downloads().

Referenced by circuit_get_open_circ_or_launch(), and learned_bridge_descriptor().

4.45.3.107 routerinfo_t∗ routerlist_sl_choose_by_bandwidth (smartlist_t ∗ *sl*, bandwidth_weight_rule_t *rule*)

Choose a random element of router list **sl**, weighted by the advertised bandwidth of each router.

References smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

Referenced by choose_good_exit_server_general(), and choose_random_entry().

4.45.3.108 int routers_in_same_family (routerinfo_t ∗ *r1*, routerinfo_t ∗ *r2*)

Return true iff r1 and r2 are in the same family, but not the same router.

References routerinfo_t::declared_family, or_options_t::EnforceDistinctSubnets, get_options(), config_line_t::next, or_options_t::NodeFamilies, router_in_nickname_smartlist(), router_nickname_is_in_list(), routers_in_same_network_family(), and config_line_t::value.

Referenced by circuit_find_to_cannibalize().

4.45.3.109 static INLINE int routers_in_same_network_family (routerinfo_t ∗ *r1*, routerinfo_t ∗ *r2*) `[static]`

Return true iff router1 and router2 have the same /16 network.

References routerinfo_t::addr.

Referenced by routerlist_add_network_family(), and routers_in_same_family().

4.45.3.110 void routers_sort_by_identity (smartlist_t ∗ *routers*)

Sort a list of **routerinfo_t** (p. 206) in ascending order of identity digest.

References _compare_routerinfo_by_id_digest(), and smartlist_sort().

Referenced by routers_update_status_from_consensus_networkstatus().

4.45.3.111 static int routerset_contains (const routerset_t ∗ *set*, const tor_addr_t ∗ *addr*, uint16_t *orport*, const char ∗ *nickname*, const char ∗ *id_digest*, int *is_named*, country_t *country*) `[static]`

Helper. Return true iff **set** contains a router based on the other provided fields. Return higher values for more specific subentries: a single router is more specific than an address range of routers, which is more specific in turn than a country code.

(If country is -1, then we take the country from addr.)

References ADDR_POLICY_REJECTED, compare_tor_addr_to_addr_policy(), digestmap_get(), geoip_get_country_by_ip(), strmap_get_lc(), and tor_addr_to_ipv4h().

Referenced by routerset_contains_extendinfo(), routerset_contains_router(), and routerset_contains_routerstatus().

4.45.3.112 int routerset_contains_extendinfo (const routerset_t ∗ *set*, const extend_info_t ∗ *ei*)

Return true iff we can tell that **ei** is a member of **set**.

References extend_info_t::addr, extend_info_t::identity_digest, extend_info_t::nickname, extend_info_t::port, and routerset_contains().

Referenced by circuit_conforms_to_options(), and warn_if_last_router_excluded().

4.45.3.113 int routerset_contains_router (const routerset_t ∗ *set*, routerinfo_t ∗ *ri*)

Return true iff **ri** is in **set**.

References routerinfo_t::addr, routerinfo_t::cache_info, routerinfo_t::country, signed_descriptor_t::identity_digest, routerinfo_t::is_named, routerinfo_t::nickname, routerinfo_t::or_port, routerset_contains(), and tor_addr_from_ipv4h.

Referenced by choose_good_exit_server_general(), choose_random_entry(), connection_ap_can_use_exit(), entry_guard_set_status(), entry_is_live(), routerset_subtract_routers(), and routersets_get_disjunction().

4.45.3.114 int routerset_contains_routerstatus (const routerset_t ∗ *set*, routerstatus_t ∗ *rs*)

Return true iff **rs** is in **set**.

References routerstatus_t::addr, routerstatus_t::identity_digest, routerstatus_t::is_named, routerstatus_t::nickname, routerstatus_t::or_port, routerset_contains(), and tor_addr_from_ipv4h.

4.45.3.115 int routerset_equal (const routerset_t ∗ *old*, const routerset_t ∗ *new*)

Helper: return true iff old and new are both NULL, or both non-NULL equal routersets.

References routerset_t::list.

Referenced by options_act().

4.45.3.116 void routerset_free (routerset_t ∗ *routerset*)

Free all storage held in **routerset**.

References addr_policy_free(), routerset_t::countries, routerset_t::country_names, digestmap_free(), routerset_t::digests, routerset_t::list, routerset_t::names, routerset_t::policies, smartlist_free(), strmap_free(), and tor_free.

Referenced by config_assign_value(), option_clear(), and or_options_free().

4.45.3.117 void routerset_get_all_routers (smartlist_t * *out*, const routerset_t * *routerset*, int *running_only*)

Add every known **routerinfo_t** (p. 206) that is a member of **routerset** to **out**. If **running_only**, only add the running ones.

References routerset_t::list, router_get_by_nickname(), routerset_is_list(), smartlist_add(), smartlist_create(), tor_assert, and warned_nicknames.

Referenced by entry_guards_prepend_from_config().

4.45.3.118 static char* routerset_get_countryname (const char * *c*) `[static]`

If **c** is a country code in the form {cc}, return a newly allocated string holding the "cc" part. Else, return NULL.

References tor_strlower().

Referenced by routerset_parse().

4.45.3.119 static int routerset_is_empty (const routerset_t * *set*) `[static]`

Return true iff there are no entries in **set**.

Referenced by routersets_get_disjunction().

4.45.3.120 int routerset_is_list (const routerset_t * *set*)

Return true iff **set** lists only nicknames and digests, and includes no IP ranges or countries.

Referenced by options_validate(), and routerset_get_all_routers().

4.45.3.121 int routerset_needs_geoip (const routerset_t * *set*)

Return true iff we need a GeoIP IP-to-country database to make sense of **set**.

Referenced by options_need_geoip_info().

4.45.3.122 routerset_t* routerset_new (void)

Return a new empty routerset.

References routerset_t::country_names, digestmap_new(), routerset_t::digests, routerset_t::list, routerset_t::names, routerset_t::policies, and smartlist_create().

Referenced by config_assign_value(), and options_validate().

4.45.3.123 int routerset_parse (routerset_t * *target*, const char * *s*, const char * *description*)

Parse the string **s** to create a set of routerset entries, and add them to **target**. In log messages, refer to the string as **description**. Return 0 on success, -1 on failure.

Three kinds of elements are allowed in routersets: nicknames, IP address patterns, and fingerprints. They may be surrounded by optional space, and must be separated by commas.

References base16_decode(), routerset_t::country_names, DIGEST_LEN, digestmap_set(), routerset_t::digests, HEX_DIGEST_LEN, is_legal_hexdigest(), is_legal_nickname(), LD_CONFIG, routerset_t::list, routerset_t::names, routerset_t::policies, router_parse_addr_policy_item_from_string(), routerset_get_countryname(), routerset_refresh_countries(), smartlist_add(), smartlist_add_all(), smartlist_create(), smartlist_free(), smartlist_split_string(), strmap_set_lc(), and tor_free.

Referenced by config_assign_value(), and routerset_union().

4.45.3.124 void routerset_refresh_countries (routerset_t * *target*)

Update the routerset's **countries** bitarray_t. Called whenever the GeoIP database is reloaded.

References routerset_t::countries, routerset_t::country_names, geoip_get_country(), geoip_get_n_countries(), geoip_is_loaded(), LD_CONFIG, routerset_t::n_countries, and tor_assert.

Referenced by refresh_all_country_info(), and routerset_parse().

4.45.3.125 void routerset_subtract_routers (smartlist_t * *lst*, const routerset_t * *routerset*)

Remove every **routerinfo_t** (p. 206) from **lst** that is in **routerset**.

References routerset_contains_router(), and tor_assert.

4.45.3.126 char* routerset_to_string (const routerset_t * *set*)

Return a new string that when parsed by routerset_parse_string() will yield **set**.

References smartlist_join_strings().

Referenced by entry_guards_prepend_from_config(), get_assigned_option(), and routerset_union().

4.45.3.127 void routerset_union (routerset_t * *target*, const routerset_t * *source*)

Add all members of the set **source** to **target**.

References routerset_t::list, routerset_parse(), routerset_to_string(), tor_assert, and tor_free.

Referenced by options_validate().

4.45.3.128 void routersets_get_disjunction (smartlist_t * *target*, const smartlist_t * *source*, const routerset_t * *include*, const routerset_t * *exclude*, int *running_only*)

Add to **target** every **routerinfo_t** (p. 206) from **source** except:

1) Don't add it if **include** is non-empty and the relay isn't in **include**; and 2) Don't add it if **exclude** is non-empty and the relay is excluded in a more specific fashion by **exclude**. 3) If **running_only**, don't add non-running routers.

References routerset_contains_router(), routerset_is_empty(), and smartlist_add().

Referenced by choose_good_exit_server_general().

4.45.3.129 routerstatus_t∗ routerstatus_sl_choose_by_bandwidth (smartlist_t ∗ *sl*, bandwidth_weight_rule_t *rule*)

Choose a random element of status list **sl**, weighted by the advertised bandwidth of each status.

References smartlist_choose_by_bandwidth(), and smartlist_choose_by_bandwidth_weights().

Referenced by router_pick_directory_server_impl().

4.45.3.130 static int signed_desc_append_to_journal (signed_descriptor_t ∗ *desc*, desc_store_t ∗ *store*) `[static]`

Add the **signed_descriptor_t** (p. 227) in **desc** to the router journal; change its saved_location to SAVED_IN_JOURNAL and set its offset appropriately.

References signed_descriptor_t::annotations_len, append_bytes_to_file(), desc_store_t::fname_base, get_datadir_fname_suffix, desc_store_t::journal_len, LD_FS, SAVED_IN_JOURNAL, signed_descriptor_t::saved_location, signed_descriptor_t::saved_offset, signed_descriptor_get_body_impl(), signed_descriptor_t::signed_descriptor_len, and tor_free.

Referenced by router_add_extrainfo_to_routerlist(), and router_add_to_routerlist().

4.45.3.131 static int signed_desc_digest_is_recognized (signed_descriptor_t ∗ *desc*) `[static]`

Return true iff any networkstatus includes a descriptor whose digest is that of **desc**.

References routerstatus_t::descriptor_digest, DIGEST_LEN, directory_caches_dir_info(), get_options(), signed_descriptor_t::identity_digest, networkstatus_get_latest_consensus(), networkstatus_get_v2_list(), networkstatus_v2_find_entry(), networkstatus_v2_list, networkstatus_vote_find_entry(), and signed_descriptor_t::signed_descriptor_digest.

Referenced by router_add_to_routerlist().

4.45.3.132 static void signed_descriptor_free (signed_descriptor_t ∗ *sd*) `[static]`

Release storage held by **sd**.

References signed_descriptor_t::signed_descriptor_body, and tor_free.

Referenced by routerlist_free(), and routerlist_remove_old().

4.45.3.133 static signed_descriptor_t∗ signed_descriptor_from_routerinfo (routerinfo_t ∗ *ri*) `[static]`

Extract a **signed_descriptor_t** (p. 227) from a routerinfo, and free the routerinfo.

References routerinfo_t::cache_info, routerinfo_free(), signed_descriptor_t::routerlist_index, and signed_descriptor_t::signed_descriptor_body.

Referenced by routerlist_insert_old(), routerlist_remove(), and routerlist_replace().

4.45.3.134 const char∗ signed_descriptor_get_annotations (signed_descriptor_t ∗ *desc*)

As **signed_descriptor_get_body()** (p. 1016), but points to the beginning of the annotations section rather than the beginning of the descriptor.

References signed_descriptor_get_body_impl().

Referenced by routerlist_reparse_old().

4.45.3.135 const char∗ signed_descriptor_get_body (signed_descriptor_t ∗ *desc*)

Return a pointer to the signed textual representation of a descriptor. The returned string is not guaranteed to be NUL-terminated: the string's length will be in desc->signed_descriptor_len.

The caller must not free the string returned.

References signed_descriptor_get_body_impl().

Referenced by connection_dirserv_add_servers_to_outbuf(), getinfo_helper_dir(), munge_extrainfo_into_routerinfo(), router_get_my_descriptor(), and router_rebuild_store().

4.45.3.136 static const char ∗ signed_descriptor_get_body_impl (signed_descriptor_t ∗ *desc*, int *with_annotations*) `[static]`

Return a pointer to the signed textual representation of a descriptor. The returned string is not guaranteed to be NUL-terminated: the string's length will be in desc->signed_descriptor_len.

If **with_annotations** is set, the returned string will include the annotations (if any) preceding the descriptor. This will increase the length of the string by desc->annotations_len.

The caller must not free the string returned.

References signed_descriptor_t::annotations_len, tor_mmap_t::data, desc_get_store(), escaped(), LD_DIR, desc_store_t::mmap, router_get_routerlist(), routerlist, SAVED_IN_CACHE, signed_descriptor_t::saved_location, signed_descriptor_t::saved_offset, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_len, and tor_assert.

Referenced by router_rebuild_store(), signed_desc_append_to_journal(), signed_descriptor_get_annotations(), and signed_descriptor_get_body().

4.45.3.137 static void∗ smartlist_choose_by_bandwidth (smartlist_t ∗ *sl*, bandwidth_weight_rule_t *rule*, int *statuses*) `[static]`

Helper function: choose a random element of smartlist **sl**, weighted by the advertised bandwidth of each element.

If **statuses** is zero, then **sl** is a list of routerinfo_t's. Otherwise it's a list of routerstatus_t's.

If **rule**==WEIGHT_FOR_EXIT. we're picking an exit node: consider all nodes' bandwidth equally regardless of their Exit status, since there may be some in the list because they exit to obscure ports. If **rule**==NO_WEIGHTING, we're picking a non-exit node: weight exit-node's bandwidth less depending on the smallness of the fraction of Exit-to-total bandwidth. If **rule**==WEIGHT_FOR_GUARD, we're picking a guard node: consider all guard's bandwidth equally. Otherwise, weight guards proportionally less.

References routerstatus_t::bandwidth, bandwidth_weight_rule_to_string(), routerinfo_t::cache_info, crypto_rand_uint64(), routerstatus_t::has_bandwidth, signed_descriptor_t::identity_digest, routerstatus_t::identity_digest, routerinfo_t::is_exit, routerstatus_t::is_exit, routerinfo_t::is_fast, routerstatus_t::is_fast, routerinfo_t::is_possible_guard, routerstatus_t::is_possible_guard, kb_to_bytes(), LD_BUG, LD_CIRC, router_digest_is_me(), router_get_advertised_bandwidth_capped(), router_get_by_digest(), router_get_consensus_status_by_id(), sl_last_total_weighted_bw, smartlist_choose(), tor_assert, tor_fragile_assert, and tor_free.

Referenced by routerlist_sl_choose_by_bandwidth(), and routerstatus_sl_choose_by_bandwidth().

4.45.3.138 static void∗ smartlist_choose_by_bandwidth_weights (smartlist_t ∗ *sl*, bandwidth_weight_rule_t *rule*, int *statuses*) [static]

Helper function: choose a random element of smartlist **sl**, weighted by the advertised bandwidth of each element using the consensus bandwidth weights.

If **statuses** is zero, then **sl** is a list of routerinfo_t's. Otherwise it's a list of routerstatus_t's.

If **rule**==WEIGHT_FOR_EXIT. we're picking an exit node: consider all nodes' bandwidth equally regardless of their Exit status, since there may be some in the list because they exit to obscure ports. If **rule**==NO_WEIGHTING, we're picking a non-exit node: weight exit-node's bandwidth less depending on the smallness of the fraction of Exit-to-total bandwidth. If **rule**==WEIGHT_FOR_GUARD, we're picking a guard node: consider all guard's bandwidth equally. Otherwise, weight guards proportionally less.

References routerstatus_t::bandwidth, bandwidth_weight_rule_to_string(), BW_WEIGHT_SCALE, routerinfo_t::cache_info, crypto_rand_uint64(), routerinfo_t::dir_port, routerstatus_t::dir_port, routerstatus_t::has_bandwidth, signed_descriptor_t::identity_digest, routerinfo_t::is_exit, routerstatus_t::is_exit, routerinfo_t::is_possible_guard, routerstatus_t::is_possible_guard, kb_to_bytes(), LD_BUG, LD_CIRC, networkstatus_get_bw_weight(), networkstatus_get_param(), router_get_advertised_bandwidth_capped(), router_get_consensus_status_by_id(), smartlist_choose(), tor_assert, tor_fragile_assert, and tor_free.

Referenced by routerlist_sl_choose_by_bandwidth(), and routerstatus_sl_choose_by_bandwidth().

4.45.3.139 static void trusted_dir_server_free (trusted_dir_server_t ∗ *ds*) [static]

Free storage held in **ds**.

References trusted_dir_server_t::address, trusted_dir_server_t::description, trusted_dir_server_t::nickname, and tor_free.

Referenced by clear_trusted_dir_servers(), and routerlist_free_all().

4.45.3.140 void trusted_dirs_flush_certs_to_disk (void)

Save all v3 key certificates to the cached-certs file.

References smartlist_add(), smartlist_create(), trusted_dir_certs, and trusted_dir_servers_certs_changed.

Referenced by trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

4.45.3.141 int trusted_dirs_load_certs_from_string (const char ∗ *contents*, int *from_store*, int *flush*)

Load a bunch of new key certificates from the string **contents**. If **from_store** is true, the certificates are from the cache, and we don't need to flush them to disk. If **flush** is true, we need to flush any changed certificates to disk now. Return 0 on success, -1 if any certs fail to parse.

References trusted_dir_server_t::addr, authority_cert_t::addr, trusted_dir_server_t::addr_current_at, trusted_dir_server_t::address, already_have_cert(), authority_cert_dl_failed(), authority_cert_free(), authority_cert_parse_from_string(), authority_cert_t::cache_info, cert_list_t::certs, DIGEST_LEN, trusted_dir_server_t::dir_port, authority_cert_t::dir_port, directory_caches_dir_info(), get_cert_list(), get_options(), hex_str(), signed_descriptor_t::identity_digest, LD_DIR, networkstatus_note_certs_arrived(), trusted_dir_server_t::nickname, signed_descriptor_t::published_on, authority_cert_t::signing_key_digest, smartlist_add(), tor_dup_ip(), tor_free, trusted_dir_servers_certs_changed, trusted_dirs_flush_certs_to_disk(), and trusteddirserver_get_by_v3_auth_digest().

Referenced by connection_dir_client_reached_eof(), dirvote_add_vote(), init_keys(), and trusted_dirs_reload_certs().

4.45.3.142 int trusted_dirs_reload_certs (void)

Reload the cached v3 key certificates from the cached-certs file in the data directory. Return 0 on success, -1 on failure.

References get_datadir_fname, read_file_to_str(), RFTS_IGNORE_MISSING, tor_free, and trusted_dirs_load_certs_from_string().

Referenced by do_main_loop().

4.45.3.143 static void trusted_dirs_remove_old_certs (void) `[static]`

Remove all v3 authority certificates that have been superseded for more than 48 hours. (If the most recent cert was published more than 48 hours ago, then we aren't going to get any consensuses signed with older keys.)

References authority_cert_free(), authority_cert_t::cache_info, ftime_definitely_after(), signed_descriptor_t::published_on, trusted_dir_certs, trusted_dir_servers_certs_changed, and trusted_dirs_flush_certs_to_disk().

Referenced by routerlist_remove_old_routers().

4.45.3.144 trusted_dir_server_t∗ trusteddirserver_get_by_v3_auth_digest (const char ∗ *digest*)

Return the **trusted_dir_server_t** (p. 247) for the directory authority whose v3 identity key hashes to **digest**, or NULL if no such authority is known.

References DIGEST_LEN, trusted_dir_servers, and V3_AUTHORITY.

Referenced by authority_certs_fetch_missing(), dirvote_add_vote(), networkstatus_check_consensus_signature(), trusted_dirs_load_certs_from_string(), and update_consensus_router_descriptor_downloads().

4.45.3.145 void update_consensus_router_descriptor_downloads (time_t *now*, int *is_vote*, networkstatus_t ∗ *consensus*)

For any descriptor that we want that's currently listed in **consensus**, download it as appropriate.

References authdir_mode(), DIGEST_LEN, digestmap_new(), directory_too_idle_to_fetch_descriptors(), trusted_dir_server_t::fake_status, get_options(), networkstatus_voter_info_t::identity_digest, LD_DIR, list_pending_descriptor_downloads(), router_get_by_descriptor_digest(), router_get_by_digest(), networkstatus_t::routerstatus_list, smartlist_add(), smartlist_create(), tor_assert, trusteddirserver_get_by_v3_auth_digest(), and networkstatus_t::voters.

Referenced by dirvote_add_vote(), and update_router_descriptor_downloads().

4.45.3.146 void update_extrainfo_downloads (time_t *now*)

Launch extrainfo downloads as needed.

References digestmap_free(), digestmap_get(), digestmap_new(), DIR_PURPOSE_FETCH_EXTRAINFO, download_status_is_ready(), or_options_t::DownloadExtraInfo, signed_descriptor_t::ei_dl_status, signed_descriptor_t::extra_info_digest, routerlist_t::extra_info_map, signed_descriptor_t::extrainfo_is_bogus, get_options(), signed_descriptor_t::identity_digest, initiate_descriptor_downloads(), signed_descriptor_t::is_extrainfo, LD_DIR, list_pending_descriptor_downloads(), MAX_DL_PER_REQUEST, MAX_ROUTERDESC_DOWNLOAD_FAILURES, routerlist_t::old_routers, PDS_NO_EXISTING_SERVERDESC_FETCH, PDS_RETRY_IF_NO_SERVERS, router_get_by_digest(), router_get_routerlist(), router_have_minimum_dir_info(), routerlist_t::routers, should_delay_dir_fetches(), smartlist_add(), smartlist_create(), smartlist_free(), smartlist_shuffle(), and tor_digest_is_zero().

Referenced by directory_info_has_arrived(), and run_scheduled_events().

4.45.3.147 static void update_router_descriptor_cache_downloads_v2 (time_t *now*) `[static]`

Launch downloads for router status as needed, using the strategy used by authorities and caches: based on the v2 networkstatuses we have, download every descriptor we don't have but would serve, from a random authority that lists it.

References authdir_mode(), digestmap_get(), digestmap_new(), directory_fetches_dir_info_early(), dirserv_would_reject_router(), get_options(), trusted_dir_server_t::is_running, LD_BUG, list_pending_descriptor_downloads(), MAX_NETWORKSTATUS_AGE, networkstatus_get_v2_list(), networkstatus_v2_list, router_get_by_descriptor_digest(), router_get_trusteddirserver_by_digest(), and smartlist_create().

Referenced by update_router_descriptor_downloads().

4.45.3.148 void update_router_descriptor_downloads (time_t *now*)

Launch downloads for router status as needed.

References or_options_t::Address, DIR_PURPOSE_FETCH_SERVERDESC, directory_fetches_dir_info_early(), directory_get_from_dirserver(), DUMMY_DOWNLOAD_INTERVAL, get_options(), last_routerdesc_download_attempted, networkstatus_get_reasonably_live_consensus(), PDS_RETRY_IF_NO_SERVERS, ROUTER_PURPOSE_-

GENERAL, server_mode(), should_delay_dir_fetches(), update_consensus_router_descriptor_downloads(), and update_router_descriptor_cache_downloads_v2().

Referenced by directory_info_has_arrived(), routerlist_retry_directory_downloads(), and run_scheduled_events().

4.45.3.149 static void update_router_have_minimum_dir_info (void) `[static]`

Change the value of have_min_dir_info, setting it true iff we have enough network and router information to build circuits. Clear the value of need_to_update_have_min_dir_info.

References control_event_bootstrap(), control_event_client_status(), count_usable_descriptors(), dir_info_status, directory_too_idle_to_fetch_descriptors(), get_options(), have_min_dir_info, LD_DIR, need_to_update_have_min_dir_info, networkstatus_get_latest_consensus(), networkstatus_get_reasonably_live_consensus(), should_delay_dir_fetches(), and tor_snprintf().

Referenced by router_have_minimum_dir_info().

4.45.4 Variable Documentation

4.45.4.1 char dir_info_status[128] = "" `[static]`

String describing what we're missing before we have enough directory info.

Referenced by update_router_have_minimum_dir_info().

4.45.4.2 int have_min_dir_info = 0 `[static]`

True iff, the last time we checked whether we had enough directory info to build circuits, the answer was "yes".

Referenced by update_router_have_minimum_dir_info().

4.45.4.3 time_t last_routerdesc_download_attempted = 0 `[static]`

The last time we tried to download any routerdesc, or 0 for "never". We use this to rate-limit download attempts when the number of routerdescs to download is low.

Referenced by launch_router_descriptor_downloads(), router_reset_descriptor_download_failures(), and update_router_descriptor_downloads().

4.45.4.4 int need_to_update_have_min_dir_info = 1 `[static]`

True iff enough has changed since the last time we checked whether we had enough directory info to build circuits that our old answer can no longer be trusted.

Referenced by update_router_have_minimum_dir_info().

4.45.4.5 routerlist_t* routerlist = NULL `[static]`

Global list of all of the routers that we know about.

Referenced by dump_routerlist_mem_usage(), extrainfo_get_by_descriptor_digest(), getinfo_helper_dir(), mark_all_trusteddirservers_up(), router_add_extrainfo_to_routerlist(), router_add_running_routers_to_smartlist(), router_add_to_routerlist(), router_exit_policy_all_routers_reject(), router_find_exact_exit_enclave(), router_get_by_descriptor_digest(), router_get_by_digest(), router_get_by_extrainfo_digest(), router_get_by_hexdigest(), router_get_by_nickname(), router_get_routerlist(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_pick_directory_server(), router_rebuild_store(), router_reset_descriptor_download_failures(), routerlist_add_network_family(), routerlist_find_my_routerinfo(), routerlist_free(), routerlist_free_all(), routerlist_remove_old_cached_routers_with_id(), routerlist_remove_old_routers(), and signed_descriptor_get_body_impl().

4.45.4.6 uint64_t sl_last_total_weighted_bw = 0 `[static]`

When we last computed the weights to use for bandwidths on directory requests, what were the total weighted bandwidth, and our share of that bandwidth? Used to determine what fraction of directory requests we should expect to see.

Referenced by router_get_my_share_of_directory_requests(), and smartlist_choose_by_bandwidth().

4.45.4.7 digestmap_t∗ trusted_dir_certs = NULL `[static]`

Map from v3 identity key digest to **cert_list_t** (p. 31).

Referenced by authority_cert_dl_failed(), authority_cert_get_all(), authority_cert_get_by_digests(), authority_cert_get_by_sk_digest(), authority_cert_get_newest_by_id(), routerlist_free_all(), trusted_dirs_flush_certs_to_disk(), and trusted_dirs_remove_old_certs().

4.45.4.8 smartlist_t∗ trusted_dir_servers = NULL `[static]`

Global list of a **trusted_dir_server_t** (p. 247) object for each trusted directory server.

Referenced by add_trusted_dir_server(), any_trusted_dir_is_v1_authority(), authority_certs_fetch_missing(), clear_trusted_dir_servers(), router_addr_is_trusted_dir(), router_digest_is_trusted_dir_type(), router_get_trusted_dir_servers(), router_get_trusteddirserver_by_digest(), router_pick_trusteddirserver_impl(), router_set_status(), routerlist_free_all(), trusteddirserver_get_by_v3_auth_digest(), and update_v2_networkstatus_cache_downloads().

4.45.4.9 int trusted_dir_servers_certs_changed = 0 `[static]`

True iff any key certificate in at least one member of **trusted_dir_certs** has changed since we last flushed the certificates to disk.

Referenced by trusted_dirs_flush_certs_to_disk(), trusted_dirs_load_certs_from_string(), and trusted_dirs_remove_old_certs().

4.45.4.10 smartlist_t∗ warned_nicknames = NULL `[static]`

List of strings for nicknames we've already warned about and that are still unknown / unavailable.

Referenced by add_nickname_list_to_smartlist(), routerlist_free_all(), routerlist_reset_warnings(), and routerset_get_all_routers().

4.46 routerparse.c File Reference

Code to parse and validate router descriptors and directories.

`#include "or.h"`

`#include "memarea.h"`

`#include <math.h>`

Data Structures

- struct **directory_token_t**
- struct **token_rule_t**

Defines

- #define **MIN_ANNOTATION** A_PURPOSE
- #define **MAX_ANNOTATION** _A_UNKNOWN
- #define **AT_START** 1
- #define **AT_END** 2
- #define **END_OF_TABLE** { NULL, _NIL, 0,0,0, NO_OBJ, 0, INT_MAX, 0, 0 }
- #define **T**(s, t, a, o) { s, t, a, o, 0, INT_MAX, 0, 0 }
- #define **T0N**(s, t, a, o) { s, t, a, o, 0, INT_MAX, 0, 0 }
- #define **T1**(s, t, a, o) { s, t, a, o, 1, 1, 0, 0 }
- #define **T1_START**(s, t, a, o) { s, t, a, o, 1, 1, AT_START, 0 }
- #define **T1_END**(s, t, a, o) { s, t, a, o, 1, 1, AT_END, 0 }
- #define **T1N**(s, t, a, o) { s, t, a, o, 1, INT_MAX, 0, 0 }
- #define **T01**(s, t, a, o) { s, t, a, o, 0, 1, 0, 0 }
- #define **A01**(s, t, a, o) { s, t, a, o, 0, 1, 0, 1 }
- #define **ARGS** 0,INT_MAX,0
- #define **NO_ARGS** 0,0,0
- #define **CONCAT_ARGS** 1,1,1
- #define **GE**(n) n,INT_MAX,0
- #define **EQ**(n) n,n,0
- #define **CERTIFICATE_MEMBERS**
- #define **find_by_keyword**(s, keyword) _find_by_keyword((s), (keyword), #keyword)
- #define **TS_ANNOTATIONS_OK** 1
- #define **TS_NOCHECK** 2
- #define **TS_NO_NEW_ANNOTATIONS** 4
- #define **CST_CHECK_AUTHORITY** (1<<0)
- #define **CST_NO_CHECK_OBJTYPE** (1<<1)
- #define **DUMP_AREA**(a, name) STMT_NIL
- #define **ALLOC_ZERO**(sz) memarea_alloc_zero(area,sz)
- #define **ALLOC**(sz) memarea_alloc(area,sz)
- #define **STRDUP**(str) memarea_strdup(area,str)
- #define **STRNDUP**(str, n) memarea_strndup(area,(str),(n))
- #define **RET_ERR**(msg)
- #define **MAX_ARGS** 512
- #define **CHECK_LENGTH**()
- #define **NEXT_LINE**()

Enumerations

- enum **directory_keyword** {

  K_ACCEPT = 0, **K_ACCEPT6**, **K_DIRECTORY_SIGNATURE**, **K_RECOMMENDED_SOFTWARE**,

  K_REJECT, **K_REJECT6**, **K_ROUTER**, **K_SIGNED_DIRECTORY**,

  K_SIGNING_KEY, **K_ONION_KEY**, **K_ROUTER_SIGNATURE**, **K_PUBLISHED**,

  K_RUNNING_ROUTERS, **K_ROUTER_STATUS**, **K_PLATFORM**, **K_OPT**,

  K_BANDWIDTH, **K_CONTACT**, **K_NETWORK_STATUS**, **K_UPTIME**,

  K_DIR_SIGNING_KEY, **K_FAMILY**, **K_FINGERPRINT**, **K_HIBERNATING**,

  K_READ_HISTORY, **K_WRITE_HISTORY**, **K_NETWORK_STATUS_VERSION**, **K_DIR_SOURCE**,

  K_DIR_OPTIONS, **K_CLIENT_VERSIONS**, **K_SERVER_VERSIONS**, **K_P**,

  K_R, **K_S**, **K_V**, **K_W**,

  K_M, **K_EVENTDNS**, **K_EXTRA_INFO**, **K_EXTRA_INFO_DIGEST**,

  K_CACHES_EXTRA_INFO, **K_HIDDEN_SERVICE_DIR**, **K_ALLOW_SINGLE_HOP_EXITS**, **K_DIRREQ_END**,

  K_DIRREQ_V2_IPS, **K_DIRREQ_V3_IPS**, **K_DIRREQ_V2_REQS**, **K_DIRREQ_V3_REQS**,

  K_DIRREQ_V2_SHARE, **K_DIRREQ_V3_SHARE**, **K_DIRREQ_V2_RESP**, **K_DIRREQ_V3_RESP**,

  K_DIRREQ_V2_DIR, **K_DIRREQ_V3_DIR**, **K_DIRREQ_V2_TUN**, **K_DIRREQ_V3_TUN**,

  K_ENTRY_END, **K_ENTRY_IPS**, **K_CELL_END**, **K_CELL_PROCESSED**,

  K_CELL_QUEUED, **K_CELL_TIME**, **K_CELL_CIRCS**, **K_EXIT_END**,

  K_EXIT_WRITTEN, **K_EXIT_READ**, **K_EXIT_OPENED**, **K_DIR_KEY_CERTIFICATE_VERSION**,

  K_DIR_IDENTITY_KEY, **K_DIR_KEY_PUBLISHED**, **K_DIR_KEY_EXPIRES**, **K_DIR_KEY_CERTIFICATION**,

  K_DIR_KEY_CROSSCERT, **K_DIR_ADDRESS**, **K_VOTE_STATUS**, **K_VALID_AFTER**,

  K_FRESH_UNTIL, **K_VALID_UNTIL**, **K_VOTING_DELAY**, **K_KNOWN_FLAGS**,

  K_PARAMS, **K_BW_WEIGHTS**, **K_VOTE_DIGEST**, **K_CONSENSUS_DIGEST**,

  K_ADDITIONAL_DIGEST, **K_ADDITIONAL_SIGNATURE**, **K_CONSENSUS_METHODS**, **K_CONSENSUS_METHOD**,

  K_LEGACY_DIR_KEY, **K_DIRECTORY_FOOTER**, **A_PURPOSE**, **A_LAST_LISTED**,

  _A_UNKNOWN, **R_RENDEZVOUS_SERVICE_DESCRIPTOR**, **R_VERSION**, **R_PERMANENT_KEY**,

R_SECRET_ID_PART, **R_PUBLICATION_TIME**, **R_PROTOCOL_VERSIONS**, **R_INTRODUCTION_POINTS**,

R_SIGNATURE, **R_IPO_IDENTIFIER**, **R_IPO_IP_ADDRESS**, **R_IPO_ONION_PORT**,

R_IPO_ONION_KEY, **R_IPO_SERVICE_KEY**, **C_CLIENT_NAME**, **C_DESCRIPTOR_COOKIE**,

C_CLIENT_KEY, **_ERR**, **_EOF**, **_NIL** }

- enum **obj_syntax** {

NO_OBJ, **NEED_OBJ**, **NEED_SKEY_1024**, **NEED_KEY_1024**,

NEED_KEY, **OBJ_OK** }

Functions

- static int **router_add_exit_policy** (**routerinfo_t** ∗router, **directory_token_t** ∗tok)
- static **addr_policy_t** ∗ **router_parse_addr_policy** (**directory_token_t** ∗tok)
- static **addr_policy_t** ∗ **router_parse_addr_policy_private** (**directory_token_t** ∗tok)
- static int **router_get_hash_impl** (const char ∗s, size_t s_len, char ∗digest, const char ∗start_str, const char ∗end_str, char end_char, digest_algorithm_t alg)
- static int **router_get_hashes_impl** (const char ∗s, size_t s_len, **digests_t** ∗digests, const char ∗start_str, const char ∗end_str, char end_char)
- static void **token_clear** (**directory_token_t** ∗tok)
- static **smartlist_t** ∗ **find_all_exitpolicy** (**smartlist_t** ∗s)
- static **directory_token_t** ∗ **_find_by_keyword** (**smartlist_t** ∗s, **directory_keyword** keyword, const char ∗keyword_str)
- static **directory_token_t** ∗ **find_opt_by_keyword** (**smartlist_t** ∗s, **directory_keyword** keyword)
- static int **tokenize_string** (**memarea_t** ∗area, const char ∗start, const char ∗end, **smartlist_t** ∗out, **token_rule_t** ∗table, int flags)
- static **directory_token_t** ∗ **get_next_token** (**memarea_t** ∗area, const char ∗∗s, const char ∗eos, **token_rule_t** ∗table)
- static int **check_signature_token** (const char ∗digest, ssize_t digest_len, **directory_token_t** ∗tok, **crypto_pk_env_t** ∗pkey, int flags, const char ∗doctype)
- static **crypto_pk_env_t** ∗ **find_dir_signing_key** (const char ∗str, const char ∗eos)
- static int **tor_version_same_series** (**tor_version_t** ∗a, **tor_version_t** ∗b)
- static void **dump_desc** (const char ∗desc, const char ∗type)
- int **router_get_dir_hash** (const char ∗s, char ∗digest)
- int **router_get_router_hash** (const char ∗s, size_t s_len, char ∗digest)
- int **router_get_runningrouters_hash** (const char ∗s, char ∗digest)
- int **router_get_networkstatus_v2_hash** (const char ∗s, char ∗digest)
- int **router_get_networkstatus_v3_hashes** (const char ∗s, **digests_t** ∗digests)
- int **router_get_networkstatus_v3_hash** (const char ∗s, char ∗digest, digest_algorithm_t alg)
- int **router_get_extrainfo_hash** (const char ∗s, char ∗digest)
- int **router_append_dirobj_signature** (char ∗buf, size_t buf_len, const char ∗digest, size_t digest_len, **crypto_pk_env_t** ∗private_key)
- **version_status_t** **tor_version_is_obsolete** (const char ∗myversion, const char ∗versionlist)
- int **router_parse_directory** (const char ∗str)

- int **router_parse_runningrouters** (const char *str)
- static int **dir_signing_key_is_trusted** (**crypto_pk_env_t** *key)
- static int **find_start_of_next_router_or_extrainfo** (const char **s_ptr, const char *eos, int *is_extrainfo_out)
- int **router_parse_list_from_string** (const char **s, const char *eos, **smartlist_t** *dest, **saved_location_t** saved_location, int want_extrainfo, int allow_annotations, const char *prepend_annotations)
- void **dump_distinct_digest_count** (int severity)
- **routerinfo_t** * **router_parse_entry_from_string** (const char *s, const char *end, int cache_copy, int allow_annotations, const char *prepend_annotations)
- **extrainfo_t** * **extrainfo_parse_entry_from_string** (const char *s, const char *end, int cache_copy, struct digest_ri_map_t *routermap)
- **authority_cert_t** * **authority_cert_parse_from_string** (const char *s, const char **end_of_string)
- static INLINE const char * **find_start_of_next_routerstatus** (const char *s)
- static **routerstatus_t** * **routerstatus_parse_entry_from_string** (**memarea_t** *area, const char **s, **smartlist_t** *tokens, **networkstatus_t** *vote, **vote_routerstatus_t** *vote_rs, int consensus_method, **consensus_flavor_t** flav)
- int **compare_routerstatus_entries** (const void **_a, const void **_b)
- static void **_free_duplicate_routerstatus_entry** (void *e)
- **networkstatus_v2_t** * **networkstatus_v2_parse_from_string** (const char *s)
- int **networkstatus_verify_bw_weights** (**networkstatus_t** *ns)
- **networkstatus_t** * **networkstatus_parse_vote_from_string** (const char *s, const char **eos_out, **networkstatus_type_t** ns_type)
- static **digests_t** * **detached_get_digests** (**ns_detached_signatures_t** *sigs, const char *flavor_name)
- static **smartlist_t** * **detached_get_signatures** (**ns_detached_signatures_t** *sigs, const char *flavor_name)
- **ns_detached_signatures_t** * **networkstatus_parse_detached_signatures** (const char *s, const char *eos)
- **addr_policy_t** * **router_parse_addr_policy_item_from_string** (const char *s, int assume_action)
- void **assert_addr_policy_ok** (**smartlist_t** *lst)
- static INLINE **directory_token_t** * **token_check_object** (**memarea_t** *area, const char *kwd, **directory_token_t** *tok, **obj_syntax** o_syn)
- static INLINE int **get_token_arguments** (**memarea_t** *area, **directory_token_t** *tok, const char *s, const char *eol)
- static int **router_get_hash_impl_helper** (const char *s, size_t s_len, const char *start_str, const char *end_str, char end_c, const char **start_out, const char **end_out)
- static const char * **find_start_of_next_microdesc** (const char *s, const char *eos)
- **smartlist_t** * **microdescs_parse_from_string** (const char *s, const char *eos, int allow_annotations, int copy_body)
- int **tor_version_as_new_as** (const char *platform, const char *cutoff)
- int **tor_version_parse** (const char *s, **tor_version_t** *out)
- int **tor_version_compare** (**tor_version_t** *a, **tor_version_t** *b)
- static int **_compare_tor_version_str_ptr** (const void **_a, const void **_b)
- void **sort_version_list** (**smartlist_t** *versions, int remove_duplicates)
- int **rend_parse_v2_service_descriptor** (**rend_service_descriptor_t** **parsed_out, char *desc_id_out, char **intro_points_encrypted_out, size_t *intro_points_encrypted_size_out, size_t *encoded_size_out, const char **next_out, const char *desc)

- int **rend_decrypt_introduction_points** (char ∗∗ipos_decrypted, size_t ∗ipos_decrypted_size, const char ∗descriptor_cookie, const char ∗ipos_encrypted, size_t ipos_encrypted_size)
- int **rend_parse_introduction_points** (**rend_service_descriptor_t** ∗parsed, const char ∗intro_points_encoded, size_t intro_points_encoded_size)
- int **rend_parse_client_keys** (strmap_t ∗parsed_clients, const char ∗ckstr)

Variables

- static **token_rule_t routerdesc_token_table** []
- static **token_rule_t extrainfo_token_table** []
- static **token_rule_t rtrstatus_token_table** []
- static **token_rule_t netstatus_token_table** []
- static **token_rule_t dir_footer_token_table** []
- static **token_rule_t dir_token_table** []
- static **token_rule_t dir_key_certificate_table** []
- static **token_rule_t desc_token_table** []
- static **token_rule_t ipo_token_table** []
- static **token_rule_t client_keys_token_table** []
- static **token_rule_t networkstatus_token_table** []
- static **token_rule_t networkstatus_consensus_token_table** []
- static **token_rule_t networkstatus_vote_footer_token_table** []
- static **token_rule_t networkstatus_detached_signature_token_table** []
- static **token_rule_t microdesc_token_table** []
- static time_t **last_desc_dumped** = 0

4.46.1 Detailed Description

Code to parse and validate router descriptors and directories.

4.46.2 Define Documentation

4.46.2.1 #define A01(s, t, a, o) { s, t, a, o, 0, 1, 0, 1 }

An annotation that must appear no more than once

4.46.2.2 #define CERTIFICATE_MEMBERS

Value:

```
T1("dir-key-certificate-version", K_DIR_KEY_CERTIFICATE_VERSION,           \
                                                     GE(1),       NO_OBJ ),  \
  T1("dir-identity-key", K_DIR_IDENTITY_KEY,         NO_ARGS,     NEED_KEY ),\
  T1("dir-key-published",K_DIR_KEY_PUBLISHED,        CONCAT_ARGS, NO_OBJ),   \
  T1("dir-key-expires",  K_DIR_KEY_EXPIRES,          CONCAT_ARGS, NO_OBJ),   \
  T1("dir-signing-key",  K_DIR_SIGNING_KEY,          NO_ARGS,     NEED_KEY ),\
  T01("dir-key-crosscert", K_DIR_KEY_CROSSCERT,       NO_ARGS,    NEED_OBJ ),\
  T1("dir-key-certification", K_DIR_KEY_CERTIFICATION,                       \
                                                     NO_ARGS,     NEED_OBJ), \
  T01("dir-address",     K_DIR_ADDRESS,              GE(1),       NO_OBJ),
```

List of tokens common to V3 authority certificates and V3 consensuses.

4.46.2.3 #define CHECK_LENGTH()

Value:

```
STMT_BEGIN \
    if (s+32 > eos)              \
      return NULL;               \
  STMT_END
```

4.46.2.4 #define END_OF_TABLE { NULL, _NIL, 0,0,0, NO_OBJ, 0, INT_MAX, 0, 0 }

Appears to indicate the end of a table.

4.46.2.5 #define NEXT_LINE()

Value:

```
STMT_BEGIN            \
    s = memchr(s, '\n', eos-s);          \
    if (!s || s+1 >= eos)                \
      return NULL;                       \
    s++;                                 \
  STMT_END
```

4.46.2.6 #define RET_ERR(msg)

Value:

```
STMT_BEGIN                                                       \
    if (tok) token_clear(tok);                                     \
    tok = ALLOC_ZERO(sizeof(directory_token_t));                  \
    tok->tp = _ERR;                                               \
    tok->error = STRDUP(msg);                                     \
    goto done_tokenizing;                                         \
  STMT_END
```

4.46.2.7 #define T(s, t, a, o) { s, t, a, o, 0, INT_MAX, 0, 0 }

An item with no restrictions: used for obsolete document types

Referenced by networkstatus_verify_bw_weights().

4.46.2.8 #define T01(s, t, a, o) { s, t, a, o, 0, 1, 0, 0 }

An item that must appear no more than once

4.46.2.9 #define T0N(s, t, a, o) { s, t, a, o, 0, INT_MAX, 0, 0 }

An item with no restrictions on multiplicity or location.

4.46.2.10 #define T1(s, t, a, o) { s, t, a, o, 1, 1, 0, 0 }

An item that must appear exactly once

4.46.2.11 #define T1_END(s, t, a, o) { s, t, a, o, 1, 1, AT_END, 0 }

An item that must appear exactly once, at the end of the document

4.46.2.12 #define T1_START(s, t, a, o) { s, t, a, o, 1, 1, AT_START, 0 }

An item that must appear exactly once, at the start of the document

4.46.2.13 #define T1N(s, t, a, o) { s, t, a, o, 1, INT_MAX, 0, 0 }

An item that must appear one or more times

4.46.3 Enumeration Type Documentation

4.46.3.1 enum directory_keyword

Enumeration of possible token types. The ones starting with K_ correspond to directory 'keywords'. _ERR is an error in the tokenizing process, _EOF is an end-of-file marker, and _NIL is used to encode not-a-token.

4.46.3.2 enum obj_syntax

We use a table of rules to decide how to parse each token type. Rules for whether the keyword needs an object.

Enumerator:

NO_OBJ No object, ever.

NEED_OBJ Object is required.

NEED_SKEY_1024 Object is required, and must be a 1024 bit private key

NEED_KEY_1024 Object is required, and must be a 1024 bit public key

NEED_KEY Object is required, and must be a public key.

OBJ_OK Object is optional.

4.46.4 Function Documentation

4.46.4.1 static int _compare_tor_version_str_ptr (const void ** *_a*, const void ** *_b*) `[static]`

Helper: Given pointers to two strings describing tor versions, return -1 if _a precedes _b, 1 if _b precedes _a, and 0 if they are equivalent. Used to sort a list of versions.

References tor_version_compare(), and tor_version_parse().

Referenced by sort_version_list().

4.46.4.2 static directory_token_t * _find_by_keyword (smartlist_t * *s*, directory_keyword *keyword*, const char * *keyword_as_string*) [static]

Find the first token in **s** whose keyword is **keyword**; fail with an assert if no such keyword is found.

References find_opt_by_keyword(), LD_BUG, and tor_assert.

4.46.4.3 static void _free_duplicate_routerstatus_entry (void * *e*) [static]

Helper: used in call to _smartlist_uniq to clear out duplicate entries.

References LD_DIR, and routerstatus_free().

Referenced by networkstatus_v2_parse_from_string().

4.46.4.4 void assert_addr_policy_ok (smartlist_t * *lst*)

Log and exit if **t** is malformed

References tor_assert.

4.46.4.5 authority_cert_t* authority_cert_parse_from_string (const char * *s*, const char ** *end_of_string*)

Parse a key certificate from **s**; point **end-of-string** to the first character after the certificate.

References authority_cert_t::addr, directory_token_t::args, authority_cert_free(), authority_cert_get_by_digests(), base16_decode(), authority_cert_t::cache_info, check_signature_token(), crypto_pk_get_digest(), DIGEST_LEN, authority_cert_t::dir_port, dump_desc(), eat_whitespace(), escaped(), authority_cert_t::expires, find_opt_by_keyword(), signed_descriptor_t::identity_digest, authority_cert_t::identity_key, authority_cert_t::is_cross_certified, directory_token_t::key, LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, parse_addr_port(), parse_iso_time(), signed_descriptor_t::published_on, router_get_hash_impl(), signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, authority_cert_t::signing_key, authority_cert_t::signing_key_digest, smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, tor_free, tor_inet_aton(), and directory_token_t::tp.

Referenced by load_authority_keyset(), networkstatus_parse_vote_from_string(), and trusted_dirs_load_certs_from_string().

4.46.4.6 static int check_signature_token (const char * *digest*, ssize_t *digest_len*, directory_token_t * *tok*, crypto_pk_env_t * *pkey*, int *flags*, const char * *doctype*) [static]

Check whether the object body of the token in **tok** has a good signature for **digest** using key **pkey**. If **CST_CHECK_AUTHORITY** is set, make sure that **pkey** is the key of a directory authority. If **CST_NO_CHECK_OBJTYPE** is set, do not check the object type of the signature object. Use **doctype** as the type of the document when generating log messages. Return 0 on success, negative on failure.

References crypto_pk_public_checksig(), dir_signing_key_is_trusted(), LD_DIR, directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, tor_-

assert, and tor_free.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), rend_parse_v2_service_descriptor(), router_parse_directory(), router_parse_entry_from_string(), and router_parse_runningrouters().

4.46.4.7 int compare_routerstatus_entries (const void ** _*a*, const void ** _*b*)

Helper to sort a smartlist of pointers to **routerstatus_t** (p. 218)

References DIGEST_LEN, and routerstatus_t::identity_digest.

Referenced by dirserv_read_measured_bandwidths(), and networkstatus_v2_parse_from_string().

4.46.4.8 static digests_t* detached_get_digests (ns_detached_signatures_t * *sigs*, const char * *flavor_name*) [static]

Return the **digests_t** (p. 77) that holds the digests of the **flavor_name**-flavored networkstatus according to the detached signatures document **sigs**, allocating a new **digests_t** (p. 77) as neeeded.

References ns_detached_signatures_t::digests, strmap_get(), and strmap_set().

Referenced by networkstatus_parse_detached_signatures().

4.46.4.9 static smartlist_t* detached_get_signatures (ns_detached_signatures_t * *sigs*, const char * *flavor_name*) [static]

Return the list of signatures of the **flavor_name**-flavored networkstatus according to the detached signatures document **sigs**, allocating a new **digests_t** (p. 77) as neeeded.

References ns_detached_signatures_t::signatures, smartlist_create(), strmap_get(), and strmap_set().

Referenced by networkstatus_parse_detached_signatures().

4.46.4.10 static int dir_signing_key_is_trusted (crypto_pk_env_t * *key*) [static]

Return true iff **key** is allowed to sign directories.

References crypto_pk_get_digest(), DIGEST_LEN, and LD_DIR.

Referenced by check_signature_token().

4.46.4.11 static void dump_desc (const char * *desc*, const char * *type*) [static]

For debugging purposes, dump unparseable descriptor ***desc** of type ***type** to file $DATADIR/unparseable-desc. Do not write more than one descriptor to disk per minute. If there is already such a file in the data directory, overwrite it.

References get_datadir_fname, last_desc_dumped, LD_DIR, tor_assert, tor_free, tor_snprintf(), and write_str_to_file().

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

4.46.4.12 void dump_distinct_digest_count (int *severity*)

Log the total count of the number of distinct router digests we've ever verified. When compared to the number of times we've verified routerdesc signatures *in toto*, this will tell us if we're doing too much multiple-verification.

References digestmap_new(), digestmap_size(), and LD_GENERAL.

Referenced by dumpstats().

4.46.4.13 extrainfo_t∗ extrainfo_parse_entry_from_string (const char ∗ *s*, const char ∗ *end*, int *cache_copy*, struct digest_ri_map_t ∗ *routermap*)

Parse a single extrainfo entry from the string **s**, ending at **end**. (If **end** is NULL, parse up to the end of **s**.) If **cache_copy** is true, make a copy of the extra-info document in the cache_info fields of the result. If **routermap** is provided, use it as a map from router identity to **routerinfo_t** (p. 206) when looking up signing keys.

References directory_token_t::args, base16_decode(), routerinfo_t::cache_info, extrainfo_t::cache_info, check_signature_token(), DIGEST_LEN, digestmap_get(), dump_desc(), escaped(), extrainfo_free(), HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, signed_descriptor_t::is_extrainfo, is_legal_nickname(), LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, extrainfo_t::nickname, note_crypto_pk_op(), directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, parse_iso_time(), extrainfo_t::pending_sig, extrainfo_t::pending_sig_len, signed_descriptor_t::published_on, router_get_extrainfo_hash(), signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, and directory_token_t::tp.

Referenced by extrainfo_dump_to_string(), and router_parse_list_from_string().

4.46.4.14 static smartlist_t ∗ find_all_exitpolicy (smartlist_t ∗ *s*) `[static]`

Return a newly allocated smartlist of all accept or reject tokens in **s**.

References smartlist_add(), and smartlist_create().

Referenced by router_parse_entry_from_string().

4.46.4.15 static crypto_pk_env_t ∗ find_dir_signing_key (const char ∗ *str*, const char ∗ *eos*) `[static]`

Given a directory or running-routers string in **str**, try to find the its dir-signing-key token (if any). If this token is present, extract and return the key. Return NULL on failure.

References get_next_token(), directory_token_t::key, LD_DIR, memarea_drop_all(), memarea_new(), token_clear(), tor_assert, and directory_token_t::tp.

Referenced by router_parse_directory(), and router_parse_runningrouters().

4.46.4.16 static directory_token_t * find_opt_by_keyword (smartlist_t * *s*, directory_keyword *keyword*) [static]

Find the first token in **s** whose keyword is **keyword**; return NULL if no such keyword is found.

Referenced by _find_by_keyword(), authority_cert_parse_from_string(), microdescs_parse_from_string(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), rend_parse_client_keys(), rend_parse_v2_service_descriptor(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

4.46.4.17 static const char* find_start_of_next_microdesc (const char * *s*, const char * *eos*) [static]

Assuming that s starts with a microdesc, return the start of the *NEXT* one. Return NULL on "not found."

References eat_whitespace_eos(), and strcmpstart().

Referenced by microdescs_parse_from_string().

4.46.4.18 static int find_start_of_next_router_or_extrainfo (const char ** *s_ptr*, const char * *eos*, int * *is_extrainfo_out*) [static]

Helper: move ***s_ptr** ahead to the next router, the next extra-info, or to the first of the annotations proceeding the next router or extra-info—whichever comes first. Set **is_extrainfo_out** to true if we found an extrainfo, or false if found a router. Do not scan beyond **eos**. Return -1 if we found nothing; 0 if we found something.

References eat_whitespace_eos(), strcmpstart(), and tor_assert.

Referenced by router_parse_list_from_string().

4.46.4.19 static INLINE const char* find_start_of_next_routerstatus (const char * *s*) [static]

Helper: given a string **s**, return the start of the next router-status object (starting with "r " at the start of a line). If none is found, return the start of the directory footer, or the next directory signature. If none is found, return the end of the string.

Referenced by networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), and routerstatus_parse_entry_from_string().

4.46.4.20 static directory_token_t * get_next_token (memarea_t * *area*, const char ** *s*, const char * *eos*, token_rule_t * *table*) [static]

Helper function: read the next token from *s, advance *s to the end of the token, and return the parsed token. Parse ***s** according to the list of tokens in **table**.

References directory_token_t::args, base64_decode(), crypto_new_pk_env(), crypto_pk_read_private_key_from_string(), crypto_pk_read_public_key_from_string(), eat_whitespace_eos(), eat_whitespace_eos_no_nl(), find_whitespace_eos(), get_token_-

arguments(), directory_token_t::key, token_rule_t::max_args, token_rule_t::min_args, directory_token_t::n_args, NO_OBJ, OBJ_OK, directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, token_rule_t::os, strcmp_len(), strcmpstart(), token_rule_t::t, token_check_object(), tor_assert, tor_snprintf(), directory_token_t::tp, and token_rule_t::v.

Referenced by find_dir_signing_key(), router_parse_addr_policy_item_from_string(), and tokenize_string().

4.46.4.21 static INLINE int get_token_arguments (memarea_t ∗ *area*, directory_token_t ∗ *tok*, const char ∗ *s*, const char ∗ *eol*) `[static]`

Helper: parse space-separated arguments from the string **s** ending at **eol**, and store them in the args field of **tok**. Store the number of parsed elements into the n_args field of **tok**. Allocate all storage in **area**. Return the number of arguments parsed, or return -1 if there was an insanely high number of arguments.

Largest number of arguments we'll accept to any token, ever.

References directory_token_t::args, eat_whitespace(), find_whitespace(), memarea_memdup(), memarea_strndup(), and directory_token_t::n_args.

Referenced by get_next_token().

4.46.4.22 smartlist_t∗ microdescs_parse_from_string (const char ∗ *s*, const char ∗ *eos*, int *allow_annotations*, int *copy_body*)

Parse as many microdescriptors as are found from the string starting at **s** and ending at **eos**. If allow_annotations is set, read any annotations we recognize and ignore ones we don't. If **copy_body** is true, then strdup the bodies of the microdescriptors. Return all newly parsed microdescriptors in a newly allocated **smartlist_t** (p. 232).

References directory_token_t::args, microdesc_t::body, microdesc_t::bodylen, microdesc_t::digest, eat_whitespace_eos(), escaped(), microdesc_t::exitsummary, microdesc_t::family, find_opt_by_keyword(), find_start_of_next_microdesc(), is_legal_nickname_or_hexdigest(), directory_token_t::key, microdesc_t::last_listed, LD_DIR, memarea_clear(), memarea_drop_all(), memarea_new(), directory_token_t::n_args, microdesc_t::off, microdesc_t::onion_pkey, parse_iso_time(), smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), tokenize_string(), and tor_assert.

Referenced by dirvote_create_microdescriptor(), and microdescs_add_to_cache().

4.46.4.23 ns_detached_signatures_t∗ networkstatus_parse_detached_signatures (const char ∗ *s*, const char ∗ *eos*)

Parse a detached v3 networkstatus signature document between **s** and **eos** and return the result. Return -1 on failure.

References directory_token_t::args, base16_decode(), crypto_digest_algorithm_parse_name(), digests_t::d, detached_get_digests(), detached_get_signatures(), DIGEST256_LEN, DIGEST_LEN, ns_detached_signatures_t::digests, escaped(), ns_detached_signatures_t::fresh_until, HEX_DIGEST256_LEN, HEX_DIGEST_LEN, LD_DIR, memarea_new(), directory_token_t::n_args, directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, parse_iso_time(), ns_detached_signatures_t::signatures, smartlist_add(), smartlist_create(), tokenize_string(), tor_assert, tor_free, tor_mem_is_-

zero(), directory_token_t::tp, ns_detached_signatures_t::valid_after, and ns_detached_signatures_t::valid_until.

4.46.4.24 networkstatus_t* networkstatus_parse_vote_from_string (const char * *s*, const char ** *eos_out*, networkstatus_type_t *ns_type*)

Parse a v3 networkstatus vote, opinion, or consensus (depending on ns_type), from **s**, and return the result. Return NULL on failure.

References networkstatus_voter_info_t::addr, networkstatus_voter_info_t::address, document_signature_t::alg, directory_token_t::args, authority_cert_parse_from_string(), base16_decode(), authority_cert_t::cache_info, networkstatus_t::cert, check_signature_token(), networkstatus_t::client_versions, networkstatus_t::consensus_method, networkstatus_voter_info_t::contact, crypto_digest_algorithm_parse_name(), digests_t::d, DIGEST_LEN, networkstatus_t::digests, networkstatus_voter_info_t::dir_port, networkstatus_t::dist_seconds, document_signature_free(), dump_desc(), escaped(), find_opt_by_keyword(), find_start_of_next_routerstatus(), networkstatus_t::flavor, networkstatus_t::fresh_until, document_signature_t::good_signature, HEX_DIGEST_LEN, document_signature_t::identity_digest, routerstatus_t::identity_digest, signed_descriptor_t::identity_digest, networkstatus_voter_info_t::identity_digest, networkstatus_t::known_flags, LD_DIR, networkstatus_voter_info_t::legacy_id_digest, memarea_drop_all(), memarea_new(), MIN_DIST_SECONDS, MIN_VOTE_INTERVAL, MIN_VOTE_SECONDS, directory_token_t::n_args, networkstatus_t::net_params, networkstatus_get_voter_by_id(), networkstatus_parse_flavor_name(), networkstatus_vote_free(), networkstatus_voter_info_t::nickname, directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, networkstatus_voter_info_t::or_port, parse_iso_time(), networkstatus_t::published, router_get_networkstatus_v3_hashes(), networkstatus_t::routerstatus_list, routerstatus_parse_entry_from_string(), networkstatus_t::server_versions, document_signature_t::signature, document_signature_t::signature_len, authority_cert_t::signing_key, document_signature_t::signing_key_digest, networkstatus_voter_info_t::sigs, smartlist_add(), smartlist_create(), smartlist_free(), vote_routerstatus_t::status, strcmpstart(), networkstatus_t::supported_methods, token_clear(), tokenize_string(), tor_assert, tor_digest_is_zero(), tor_free, tor_inet_aton(), tor_parse_long(), directory_token_t::tp, networkstatus_t::type, networkstatus_t::valid_after, networkstatus_t::valid_until, vote_routerstatus_t::version, networkstatus_voter_info_t::vote_digest, networkstatus_t::vote_seconds, voter_get_sig_by_algorithm(), networkstatus_t::voters, and networkstatus_t::weight_params.

Referenced by dirvote_add_signatures_to_pending_consensus(), dirvote_add_vote(), dirvote_compute_consensuses(), format_networkstatus_vote(), and networkstatus_set_current_consensus().

4.46.4.25 networkstatus_v2_t* networkstatus_v2_parse_from_string (const char * *s*)

Given a v2 network-status object in **s**, try to parse it and return the result. Return NULL on failure. Check the signature of the network status, but do not (yet) check the signing key for authority.

References _free_duplicate_routerstatus_entry(), base16_decode(), networkstatus_v2_t::binds_names, check_signature_token(), networkstatus_v2_t::client_versions, compare_routerstatus_entries(), networkstatus_v2_t::contact, crypto_pk_get_digest(), DIGEST_LEN, dump_desc(), networkstatus_v2_t::entries, escaped(), find_opt_by_keyword(), find_start_of_next_routerstatus(), networkstatus_v2_t::identity_digest, crypto_pk_env_t::key, LD_BUG, LD_DIR, networkstatus_v2_t::lists_bad_directories, networkstatus_v2_t::lists_bad_exits, memarea_clear(), memarea_drop_all(), memarea_new(), networkstatus_v2_t::networkstatus_digest, networkstatus_v2_free(), note_crypto_pk_op(), parse_iso_time(), networkstatus_v2_t::published_on, networkstatus_v2_t::recommends_versions, router_get_networkstatus_v2_hash(), routerstatus_parse_entry_from_string(), networkstatus_v2_t::server_versions, networkstatus_v2_t::signing_key, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), smartlist_sort(), smartlist_uniq(), networkstatus_v2_t::source_addr, networkstatus_v2_t::source_address, networkstatus_v2_t::source_dirport, strcmpstart(), token_clear(), tokenize_string(), tor_assert, tor_inet_aton(), and tor_parse_long().

Referenced by router_set_networkstatus_v2().

4.46.4.26 int networkstatus_verify_bw_weights (networkstatus_t ∗ *ns*)

Verify the bandwidth weights of a network status document

References BW_WEIGHT_SCALE, LD_BUG, LD_DIR, networkstatus_get_bw_weight(), networkstatus_get_param(), networkstatus_t::routerstatus_list, and T.

4.46.4.27 int rend_decrypt_introduction_points (char ∗∗ *ipos_decrypted*, size_t ∗ *ipos_decrypted_size*, const char ∗ *descriptor_cookie*, const char ∗ *ipos_encrypted*, size_t *ipos_encrypted_size*)

Decrypt the encrypted introduction points in **ipos_encrypted** of length **ipos_encrypted_size** using **descriptor_cookie** and write the result to a newly allocated string that is pointed to by **ipos_decrypted** and its length to **ipos_decrypted_size**. Return 0 if decryption was successful and -1 otherwise.

References CIPHER_IV_LEN, CIPHER_KEY_LEN, crypto_cipher_decrypt(), crypto_cipher_decrypt_with_iv(), crypto_create_init_cipher(), crypto_digest_add_bytes(), crypto_digest_get_digest(), crypto_free_cipher_env(), crypto_free_digest_env(), crypto_new_digest_env(), LD_REND, memcmpstart(), REND_BASIC_AUTH_CLIENT_ENTRY_LEN, REND_BASIC_AUTH_CLIENT_ID_LEN, REND_BASIC_AUTH_CLIENT_MULTIPLE, REND_DESC_COOKIE_LEN, tor_assert, and tor_free.

Referenced by rend_cache_store_v2_desc_as_client().

4.46.4.28 int rend_parse_client_keys (strmap_t ∗ *parsed_clients*, const char ∗ *ckstr*)

Parse the content of a client_key file in **ckstr** and add rend_authorized_client_t's for each parsed client to **parsed_clients**. Return the number of parsed clients as result or -1 for failure.

References directory_token_t::args, base64_decode(), rend_authorized_client_t::client_key, rend_authorized_client_t::client_name, rend_authorized_client_t::descriptor_cookie, eat_whitespace(), escaped(), find_opt_by_keyword(), directory_token_t::key, LD_CONFIG, LD_REND, memarea_clear(), memarea_drop_all(), memarea_new(), directory_token_t::n_args, REND_DESC_COOKIE_LEN, REND_DESC_COOKIE_LEN_BASE64, REND_LEGAL_CLIENTNAME_CHARACTERS, smartlist_clear(), smartlist_create(), smartlist_free(), strcmpstart(), strmap_get(), strmap_set(), strmap_size(), token_clear(), tokenize_string(), and tor_assert.

Referenced by rend_service_load_keys().

4.46.4.29 int rend_parse_introduction_points (rend_service_descriptor_t * *parsed*, const char * *intro_points_encoded*, size_t *intro_points_encoded_size*)

Parse the encoded introduction points in **intro_points_encoded** of length **intro_points_encoded_size** and write the result to the descriptor in **parsed**; return the number of successfully parsed introduction points or -1 in case of a failure.

Function may only be invoked once.

References extend_info_t::addr, directory_token_t::args, base16_encode(), base32_decode(), DIGEST_LEN, escaped(), rend_intro_point_t::extend_info, extend_info_t::identity_digest, rend_intro_point_t::intro_key, rend_service_descriptor_t::intro_nodes, directory_token_t::key, LD_REND, memarea_clear(), memarea_drop_all(), memarea_new(), memcmpstart(), extend_info_t::nickname, extend_info_t::onion_key, extend_info_t::port, rend_intro_point_free(), REND_INTRO_POINT_ID_LEN_BASE32, smartlist_add(), smartlist_clear(), smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_addr_family(), tor_addr_from_str(), tor_assert, and tor_parse_long().

Referenced by rend_cache_store_v2_desc_as_client().

4.46.4.30 int rend_parse_v2_service_descriptor (rend_service_descriptor_t ** *parsed_out*, char * *desc_id_out*, char ** *intro_points_encrypted_out*, size_t * *intro_points_encrypted_size_out*, size_t * *encoded_size_out*, const char ** *next_out*, const char * *desc*)

Parse and validate the ASCII-encoded v2 descriptor in **desc**, write the parsed descriptor to the newly allocated ∗**parsed_out**, the binary descriptor ID of length DIGEST_LEN to **desc_id_out**, the encrypted introduction points to the newly allocated ∗**intro_points_encrypted_out**, their encrypted size to ∗**intro_points_encrypted_size_out**, the size of the encoded descriptor to ∗**encoded_size_out**, and a pointer to the possibly next descriptor to ∗**next_out**; return 0 for success (including validation) and -1 for failure.

References directory_token_t::args, BASE32_CHARS, base32_decode(), check_signature_token(), crypto_pk_get_digest(), DIGEST_LEN, escaped(), find_opt_by_keyword(), directory_token_t::key, LD_DIR, LD_REND, memarea_drop_all(), memarea_new(), directory_token_t::n_args, note_crypto_pk_op(), directory_token_t::object_body, directory_token_t::object_size, directory_token_t::object_type, parse_iso_time(), rend_service_descriptor_t::pk, rend_service_descriptor_t::protocols, REND_DESC_ID_V2_LEN_BASE32, REND_DESC_MAX_SIZE, rend_get_descriptor_id_bytes(), REND_SECRET_ID_PART_LEN_BASE32, rend_service_descriptor_free(), router_get_hash_impl(), smartlist_create(), smartlist_free(), smartlist_split_string(), rend_service_descriptor_t::timestamp, token_clear(), tokenize_string(), tor_assert, tor_free, tor_parse_long(), and rend_service_descriptor_t::version.

Referenced by rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), and rend_desc_v2_is_parsable().

4.46.4.31 static int router_add_exit_policy (routerinfo_t * *router*, directory_token_t * *tok*) `[static]`

Add an exit policy stored in the token **tok** to the router info in **router**. Return 0 on success, -1 on failure.

References addr_policy_t::addr, addr_policy_free(), routerinfo_t::exit_policy, LD_DIR, router_parse_addr_policy(), smartlist_add(), smartlist_create(), tor_addr_family(), and directory_token_t::tp.

Referenced by router_parse_entry_from_string().

4.46.4.32 int router_append_dirobj_signature (char ∗ *buf*, size_t *buf_len*, const char ∗ *digest*, size_t *digest_len*, crypto_pk_env_t ∗ *private_key*)

Helper: used to generate signatures for routers, directories and network-status objects. Given a digest in **digest** and a secret **private_key**, generate an PKCS1-padded signature, BASE64-encode it, surround it with —–BEGIN/END—– pairs, and write it to the **buf_len**-byte buffer at **buf**. Return 0 on success, -1 on failure.

References base64_encode(), crypto_pk_keysize(), crypto_pk_private_sign(), LD_BUG, and tor_free.

Referenced by dirserv_dump_directory_to_string(), extrainfo_dump_to_string(), format_networkstatus_vote(), generate_runningrouters(), rend_encode_v2_descriptors(), and router_dump_router_to_string().

4.46.4.33 int router_get_dir_hash (const char ∗ *s*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the directory in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by dirserv_dump_directory_to_string(), and router_parse_directory().

4.46.4.34 int router_get_extrainfo_hash (const char ∗ *s*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the extrainfo string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by extrainfo_dump_to_string(), extrainfo_parse_entry_from_string(), and router_rebuild_descriptor().

4.46.4.35 static int router_get_hash_impl (const char ∗ *s*, size_t *s_len*, char ∗ *digest*, const char ∗ *start_str*, const char ∗ *end_str*, char *end_c*, digest_algorithm_t *alg*) `[static]`

Compute the digest of the substring of **s** taken from the first occurrence of **start_str** through the first instance of c after the first subsequent occurrence of **end_str**; store the 20-byte result in **digest**; return 0 on success.

If no such substring exists, return -1.

References crypto_digest(), and LD_BUG.

Referenced by authority_cert_parse_from_string(), rend_parse_v2_service_descriptor(), router_get_dir_hash(), router_get_extrainfo_hash(), router_get_networkstatus_v2_hash(), router_get_networkstatus_v3_hash(), router_get_router_hash(), and router_get_runningrouters_hash().

4.46.4.36 static int router_get_hashes_impl (const char ∗ *s*, size_t *s_len*, digests_t ∗ *digests*, const char ∗ *start_str*, const char ∗ *end_str*, char *end_c*) `[static]`

As router_get_hash_impl, but compute all hashes.

References crypto_digest_all(), and LD_BUG.

Referenced by router_get_networkstatus_v3_hashes().

4.46.4.37 int router_get_networkstatus_v2_hash (const char ∗ *s*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the network-status string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by networkstatus_v2_parse_from_string().

4.46.4.38 int router_get_networkstatus_v3_hash (const char ∗ *s*, char ∗ *digest*, digest_algorithm_t *alg*)

Set **digest** to the SHA-1 digest of the hash of the network-status string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by format_networkstatus_vote().

4.46.4.39 int router_get_networkstatus_v3_hashes (const char ∗ *s*, digests_t ∗ *digests*)

Set **digests** to all the digests of the consensus document in **s**

References router_get_hashes_impl().

Referenced by networkstatus_parse_vote_from_string().

4.46.4.40 int router_get_router_hash (const char ∗ *s*, size_t *s_len*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the first router in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by router_dump_router_to_string(), router_parse_entry_from_string(), and router_rebuild_descriptor().

4.46.4.41 int router_get_runningrouters_hash (const char ∗ *s*, char ∗ *digest*)

Set **digest** to the SHA-1 digest of the hash of the running-routers string in **s**. Return 0 on success, -1 on failure.

References router_get_hash_impl().

Referenced by generate_runningrouters(), and router_parse_runningrouters().

4.46.4.42 static addr_policy_t ∗ router_parse_addr_policy (directory_token_t ∗ *tok*) `[static]`

Given a K_ACCEPT or K_REJECT token and a router, create and return a new exit_policy_t corresponding to the token.

References addr_policy_t::addr, directory_token_t::args, escaped(), LD_DIR, addr_policy_t::maskbits, directory_token_t::n_args, addr_policy_t::policy_type, addr_policy_t::prt_max, addr_policy_t::prt_min, router_parse_addr_policy_private(), strcmpstart(), tor_addr_parse_mask_ports(), tor_assert, and directory_token_t::tp.

Referenced by router_add_exit_policy(), and router_parse_addr_policy_item_from_string().

4.46.4.43 addr_policy_t∗ router_parse_addr_policy_item_from_string (const char ∗ *s*, int *assume_action*)

Parse the addr policy in the string **s** and return it. If assume_action is nonnegative, then insert its action (ADDR_POLICY_ACCEPT or ADDR_POLICY_REJECT) for items that specify no action.

References eat_whitespace(), directory_token_t::error, escaped(), get_next_token(), LD_DIR, memarea_drop_all(), memarea_new(), router_parse_addr_policy(), token_clear(), tor_snprintf(), tor_strlower(), and directory_token_t::tp.

Referenced by parse_addr_policy(), policies_set_router_exitpolicy_to_reject_all(), and routerset_parse().

4.46.4.44 static addr_policy_t ∗ router_parse_addr_policy_private (directory_token_t ∗ *tok*) `[static]`

Parse an exit policy line of the format "accept/reject private:...". This didn't exist until Tor 0.1.1.15, so nobody should generate it in router descriptors until earlier versions are obsolete.

References directory_token_t::args, eat_whitespace(), addr_policy_t::is_private, parse_port_range(), addr_policy_t::policy_type, addr_policy_t::prt_max, addr_policy_t::prt_min, strcmpstart(), and directory_token_t::tp.

Referenced by router_parse_addr_policy().

4.46.4.45 int router_parse_directory (const char ∗ *str*)

Read a signed directory from **str**. If it's well-formed, return 0. Otherwise, return -1. If we're a directory cache, cache it.

References directory_token_t::args, authdir_mode_v1(), check_signature_token(), crypto_free_pk_env(), DIGEST_LEN, dirserv_set_cached_directory(), dump_desc(), find_dir_signing_key(), get_options(), hex_str(), LD_DIR, memarea_clear(), memarea_drop_all(), memarea_new(), directory_token_t::n_args, note_crypto_pk_op(), parse_iso_time(), router_get_dir_hash(), smartlist_clear(), smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, and directory_token_t::tp.

4.46.4.46 routerinfo_t* router_parse_entry_from_string (const char * *s*, const char * *end*, int *cache_copy*, int *allow_annotations*, const char * *prepend_annotations*)

Helper function: reads a single router entry from ***s** ... ***end**. Mallocs a new router and returns it if all goes well, else returns NULL. If **cache_copy** is true, duplicate the contents of s through end into the signed_descriptor_body of the resulting **routerinfo_t** (p. 206).

If **end** is NULL, **s** must be properly NULL-terminated.

If **allow_annotations**, it's okay to encounter annotations in **s** before the router; if it's false, reject the router if it's annotated. If **prepend_annotations** is set, it should contain some annotations: append them to the front of the router before parsing it, and keep them around when caching the router.

Only one of allow_annotations and prepend_annotations may be set.

References routerinfo_t::addr, routerinfo_t::address, routerinfo_t::allow_single_hop_exits, signed_descriptor_t::annotations_len, directory_token_t::args, routerinfo_t::bandwidthburst, routerinfo_t::bandwidthcapacity, routerinfo_t::bandwidthrate, base16_decode(), routerinfo_t::cache_info, routerinfo_t::caches_extra_info, check_signature_token(), routerinfo_t::contact_info, routerinfo_t::country, crypto_pk_get_digest(), routerinfo_t::declared_family, DIGEST_LEN, digestmap_new(), digestmap_set(), routerinfo_t::dir_port, dump_desc(), escaped(), routerinfo_t::exit_policy, signed_descriptor_t::extra_info_digest, find_all_exitpolicy(), find_opt_by_keyword(), routerinfo_t::has_old_dnsworkers, HEX_DIGEST_LEN, signed_descriptor_t::identity_digest, routerinfo_t::identity_pkey, routerinfo_t::is_hibernating, is_legal_nickname(), is_legal_nickname_or_hexdigest(), directory_token_t::key, LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, routerinfo_t::nickname, note_crypto_pk_op(), routerinfo_t::onion_pkey, routerinfo_t::or_port, parse_iso_time(), routerinfo_t::platform, policy_expand_private(), routerinfo_t::policy_is_reject_star, policy_is_reject_star(), signed_descriptor_t::published_on, routerinfo_t::purpose, router_add_exit_policy(), router_get_router_hash(), router_purpose_from_string(), ROUTER_PURPOSE_GENERAL, routerinfo_free(), routerinfo_set_country(), signed_descriptor_t::routerlist_index, signed_descriptor_t::send_unencrypted, signed_descriptor_t::signed_descriptor_body, signed_descriptor_t::signed_descriptor_digest, signed_descriptor_t::signed_descriptor_len, smartlist_add(), smartlist_create(), smartlist_free(), strcmpstart(), token_clear(), tokenize_string(), tor_assert, tor_inet_aton(), tor_parse_long(), tor_strstrip(), tor_version_as_new_as(), routerinfo_t::uptime, and routerinfo_t::wants_to_be_hs_dir.

Referenced by init_keys(), router_dump_router_to_string(), router_load_single_router(), router_parse_list_from_string(), and routerlist_reparse_old().

4.46.4.47 int router_parse_list_from_string (const char ** *s*, const char * *eos*, smartlist_t * *dest*, saved_location_t *saved_location*, int *want_extrainfo*, int *allow_annotations*, const char * *prepend_annotations*)

Given a string ***s** containing a concatenated sequence of router descriptors (or extra-info documents if **is_extrainfo** is set), parses them and stores the result in **dest**. All routers are marked running and valid. Advances *s to a point immediately following the last router entry. Ignore any trailing router entries that are not complete.

If **saved_location** isn't SAVED_IN_CACHE, make a local copy of each descriptor in the signed_descriptor_body field of each **routerinfo_t** (p. 206). If it isn't SAVED_NOWHERE, remember the offset of each descriptor.

Returns 0 on success and -1 on failure.

References routerinfo_t::cache_info, extrainfo_t::cache_info, extrainfo_parse_entry_from_string(), find_start_of_next_router_or_extrainfo(), routerlist_t::identity_map, LD_DIR, routerinfo_t::nickname, routerinfo_t::purpose, router_get_routerlist(), router_parse_entry_from_string(), router_purpose_to_string(), SAVED_IN_CACHE, signed_descriptor_t::saved_location, SAVED_NOWHERE, signed_descriptor_t::saved_offset, smartlist_add(), and tor_assert.

Referenced by dirserv_add_multiple_descriptors(), router_load_extrainfo_from_string(), and router_load_routers_from_string().

4.46.4.48 int router_parse_runningrouters (const char * *str*)

Read a signed router status statement from **str**. If it's well-formed, return 0. Otherwise, return -1. If we're a directory cache, cache it.

References directory_token_t::args, authdir_mode_v1(), check_signature_token(), crypto_free_pk_env(), DIGEST_LEN, dirserv_set_cached_directory(), dump_desc(), find_dir_signing_key(), find_opt_by_keyword(), get_options(), LD_DIR, memarea_drop_all(), memarea_new(), directory_token_t::n_args, note_crypto_pk_op(), parse_iso_time(), router_get_runningrouters_hash(), smartlist_create(), smartlist_free(), token_clear(), tokenize_string(), tor_assert, and directory_token_t::tp.

4.46.4.49 static routerstatus_t* routerstatus_parse_entry_from_string (memarea_t * *area*, const char ** *s*, smartlist_t * *tokens*, networkstatus_t * *vote*, vote_routerstatus_t * *vote_rs*, int *consensus_method*, consensus_flavor_t *flav*) `[static]`

Given a string at *****s**, containing a routerstatus object, and an empty smartlist at **tokens**, parse and return the first router status object in the string, and advance *****s** to just after the end of the router status. Return NULL and advance *****s** on error.

If **vote** and **vote_rs** are provided, don't allocate a fresh routerstatus but use **vote_rs** instead.

If **consensus_method** is nonzero, this routerstatus is part of a consensus, and we should parse it according to the method used to make that consensus.

Parse according to the syntax used by the consensus flavor **flav**.

References routerstatus_t::addr, directory_token_t::args, routerstatus_t::bandwidth, bool_eq, routerstatus_t::descriptor_digest, digest_from_base64(), routerstatus_t::dir_port, dump_desc(), escaped(), routerstatus_t::exitsummary, find_opt_by_keyword(), find_start_of_next_routerstatus(), vote_routerstatus_t::flags, routerstatus_t::has_bandwidth, routerstatus_t::has_exitsummary, routerstatus_t::has_measured_bw, routerstatus_t::identity_digest, routerstatus_t::is_authority, routerstatus_t::is_bad_directory, routerstatus_t::is_bad_exit, routerstatus_t::is_exit, routerstatus_t::is_fast, routerstatus_t::is_hs_dir, is_legal_nickname(), routerstatus_t::is_named, routerstatus_t::is_possible_guard, routerstatus_t::is_running, routerstatus_t::is_stable, routerstatus_t::is_unnamed, routerstatus_t::is_v2_dir, routerstatus_t::is_valid, networkstatus_t::known_flags, LD_DIR, routerstatus_t::measured_bw, memarea_clear(), vote_routerstatus_t::microdesc, directory_token_t::n_args, routerstatus_t::nickname, routerstatus_t::or_port, parse_iso_time(), routerstatus_t::published_on, routerstatus_free(), smartlist_clear(), smartlist_string_pos(), vote_routerstatus_t::status, strcmpstart(), token_clear(), tokenize_string(), tor_assert, tor_inet_aton(), tor_parse_long(), tor_parse_ulong(), tor_snprintf(), tor_version_as_new_as(), vote_routerstatus_t::version, routerstatus_t::version_known, routerstatus_t::version_supports_begindir, routerstatus_t::version_supports_conditional_consensus, routerstatus_-

t::version_supports_extrainfo_upload, and routerstatus_t::version_supports_v3_dir.

Referenced by networkstatus_parse_vote_from_string(), and networkstatus_v2_parse_from_string().

4.46.4.50 void sort_version_list (smartlist_t ∗ *versions*, int *remove_duplicates*)

Sort a list of string-representations of versions in ascending order.

References _compare_tor_version_str_ptr(), _tor_free(), smartlist_sort(), and smartlist_uniq().

Referenced by compute_consensus_versions_list(), format_versions_list(), and networkstatus_compute_consensus().

4.46.4.51 static INLINE directory_token_t∗ token_check_object (memarea_t ∗ *area*, const char ∗ *kwd*, directory_token_t ∗ *tok*, obj_syntax *o_syn*) `[static]`

Helper: make sure that the token **tok** with keyword **kwd** obeys the object syntax of **o_syn**. Allocate all storage in **area**. Return **tok** on success, or a new _ERR token if the token didn't conform to the syntax we wanted.

References crypto_pk_key_is_private(), crypto_pk_keysize(), directory_token_t::key, NEED_KEY, NEED_KEY_1024, NEED_OBJ, NEED_SKEY_1024, NO_OBJ, OBJ_OK, directory_token_t::object_body, PK_BYTES, and tor_snprintf().

Referenced by get_next_token().

4.46.4.52 static void token_clear (directory_token_t ∗ *tok*) `[static]`

Free all resources allocated for **tok**

References crypto_free_pk_env(), and directory_token_t::key.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), find_dir_signing_key(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), router_parse_addr_policy_item_from_string(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), routerstatus_parse_entry_from_string(), and tokenize_string().

4.46.4.53 static int tokenize_string (memarea_t ∗ *area*, const char ∗ *start*, const char ∗ *end*, smartlist_t ∗ *out*, token_rule_t ∗ *table*, int *flags*) `[static]`

Read all tokens from a string between **start** and **end**, and add them to **out**. Parse according to the token rules in **table**. Caller must free tokens in **out**. If **end** is NULL, use the entire string.

References eat_whitespace_eos(), directory_token_t::error, get_next_token(), LD_DIR, smartlist_add(), token_rule_t::t, token_clear(), tor_assert, and directory_token_t::tp.

Referenced by authority_cert_parse_from_string(), extrainfo_parse_entry_from_string(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

4.46.4.54 int tor_version_as_new_as (const char ∗ *platform*, const char ∗ *cutoff*)

Parse the Tor version of the platform string **platform**, and compare it to the version in **cutoff**. Return 1 if the router is at least as new as the cutoff, else return 0.

References eat_whitespace(), find_whitespace(), LD_BUG, LD_DIR, strcmpstart(), tor_assert, tor_version_compare(), and tor_version_parse().

Referenced by dirserv_get_status_impl(), remove_obsolete_entry_guards(), router_digest_version_as_new_as(), router_parse_entry_from_string(), router_supports_extrainfo(), routerstatus_parse_entry_from_string(), and set_routerstatus_from_routerinfo().

4.46.4.55 int tor_version_compare (tor_version_t ∗ *a*, tor_version_t ∗ *b*)

Compare two tor versions; Return <0 if a < b; 0 if a ==b, >0 if a > b.

References tor_version_t::git_tag, tor_version_t::git_tag_len, tor_version_t::major, tor_version_t::micro, tor_version_t::minor, tor_version_t::patchlevel, tor_version_t::status, tor_version_t::status_tag, tor_version_t::svn_revision, and tor_assert.

Referenced by _compare_tor_version_str_ptr(), tor_version_as_new_as(), and tor_version_is_obsolete().

4.46.4.56 version_status_t tor_version_is_obsolete (const char ∗ *myversion*, const char ∗ *versionlist*)

Return VS_RECOMMENDED if **myversion** is contained in **versionlist**. Else, return VS_EMPTY if versionlist has no entries. Else, return VS_OLD if every member of **versionlist** is newer than **myversion**. Else, return VS_NEW_IN_SERIES if there is at least one member of **versionlist** in the same series (major.minor.micro) as **myversion**, but no such member is newer than **myversion.**. Else, return VS_NEW if every member of **versionlist** is older than **myversion**. Else, return VS_UNRECOMMENDED.

(versionlist is a comma-separated list of version strings, optionally prefixed with "Tor". Versions that can't be parsed are ignored.)

References LD_BUG, LD_CONFIG, smartlist_create(), smartlist_free(), smartlist_split_string(), strcmpstart(), tor_assert, tor_free, tor_version_compare(), tor_version_parse(), tor_version_same_series(), VS_EMPTY, VS_NEW, VS_NEW_IN_SERIES, VS_OLD, VS_RECOMMENDED, and VS_UNRECOMMENDED.

Referenced by getinfo_helper_events(), and routers_update_all_from_networkstatus().

4.46.4.57 int tor_version_parse (const char ∗ *s*, tor_version_t ∗ *out*)

Parse a tor version from **s**, and store the result in **out**. Return 0 on success, -1 on failure.

References base16_decode(), DIGEST_LEN, eat_whitespace(), find_whitespace(), tor_version_t::git_tag, tor_version_t::git_tag_len, HEX_DIGEST_LEN, tor_version_t::major, tor_version_t::micro, tor_version_t::minor, tor_version_t::patchlevel, tor_version_t::status, tor_version_t::status_tag, strcasecmpstart(), strcmpstart(), tor_version_t::svn_revision, and tor_assert.

Referenced by _compare_tor_version_str_ptr(), remove_obsolete_entry_guards(), tor_version_as_new_as(), and tor_version_is_obsolete().

4.46.4.58 static int tor_version_same_series (tor_version_t ∗ *a*, tor_version_t ∗ *b*) [static]

Return true iff versions **a** and **b** belong to the same series.

References tor_version_t::major, tor_version_t::micro, tor_version_t::minor, and tor_assert.

Referenced by tor_version_is_obsolete().

4.46.5 Variable Documentation

4.46.5.1 token_rule_t client_keys_token_table[ ] [static]

Initial value:

```
 {
  T1_START("client-name", C_CLIENT_NAME, CONCAT_ARGS, NO_OBJ),
  T1("descriptor-cookie", C_DESCRIPTOR_COOKIE, EQ(1), NO_OBJ),
  T01("client-key", C_CLIENT_KEY, NO_ARGS, NEED_SKEY_1024),
  END_OF_TABLE
}
```

List of tokens allowed in the (possibly encrypted) list of introduction points of rendezvous service descriptors

4.46.5.2 token_rule_t desc_token_table[ ] [static]

Initial value:

```
 {
  T1_START("rendezvous-service-descriptor", R_RENDEZVOUS_SERVICE_DESCRIPTOR,
          EQ(1), NO_OBJ),
  T1("version", R_VERSION, EQ(1), NO_OBJ),
  T1("permanent-key", R_PERMANENT_KEY, NO_ARGS, NEED_KEY_1024),
  T1("secret-id-part", R_SECRET_ID_PART, EQ(1), NO_OBJ),
  T1("publication-time", R_PUBLICATION_TIME, CONCAT_ARGS, NO_OBJ),
  T1("protocol-versions", R_PROTOCOL_VERSIONS, EQ(1), NO_OBJ),
  T01("introduction-points", R_INTRODUCTION_POINTS, NO_ARGS, NEED_OBJ),
  T1_END("signature", R_SIGNATURE, NO_ARGS, NEED_OBJ),
  END_OF_TABLE
}
```

List of tokens allowable in rendezvous service descriptors

4.46.5.3 token_rule_t dir_footer_token_table[ ] [static]

Initial value:

```
 {
  T1("directory-signature", K_DIRECTORY_SIGNATURE, EQ(1), NEED_OBJ ),
  END_OF_TABLE
}
```

List of tokens allowable in the footer of v1/v2 directory/networkstatus footers.

4.46.5.4 token_rule_t dir_key_certificate_table[] [static]

Initial value:

```
 {
  CERTIFICATE_MEMBERS
  T1("fingerprint",      K_FINGERPRINT,              CONCAT_ARGS, NO_OBJ ),
  END_OF_TABLE
}
```

List of tokens allowable in V3 authority certificates.

4.46.5.5 token_rule_t dir_token_table[] [static]

Initial value:

```
 {

  T( "network-status",      K_NETWORK_STATUS,      NO_ARGS, NO_OBJ ),
  T( "directory-signature", K_DIRECTORY_SIGNATURE, ARGS,    NEED_OBJ ),
  T( "recommended-software",K_RECOMMENDED_SOFTWARE,CONCAT_ARGS, NO_OBJ ),
  T( "signed-directory",    K_SIGNED_DIRECTORY,    NO_ARGS, NO_OBJ ),

  T( "running-routers",     K_RUNNING_ROUTERS,     ARGS,    NO_OBJ ),
  T( "router-status",       K_ROUTER_STATUS,       ARGS,    NO_OBJ ),
  T( "published",           K_PUBLISHED,       CONCAT_ARGS, NO_OBJ ),
  T( "opt",                 K_OPT,             CONCAT_ARGS, OBJ_OK ),
  T( "contact",             K_CONTACT,         CONCAT_ARGS, NO_OBJ ),
  T( "dir-signing-key",     K_DIR_SIGNING_KEY,     ARGS,    OBJ_OK ),
  T( "fingerprint",         K_FINGERPRINT,     CONCAT_ARGS, NO_OBJ ),

  END_OF_TABLE
}
```

List of tokens allowable in v1 directory headers/footers.

4.46.5.6 token_rule_t extrainfo_token_table[] [static]

List of tokens allowable in extra-info documents.

4.46.5.7 token_rule_t ipo_token_table[] [static]

Initial value:

```
 {
  T1_START("introduction-point", R_IPO_IDENTIFIER, EQ(1), NO_OBJ),
  T1("ip-address", R_IPO_IP_ADDRESS, EQ(1), NO_OBJ),
  T1("onion-port", R_IPO_ONION_PORT, EQ(1), NO_OBJ),
  T1("onion-key", R_IPO_ONION_KEY, NO_ARGS, NEED_KEY_1024),
  T1("service-key", R_IPO_SERVICE_KEY, NO_ARGS, NEED_KEY_1024),
  END_OF_TABLE
}
```

List of tokens allowed in the (encrypted) list of introduction points of rendezvous service descriptors

4.46.5.8 time_t last_desc_dumped = 0 [static]

Last time we dumped a descriptor to disk.

Referenced by dump_desc().

4.46.5.9 token_rule_t microdesc_token_table[] [static]

Initial value:

```
 {
  T1_START("onion-key",        K_ONION_KEY,        NO_ARGS,     NEED_KEY_1024),
  T01("family",                K_FAMILY,           ARGS,        NO_OBJ ),
  T01("p",                     K_P,                CONCAT_ARGS, NO_OBJ ),
  A01("@last-listed",          A_LAST_LISTED,      CONCAT_ARGS, NO_OBJ ),
  END_OF_TABLE
}
```

4.46.5.10 token_rule_t netstatus_token_table[] [static]

Initial value:

```
 {
  T1( "published",           K_PUBLISHED,       CONCAT_ARGS, NO_OBJ ),
  T0N("opt",                 K_OPT,             CONCAT_ARGS, OBJ_OK ),
  T1( "contact",             K_CONTACT,         CONCAT_ARGS, NO_OBJ ),
  T1( "dir-signing-key",     K_DIR_SIGNING_KEY,  NO_ARGS,    NEED_KEY_1024 ),
  T1( "fingerprint",         K_FINGERPRINT,     CONCAT_ARGS, NO_OBJ ),
  T1_START("network-status-version", K_NETWORK_STATUS_VERSION,
                                                   GE(1),    NO_OBJ ),
  T1( "dir-source",          K_DIR_SOURCE,         GE(3),    NO_OBJ ),
  T01("dir-options",         K_DIR_OPTIONS,        ARGS,     NO_OBJ ),
  T01("client-versions",     K_CLIENT_VERSIONS, CONCAT_ARGS, NO_OBJ ),
  T01("server-versions",     K_SERVER_VERSIONS, CONCAT_ARGS, NO_OBJ ),

  END_OF_TABLE
}
```

List of tokens allowable in the header part of v2 networkstatus documents.

4.46.5.11 token_rule_t networkstatus_consensus_token_table[] [static]

Initial value:

```
 {
  T1_START("network-status-version", K_NETWORK_STATUS_VERSION,
                                                GE(1),       NO_OBJ ),
  T1("vote-status",            K_VOTE_STATUS,      GE(1),       NO_OBJ ),
  T1("valid-after",            K_VALID_AFTER,      CONCAT_ARGS, NO_OBJ ),
  T1("fresh-until",            K_FRESH_UNTIL,      CONCAT_ARGS, NO_OBJ ),
  T1("valid-until",            K_VALID_UNTIL,      CONCAT_ARGS, NO_OBJ ),
  T1("voting-delay",           K_VOTING_DELAY,     GE(2),       NO_OBJ ),

  T0N("opt",                 K_OPT,             CONCAT_ARGS, OBJ_OK ),

  T1N("dir-source",          K_DIR_SOURCE,         GE(6),    NO_OBJ ),
  T1N("contact",             K_CONTACT,         CONCAT_ARGS, NO_OBJ ),
  T1N("vote-digest",         K_VOTE_DIGEST,        GE(1),    NO_OBJ ),
```

```
  T1( "known-flags",         K_KNOWN_FLAGS,     CONCAT_ARGS, NO_OBJ ),

  T01("client-versions",     K_CLIENT_VERSIONS, CONCAT_ARGS, NO_OBJ ),
  T01("server-versions",     K_SERVER_VERSIONS, CONCAT_ARGS, NO_OBJ ),
  T01("consensus-method",    K_CONSENSUS_METHOD,    EQ(1),   NO_OBJ),
  T01("params",                K_PARAMS,           ARGS,        NO_OBJ ),

  END_OF_TABLE
}
```

List of tokens allowed in V3 networkstatus consensuses.

4.46.5.12 token_rule_t networkstatus_detached_signature_token_table[] [static]

Initial value:

```
 {
  T1_START("consensus-digest", K_CONSENSUS_DIGEST, GE(1),       NO_OBJ ),
  T("additional-digest",       K_ADDITIONAL_DIGEST,GE(3),       NO_OBJ ),
  T1("valid-after",            K_VALID_AFTER,      CONCAT_ARGS, NO_OBJ ),
  T1("fresh-until",            K_FRESH_UNTIL,      CONCAT_ARGS, NO_OBJ ),
  T1("valid-until",            K_VALID_UNTIL,      CONCAT_ARGS, NO_OBJ ),
  T("additional-signature",  K_ADDITIONAL_SIGNATURE, GE(4),   NEED_OBJ ),
  T1N("directory-signature", K_DIRECTORY_SIGNATURE,  GE(2),   NEED_OBJ ),
  END_OF_TABLE
}
```

List of tokens allowable in detached networkstatus signature documents.

4.46.5.13 token_rule_t networkstatus_token_table[] [static]

Initial value:

```
 {
  T1_START("network-status-version", K_NETWORK_STATUS_VERSION,
                                                   GE(1),       NO_OBJ ),
  T1("vote-status",            K_VOTE_STATUS,      GE(1),       NO_OBJ ),
  T1("published",              K_PUBLISHED,        CONCAT_ARGS, NO_OBJ ),
  T1("valid-after",            K_VALID_AFTER,      CONCAT_ARGS, NO_OBJ ),
  T1("fresh-until",            K_FRESH_UNTIL,      CONCAT_ARGS, NO_OBJ ),
  T1("valid-until",            K_VALID_UNTIL,      CONCAT_ARGS, NO_OBJ ),
  T1("voting-delay",           K_VOTING_DELAY,     GE(2),       NO_OBJ ),
  T1("known-flags",            K_KNOWN_FLAGS,      ARGS,        NO_OBJ ),
  T01("params",                K_PARAMS,           ARGS,        NO_OBJ ),
  T( "fingerprint",            K_FINGERPRINT,      CONCAT_ARGS, NO_OBJ ),

  CERTIFICATE_MEMBERS

  TON("opt",                 K_OPT,             CONCAT_ARGS, OBJ_OK ),
  T1( "contact",             K_CONTACT,         CONCAT_ARGS, NO_OBJ ),
  T1( "dir-source",          K_DIR_SOURCE,      GE(6),       NO_OBJ ),
  T01("legacy-dir-key",      K_LEGACY_DIR_KEY,  GE(1),       NO_OBJ ),
  T1( "known-flags",         K_KNOWN_FLAGS,     CONCAT_ARGS, NO_OBJ ),
  T01("client-versions",     K_CLIENT_VERSIONS, CONCAT_ARGS, NO_OBJ ),
  T01("server-versions",     K_SERVER_VERSIONS, CONCAT_ARGS, NO_OBJ ),
  T1( "consensus-methods",   K_CONSENSUS_METHODS, GE(1),     NO_OBJ ),

  END_OF_TABLE
}
```

List of tokens allowed in V3 networkstatus votes.

4.46.5.14 token_rule_t networkstatus_vote_footer_token_table[] [static]

Initial value:

```
 {
  T01("directory-footer",    K_DIRECTORY_FOOTER,    NO_ARGS,   NO_OBJ ),
  T01("bandwidth-weights",   K_BW_WEIGHTS,          ARGS,      NO_OBJ ),
  T(  "directory-signature", K_DIRECTORY_SIGNATURE, GE(2),     NEED_OBJ ),
  END_OF_TABLE
}
```

List of tokens allowable in the footer of v1/v2 directory/networkstatus footers.

4.46.5.15 token_rule_t routerdesc_token_table[] [static]

List of tokens allowable in router descriptors

4.46.5.16 token_rule_t rtrstatus_token_table[] [static]

Initial value:

```
 {
  T01("p",                   K_P,             CONCAT_ARGS, NO_OBJ ),
  T1( "r",                   K_R,                 GE(7),   NO_OBJ ),
  T1( "s",                   K_S,                 ARGS,    NO_OBJ ),
  T01("v",                   K_V,             CONCAT_ARGS, NO_OBJ ),
  T01("w",                   K_W,                 ARGS,    NO_OBJ ),
  TON("m",                   K_M,             CONCAT_ARGS, NO_OBJ ),
  TON("opt",                 K_OPT,           CONCAT_ARGS, OBJ_OK ),
  END_OF_TABLE
}
```

List of tokens allowable in the body part of v2 and v3 networkstatus documents.

4.47 sha256.c File Reference

Data Structures

- struct **sha256_state**

Defines

- #define **CRYPT_OK** 0
- #define **CRYPT_NOP** -1
- #define **CRYPT_INVALID_ARG** -2
- #define **LOAD32H**(x, y) STMT_BEGIN x = ntohl(get_uint32((const char*)y)); STMT_END
- #define **STORE32H**(x, y) STMT_BEGIN set_uint32((char*)y, htonl(x)); STMT_END
- #define **STORE64H**(x, y)
- #define **RORc**(x, y) (((((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)((y)&31)) | ((unsigned long)(x)<<(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)
- #define **MIN**(x, y) (((x)<(y))?(x):(y))
- #define **Ch**(x, y, z) (z ^ (x & (y ^ z)))
- #define **Maj**(x, y, z) (((x | y) & z) | (x & y))
- #define **S**(x, n) RORc((x),(n))
- #define **R**(x, n) (((x)&0xFFFFFFFFUL)>>(n))
- #define **Sigma0**(x) (S(x, 2) ^ S(x, 13) ^ S(x, 22))
- #define **Sigma1**(x) (S(x, 6) ^ S(x, 11) ^ S(x, 25))
- #define **Gamma0**(x) (S(x, 7) ^ S(x, 18) ^ R(x, 3))
- #define **Gamma1**(x) (S(x, 17) ^ S(x, 19) ^ R(x, 10))
- #define **RND**(a, b, c, d, e, f, g, h, i, ki)

Functions

- static int **sha256_compress** (sha256_state *md, unsigned char *buf)
- static int **sha256_init** (sha256_state *md)
- static int **sha256_process** (sha256_state *md, const unsigned char *in, unsigned long inlen)
- static int **sha256_done** (sha256_state *md, unsigned char *out)

4.47.1 Detailed Description

SHA256 by Tom St Denis

4.47.2 Define Documentation

4.47.2.1 #define RND(a, b, c, d, e, f, g, h, i, ki)

Value:

```
t0 = h + Sigma1(e) + Ch(e, f, g) + ki + W[i];           \
      t1 = Sigma0(a) + Maj(a, b, c);                           \
      d += t0;                                                 \
      h  = t0 + t1;
```

4.47.2.2 #define STORE64H(x, y)

Value:

```
STMT_BEGIN                                \
  set_uint32((char*)y, htonl((uint32_t)((x)>>32)));             \
  set_uint32(((char*)y)+4, htonl((uint32_t)((x)&0xffffffff)));  \
  STMT_END
```

4.47.3 Function Documentation

4.47.3.1 static int sha256_done (sha256_state ∗ *md*, unsigned char ∗ *out*) `[static]`

Terminate the hash to get the digest

Parameters:

md The hash state

out [out] The destination of the hash (32 bytes)

Returns:

CRYPT_OK if successful

4.47.3.2 static int sha256_init (sha256_state ∗ *md*) `[static]`

Initialize the hash state

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

4.47.3.3 static int sha256_process (sha256_state ∗ *md*, const unsigned char ∗ *in*, unsigned long *inlen*) `[static]`

Process a block of memory though the hash

Parameters:

md The hash state

in The data to hash

inlen The length of the data (octets)

Returns:

CRYPT_OK if successful

4.48 tor_main.c File Reference

Stub module containing a **main()** (p. 1210) function. Allows unit test binary to link against **main.c** (p. 690).

`#include "micro-revision.i"`

Functions

- int **tor_main** (int argc, char ∗argv[ ])
- int **main** (int argc, char ∗argv[ ])

Variables

- const char **tor_git_revision** [ ]

4.48.1 Detailed Description

Stub module containing a **main()** (p. 1210) function. Allows unit test binary to link against **main.c** (p. 690).

4.48.2 Function Documentation

4.48.2.1 int main (int *argc*, char ∗ *argv*[ ])

We keep **main()** (p. 1210) in a separate file so that our unit tests can use functions from **main.c** (p. 690))

References tor_main().

4.48.2.2 int tor_main (int *argc*, char ∗ *argv*[ ])

Main entry point for the Tor process. Called from **main()** (p. 1210).

References _tor_free(), _tor_malloc(), _tor_realloc(), do_hash_password(), do_list_fingerprint(), do_main_loop(), get_options(), init_logging(), LD_BUG, tor_assert, tor_cleanup(), tor_init(), tor_threads_init(), and update_approx_time().

Referenced by main().

4.48.3 Variable Documentation

4.48.3.1 const char tor_git_revision[ ]

Initial value:

```
# 12 "/home/arma/work/onion/git/tor/src/or/tor_main.c" 2
  ""
```

String describing which Tor subversion repository version the source was built from. This string is generated by a bit of shell kludging int src/or/Makefile.am, and is usually right.

4.49 torgzip.c File Reference

A simple in-memory gzip implementation.

`#include "orconfig.h"`

`#include <stdlib.h>`

`#include <stdio.h>`

`#include <assert.h>`

`#include <zlib.h>`

`#include <string.h>`

`#include <netinet/in.h>`

`#include "util.h"`

`#include "log.h"`

`#include "torgzip.h"`

Data Structures

- struct **tor_zlib_state_t**

Functions

- int **is_gzip_supported** (void)
- static INLINE int **method_bits** (**compress_method_t** method)
- int **tor_gzip_compress** (char ∗∗out, size_t ∗out_len, const char ∗in, size_t in_len, **compress_method_t** method)
- int **tor_gzip_uncompress** (char ∗∗out, size_t ∗out_len, const char ∗in, size_t in_len, **compress_method_t** method, int complete_only, int protocol_warn_level)
- **compress_method_t detect_compression_method** (const char ∗in, size_t in_len)
- **tor_zlib_state_t** ∗ **tor_zlib_new** (int compress, **compress_method_t** method)
- **tor_zlib_output_t tor_zlib_process** (**tor_zlib_state_t** ∗state, char ∗∗out, size_t ∗out_len, const char ∗∗in, size_t ∗in_len, int finish)
- void **tor_zlib_free** (**tor_zlib_state_t** ∗state)

Variables

- static int **gzip_is_supported** = -1

4.49.1 Detailed Description

A simple in-memory gzip implementation.

4.49.2 Function Documentation

4.49.2.1 compress_method_t detect_compression_method (const char ∗ *in*, size_t *in_len*)

Try to tell whether the **in_len**-byte string in **in** is likely to be compressed or not. If it is, return the likeliest compression method. Otherwise, return UNKNOWN_METHOD.

References get_uint16().

Referenced by connection_dir_client_reached_eof().

4.49.2.2 int is_gzip_supported (void)

Return true iff we support gzip-based compression. Otherwise, we need to use zlib.

References gzip_is_supported, and strcmpstart().

Referenced by tor_gzip_compress(), tor_gzip_uncompress(), and tor_zlib_new().

4.49.2.3 static INLINE int method_bits (compress_method_t *method*) `[static]`

Return the 'bits' value to tell zlib to use **method**.

Referenced by tor_gzip_compress(), tor_gzip_uncompress(), and tor_zlib_new().

4.49.2.4 int tor_gzip_compress (char ∗∗ *out*, size_t ∗ *out_len*, const char ∗ *in*, size_t *in_len*, compress_method_t *method*)

Given **in_len** bytes at **in**, compress them into a newly allocated buffer, using the method described in **method**. Store the compressed string in ∗**out**, and its length in ∗**out_len**. Return 0 on success, -1 on failure.

References is_gzip_supported(), LD_BUG, LD_GENERAL, method_bits(), tor_assert, and tor_free.

Referenced by new_cached_dir(), and set_cached_dir().

4.49.2.5 int tor_gzip_uncompress (char ∗∗ *out*, size_t ∗ *out_len*, const char ∗ *in*, size_t *in_len*, compress_method_t *method*, int *complete_only*, int *protocol_warn_level*)

Given zero or more zlib-compressed or gzip-compressed strings of total length **in_len** bytes at **in**, uncompress them into a newly allocated buffer, using the method described in **method**. Store the uncompressed string in ∗**out**, and its length in ∗**out_len**. Return 0 on success, -1 on failure.

If **complete_only** is true, we consider a truncated input as a failure; otherwise we decompress as much as we can. Warn about truncated or corrupt inputs at **protocol_warn_level**.

References is_gzip_supported(), LD_BUG, LD_GENERAL, LD_PROTOCOL, method_bits(), tor_assert, and tor_free.

Referenced by connection_dir_client_reached_eof().

4.49.2.6 void tor_zlib_free (tor_zlib_state_t * *state*)

Deallocate **state**.

References tor_zlib_state_t::compress, tor_zlib_state_t::stream, and tor_free.

Referenced by _connection_free(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), and connection_dirserv_finish_spooling().

4.49.2.7 tor_zlib_state_t* tor_zlib_new (int *compress*, compress_method_t *method*)

Construct and return a **tor_zlib_state_t** (p. 246) object using **method**. If **compress**, it's for compression; otherwise it's for decompression.

References tor_zlib_state_t::compress, is_gzip_supported(), LD_BUG, method_bits(), tor_zlib_state_t::stream, and tor_free.

Referenced by connection_dirserv_add_networkstatus_bytes_to_outbuf(), and directory_handle_command_get().

4.49.2.8 tor_zlib_output_t tor_zlib_process (tor_zlib_state_t * *state*, char ** *out*, size_t * *out_len*, const char ** *in*, size_t * *in_len*, int *finish*)

Compress/decompress some bytes using **state**. Read up to *__in_len__ bytes from *__in__, and write up to *__out_len__ bytes to *__out__, adjusting the values as we go. If **finish** is true, we've reached the end of the input.

Return TOR_ZLIB_DONE if we've finished the entire compression/decompression. Return TOR_ZLIB_OK if we're processed everything from the input. Return TOR_ZLIB_BUF_FULL if we're out of space on **out**. Return TOR_ZLIB_ERR if the stream is corrupt.

References tor_zlib_state_t::compress, LD_GENERAL, tor_zlib_state_t::stream, and tor_assert.

Referenced by write_to_buf_zlib().

4.49.3 Variable Documentation

4.49.3.1 int gzip_is_supported = -1 `[static]`

Set to 1 if zlib is a version that supports gzip; set to 0 if it doesn't; set to -1 if we haven't checked yet.

Referenced by is_gzip_supported().

4.50 torgzip.h File Reference

Headers for **torgzip.h** (p. 1215).

Typedefs

- typedef struct **tor_zlib_state_t** **tor_zlib_state_t**

Enumerations

- enum **compress_method_t** { **NO_METHOD** = 0, **GZIP_METHOD** = 1, **ZLIB_METHOD** = 2, **UNKNOWN_METHOD** = 3 }
- enum **tor_zlib_output_t** { **TOR_ZLIB_OK**, **TOR_ZLIB_DONE**, **TOR_ZLIB_BUF_FULL**, **TOR_ZLIB_ERR** }

Functions

- int **tor_gzip_compress** (char ∗∗out, size_t ∗out_len, const char ∗in, size_t in_len, **compress_method_t** method)
- int **tor_gzip_uncompress** (char ∗∗out, size_t ∗out_len, const char ∗in, size_t in_len, **compress_method_t** method, int complete_only, int protocol_warn_level)
- int **is_gzip_supported** (void)
- **compress_method_t** **detect_compression_method** (const char ∗in, size_t in_len)
- **tor_zlib_state_t** ∗ **tor_zlib_new** (int compress, **compress_method_t** method)
- **tor_zlib_output_t** **tor_zlib_process** (**tor_zlib_state_t** ∗state, char ∗∗out, size_t ∗out_len, const char ∗∗in, size_t ∗in_len, int finish)
- void **tor_zlib_free** (**tor_zlib_state_t** ∗state)

4.50.1 Detailed Description

Headers for **torgzip.h** (p. 1215).

4.50.2 Typedef Documentation

4.50.2.1 typedef struct tor_zlib_state_t tor_zlib_state_t

Internal state for an incremental zlib compression/decompression.

4.50.3 Enumeration Type Documentation

4.50.3.1 enum compress_method_t

Enumeration of what kind of compression to use. Only ZLIB_METHOD is guaranteed to be supported by the compress/uncompress functions here; GZIP_METHOD may be supported if we built against zlib version 1.2 or later and **is_gzip_supported()** (p. 1213) returns true.

4.50.3.2 enum tor_zlib_output_t

Return values from tor_zlib_process; see that function's documentation for details.

4.50.4 Function Documentation

4.50.4.1 compress_method_t detect_compression_method (const char ∗ *in*, size_t *in_len*)

Try to tell whether the **in_len**-byte string in **in** is likely to be compressed or not. If it is, return the likeliest compression method. Otherwise, return UNKNOWN_METHOD.

References get_uint16().

Referenced by connection_dir_client_reached_eof().

4.50.4.2 int is_gzip_supported (void)

Return true iff we support gzip-based compression. Otherwise, we need to use zlib.

References gzip_is_supported, and strcmpstart().

Referenced by tor_gzip_compress(), tor_gzip_uncompress(), and tor_zlib_new().

4.50.4.3 int tor_gzip_compress (char ∗∗ *out*, size_t ∗ *out_len*, const char ∗ *in*, size_t *in_len*, compress_method_t *method*)

Given **in_len** bytes at **in**, compress them into a newly allocated buffer, using the method described in **method**. Store the compressed string in ∗**out**, and its length in ∗**out_len**. Return 0 on success, -1 on failure.

References is_gzip_supported(), LD_BUG, LD_GENERAL, method_bits(), tor_assert, and tor_free.

Referenced by new_cached_dir(), and set_cached_dir().

4.50.4.4 int tor_gzip_uncompress (char ∗∗ *out*, size_t ∗ *out_len*, const char ∗ *in*, size_t *in_len*, compress_method_t *method*, int *complete_only*, int *protocol_warn_level*)

Given zero or more zlib-compressed or gzip-compressed strings of total length **in_len** bytes at **in**, uncompress them into a newly allocated buffer, using the method described in **method**. Store the uncompressed string in ∗**out**, and its length in ∗**out_len**. Return 0 on success, -1 on failure.

If **complete_only** is true, we consider a truncated input as a failure; otherwise we decompress as much as we can. Warn about truncated or corrupt inputs at **protocol_warn_level**.

References is_gzip_supported(), LD_BUG, LD_GENERAL, LD_PROTOCOL, method_bits(), tor_assert, and tor_free.

Referenced by connection_dir_client_reached_eof().

4.50.4.5 void tor_zlib_free (tor_zlib_state_t ∗ *state*)

Deallocate **state**.

References tor_zlib_state_t::compress, tor_zlib_state_t::stream, and tor_free.

Referenced by _connection_free(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), and connection_dirserv_finish_spooling().

4.50.4.6 tor_zlib_state_t∗ tor_zlib_new (int *compress*, compress_method_t *method*)

Construct and return a **tor_zlib_state_t** (p. 246) object using **method**. If **compress**, it's for compression; otherwise it's for decompression.

References tor_zlib_state_t::compress, is_gzip_supported(), LD_BUG, method_bits(), tor_zlib_state_t::stream, and tor_free.

Referenced by connection_dirserv_add_networkstatus_bytes_to_outbuf(), and directory_handle_command_get().

4.50.4.7 tor_zlib_output_t tor_zlib_process (tor_zlib_state_t ∗ *state*, char ∗∗ *out*, size_t ∗ *out_len*, const char ∗∗ *in*, size_t ∗ *in_len*, int *finish*)

Compress/decompress some bytes using **state**. Read up to ∗**in_len** bytes from ∗**in**, and write up to ∗**out_len** bytes to ∗**out**, adjusting the values as we go. If **finish** is true, we've reached the end of the input.

Return TOR_ZLIB_DONE if we've finished the entire compression/decompression. Return TOR_ZLIB_OK if we're processed everything from the input. Return TOR_ZLIB_BUF_FULL if we're out of space on **out**. Return TOR_ZLIB_ERR if the stream is corrupt.

References tor_zlib_state_t::compress, LD_GENERAL, tor_zlib_state_t::stream, and tor_assert.

Referenced by write_to_buf_zlib().

4.51 torint.h File Reference

Header file to define uint32_t and friends.

`#include "orconfig.h"`

`#include <stdint.h>`

`#include <sys/types.h>`

`#include <limits.h>`

`#include <inttypes.h>`

Defines

- #define **HAVE_INT8_T**
- #define **HAVE_INT16_T**
- #define **HAVE_INT32_T**
- #define **HAVE_INT64_T**
- #define **HAVE_UINT8_T**
- #define **HAVE_UINT16_T**
- #define **HAVE_UINT32_T**
- #define **HAVE_UINT64_T**
- #define **HAVE_INTPTR_T**
- #define **HAVE_UINTPTR_T**
- #define **UINT32_MAX** 0xffffffffu
- #define **INT32_MAX** 0x7fffffffu
- #define **INT32_MIN** (-2147483647-1)
- #define **UINT64_MAX** 0xfffffffffffffffful
- #define **INT64_MAX** 0x7fffffffffffffffll
- #define **LONG_MAX** 0x7fffffffffffffffL
- #define **INT_MAX** 0x7fffffffL
- #define **UINT_MAX** 0xffffffffu
- #define **SHORT_MAX** 0x7fff
- #define **TIME_MAX** ((time_t)LONG_MAX)
- #define **SIZE_T_MAX** UINT64_MAX
- #define **SSIZE_T_MAX** INT64_MAX
- #define **SIZE_T_CEILING** (sizeof(char)<<(sizeof(size_t)*8 - 1))

4.51.1 Detailed Description

Header file to define uint32_t and friends.

4.52 tortls.c File Reference

Wrapper functions to present a consistent interface to TLS, SSL, and X.509 functions from OpenSSL.

`#include "orconfig.h"`

`#include <assert.h>`

`#include <openssl/ssl.h>`

`#include <openssl/ssl3.h>`

`#include <openssl/err.h>`

`#include <openssl/tls1.h>`

`#include <openssl/asn1.h>`

`#include <openssl/bio.h>`

`#include <openssl/opensslv.h>`

`#include "crypto.h"`

`#include "tortls.h"`

`#include "util.h"`

`#include "log.h"`

`#include "container.h"`

`#include "ht.h"`

`#include <string.h>`

`#include "tortls_states.h"`

`#include "./ciphers.inc"`

Data Structures

- struct **tor_tls_context_t**
- struct **tor_tls_t**
- struct **cipher_info_t**

Defines

- #define **CRYPTO_PRIVATE**
- #define **V2_HANDSHAKE_SERVER**
- #define **V2_HANDSHAKE_CLIENT**
- #define **LEGAL_NICKNAME_CHARACTERS** "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
- #define **IDENTITY_CERT_LIFETIME** (365∗24∗60∗60)
- #define **ADDR**(tls) (((tls) && (tls) → address) ? tls → address : "peer")
- #define **SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION** 0x00040000L
- #define **SSL3_FLAGS_ALLOW_UNSAFE_LEGACY_RENEGOTIATION** 0x0010
- #define **_TOR_TLS_SYSCALL** (_MIN_TOR_TLS_ERROR_VAL - 2)
- #define **_TOR_TLS_ZERORETURN** (_MIN_TOR_TLS_ERROR_VAL - 1)

- #define **CATCH_SYSCALL** 1
- #define **CATCH_ZERO** 2
- #define **SERVER_CIPHER_LIST**
- #define **CIPHER**(id, name) name ":"
- #define **XCIPHER**(id, name)
- #define **CIPHER**(id, name) { id, name },
- #define **XCIPHER**(id, name) { id, #name },

Functions

- static **STACK_OF** (SSL_CIPHER)
- static INLINE unsigned int **tor_tls_entry_hash** (const **tor_tls_t** ∗a)
- static **HT_HEAD** (HT_PROTOTYPE(tlsmap, HT_PROTOTYPE(**tor_tls_t**)
- static void **tor_tls_context_decref** (**tor_tls_context_t** ∗ctx)
- static void **tor_tls_context_incref** (**tor_tls_context_t** ∗ctx)
- static X509 ∗ **tor_tls_create_certificate** (**crypto_pk_env_t** ∗rsa, **crypto_pk_env_t** ∗rsa_sign, const char ∗cname, const char ∗cname_sign, unsigned int lifetime)
- static void **tor_tls_unblock_renegotiation** (**tor_tls_t** ∗tls)
- static const char ∗ **ssl_state_to_string** (int ssl_state)
- static void **tls_log_errors** (**tor_tls_t** ∗tls, int severity, int domain, const char ∗doing)
- static int **tor_errno_to_tls_error** (int e)
- const char ∗ **tor_tls_err_to_string** (int err)
- static int **tor_tls_get_error** (**tor_tls_t** ∗tls, int r, int extra, const char ∗doing, int severity, int domain)
- static void **tor_tls_init** (void)
- void **tor_tls_free_all** (void)
- static int **always_accept_verify_cb** (int preverify_ok, X509_STORE_CTX ∗x509_ctx)
- static X509_NAME ∗ **tor_x509_name_new** (const char ∗cname)
- int **tor_tls_context_new** (**crypto_pk_env_t** ∗identity, unsigned int key_lifetime)
- static int **tor_tls_client_is_using_v2_ciphers** (const SSL ∗ssl, const char ∗address)
- static void **tor_tls_server_info_callback** (const SSL ∗ssl, int type, int val)
- static void **rectify_client_ciphers** (STACK_OF(SSL_CIPHER)∗∗ciphers)
- **tor_tls_t** ∗ **tor_tls_new** (int sock, int isServer)
- void **tor_tls_set_logged_address** (**tor_tls_t** ∗tls, const char ∗address)
- void **tor_tls_set_renegotiate_callback** (**tor_tls_t** ∗tls, void(∗cb)(**tor_tls_t** ∗, void ∗arg), void ∗arg)
- void **tor_tls_block_renegotiation** (**tor_tls_t** ∗tls)
- int **tor_tls_is_server** (**tor_tls_t** ∗tls)
- void **tor_tls_free** (**tor_tls_t** ∗tls)
- int **tor_tls_read** (**tor_tls_t** ∗tls, char ∗cp, size_t len)
- int **tor_tls_write** (**tor_tls_t** ∗tls, const char ∗cp, size_t n)
- int **tor_tls_handshake** (**tor_tls_t** ∗tls)
- int **tor_tls_renegotiate** (**tor_tls_t** ∗tls)
- int **tor_tls_shutdown** (**tor_tls_t** ∗tls)
- int **tor_tls_peer_has_cert** (**tor_tls_t** ∗tls)
- static void **log_cert_lifetime** (X509 ∗cert, const char ∗problem)
- static void **try_to_extract_certs_from_tls** (int severity, **tor_tls_t** ∗tls, X509 ∗∗cert_out, X509 ∗∗id_cert_out)
- int **tor_tls_verify** (int severity, **tor_tls_t** ∗tls, **crypto_pk_env_t** ∗∗identity_key)

- int **tor_tls_check_lifetime** (**tor_tls_t** *tls, int tolerance)
- int **tor_tls_get_pending_bytes** (**tor_tls_t** *tls)
- size_t **tor_tls_get_forced_write_size** (**tor_tls_t** *tls)
- void **tor_tls_get_n_raw_bytes** (**tor_tls_t** *tls, size_t *n_read, size_t *n_written)
- void **_check_no_tls_errors** (const char *fname, int line)
- int **tor_tls_used_v1_handshake** (**tor_tls_t** *tls)
- void **tor_tls_get_buffer_sizes** (**tor_tls_t** *tls, size_t *rbuf_capacity, size_t *rbuf_bytes, size_t *wbuf_capacity, size_t *wbuf_bytes)

Variables

- static int **use_unsafe_renegotiation_op** = 0
- static int **use_unsafe_renegotiation_flag** = 0
- static SSL_CIPHER * **CLIENT_CIPHER_DUMMIES** = NULL
- static **tor_tls_context_t** * **global_tls_context** = NULL
- static int **tls_library_is_initialized** = 0
- static const char **CLIENT_CIPHER_LIST** [ ]
- static const **cipher_info_t** **CLIENT_CIPHER_INFO_LIST** [ ]
- static const int **N_CLIENT_CIPHERS**

4.52.1 Detailed Description

Wrapper functions to present a consistent interface to TLS, SSL, and X.509 functions from OpenSSL.

4.52.2 Define Documentation

4.52.2.1 #define IDENTITY_CERT_LIFETIME (365*24*60*60)

How long do identity certificates live? (sec)

Referenced by tor_tls_context_new().

4.52.2.2 #define SERVER_CIPHER_LIST

Value:

```
(TLS1_TXT_DHE_RSA_WITH_AES_256_SHA ":"          \
   TLS1_TXT_DHE_RSA_WITH_AES_128_SHA ":"          \
   SSL3_TXT_EDH_RSA_DES_192_CBC3_SHA)
```

List of ciphers that servers should select from.

Referenced by tor_tls_handshake(), and tor_tls_new().

4.52.3 Function Documentation

4.52.3.1 void _check_no_tls_errors (const char * *fname*, int *line*)

Implement check_no_tls_errors: If there are any pending OpenSSL errors, log an error message.

References LD_CRYPTO, LD_NET, and tls_log_errors().

4.52.3.2 static int always_accept_verify_cb (int *preverify_ok*, X509_STORE_CTX * *x509_ctx*) `[static]`

We need to give OpenSSL a callback to verify certificates. This is it: We always accept peer certs and complete the handshake. We don't validate them until later.

Referenced by tor_tls_context_new(), and tor_tls_handshake().

4.52.3.3 static HT_HEAD (HT_PROTOTYPE(*tlsmap*, HT_PROTOTYPE(*tor_tls_t*) `[static]`

Map from SSL* pointers to **tor_tls_t** (p. 243) objects using those pointers. Helper: given a SSL* pointer, return the **tor_tls_t** (p. 243) object using that pointer.

4.52.3.4 static void log_cert_lifetime (X509 * *cert*, const char * *problem*) `[static]`

Warn that a certificate lifetime extends through a certain range.

References LD_GENERAL, LD_NET, tls_log_errors(), and tor_free.

Referenced by tor_tls_check_lifetime().

4.52.3.5 static void rectify_client_ciphers (STACK_OF(SSL_CIPHER)** *ciphers*) `[static]`

Replace ***ciphers** with a new list of SSL ciphersuites: specifically, a list designed to mimic a common web browser. Some of the ciphers in the list won't actually be implemented by OpenSSL: that's okay so long as the server doesn't select them, and the server won't select anything besides what's in SERVER_CIPHER_LIST.

[If the server **does** select a bogus cipher, we won't crash or anything; we'll just fail later when we try to look up the cipher in ssl->cipher_list_by_id.]

References CLIENT_CIPHER_DUMMIES, CLIENT_CIPHER_LIST, cipher_info_t::id, LD_NET, N_CLIENT_CIPHERS, cipher_info_t::name, and tor_assert.

Referenced by tor_tls_new().

4.52.3.6 static const char* ssl_state_to_string (int *ssl_state*) `[static]`

Return the symbolic name of an OpenSSL state.

References tor_snprintf().

Referenced by tls_log_errors(), tor_tls_get_error(), and tor_tls_handshake().

4.52.3.7 static STACK_OF (SSL_CIPHER) `[static]`

A stack of SSL_CIPHER objects, some real, some fake. See **rectify_client_ciphers()** (p. 1222) for details.

Referenced by tor_tls_handshake(), and try_to_extract_certs_from_tls().

4.52.3.8 static void tls_log_errors (tor_tls_t * *tls*, int *severity*, int *domain*, const char * *doing*) [static]

Log all pending tls errors at level **severity**. Use **doing** to describe our current activities.

References ssl_state_to_string().

Referenced by _check_no_tls_errors(), log_cert_lifetime(), tor_tls_check_lifetime(), tor_tls_context_new(), tor_tls_create_certificate(), tor_tls_get_error(), tor_tls_handshake(), tor_tls_new(), tor_tls_peer_has_cert(), and tor_tls_verify().

4.52.3.9 static int tor_errno_to_tls_error (int *e*) [static]

Convert an errno (or a WSAerrno on windows) into a TOR_TLS_* error code.

Referenced by tor_tls_get_error().

4.52.3.10 void tor_tls_block_renegotiation (tor_tls_t * *tls*)

If this version of openssl supports it, turn off renegotiation on **tls**. (Our protocol never requires this for security, but it's nice to use belt-and-suspenders here.)

Referenced by connection_or_tls_renegotiated_cb(), and connection_tls_finish_handshake().

4.52.3.11 int tor_tls_check_lifetime (tor_tls_t * *tls*, int *tolerance*)

Check whether the certificate set on the connection **tls** is expired or not-yet-valid, give or take **tolerance** seconds. Return 0 for valid, -1 for failure.

NOTE: you should call tor_tls_verify before tor_tls_check_lifetime.

References LD_NET, log_cert_lifetime(), and tls_log_errors().

4.52.3.12 static int tor_tls_client_is_using_v2_ciphers (const SSL * *ssl*, const char * *address*) [static]

Return true iff the cipher list suggested by the client for **ssl** is a list that indicates that the client knows how to do the v2 TLS connection handshake.

References LD_NET, smartlist_add(), smartlist_create(), smartlist_free(), smartlist_join_strings(), and tor_free.

Referenced by tor_tls_handshake(), and tor_tls_server_info_callback().

4.52.3.13 static void tor_tls_context_decref (tor_tls_context_t * *ctx*) [static]

Remove a reference to **ctx**, and free it if it has no more references.

References crypto_free_pk_env(), tor_tls_context_t::ctx, tor_tls_context_t::key, tor_tls_context_t::my_cert, tor_tls_context_t::my_id_cert, tor_tls_context_t::refcnt, tor_assert, and tor_free.

Referenced by tor_tls_context_new(), tor_tls_free(), and tor_tls_free_all().

4.52.3.14 static void tor_tls_context_incref (tor_tls_context_t ∗ *ctx*) [static]

Increase the reference count of **ctx**.

References tor_tls_context_t::refcnt.

Referenced by tor_tls_new().

4.52.3.15 int tor_tls_context_new (crypto_pk_env_t ∗ *identity*, unsigned int *key_lifetime*)

Create a new TLS context for use with Tor TLS handshakes. **identity** should be set to the identity key used to sign the certificate, and **nickname** set to the nickname to use.

You can call this function multiple times. Each time you call it, it generates new certificates; all new connections will use the new SSL context.

References _crypto_dh_env_get_dh(), _crypto_pk_env_get_evp_pkey(), always_accept_verify_cb(), crypto_dh_free(), crypto_dh_new(), crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_dup_key(), crypto_random_hostname(), tor_tls_context_t::ctx, IDENTITY_CERT_LIFETIME, tor_tls_context_t::key, LD_CRYPTO, LD_NET, tor_tls_context_t::my_cert, tor_tls_context_t::my_id_cert, tor_tls_context_t::refcnt, tls_log_errors(), tor_assert, tor_free, tor_tls_context_decref(), tor_tls_create_certificate(), tor_tls_init(), and use_unsafe_renegotiation_op.

Referenced by init_keys(), and run_scheduled_events().

4.52.3.16 static X509 ∗ tor_tls_create_certificate (crypto_pk_env_t ∗ *rsa*, crypto_pk_env_t ∗ *rsa_sign*, const char ∗ *cname*, const char ∗ *cname_sign*, unsigned int *cert_lifetime*) [static]

Generate and sign an X509 certificate with the public key **rsa**, signed by the private key **rsa_sign**. The commonName of the certificate will be **cname**; the commonName of the issuer will be **cname_sign**. The cert will be valid for **cert_lifetime** seconds starting from now. Return a certificate on success, NULL on failure.

References _crypto_pk_env_get_evp_pkey(), LD_NET, tls_log_errors(), tor_assert, tor_tls_init(), and tor_x509_name_new().

Referenced by tor_tls_context_new().

4.52.3.17 static INLINE unsigned int tor_tls_entry_hash (const tor_tls_t ∗ *a*) [static]

Helper: return a hash value for a **tor_tls_t** (p. 243) by its SSL.

4.52.3.18 const char∗ tor_tls_err_to_string (int *err*)

Given a TOR_TLS_∗ error code, return a string equivalent.

Referenced by connection_read_to_buf(), and connection_tls_continue_handshake().

4.52.3.19 void tor_tls_free (tor_tls_t ∗ *tls*)

Release resources associated with a TLS object. Does not close the underlying file descriptor.

References LD_BUG, tor_tls_t::negotiated_callback, tor_assert, tor_free, and tor_tls_context_decref().

Referenced by _connection_free().

4.52.3.20 void tor_tls_free_all (void)

Free all global TLS structures.

References CLIENT_CIPHER_DUMMIES, LD_MM, tor_free, and tor_tls_context_decref().

Referenced by tor_free_all().

4.52.3.21 void tor_tls_get_buffer_sizes (tor_tls_t * *tls*, size_t * *rbuf_capacity*, size_t * *rbuf_bytes*, size_t * *wbuf_capacity*, size_t * *wbuf_bytes*)

Examine the amount of memory used and available for buffers in **tls**. Set ∗**rbuf_capacity** to the amount of storage allocated for the read buffer and ∗**rbuf_bytes** to the amount actually used. Set ∗**wbuf_capacity** to the amount of storage allocated for the write buffer and ∗**wbuf_bytes** to the amount actually used.

Referenced by dumpstats().

4.52.3.22 static int tor_tls_get_error (tor_tls_t * *tls*, int *r*, int *extra*, const char * *doing*, int *severity*, int *domain*) `[static]`

Given a TLS object and the result of an SSL_∗ call, use SSL_get_error to determine whether an error has occurred, and if so which one. Return one of TOR_TLS_{DONE|WANTREAD|WANTWRITE|ERROR}. If extra&CATCH_SYSCALL is true, return _TOR_TLS_SYSCALL instead of reporting syscall errors. If extra&CATCH_ZERO is true, return _TOR_TLS_ZERORETURN instead of reporting zero-return errors.

If an error has occurred, log it at level **severity** and describe the current action as **doing**.

References LD_NET, ssl_state_to_string(), tls_log_errors(), and tor_errno_to_tls_error().

Referenced by tor_tls_handshake(), tor_tls_read(), tor_tls_renegotiate(), tor_tls_shutdown(), and tor_tls_write().

4.52.3.23 size_t tor_tls_get_forced_write_size (tor_tls_t * *tls*)

If **tls** requires that the next write be of a particular size, return that size. Otherwise, return 0.

References tor_tls_t::wantwrite_n.

Referenced by flush_chunk_tls().

4.52.3.24 void tor_tls_get_n_raw_bytes (tor_tls_t * *tls*, size_t * *n_read*, size_t * *n_written*)

Sets n_read and n_written to the number of bytes read and written, respectively, on the raw socket used by **tls** since the last time this function was called on **tls**.

References tor_tls_t::last_read_count, tor_tls_t::last_write_count, and LD_BUG.

Referenced by connection_handle_write_impl(), and connection_read_to_buf().

4.52.3.25 int tor_tls_get_pending_bytes (tor_tls_t ∗ *tls*)

Return the number of bytes available for reading from **tls**.

References tor_assert.

Referenced by connection_or_process_cells_from_inbuf(), and connection_read_to_buf().

4.52.3.26 int tor_tls_handshake (tor_tls_t ∗ *tls*)

Perform initial handshake on **tls**. When finished, returns TOR_TLS_DONE. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References always_accept_verify_cb(), tor_tls_t::isServer, LD_BUG, LD_HANDSHAKE, SERVER_CIPHER_LIST, ssl_state_to_string(), STACK_OF(), tls_log_errors(), tor_assert, tor_tls_client_is_using_v2_ciphers(), tor_tls_get_error(), tor_tls_unblock_renegotiation(), and tor_tls_t::wasV2Handshake.

Referenced by connection_tls_continue_handshake().

4.52.3.27 static void tor_tls_init (void) `[static]`

Initialize OpenSSL, unless it has already been initialized.

References LD_GENERAL, tls_library_is_initialized, use_unsafe_renegotiation_flag, and use_unsafe_renegotiation_op.

Referenced by tor_tls_context_new(), and tor_tls_create_certificate().

4.52.3.28 int tor_tls_is_server (tor_tls_t ∗ *tls*)

Return whether this tls initiated the connect (client) or received it (server).

References tor_tls_t::isServer, and tor_assert.

Referenced by connection_or_nonopen_was_started_here(), and connection_tls_continue_handshake().

4.52.3.29 tor_tls_t∗ tor_tls_new (int *sock*, int *isServer*)

Create a new TLS object from a file descriptor, and a flag to determine whether it is functioning as a server.

References CLIENT_CIPHER_LIST, crypto_random_hostname(), tor_tls_context_t::ctx, tor_tls_t::isServer, tor_tls_t::last_read_count, tor_tls_t::last_write_count, LD_NET, rectify_client_ciphers(), SERVER_CIPHER_LIST, tls_log_errors(), tor_assert, tor_free, tor_tls_context_incref(), tor_tls_server_info_callback(), and tor_tls_t::wantwrite_n.

Referenced by connection_tls_start_handshake().

4.52.3.30 int tor_tls_peer_has_cert (tor_tls_t ∗ *tls*)

Return true iff this TLS connection is authenticated.

References LD_HANDSHAKE, and tls_log_errors().

Referenced by connection_or_check_valid_tls_handshake().

4.52.3.31 int tor_tls_read (tor_tls_t * *tls*, char * *cp*, size_t *len*)

Underlying function for TLS reading. Reads up to **len** characters from **tls** into **cp**. On success, returns the number of characters read. On failure, returns TOR_TLS_ERROR, TOR_TLS_-CLOSE, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References tor_tls_t::callback_arg, tor_tls_t::got_renegotiate, LD_NET, tor_tls_-t::negotiated_callback, tor_assert, and tor_tls_get_error().

Referenced by read_to_chunk_tls().

4.52.3.32 int tor_tls_renegotiate (tor_tls_t * *tls*)

Client only: Renegotiate a TLS session. When finished, returns TOR_TLS_DONE. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References tor_tls_t::isServer, LD_HANDSHAKE, tor_assert, and tor_tls_get_error().

Referenced by connection_tls_continue_handshake().

4.52.3.33 static void tor_tls_server_info_callback (const SSL * *ssl*, int *type*, int *val*) `[static]`

Invoked when we're accepting a connection on **ssl**, and the connection changes state. We use this:

- To alter the state of the handshake partway through, so we do not send or request extra certificates in v2 handshakes.
- To detect renegotiation

References tor_tls_t::got_renegotiate, LD_BUG, tor_tls_t::negotiated_callback, tor_tls_-client_is_using_v2_ciphers(), and tor_tls_t::wasV2Handshake.

Referenced by tor_tls_new(), and tor_tls_set_renegotiate_callback().

4.52.3.34 void tor_tls_set_logged_address (tor_tls_t * *tls*, const char * *address*)

Make future log messages about **tls** display the address **address**.

References tor_assert, and tor_free.

Referenced by connection_tls_start_handshake().

4.52.3.35 void tor_tls_set_renegotiate_callback (tor_tls_t * *tls*, void(*)(tor_tls_t *, void *arg) *cb*, void * *arg*)

Set **cb** to be called with argument **arg** whenever **tls** next gets a client-side renegotiate in the middle of a read. Do not invoke this function until *after* initial handshaking is done!

References tor_tls_t::callback_arg, tor_tls_t::got_renegotiate, tor_tls_t::negotiated_callback, and tor_tls_server_info_callback().

Referenced by connection_or_tls_renegotiated_cb(), and connection_tls_continue_-handshake().

4.52.3.36 int tor_tls_shutdown (tor_tls_t * *tls*)

Shut down an open tls connection **tls**. When finished, returns TOR_TLS_DONE. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References LD_NET, tor_assert, and tor_tls_get_error().

4.52.3.37 static void tor_tls_unblock_renegotiation (tor_tls_t * *tls*) `[static]`

If this version of openssl requires it, turn on renegotiation on **tls**.

References use_unsafe_renegotiation_flag, and use_unsafe_renegotiation_op.

Referenced by tor_tls_handshake().

4.52.3.38 int tor_tls_used_v1_handshake (tor_tls_t * *tls*)

Return true iff the initial TLS connection at **tls** did not use a v2 TLS handshake. Output is undefined if the handshake isn't finished.

References tor_tls_t::isServer, and tor_tls_t::wasV2Handshake.

Referenced by connection_tls_continue_handshake(), and connection_tls_finish_handshake().

4.52.3.39 int tor_tls_verify (int *severity*, tor_tls_t * *tls*, crypto_pk_env_t ** *identity_key*)

If the provided tls connection is authenticated and has a certificate chain that is currently valid and signed, then set ***identity_key** to the identity certificate's key and return 0. Else, return -1 and log complaints with log-level **severity**.

References _crypto_new_pk_env_rsa(), LD_HANDSHAKE, LD_PROTOCOL, tls_log_errors(), and try_to_extract_certs_from_tls().

Referenced by connection_or_check_valid_tls_handshake().

4.52.3.40 int tor_tls_write (tor_tls_t * *tls*, const char * *cp*, size_t *n*)

Underlying function for TLS writing. Write up to **n** characters from **cp** onto **tls**. On success, returns the number of characters written. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References LD_NET, tor_assert, tor_tls_get_error(), and tor_tls_t::wantwrite_n.

Referenced by flush_chunk_tls().

4.52.3.41 static X509_NAME* tor_x509_name_new (const char * *cname*) `[static]`

Return a newly allocated X509 name with commonName **cname**.

Referenced by tor_tls_create_certificate().

4.52.3.42 static void try_to_extract_certs_from_tls (int *severity*, tor_tls_t * *tls*, X509 ** *cert_out*, X509 ** *id_cert_out*) `[static]`

Helper function: try to extract a link certificate and an identity certificate from **tls**, and store them in ***cert_out** and ***id_cert_out** respectively. Log all messages at level **severity**.

Note that a reference is added to cert_out, so it needs to be freed. id_cert_out doesn't.

References LD_PROTOCOL, and STACK_OF().

Referenced by tor_tls_verify().

4.52.4 Variable Documentation

4.52.4.1 SSL_CIPHER* CLIENT_CIPHER_DUMMIES = NULL `[static]`

An array of fake SSL_CIPHER objects that we use in order to trick OpenSSL in client mode into advertising the ciphers we want. See **rectify_client_ciphers()** (p. 1222) for details.

Referenced by rectify_client_ciphers(), and tor_tls_free_all().

4.52.4.2 const cipher_info_t CLIENT_CIPHER_INFO_LIST[ ] `[static]`

A list of all the ciphers that clients should advertise, including items that OpenSSL might not know about.

4.52.4.3 const char CLIENT_CIPHER_LIST[ ] `[static]`

Initial value:

```
# 554 "/home/arma/work/onion/git/tor/src/common/tortls.c" 2
```

List of ciphers that clients should advertise, omitting items that our OpenSSL doesn't know about.

Referenced by rectify_client_ciphers(), and tor_tls_new().

4.52.4.4 tor_tls_context_t* global_tls_context = NULL `[static]`

Global tls context. We keep it here because nobody else needs to touch it.

4.52.4.5 const int N_CLIENT_CIPHERS `[static]`

Initial value:

```
  sizeof(CLIENT_CIPHER_INFO_LIST)/sizeof(CLIENT_CIPHER_INFO_LIST[0])
```

The length of CLIENT_CIPHER_INFO_LIST and CLIENT_CIPHER_DUMMIES.

Referenced by rectify_client_ciphers().

4.52.4.6 int tls_library_is_initialized = 0 `[static]`

True iff **tor_tls_init()** (p. 1226) has been called.

Referenced by tor_tls_init().

4.52.4.7 int use_unsafe_renegotiation_flag = 0 `[static]`

Does the run-time openssl version look like we need SSL3_FLAGS_ALLOW_UNSAFE_LEGACY_RENEGOTIATION?

Referenced by tor_tls_init(), and tor_tls_unblock_renegotiation().

4.52.4.8 int use_unsafe_renegotiation_op = 0 `[static]`

Does the run-time openssl version look like we need SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION?

Referenced by tor_tls_context_new(), tor_tls_init(), and tor_tls_unblock_renegotiation().

4.53 tortls.h File Reference

Headers for **tortls.c** (p. 1219).

`#include "crypto.h"`

`#include "compat.h"`

Defines

- #define **_MIN_TOR_TLS_ERROR_VAL** -9
- #define **TOR_TLS_ERROR_MISC** -9
- #define **TOR_TLS_ERROR_IO** -8
- #define **TOR_TLS_ERROR_CONNREFUSED** -7
- #define **TOR_TLS_ERROR_CONNRESET** -6
- #define **TOR_TLS_ERROR_NO_ROUTE** -5
- #define **TOR_TLS_ERROR_TIMEOUT** -4
- #define **TOR_TLS_CLOSE** -3
- #define **TOR_TLS_WANTREAD** -2
- #define **TOR_TLS_WANTWRITE** -1
- #define **TOR_TLS_DONE** 0
- #define **CASE_TOR_TLS_ERROR_ANY_NONIO**
- #define **CASE_TOR_TLS_ERROR_ANY**
- #define **TOR_TLS_IS_ERROR**(rv) ((rv) < TOR_TLS_CLOSE)
- #define **check_no_tls_errors**() _check_no_tls_errors(__FILE__,__LINE__)

Typedefs

- typedef struct **tor_tls_t** **tor_tls_t**

Functions

- const char * **tor_tls_err_to_string** (int err)
- void **tor_tls_free_all** (void)
- int **tor_tls_context_new** (**crypto_pk_env_t** *rsa, unsigned int key_lifetime)
- **tor_tls_t** * **tor_tls_new** (int sock, int is_server)
- void **tor_tls_set_logged_address** (**tor_tls_t** *tls, const char *address)
- void **tor_tls_set_renegotiate_callback** (**tor_tls_t** *tls, void(*cb)(**tor_tls_t** *, void *arg), void *arg)
- int **tor_tls_is_server** (**tor_tls_t** *tls)
- void **tor_tls_free** (**tor_tls_t** *tls)
- int **tor_tls_peer_has_cert** (**tor_tls_t** *tls)
- int **tor_tls_verify** (int severity, **tor_tls_t** *tls, **crypto_pk_env_t** **identity)
- int **tor_tls_check_lifetime** (**tor_tls_t** *tls, int tolerance)
- int **tor_tls_read** (**tor_tls_t** *tls, char *cp, size_t len)
- int **tor_tls_write** (**tor_tls_t** *tls, const char *cp, size_t n)
- int **tor_tls_handshake** (**tor_tls_t** *tls)
- int **tor_tls_renegotiate** (**tor_tls_t** *tls)
- void **tor_tls_block_renegotiation** (**tor_tls_t** *tls)
- int **tor_tls_shutdown** (**tor_tls_t** *tls)

- int **tor_tls_get_pending_bytes** (**tor_tls_t** *tls)
- size_t **tor_tls_get_forced_write_size** (**tor_tls_t** *tls)
- void **tor_tls_get_n_raw_bytes** (**tor_tls_t** *tls, size_t *n_read, size_t *n_written)
- void **tor_tls_get_buffer_sizes** (**tor_tls_t** *tls, size_t *rbuf_capacity, size_t *rbuf_bytes, size_t *wbuf_capacity, size_t *wbuf_bytes)
- int **tor_tls_used_v1_handshake** (**tor_tls_t** *tls)
- void **_check_no_tls_errors** (const char *fname, int line)

4.53.1 Detailed Description

Headers for **tortls.c** (p. 1219).

4.53.2 Define Documentation

4.53.2.1 #define CASE_TOR_TLS_ERROR_ANY

Value:

```
CASE_TOR_TLS_ERROR_ANY_NONIO:                 \
  case TOR_TLS_ERROR_IO
```

Use this macro in a switch statement to catch _any_ TLS error. That way, if more errors are added, your switches will still work.

Referenced by connection_handle_write_impl(), and connection_tls_continue_handshake().

4.53.2.2 #define CASE_TOR_TLS_ERROR_ANY_NONIO

Value:

```
case TOR_TLS_ERROR_MISC:                      \
  case TOR_TLS_ERROR_CONNREFUSED:               \
  case TOR_TLS_ERROR_CONNRESET:                 \
  case TOR_TLS_ERROR_NO_ROUTE:                  \
  case TOR_TLS_ERROR_TIMEOUT
```

Collection of case statements for all TLS errors that are not due to underlying IO failure.

Referenced by connection_read_to_buf().

4.53.3 Function Documentation

4.53.3.1 void _check_no_tls_errors (const char * *fname*, int *line*)

Implement check_no_tls_errors: If there are any pending OpenSSL errors, log an error message.

References LD_CRYPTO, LD_NET, and tls_log_errors().

4.53.3.2 void tor_tls_block_renegotiation (tor_tls_t * *tls*)

If this version of openssl supports it, turn off renegotiation on **tls**. (Our protocol never requires this for security, but it's nice to use belt-and-suspenders here.)

Referenced by connection_or_tls_renegotiated_cb(), and connection_tls_finish_handshake().

4.53.3.3 int tor_tls_check_lifetime (tor_tls_t ∗ *tls*, int *tolerance*)

Check whether the certificate set on the connection **tls** is expired or not-yet-valid, give or take **tolerance** seconds. Return 0 for valid, -1 for failure.

NOTE: you should call tor_tls_verify before tor_tls_check_lifetime.

References LD_NET, log_cert_lifetime(), and tls_log_errors().

4.53.3.4 int tor_tls_context_new (crypto_pk_env_t ∗ *identity*, unsigned int *key_lifetime*)

Create a new TLS context for use with Tor TLS handshakes. **identity** should be set to the identity key used to sign the certificate, and **nickname** set to the nickname to use.

You can call this function multiple times. Each time you call it, it generates new certificates; all new connections will use the new SSL context.

References _crypto_dh_env_get_dh(), _crypto_pk_env_get_evp_pkey(), always_accept_verify_cb(), crypto_dh_free(), crypto_dh_new(), crypto_free_pk_env(), crypto_new_pk_env(), crypto_pk_dup_key(), crypto_random_hostname(), tor_tls_context_t::ctx, IDENTITY_CERT_LIFETIME, tor_tls_context_t::key, LD_CRYPTO, LD_NET, tor_tls_context_t::my_cert, tor_tls_context_t::my_id_cert, tor_tls_context_t::refcnt, tls_log_errors(), tor_assert, tor_free, tor_tls_context_decref(), tor_tls_create_certificate(), tor_tls_init(), and use_unsafe_renegotiation_op.

Referenced by init_keys(), and run_scheduled_events().

4.53.3.5 const char∗ tor_tls_err_to_string (int *err*)

Given a TOR_TLS_∗ error code, return a string equivalent.

Referenced by connection_read_to_buf(), and connection_tls_continue_handshake().

4.53.3.6 void tor_tls_free (tor_tls_t ∗ *tls*)

Release resources associated with a TLS object. Does not close the underlying file descriptor.

References LD_BUG, tor_tls_t::negotiated_callback, tor_assert, tor_free, and tor_tls_context_decref().

Referenced by _connection_free().

4.53.3.7 void tor_tls_free_all (void)

Free all global TLS structures.

References CLIENT_CIPHER_DUMMIES, LD_MM, tor_free, and tor_tls_context_decref().

Referenced by tor_free_all().

4.53.3.8 void tor_tls_get_buffer_sizes (tor_tls_t ∗ *tls*, size_t ∗ *rbuf_capacity*, size_t ∗ *rbuf_bytes*, size_t ∗ *wbuf_capacity*, size_t ∗ *wbuf_bytes*)

Examine the amount of memory used and available for buffers in **tls**. Set ∗**rbuf_capacity** to the amount of storage allocated for the read buffer and ∗**rbuf_bytes** to the amount actually used.

Set ∗**wbuf_capacity** to the amount of storage allocated for the write buffer and ∗**wbuf_bytes** to the amount actually used.

Referenced by dumpstats().

4.53.3.9 size_t tor_tls_get_forced_write_size (tor_tls_t ∗ *tls*)

If **tls** requires that the next write be of a particular size, return that size. Otherwise, return 0.

References tor_tls_t::wantwrite_n.

Referenced by flush_chunk_tls().

4.53.3.10 void tor_tls_get_n_raw_bytes (tor_tls_t ∗ *tls*, size_t ∗ *n_read*, size_t ∗ *n_written*)

Sets n_read and n_written to the number of bytes read and written, respectively, on the raw socket used by **tls** since the last time this function was called on **tls**.

References tor_tls_t::last_read_count, tor_tls_t::last_write_count, and LD_BUG.

Referenced by connection_handle_write_impl(), and connection_read_to_buf().

4.53.3.11 int tor_tls_get_pending_bytes (tor_tls_t ∗ *tls*)

Return the number of bytes available for reading from **tls**.

References tor_assert.

Referenced by connection_or_process_cells_from_inbuf(), and connection_read_to_buf().

4.53.3.12 int tor_tls_handshake (tor_tls_t ∗ *tls*)

Perform initial handshake on **tls**. When finished, returns TOR_TLS_DONE. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References always_accept_verify_cb(), tor_tls_t::isServer, LD_BUG, LD_HANDSHAKE, SERVER_CIPHER_LIST, ssl_state_to_string(), STACK_OF(), tls_log_errors(), tor_assert, tor_tls_client_is_using_v2_ciphers(), tor_tls_get_error(), tor_tls_unblock_renegotiation(), and tor_tls_t::wasV2Handshake.

Referenced by connection_tls_continue_handshake().

4.53.3.13 int tor_tls_is_server (tor_tls_t ∗ *tls*)

Return whether this tls initiated the connect (client) or received it (server).

References tor_tls_t::isServer, and tor_assert.

Referenced by connection_or_nonopen_was_started_here(), and connection_tls_continue_handshake().

4.53.3.14 tor_tls_t∗ tor_tls_new (int *sock*, int *isServer*)

Create a new TLS object from a file descriptor, and a flag to determine whether it is functioning as a server.

References CLIENT_CIPHER_LIST, crypto_random_hostname(), tor_tls_context_t::ctx, tor_tls_t::isServer, tor_tls_t::last_read_count, tor_tls_t::last_write_count, LD_NET, rectify_client_ciphers(), SERVER_CIPHER_LIST, tls_log_errors(), tor_assert, tor_free, tor_tls_context_incref(), tor_tls_server_info_callback(), and tor_tls_t::wantwrite_n.

Referenced by connection_tls_start_handshake().

4.53.3.15 int tor_tls_peer_has_cert (tor_tls_t * *tls*)

Return true iff this TLS connection is authenticated.

References LD_HANDSHAKE, and tls_log_errors().

Referenced by connection_or_check_valid_tls_handshake().

4.53.3.16 int tor_tls_read (tor_tls_t * *tls*, char * *cp*, size_t *len*)

Underlying function for TLS reading. Reads up to **len** characters from **tls** into **cp**. On success, returns the number of characters read. On failure, returns TOR_TLS_ERROR, TOR_TLS_CLOSE, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References tor_tls_t::callback_arg, tor_tls_t::got_renegotiate, LD_NET, tor_tls_t::negotiated_callback, tor_assert, and tor_tls_get_error().

Referenced by read_to_chunk_tls().

4.53.3.17 int tor_tls_renegotiate (tor_tls_t * *tls*)

Client only: Renegotiate a TLS session. When finished, returns TOR_TLS_DONE. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References tor_tls_t::isServer, LD_HANDSHAKE, tor_assert, and tor_tls_get_error().

Referenced by connection_tls_continue_handshake().

4.53.3.18 void tor_tls_set_logged_address (tor_tls_t * *tls*, const char * *address*)

Make future log messages about **tls** display the address **address**.

References tor_assert, and tor_free.

Referenced by connection_tls_start_handshake().

4.53.3.19 void tor_tls_set_renegotiate_callback (tor_tls_t * *tls*, void(*)(tor_tls_t *, void *arg) *cb*, void * *arg*)

Set **cb** to be called with argument **arg** whenever **tls** next gets a client-side renegotiate in the middle of a read. Do not invoke this function until *after* initial handshaking is done!

References tor_tls_t::callback_arg, tor_tls_t::got_renegotiate, tor_tls_t::negotiated_callback, and tor_tls_server_info_callback().

Referenced by connection_or_tls_renegotiated_cb(), and connection_tls_continue_handshake().

4.53.3.20 int tor_tls_shutdown (tor_tls_t * *tls*)

Shut down an open tls connection **tls**. When finished, returns TOR_TLS_DONE. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References LD_NET, tor_assert, and tor_tls_get_error().

4.53.3.21 int tor_tls_used_v1_handshake (tor_tls_t * *tls*)

Return true iff the initial TLS connection at **tls** did not use a v2 TLS handshake. Output is undefined if the handshake isn't finished.

References tor_tls_t::isServer, and tor_tls_t::wasV2Handshake.

Referenced by connection_tls_continue_handshake(), and connection_tls_finish_handshake().

4.53.3.22 int tor_tls_verify (int *severity*, tor_tls_t * *tls*, crypto_pk_env_t ** *identity_key*)

If the provided tls connection is authenticated and has a certificate chain that is currently valid and signed, then set ***identity_key** to the identity certificate's key and return 0. Else, return -1 and log complaints with log-level **severity**.

References _crypto_new_pk_env_rsa(), LD_HANDSHAKE, LD_PROTOCOL, tls_log_errors(), and try_to_extract_certs_from_tls().

Referenced by connection_or_check_valid_tls_handshake().

4.53.3.23 int tor_tls_write (tor_tls_t * *tls*, const char * *cp*, size_t *n*)

Underlying function for TLS writing. Write up to **n** characters from **cp** onto **tls**. On success, returns the number of characters written. On failure, returns TOR_TLS_ERROR, TOR_TLS_WANTREAD, or TOR_TLS_WANTWRITE.

References LD_NET, tor_assert, tor_tls_get_error(), and tor_tls_t::wantwrite_n.

Referenced by flush_chunk_tls().

4.54 util.c File Reference

Common functions for strings, IO, network, data structures, process control.

```
#include "orconfig.h"
#include "util.h"
#include "log.h"
#include "crypto.h"
#include "torint.h"
#include "container.h"
#include "address.h"
#include <dirent.h>
#include <pwd.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <fcntl.h>
#include <time.h>
#include <malloc.h>
```

Data Structures

- struct **open_file_t**

Defines

- #define **_GNU_SOURCE**
- #define **__USE_ISOC99** 1
- #define **DMALLOC_FN_ARGS**
- #define **CHECK_STRTOX_RESULT**()
- #define **IS_LEAPYEAR**(y) (!(y % 4) && ((y % 100) || !(y % 400)))

- #define **TOR_ISODIGIT**(c) ('0' <= (c) && (c) <= '7')
- #define **MAX_SCANF_WIDTH** 9999

Functions

- void * **_tor_malloc** (size_t size DMALLOC_PARAMS)
- void * **_tor_malloc_zero** (size_t size DMALLOC_PARAMS)
- void * **_tor_realloc** (void *ptr, size_t size DMALLOC_PARAMS)
- char * **_tor_strdup** (const char *s DMALLOC_PARAMS)
- char * **_tor_strndup** (const char *s, size_t n DMALLOC_PARAMS)
- void * **_tor_memdup** (const void *mem, size_t len DMALLOC_PARAMS)
- void **_tor_free** (void *mem)
- void * **_tor_malloc_roundup** (size_t *sizep DMALLOC_PARAMS)
- void **tor_log_mallinfo** (int severity)
- double **tor_mathlog** (double d)
- long **tor_lround** (double d)
- int **tor_log2** (uint64_t u64)
- uint64_t **round_to_power_of_2** (uint64_t u64)
- unsigned **round_to_next_multiple_of** (unsigned number, unsigned divisor)
- uint32_t **round_uint32_to_next_multiple_of** (uint32_t number, uint32_t divisor)
- uint64_t **round_uint64_to_next_multiple_of** (uint64_t number, uint64_t divisor)
- void **tor_strstrip** (char *s, const char *strip)
- const char * **hex_str** (const char *from, size_t fromlen)
- void **tor_strlower** (char *s)
- void **tor_strupper** (char *s)
- int **tor_strisprint** (const char *s)
- int **tor_strisnonupper** (const char *s)
- int **strcmpstart** (const char *s1, const char *s2)
- int **strcmp_len** (const char *s1, const char *s2, size_t s1_len)
- int **strcasecmpstart** (const char *s1, const char *s2)
- int **strcmpend** (const char *s1, const char *s2)
- int **strcasecmpend** (const char *s1, const char *s2)
- int **memcmpstart** (const void *mem, size_t memlen, const char *prefix)
- const char * **eat_whitespace** (const char *s)
- const char * **eat_whitespace_eos** (const char *s, const char *eos)
- const char * **eat_whitespace_no_nl** (const char *s)
- const char * **eat_whitespace_eos_no_nl** (const char *s, const char *eos)
- const char * **find_whitespace** (const char *s)
- const char * **find_whitespace_eos** (const char *s, const char *eos)
- const char * **find_str_at_start_of_line** (const char *haystack, const char *needle)
- int **tor_mem_is_zero** (const char *mem, size_t len)
- int **tor_digest_is_zero** (const char *digest)
- int **tor_digest256_is_zero** (const char *digest)
- long **tor_parse_long** (const char *s, int base, long min, long max, int *ok, char **next)
- unsigned long **tor_parse_ulong** (const char *s, int base, unsigned long min, unsigned long max, int *ok, char **next)
- double **tor_parse_double** (const char *s, double min, double max, int *ok, char **next)
- uint64_t **tor_parse_uint64** (const char *s, int base, uint64_t min, uint64_t max, int *ok, char **next)
- void **base16_encode** (char *dest, size_t destlen, const char *src, size_t srclen)

- static INLINE int **_hex_decode_digit** (char c)
- int **hex_decode_digit** (char c)
- int **base16_decode** (char ∗dest, size_t destlen, const char ∗src, size_t srclen)
- char ∗ **esc_for_log** (const char ∗s)
- const char ∗ **escaped** (const char ∗s)
- void **wrap_string** (**smartlist_t** ∗out, const char ∗string, size_t width, const char ∗prefix0, const char ∗prefixRest)
- double **tv_to_double** (const struct timeval ∗tv)
- int64_t **tv_to_msec** (const struct timeval ∗tv)
- int64_t **tv_to_usec** (const struct timeval ∗tv)
- long **tv_udiff** (const struct timeval ∗start, const struct timeval ∗end)
- long **tv_mdiff** (const struct timeval ∗start, const struct timeval ∗end)
- static int **n_leapdays** (int y1, int y2)
- time_t **tor_timegm** (struct tm ∗tm)
- void **format_rfc1123_time** (char ∗buf, time_t t)
- int **parse_rfc1123_time** (const char ∗buf, time_t ∗t)
- void **format_local_iso_time** (char ∗buf, time_t t)
- void **format_iso_time** (char ∗buf, time_t t)
- int **parse_iso_time** (const char ∗cp, time_t ∗t)
- int **parse_http_time** (const char ∗date, struct tm ∗tm)
- int **format_time_interval** (char ∗out, size_t out_len, long interval)
- time_t **approx_time** (void)
- void **update_approx_time** (time_t now)
- void **ftime_set_maximum_sloppiness** (int seconds)
- void **ftime_set_estimated_skew** (int seconds)
- int **ftime_maybe_after** (time_t now, time_t when)
- int **ftime_maybe_before** (time_t now, time_t when)
- int **ftime_definitely_after** (time_t now, time_t when)
- int **ftime_definitely_before** (time_t now, time_t when)
- ssize_t **write_all** (int fd, const char ∗buf, size_t count, int isSocket)
- ssize_t **read_all** (int fd, char ∗buf, size_t count, int isSocket)
- static void **clean_name_for_stat** (char ∗name)
- **file_status_t** **file_status** (const char ∗fname)
- int **check_private_dir** (const char ∗dirname, **cpd_check_t** check)
- int **write_str_to_file** (const char ∗fname, const char ∗str, int bin)
- int **start_writing_to_file** (const char ∗fname, int open_flags, int mode, **open_file_t** ∗∗data_out)
- FILE ∗ **fdopen_file** (**open_file_t** ∗file_data)
- FILE ∗ **start_writing_to_stdio_file** (const char ∗fname, int open_flags, int mode, **open_file_t** ∗∗data_out)
- static int **finish_writing_to_file_impl** (**open_file_t** ∗file_data, int abort_write)
- int **finish_writing_to_file** (**open_file_t** ∗file_data)
- int **abort_writing_to_file** (**open_file_t** ∗file_data)
- static int **write_chunks_to_file_impl** (const char ∗fname, const **smartlist_t** ∗chunks, int open_flags)
- int **write_chunks_to_file** (const char ∗fname, const **smartlist_t** ∗chunks, int bin)
- int **write_bytes_to_file** (const char ∗fname, const char ∗str, size_t len, int bin)
- int **append_bytes_to_file** (const char ∗fname, const char ∗str, size_t len, int bin)
- char ∗ **read_file_to_str** (const char ∗filename, int flags, struct stat ∗stat_out)
- static const char ∗ **unescape_string** (const char ∗s, char ∗∗result, size_t ∗size_out)

- const char * **parse_config_line_from_str** (const char *line, char **key_out, char **value_out)
- char * **expand_filename** (const char *filename)
- static int **digit_to_num** (char d)
- static int **scan_unsigned** (const char **bufp, unsigned *out, int width)
- static int **scan_string** (const char **bufp, char *out, int width)
- int **tor_vsscanf** (const char *buf, const char *pattern, va_list ap)
- int **tor_sscanf** (const char *buf, const char *pattern,...)
- **smartlist_t** * **tor_listdir** (const char *dirname)
- int **path_is_relative** (const char *filename)
- void **start_daemon** (void)
- void **finish_daemon** (const char *desired_cwd)
- void **write_pidfile** (char *filename)

Variables

- static const int **days_per_month** [ ]
- static const char * **WEEKDAY_NAMES** [ ]
- static const char * **MONTH_NAMES** [ ]
- static time_t **cached_approx_time** = 0
- static int **ftime_skew** = 0
- static int **ftime_slop** = 60
- static int **start_daemon_called** = 0
- static int **finish_daemon_called** = 0
- static int **daemon_filedes** [2]

4.54.1 Detailed Description

Common functions for strings, IO, network, data structures, process control.

4.54.2 Define Documentation

4.54.2.1 #define CHECK_STRTOX_RESULT()

Value:

```
/* Was at least one character converted? */           \
  if (endptr == s)                                      \
    goto err;                                           \
  /* Were there unexpected unconverted characters? */   \
  if (!next && *endptr)                                 \
    goto err;                                           \
  /* Is r within limits? */                             \
  if (r < min || r > max)                               \
    goto err;                                           \
  if (ok) *ok = 1;                                      \
  if (next) *next = endptr;                             \
  return r;                                             \
 err:                                                   \
  if (ok) *ok = 0;                                      \
  if (next) *next = endptr;                             \
  return 0
```

4.54.2.2 #define IS_LEAPYEAR(y) (!(y % 4) && ((y % 100) || !(y % 400)))

Yield true iff **y** is a leap-year.

Referenced by tor_timegm().

4.54.3 Function Documentation

4.54.3.1 static INLINE int _hex_decode_digit (char *c*) `[static]`

Helper: given a hex digit, return its value, or -1 if it isn't hex.

Referenced by base16_decode(), and hex_decode_digit().

4.54.3.2 void _tor_free (void ∗ *mem*)

Helper for places that need to take a function pointer to the right spelling of "free()".

References tor_free.

Referenced by dir_split_resource_into_fingerprint_pairs(), dirserv_free_fingerprint_list(), dns_reset_correctness_checks(), download_status_map_update_from_v2_networkstatus(), entry_guards_parse_state(), networkstatus_free_all(), ns_detached_signatures_free(), rend_service_free(), routerstatus_list_update_named_server_map(), smartlist_uniq_digests(), smartlist_uniq_digests256(), smartlist_uniq_strings(), sort_version_list(), and tor_main().

4.54.3.3 void∗ _tor_malloc (size_t size *DMALLOC_PARAMS*)

Allocate a chunk of **size** bytes of memory, and return a pointer to result. On error, log and terminate the process. (Same as malloc(size), but never returns NULL.)

file and **line** are used if dmalloc is enabled, and ignored otherwise.

References LD_MM.

Referenced by _tor_malloc_roundup(), _tor_malloc_zero(), _tor_memdup(), _tor_strndup(), and tor_main().

4.54.3.4 void∗ _tor_malloc_roundup (size_t ∗sizep *DMALLOC_PARAMS*)

Allocate and return a chunk of memory of size at least ∗**size**, using the same resources we would use to malloc ∗**sizep**. Set ∗**sizep** to the number of usable bytes in the chunk of memory.

References _tor_malloc().

4.54.3.5 void∗ _tor_malloc_zero (size_t size *DMALLOC_PARAMS*)

Allocate a chunk of **size** bytes of memory, fill the memory with zero bytes, and return a pointer to the result. Log and terminate the process on error. (Same as calloc(size,1), but never returns NULL.)

References _tor_malloc().

4.54.3.6 void∗ _tor_memdup (const void ∗ *mem*, size_t len *DMALLOC_PARAMS*)

Allocate a chunk of **len** bytes, with the same contents as the **len** bytes starting at **mem**.

References _tor_malloc(), and tor_assert.

4.54.3.7 void∗ _tor_realloc (void ∗ *ptr*, size_t size *DMALLOC_PARAMS*)

Change the size of the memory block pointed to by **ptr** to **size** bytes long; return the new memory block. On error, log and terminate. (Like realloc(ptr,size), but never returns NULL.)

References LD_MM.

Referenced by tor_main().

4.54.3.8 char∗ _tor_strdup (const char ∗s *DMALLOC_PARAMS*)

Return a newly allocated copy of the NUL-terminated string s. On error, log and terminate. (Like strdup(s), but never returns NULL.)

References LD_MM, and tor_assert.

4.54.3.9 char∗ _tor_strndup (const char ∗ *s*, size_t n *DMALLOC_PARAMS*)

Allocate and return a new string containing the first **n** characters of **s**. If **s** is longer than **n** characters, only the first **n** are copied. The result is always NUL-terminated. (Like strndup(s,n), but never returns NULL.)

References _tor_malloc(), and tor_assert.

4.54.3.10 int abort_writing_to_file (open_file_t ∗ *file_data*)

Finish writing to **file_data**: close the file handle, free memory as needed, and if using a temporary file, delete it.

References finish_writing_to_file_impl().

Referenced by geoip_dirreq_stats_write(), geoip_entry_stats_write(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_record_mtbf_data(), start_writing_to_stdio_file(), and write_chunks_to_file_impl().

4.54.3.11 int append_bytes_to_file (const char ∗ *fname*, const char ∗ *str*, size_t *len*, int *bin*)

As write_bytes_to_file, but if the file already exists, append the bytes to the end of the file instead of overwriting it.

References smartlist_add(), smartlist_create(), smartlist_free(), and write_chunks_to_file_impl().

Referenced by signed_desc_append_to_journal().

4.54.3.12 time_t approx_time (void)

Return a cached estimate of the current time from when **update_approx_time()** (p. 1258) was last called. This is a hack to avoid calling time(NULL) on critical paths: please do not even think of calling it anywhere else.

References cached_approx_time.

Referenced by append_cell_to_circuit_queue(), cell_ewma_get_tick(), circuit_build_times_needs_circuits_now(), circuit_build_times_network_check_live(), circuit_build_times_network_is_live(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_handle_read_impl(), connection_handle_write_impl(), connection_or_flushed_some(), connection_or_get_for_extend(), connection_or_write_cell_to_buf(), connection_or_write_var_cell_to_buf(), connection_read_to_buf(), origin_circuit_new(), and relay_send_command_from_edge().

4.54.3.13 int base16_decode (char * *dest*, size_t *destlen*, const char * *src*, size_t *srclen*)

Given a hexadecimal string of **srclen** bytes in **src**, decode it and store the result in the **destlen**-byte buffer at **dest**. Return 0 on success, -1 on failure.

References _hex_decode_digit().

Referenced by add_fingerprint_to_dir(), authority_cert_parse_from_string(), circuit_get_open_circ_or_launch(), client_likes_consensus(), decode_hashed_passwords(), dir_networkstatus_download_failed(), dir_routerdesc_download_failed(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), dirserv_load_fingerprint_file(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), handle_control_authenticate(), hex_digest_matches(), hexdigest_to_digest(), measured_bw_line_parse(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_bridge_line(), parse_dir_server_line(), rep_hist_load_mtbf_data(), router_fingerprint_is_me(), router_get_by_nickname(), router_get_consensus_status_by_nickname(), router_parse_entry_from_string(), router_reload_v2_networkstatus(), routerset_parse(), and tor_version_parse().

4.54.3.14 void base16_encode (char * *dest*, size_t *destlen*, const char * *src*, size_t *srclen*)

Encode the **srclen** bytes at **src** in a NUL-terminated, uppercase hexadecimal string; store it in the **destlen**-byte buffer **dest**.

References tor_assert.

Referenced by circuit_list_path_impl(), connection_ap_make_link(), connection_dir_client_reached_eof(), connection_or_check_valid_tls_handshake(), connection_or_init_conn_from_address(), control_event_guard(), crypto_pk_get_fingerprint(), directory_get_consensus_url(), dirserv_get_name_status(), dirserv_get_networkstatus_v2(), dirvote_fetch_missing_votes(), do_hash_password(), entry_guard_set_status(), entry_guards_update_state(), extrainfo_dump_to_string(), fetch_bridge_descriptors(), format_networkstatus_vote(), getinfo_helper_entry_guards(), getinfo_helper_misc(), hex_str(), initiate_descriptor_downloads(), list_single_server_status(), networkstatus_add_detached_signatures(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_cache_filename(), networkstatus_get_detached_signatures(), orconn_target_get_name(), remove_dead_entry_guards(), remove_obsolete_entry_guards(), rend_-

mid_establish_rendezvous(), rend_mid_rendezvous(), rend_parse_introduction_points(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_rendezvous_has_opened(), rep_hist_dump_stats(), rep_hist_record_mtbf_data(), router_dump_router_to_string(), router_get_verbose_nickname(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_rebuild_descriptor(), router_set_networkstatus_v2(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), and update_v2_networkstatus_cache_downloads().

4.54.3.15 int check_private_dir (const char ∗ *dirname*, cpd_check_t *check*)

Check whether dirname exists and is private. If yes return 0. If it does not exist, and check==CPD_CREATE is set, try to create it and return 0 on success. If it does not exist, and check==CPD_CHECK, and we think we can create it, return 0. Else return -1.

References clean_name_for_stat(), LD_FS, LD_GENERAL, tor_assert, and tor_free.

Referenced by geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), init_keys(), load_bridge_stats(), options_act_reversible(), rend_service_load_keys(), rep_hist_buffer_stats_write(), and rep_hist_exit_stats_write().

4.54.3.16 static void clean_name_for_stat (char ∗ *name*) `[static]`

Clean up **name** so that we can use it in a call to "stat". On Unix, we do nothing. On Windows, we remove a trailing slash, unless the path is the root of a disk.

Referenced by check_private_dir(), and file_status().

4.54.3.17 static int digit_to_num (char *d*) `[static]`

Helper: given an ASCII-encoded decimal digit, return its numeric value. NOTE: requires that its input be in-bounds.

References tor_assert.

Referenced by scan_unsigned(), and tor_vsscanf().

4.54.3.18 const char∗ eat_whitespace (const char ∗ *s*)

Return a pointer to the first char of s that is not whitespace and not a comment, or to the terminating NUL if no such character exists.

References tor_assert.

Referenced by authority_cert_parse_from_string(), config_parse_units(), get_token_arguments(), parse_log_severity_config(), rend_parse_client_keys(), router_parse_addr_policy_item_from_string(), router_parse_addr_policy_private(), tor_addr_port_parse(), tor_version_as_new_as(), and tor_version_parse().

4.54.3.19 const char∗ eat_whitespace_eos (const char ∗ *s*, const char ∗ *eos*)

Return a pointer to the first char of s that is not whitespace and not a comment, or to the terminating NUL if no such character exists.

References tor_assert.

Referenced by find_start_of_next_microdesc(), find_start_of_next_router_or_extrainfo(), get_next_token(), microdescs_parse_from_string(), and tokenize_string().

4.54.3.20 const char∗ eat_whitespace_eos_no_nl (const char ∗ *s*, const char ∗ *eos*)

As eat_whitespace_no_nl, but stop at **eos** whether we have found a non-whitespace character or not.

Referenced by get_next_token().

4.54.3.21 const char∗ eat_whitespace_no_nl (const char ∗ *s*)

Return a pointer to the first char of s that is not a space or a tab or a \r, or to the terminating NUL if no such character exists.

Referenced by parse_bridge_stats_controller(), and parse_http_url().

4.54.3.22 char∗ esc_for_log (const char ∗ *s*)

Allocate and return a new string representing the contents of **s**, surrounded by quotes and using standard C escapes.

Generally, we use this for logging values that come in over the network to keep them from tricking users, and for sending certain values to the controller.

We trust values from the resolver, OS, configuration file, and command line to not be maliciously ill-formed. We validate incoming routerdescs and SOCKS requests and addresses from BEGIN cells as they're parsed; afterwards, we trust them as non-malicious.

References tor_snprintf().

Referenced by control_event_logmsg(), dirserv_get_status_impl(), esc_router_info(), escaped(), evdns_callback(), get_assigned_option(), handle_control_protocolinfo(), parse_addr_port(), parse_iso_time(), and parse_rfc1123_time().

4.54.3.23 const char∗ escaped (const char ∗ *s*)

Allocate and return a new string representing the contents of **s**, surrounded by quotes and using standard C escapes.

THIS FUNCTION IS NOT REENTRANT. Don't call it from outside the main thread. Also, each call invalidates the last-returned value, so don't try log_warn(LD_GENERAL, "%s %s", escaped(a), escaped(b));

References esc_for_log(), and tor_free.

Referenced by add_fingerprint_to_dir(), authority_cert_parse_from_string(), circuit_build_failed(), command_process_create_cell(), connection_ap_process_natd(), connection_dir_client_reached_eof(), connection_read_https_proxy_response(), connection_read_proxy_handshake(), dir_networkstatus_download_failed(), dir_routerdesc_download_failed(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_handle_command(), dirserv_add_multiple_descriptors(), dirserv_get_status_impl(), dirserv_read_measured_bandwidths(), entry_guards_parse_state(), escaped_safe_str(), escaped_safe_str_client(), evdns_log_cb(), extrainfo_parse_entry_from_string(), fetch_from_buf_socks(), geoip_parse_entry(), handle_control_protocolinfo(), http_set_address_origin(), init_cookie_authentication(), load_downloaded_routers(), measured_bw_line_parse(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_addr_and_port_range(), parse_addr_port(), parse_http_response(), parse_port_config(), parse_port_range(), remove_obsolete_entry_guards(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rend_service_introduce(), rend_service_load_keys(), rep_hist_load_mtbf_data(), router_dump_router_to_string(), router_get_consensus_status_by_nickname(), router_new_address_suggestion(), router_parse_addr_policy(), router_parse_addr_policy_item_from_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerstatus_parse_entry_from_string(), signed_descriptor_get_body_impl(), tor_addr_parse_mask_ports(), and tor_free_all().

4.54.3.24 char∗ expand_filename (const char ∗ *filename*)

Expand any homedir prefix on **filename**; return a newly allocated string.

References get_user_homedir(), LD_CONFIG, strcmpend(), tor_assert, tor_free, and tor_snprintf().

Referenced by find_torrc_filename(), and normalize_data_directory().

4.54.3.25 FILE∗ fdopen_file (open_file_t ∗ *file_ data*)

Given **file_data** from **start_writing_to_file()** (p. 1252), return a stdio FILE∗ that can be used to write to the same file. The caller should not mix stdio calls with non-stdio calls.

References open_file_t::binary, open_file_t::fd, open_file_t::filename, LD_FS, open_file_t::stdio_file, and tor_assert.

Referenced by start_writing_to_stdio_file().

4.54.3.26 file_status_t file_status (const char ∗ *fname*)

Return FN_ERROR if filename can't be read, FN_NOENT if it doesn't exist, FN_FILE if it is a regular file, or FN_DIR if it's a directory. On FN_ERROR, sets errno.

References clean_name_for_stat(), and tor_free.

Referenced by dirserv_clear_old_networkstatuses(), find_torrc_filename(), init_key_from_file(), init_keys(), load_stats_file(), load_torrc_from_disk(), networkstatus_v2_list_clean(), or_state_load(), replace_file(), rotate_onion_key(), router_reload_router_list_impl(), and write_configuration_file().

4.54.3.27 const char∗ find_str_at_start_of_line (const char ∗ *haystack*, const char ∗ *needle*)

Return the first occurrence of **needle** in **haystack** that occurs at the start of a line (that is, at the beginning of **haystack** or immediately after a newline). Return NULL if no such string is found.

Referenced by parse_bridge_stats_controller().

4.54.3.28 const char* find_whitespace (const char * *s*)

Return a pointer to the first char of s that is whitespace or #, or to the terminating NUL if no such character exists.

Referenced by get_token_arguments(), parse_http_url(), tor_version_as_new_as(), tor_version_parse(), and version_from_platform().

4.54.3.29 const char* find_whitespace_eos (const char * *s*, const char * *eos*)

As find_whitespace, but stop at **eos** whether we have found a whitespace or not.

Referenced by get_next_token().

4.54.3.30 void finish_daemon (const char * *desired_cwd*)

Finish putting the process into daemon mode: drop standard fds, and tell the parent process to exit. (Note: it's safe to call this more than once: calls after the first are ignored. Calls start_daemon first if it hasn't been called already.)

References daemon_filedes, finish_daemon_called, LD_GENERAL, start_daemon(), and start_daemon_called.

Referenced by options_act().

4.54.3.31 int finish_writing_to_file (open_file_t * *file_data*)

Finish writing to **file_data**: close the file handle, free memory as needed, and if using a temporary file, replace the original file with the temporary file.

References finish_writing_to_file_impl().

Referenced by geoip_dirreq_stats_write(), geoip_entry_stats_write(), microdesc_cache_rebuild(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_record_mtbf_data(), and write_chunks_to_file_impl().

4.54.3.32 static int finish_writing_to_file_impl (open_file_t * *file_data*, int *abort_write*) `[static]`

Helper function: close and free the underlying file and memory in **file_data**. If we were writing into a temporary file, then delete that file (if abort_write is true) or replaces the target file with the temporary file (if abort_write is false).

References open_file_t::fd, open_file_t::filename, LD_FS, open_file_t::rename_on_close, replace_file(), open_file_t::stdio_file, open_file_t::tempname, tor_assert, and tor_free.

Referenced by abort_writing_to_file(), and finish_writing_to_file().

4.54.3.33 void format_iso_time (char * *buf*, time_t *t*)

Set **buf** to the ISO8601 encoding of the GMT value of **t**. The buffer must be at least ISO_TIME_LEN+1 bytes long.

Referenced by accounting_set_wakeup_time(), addressmap_get_mappings(), control_event_address_mapped(), dirserv_add_multiple_descriptors(), dirserv_dump_directory_-

to_string(), dirvote_add_vote(), dirvote_recalculate_timing(), entry_guards_update_state(), extrainfo_dump_to_string(), format_networkstatus_vote(), generate_runningrouters(), generate_v2_networkstatus_opinion(), geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), get_assigned_option(), getinfo_helper_accounting(), getinfo_helper_entry_guards(), hibernate_end_time_elapsed(), load_downloaded_routers(), networkstatus_compute_consensus(), networkstatus_get_detached_signatures(), networkstatus_set_current_consensus(), parse_bridge_stats_controller(), read_bandwidth_usage(), rend_encode_v2_descriptors(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_format_router_status(), rep_hist_get_bandwidth_lines(), rep_hist_get_router_stability_doc(), rep_hist_record_mtbf_data(), router_dump_router_to_string(), router_set_networkstatus_v2(), and routerstatus_format_entry().

4.54.3.34 void format_local_iso_time (char ∗ *buf*, time_t *t*)

Set **buf** to the ISO8601 encoding of the local value of **t**. The buffer must be at least ISO_TIME_LEN+1 bytes long.

(ISO8601 format is 2006-10-29 10:57:20)

Referenced by accounting_set_wakeup_time(), consider_hibernation(), control_event_address_mapped(), or_state_save(), remove_dead_entry_guards(), remove_file_if_very_old(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), and update_consensus_networkstatus_fetch_time().

4.54.3.35 void format_rfc1123_time (char ∗ *buf*, time_t *t*)

Set **buf** to the RFC1123 encoding of the GMT value of **t**. The buffer must be at least RFC1123_TIME_LEN+1 bytes long.

(RFC1123 format is Fri, 29 Sep 2006 15:54:20 GMT)

References MONTH_NAMES, tor_assert, and WEEKDAY_NAMES.

Referenced by directory_send_command(), and write_http_response_header_impl().

4.54.3.36 int format_time_interval (char ∗ *out*, size_t *out_len*, long *interval*)

Given an **interval** in seconds, try to write it to the **out_len**-byte buffer in **out** in a human-readable form. Return 0 on success, -1 on failure.

References tor_snprintf().

Referenced by command_process_netinfo_cell(), connection_dir_client_reached_eof(), networkstatus_set_current_consensus(), and router_set_networkstatus_v2().

4.54.3.37 int ftime_definitely_after (time_t *now*, time_t *when*)

Return true if we think that **now** is definitely after **when**.

References ftime_skew, and ftime_slop.

Referenced by authority_certs_fetch_missing(), and trusted_dirs_remove_old_certs().

4.54.3.38 int ftime_definitely_before (time_t *now*, time_t *when*)

Return true if we think that **now** is definitely before **when**.

References ftime_skew, and ftime_slop.

Referenced by networkstatus_set_current_consensus().

4.54.3.39 int ftime_maybe_after (time_t *now*, time_t *when*)

Return true iff we think that **now** might be after **when**.

References ftime_skew, and ftime_slop.

4.54.3.40 int ftime_maybe_before (time_t *now*, time_t *when*)

Return true iff we think that **now** might be before **when**.

References ftime_skew, and ftime_slop.

4.54.3.41 void ftime_set_estimated_skew (int *seconds*)

Set the amount by which we believe our system clock to differ from real time.

References ftime_skew.

4.54.3.42 void ftime_set_maximum_sloppiness (int *seconds*)

Set the largest amount of sloppiness we'll allow in fuzzy time comparisons.

References ftime_slop, and tor_assert.

4.54.3.43 int hex_decode_digit (char *c*)

Helper: given a hex digit, return its value, or -1 if it isn't hex.

References _hex_decode_digit().

Referenced by tor_addr_parse_reverse_lookup_name(), and unescape_string().

4.54.3.44 const char* hex_str (const char * *from*, size_t *fromlen*)

Return a pointer to a NUL-terminated hexadecimal string encoding the first **fromlen** bytes of **from**. (fromlen must be <= 32.) The result does not need to be deallocated, but repeated calls to hex_str will trash old results.

References base16_encode().

Referenced by authority_certs_fetch_missing(), dirvote_act(), dirvote_add_vote(), handle_control_attachstream(), list_v3_auth_ids(), networkstatus_check_consensus_signature(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), router_add_to_routerlist(), router_load_routers_from_string(), router_parse_directory(), and trusted_dirs_load_certs_from_string().

4.54.3.45 int memcmpstart (const void ∗ *mem*, size_t *memlen*, const char ∗ *prefix*)

Compare the value of the string **prefix** with the start of the **memlen**-byte memory chunk at **mem**. Return as for strcmp.

[As memcmp(mem, prefix, strlen(prefix)) but returns -1 if memlen is less than strlen(prefix).]

Referenced by rend_decrypt_introduction_points(), and rend_parse_introduction_points().

4.54.3.46 static int n_leapdays (int *y1*, int *y2*) [static]

Helper: Return the number of leap-days between Jan 1, y1 and Jan 1, y2.

Referenced by tor_timegm().

4.54.3.47 const char∗ parse_config_line_from_str (const char ∗ *line*, char ∗∗ *key_out*, char ∗∗ *value_out*)

Given a string containing part of a configuration file or similar format, advance past comments and whitespace and try to parse a single line. If we parse a line successfully, set ∗**key_out** to a new string holding the key portion and ∗**value_out** to a new string holding the value portion of the line, and return a pointer to the start of the next line. If we run out of data, return a pointer to the end of the string. If we encounter an error, return NULL.

References tor_assert, and unescape_string().

Referenced by config_get_lines().

4.54.3.48 int parse_http_time (const char ∗ *date*, struct tm ∗ *tm*)

Given a **date** in one of the three formats allowed by HTTP (ugh), parse it into **tm**. Return 0 on success, negative on failure.

References MONTH_NAMES, tor_assert, and tor_sscanf().

Referenced by directory_handle_command_get().

4.54.3.49 int parse_iso_time (const char ∗ *cp*, time_t ∗ *t*)

Given an ISO-formatted UTC time value (after the epoch) in **cp**, parse it and store its value in ∗**t**. Return 0 on success, -1 on failure. Ignore extraneous stuff in **cp** separated by whitespace from the end of the time string.

References esc_for_log(), LD_GENERAL, tor_free, tor_sscanf(), and tor_timegm().

Referenced by authority_cert_parse_from_string(), config_assign_value(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), load_stats_file(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_bridge_stats_controller(), parse_possibly_bad_iso_time(), rend_parse_v2_service_descriptor(), rep_hist_load_mtbf_data(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

4.54.3.50 int parse_rfc1123_time (const char * *buf*, time_t * *t*)

Parse the RFC1123 encoding of some time (in GMT) from **buf**, and store the result in ***t**.

Return 0 on success, -1 on failure.

References esc_for_log(), LD_GENERAL, MONTH_NAMES, tor_free, tor_sscanf(), and tor_timegm().

Referenced by parse_http_response().

4.54.3.51 int path_is_relative (const char * *filename*)

Return true iff **filename** is a relative path.

Referenced by options_validate().

4.54.3.52 ssize_t read_all (int *fd*, char * *buf*, size_t *count*, int *isSocket*)

Read from **fd** to **buf**, until we get **count** bytes or reach the end of the file. **isSocket** must be 1 if fd was returned by socket() or accept(), and 0 if fd was returned by open(). Return the number of bytes read, or -1 on error. Only use if fd is a blocking fd.

Referenced by cpuworker_main(), crypto_seed_rng(), and read_file_to_str().

4.54.3.53 char* read_file_to_str (const char * *filename*, int *flags*, struct stat * *stat_out*)

Read the contents of **filename** into a newly allocated string; return the string on success or NULL on failure.

If **stat_out** is provided, store the result of stat()ing the file into **stat_out**.

If **flags** & RFTS_BIN, open the file in binary mode. If **flags** & RFTS_IGNORE_MISSING, don't warn if the file doesn't exist.

References LD_FS, read_all(), RFTS_BIN, RFTS_IGNORE_MISSING, tor_assert, tor_free, and tor_strstrip().

Referenced by crypto_pk_read_private_key_from_filename(), dirserv_load_fingerprint_file(), init_keys(), load_authority_keyset(), load_bridge_stats(), load_stats_file(), load_torrc_from_disk(), microdesc_cache_reload(), options_act(), or_state_load(), rend_service_load_keys(), rep_hist_load_mtbf_data(), router_reload_consensus_networkstatus(), router_reload_router_list_impl(), router_reload_v2_networkstatus(), trusted_dirs_reload_certs(), and write_configuration_file().

4.54.3.54 unsigned round_to_next_multiple_of (unsigned *number*, unsigned *divisor*)

Return the lowest x such that x is at least **number**, and x modulo **divisor** == 0.

Referenced by geoip_get_client_history(), and geoip_get_request_history().

4.54.3.55 uint64_t round_to_power_of_2 (uint64_t *u64*)

Return the power of 2 closest to **u64**.

References tor_log2().

Referenced by mp_pool_new().

4.54.3.56 uint32_t round_uint32_to_next_multiple_of (uint32_t *number*, uint32_t *divisor*)

Return the lowest x such that x is at least **number**, and x modulo **divisor** == 0.

Referenced by geoip_dirreq_stats_write(), geoip_get_dirreq_history(), and rep_hist_exit_stats_write().

4.54.3.57 uint64_t round_uint64_to_next_multiple_of (uint64_t *number*, uint64_t *divisor*)

Return the lowest x such that x is at least **number**, and x modulo **divisor** == 0.

Referenced by rep_hist_exit_stats_write().

4.54.3.58 static int scan_string (const char ** *bufp*, char * *out*, int *width*) `[static]`

Helper: copy up to **width** non-space characters from **bufp** to **out**. Make sure **out** is nul-terminated. Advance **bufp** to the next non-space character or the EOS.

Referenced by tor_vsscanf().

4.54.3.59 static int scan_unsigned (const char ** *bufp*, unsigned * *out*, int *width*) `[static]`

Helper: Read an unsigned int from ***bufp** of up to **width** characters. (Handle arbitrary width if **width** is less than 0.) On success, store the result in **out**, advance bufp to the next character, and return 0. On failure, return -1.

References digit_to_num().

Referenced by tor_vsscanf().

4.54.3.60 void start_daemon (void)

Start putting the process into daemon mode: fork and drop all resources except standard fds. The parent process never returns, but stays around until finish_daemon is called. (Note: it's safe to call this more than once: calls after the first are ignored.)

References daemon_filedes, LD_GENERAL, set_main_thread(), and start_daemon_called.

Referenced by finish_daemon(), and options_act_reversible().

4.54.3.61 int start_writing_to_file (const char * *fname*, int *open_flags*, int *mode*, open_file_t ** *data_out*)

Try to start writing to the file in **fname**, passing the flags **open_flags** to the open() syscall, creating the file (if needed) with access value **mode**. If the O_APPEND flag is set, we append to the original file. Otherwise, we open a new temporary file in the same directory, and either replace the original or remove the temporary file when we're done.

Return the fd for the newly opened file, and store working data in ∗**data_out**. The caller should not close the fd manually: instead, call **finish_writing_to_file()** (p. 1247) or **abort_writing_to_file()** (p. 1242). Returns -1 on failure.

NOTE: When not appending, the flags O_CREAT and O_TRUNC are treated as true and the flag O_EXCL is treated as false.

NOTE: Ordinarily, O_APPEND means "seek to the end of the file before each write()". We don't do that.

References open_file_t::binary, open_file_t::fd, open_file_t::filename, LD_FS, LD_GENERAL, open_file_t::rename_on_close, open_file_t::tempname, tor_assert, tor_fd_seekend(), tor_free, and tor_snprintf().

Referenced by start_writing_to_stdio_file(), and write_chunks_to_file_impl().

4.54.3.62 FILE∗ start_writing_to_stdio_file (const char ∗ *fname*, int *open_flags*, int *mode*, open_file_t ∗∗ *data_out*)

Combines start_writing_to_file with **fdopen_file()** (p. 1246): arguments are as for start_writing_to_file, but

References abort_writing_to_file(), fdopen_file(), and start_writing_to_file().

Referenced by geoip_dirreq_stats_write(), geoip_entry_stats_write(), microdesc_cache_rebuild(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), and rep_hist_record_mtbf_data().

4.54.3.63 int strcasecmpend (const char ∗ *s1*, const char ∗ *s2*)

Compares the last strlen(s2) characters of s1 with s2. Returns as for strcasecmp.

Referenced by address_is_in_virtual_range(), connection_ap_handshake_rewrite_and_attach(), consider_recording_trackhost(), and tor_addr_parse_reverse_lookup_name().

4.54.3.64 int strcasecmpstart (const char ∗ *s1*, const char ∗ *s2*)

Compares the first strlen(s2) characters of s1 with s2. Returns as for strcasecmp.

Referenced by circuit_purpose_from_string(), find_element_starting_with(), handle_control_postdescriptor(), http_get_header(), parse_dir_server_line(), parse_log_severity_config(), and tor_version_parse().

4.54.3.65 int strcmp_len (const char ∗ *s1*, const char ∗ *s2*, size_t *s1_len*)

Compare the s1_len-byte string **s1** with **s2**, without depending on a terminating nul in s1. Sorting order is first by length, then lexically; return values are as for strcmp.

Referenced by get_next_token().

4.54.3.66 int strcmpend (const char ∗ *s1*, const char ∗ *s2*)

Compares the last strlen(s2) characters of s1 with s2. Returns as for strcmp.

Referenced by clear_trackexithost_mappings(), expand_filename(), and rend_parse_service_-authorization().

4.54.3.67 int strcmpstart (const char * *s1*, const char * *s2*)

Compares the first strlen(s2) characters of s1 with s2. Returns as for strcmp.

Referenced by body_is_plausible(), config_dump(), connection_ap_process_natd(), connection_dir_client_reached_eof(), connection_dir_download_networkstatus_failed(), decode_hashed_passwords(), directory_conn_is_self_reachability_test(), directory_handle_-command_get(), directory_handle_command_post(), dirserv_get_networkstatus_v2_-fingerprints(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirvote_-add_vote(), evdns_log_cb(), find_next_with(), find_start_of_next_microdesc(), find_-start_of_next_router_or_extrainfo(), get_next_token(), getinfo_helper_dir(), getinfo_-helper_events(), getinfo_helper_geoip(), handle_getinfo_helper(), is_gzip_supported(), list_pending_downloads(), measured_bw_line_parse(), networkstatus_parse_vote_from_-string(), networkstatus_v2_parse_from_string(), options_validate(), parse_http_response(), parse_http_url(), rend_cache_store_v2_desc_as_dir(), rend_parse_client_keys(), rep_-hist_load_mtbf_data(), router_parse_addr_policy(), router_parse_addr_policy_private(), router_parse_entry_from_string(), routerstatus_parse_entry_from_string(), send_control_-event_string(), tor_version_as_new_as(), tor_version_is_obsolete(), tor_version_parse(), version_from_platform(), and write_configuration_file().

4.54.3.68 int tor_digest256_is_zero (const char * *digest*)

Return true iff the DIGEST256_LEN bytes in digest are all zero.

References DIGEST256_LEN, and tor_mem_is_zero().

4.54.3.69 int tor_digest_is_zero (const char * *digest*)

Return true iff the DIGEST_LEN bytes in digest are all zero.

References DIGEST_LEN, and tor_mem_is_zero().

Referenced by _connection_free(), circuit_extend(), circuit_get_all_pending_on_or_-conn(), circuit_is_acceptable(), circuit_n_conn_done(), connection_about_to_close_-connection(), connection_ap_fail_onehop(), connection_free(), connection_or_check_-valid_tls_handshake(), connection_or_remove_from_identity_map(), connection_or_set_-identity_digest(), connection_unlink(), fetch_bridge_descriptors(), format_networkstatus_-vote(), handle_control_attachstream(), list_v3_auth_ids(), lookup_cached_dir_by_fp(), networkstatus_compute_consensus(), networkstatus_parse_vote_from_string(), orconn_-target_get_name(), routerinfo_get_configured_bridge(), routerlist_insert(), routerlist_-insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_replace(), and update_-extrainfo_downloads().

4.54.3.70 smartlist_t* tor_listdir (const char * *dirname*) `[read]`

Return a new list containing the filenames in the directory **dirname**. Return NULL on error or if **dirname** is not a directory.

References LD_FS, smartlist_add(), smartlist_create(), tor_free, and tor_snprintf().

Referenced by router_reload_v2_networkstatus().

4.54.3.71 int tor_log2 (uint64_t *u64*)

Returns floor(log2(u64)). If u64 is 0, (incorrectly) returns 0.

Referenced by digestset_new(), and round_to_power_of_2().

4.54.3.72 void tor_log_mallinfo (int *severity*)

Call the platform malloc info function, and dump the results to the log at level **severity**. If no such function exists, do nothing.

References LD_MM, and tor_log().

Referenced by dumpmemusage().

4.54.3.73 long tor_lround (double *d*)

Return the long integer closest to d. We define this wrapper here so that not all users of math.h need to use the right incancations to get the c99 functions.

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_generate_sample(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), and circuit_build_times_set_timeout().

4.54.3.74 double tor_mathlog (double *d*)

Returns the natural logarithm of d base 2. We define this wrapper here so as to make it easier not to conflict with Tor's log() macro.

Referenced by circuit_build_times_initial_alpha(), and circuit_build_times_update_alpha().

4.54.3.75 int tor_mem_is_zero (const char * *mem*, size_t *len*)

Return true iff the 'len' bytes at 'mem' are all zero.

Referenced by check_sockaddr(), get_link_history(), get_or_history(), networkstatus_add_detached_signatures(), networkstatus_get_detached_signatures(), networkstatus_parse_detached_signatures(), rend_cache_store_v2_desc_as_client(), tor_digest256_is_zero(), and tor_digest_is_zero().

4.54.3.76 double tor_parse_double (const char * *s*, double *min*, double *max*, int * *ok*, char ** *next*)

As **tor_parse_long()** (p. 1255), but return a double.

Referenced by config_parse_units().

4.54.3.77 long tor_parse_long (const char * *s*, int *base*, long *min*, long *max*, int * *ok*, char ** *next*)

Extract a long from the start of s, in the given numeric base. If there is unconverted data and next is provided, set *next to the first unconverted character. An error has occurred if no characters are converted; or if there are unconverted characters and next is NULL; or if the parsed value is not between min and max. When no error occurs, return the parsed value and set *ok (if provided) to 1. When an error occurs, return 0 and set *ok (if provided) to 0.

Referenced by _cmp_int_strings(), accounting_parse_options(), compute_consensus_method(), config_assign_value(), connection_ap_process_natd(), dirvote_compute_params(), handle_control_protocolinfo(), measured_bw_line_parse(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_addr_port(), parse_dir_server_line(), parse_port_config(), parse_port_range(), parse_possibly_bad_iso_time(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rep_hist_load_mtbf_data(), router_parse_entry_from_string(), routerstatus_parse_entry_from_string(), and tor_addr_port_parse().

4.54.3.78 uint64_t tor_parse_uint64 (const char * *s*, int *base*, uint64_t *min*, uint64_t *max*, int * *ok*, char ** *next*)

As tor_parse_long, but return a uint64_t. Only base 10 is guaranteed to work for now.

References tor_assert.

Referenced by config_parse_units(), get_stream(), and rep_hist_load_state().

4.54.3.79 unsigned long tor_parse_ulong (const char * *s*, int *base*, unsigned long *min*, unsigned long *max*, int * *ok*, char ** *next*)

As **tor_parse_long()** (p. 1255), but return an unsigned long.

Referenced by circuit_build_times_parse_state(), dirserv_read_measured_bandwidths(), get_circ(), handle_control_attachstream(), handle_control_closestream(), handle_control_redirectstream(), and routerstatus_parse_entry_from_string().

4.54.3.80 int tor_sscanf (const char * *buf*, const char * *pattern*, ...)

Minimal sscanf replacement: parse **buf** according to **pattern** and store the results in the corresponding argument fields. Differs from sscanf in that it: Only handles u and Ns. Does not handle arbitrarily long widths. u does not consume any space. Is locale-independent. Returns -1 on malformed patterns.

(As with other locale-independent functions, we need this to parse data that is in ASCII without worrying that the C library's locale-handling will make miscellaneous characters look like numbers, spaces, and so on.)

References tor_vsscanf().

Referenced by parse_http_response(), parse_http_time(), parse_iso_time(), parse_rfc1123_time(), tor_inet_aton(), and tor_inet_pton().

4.54.3.81 int tor_strisnonupper (const char * *s*)

Return 1 if no character in **s** is uppercase, else return 0.

Referenced by assert_resolve_ok().

4.54.3.82 int tor_strisprint (const char * *s*)

Return 1 if every character in **s** is printable, else return 0.

Referenced by fetch_from_buf_socks().

4.54.3.83 void tor_strlower (char * *s*)

Convert all alphabetic characters in the nul-terminated string **s** to lowercase.

Referenced by clear_trackexithost_mappings(), connection_ap_handshake_rewrite_and_attach(), connection_exit_begin_conn(), dns_resolve_impl(), geoip_add_entry(), get_default_nickname(), handle_control_mapaddress(), router_parse_addr_policy_item_from_string(), routerset_get_countryname(), strmap_get_lc(), strmap_remove_lc(), and strmap_set_lc().

4.54.3.84 void tor_strstrip (char * *s*, const char * *strip*)

Remove from the string **s** every character which appears in **strip**.

Referenced by add_fingerprint_to_dir(), dirserv_load_fingerprint_file(), read_file_to_str(), router_parse_entry_from_string(), and smartlist_split_string().

4.54.3.85 void tor_strupper (char * *s*)

Convert all alphabetic characters in the nul-terminated string **s** to lowercase.

4.54.3.86 time_t tor_timegm (struct tm * *tm*)

Return a time_t given a struct tm. The result is given in GMT, and does not account for leap seconds.

References days_per_month, IS_LEAPYEAR, LD_BUG, n_leapdays(), and tor_assert.

Referenced by directory_handle_command_get(), dirvote_get_start_of_next_interval(), parse_iso_time(), and parse_rfc1123_time().

4.54.3.87 int tor_vsscanf (const char * *buf*, const char * *pattern*, va_list *ap*)

Locale-independent, minimal, no-surprises scanf variant, accepting only a restricted pattern format. For more info on what it supports, see **tor_sscanf()** (p. 1256) documentation.

References digit_to_num(), scan_string(), and scan_unsigned().

Referenced by tor_sscanf().

4.54.3.88 long tv_mdiff (const struct timeval * *start*, const struct timeval * *end*)

Return the number of milliseconds elapsed between *start and *end.

References LD_GENERAL.

Referenced by circuit_send_next_onion_skin(), and geoip_get_dirreq_history().

4.54.3.89 double tv_to_double (const struct timeval * *tv*)

Converts struct timeval to a double value. Preserves microsecond precision, but just barely. Error is approx +/- 0.1 usec when dealing with epoch values.

4.54.3.90 int64_t tv_to_msec (const struct timeval * *tv*)

Converts timeval to milliseconds.

4.54.3.91 int64_t tv_to_usec (const struct timeval * *tv*)

Converts timeval to microseconds.

4.54.3.92 long tv_udiff (const struct timeval * *start*, const struct timeval * *end*)

Return the number of microseconds elapsed between *start and *end.

References LD_GENERAL.

4.54.3.93 static const char* unescape_string (const char * *s*, char ** *result*, size_t * *size_out*) [static]

Given a c-style double-quoted escaped string in **s**, extract and decode its contents into a newly allocated string. On success, assign this string to ***result**, assign its length to **size_out** (if provided), and return a pointer to the position in **s** immediately after the string. On failure, return NULL.

References hex_decode_digit(), tor_fragile_assert, and tor_free.

Referenced by parse_config_line_from_str().

4.54.3.94 void update_approx_time (time_t *now*)

Update the cached estimate of the current time. This function SHOULD be called once per second, and MUST be called before the first call to get_approx_time.

References cached_approx_time.

Referenced by do_main_loop(), second_elapsed_callback(), and tor_main().

4.54.3.95 void wrap_string (smartlist_t * *out*, const char * *string*, size_t *width*, const char * *prefix0*, const char * *prefixRest*)

Rudimentary string wrapping code: given a un-wrapped **string** (no newlines!), break the string into newline-terminated lines of no more than **width** characters long (not counting newline) and insert them into **out** in order. Precede the first line with prefix0, and subsequent lines with prefixRest.

References smartlist_add(), and tor_assert.

4.54.3.96 ssize_t write_all (int *fd*, const char * *buf*, size_t *count*, int *isSocket*)

Write **count** bytes from **buf** to **fd**. **isSocket** must be 1 if fd was returned by socket() or accept(), and 0 if fd was returned by open(). Return the number of bytes written, or -1 on error. Only use if fd is a blocking fd.

References tor_assert.

Referenced by cpuworker_main(), log_tor_version(), logv(), and write_chunks_to_file_impl().

4.54.3.97 int write_bytes_to_file (const char ∗ *fname*, const char ∗ *str*, size_t *len*, int *bin*)

As write_str_to_file, but does not assume a NUL-terminated string. Instead, we write **len** bytes, starting at **str**.

References smartlist_add(), smartlist_create(), smartlist_free(), and write_chunks_to_file_impl().

Referenced by init_cookie_authentication(), and write_str_to_file().

4.54.3.98 int write_chunks_to_file (const char ∗ *fname*, const smartlist_t ∗ *chunks*, int *bin*)

Given a smartlist of **sized_chunk_t** (p. 231), write them atomically to a file **fname**, overwriting or creating the file as necessary.

References write_chunks_to_file_impl().

Referenced by dirvote_compute_consensuses(), and router_rebuild_store().

4.54.3.99 static int write_chunks_to_file_impl (const char ∗ *fname*, const smartlist_t ∗ *chunks*, int *open_flags*) `[static]`

Helper: given a set of flags as passed to open(2), open the file **fname** and write all the **sized_chunk_t** (p. 231) structs in **chunks** to the file. Do so as atomically as possible e.g. by opening temp files and renaming.

References abort_writing_to_file(), finish_writing_to_file(), LD_FS, start_writing_to_file(), tor_assert, and write_all().

Referenced by append_bytes_to_file(), write_bytes_to_file(), and write_chunks_to_file().

4.54.3.100 void write_pidfile (char ∗ *filename*)

Write the current process ID, followed by NL, into **filename**.

References LD_FS.

Referenced by options_act().

4.54.3.101 int write_str_to_file (const char ∗ *fname*, const char ∗ *str*, int *bin*)

Create a file named **fname** with the contents **str**. Overwrite the previous **fname** if possible. Return 0 on success, -1 on failure.

This function replaces the old file atomically, if possible. This function, and all other functions in **util.c** (p. 1237) that create files, create them with mode 0600.

References LD_BUG, and write_bytes_to_file().

Referenced by add_networkstatus_to_cache(), crypto_pk_write_private_key_to_filename(), dump_desc(), geoip_bridge_stats_write(), init_keys(), microdesc_cache_rebuild(), networkstatus_dump_bridge_status_to_file(), networkstatus_set_current_consensus(), or_state_save(), rend_service_load_keys(), and write_configuration_file().

4.54.4 Variable Documentation

4.54.4.1 time_t cached_approx_time = 0 `[static]`

Cached estimate of the current time. Updated around once per second; may be a few seconds off if we are really busy. This is a hack to avoid calling time(NULL) (which not everybody has optimized) on critical paths.

Referenced by approx_time(), and update_approx_time().

4.54.4.2 int daemon_filedes[2] `[static]`

Socketpair used to communicate between parent and child process while daemonizing.

Referenced by finish_daemon(), and start_daemon().

4.54.4.3 const int days_per_month[ ] `[static]`

Initial value:

```
{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
```

Number of days per month in non-leap year; used by tor_timegm.

Referenced by tor_timegm().

4.54.4.4 int finish_daemon_called = 0 `[static]`

True iff we've called **finish_daemon()** (p. 1247).

Referenced by finish_daemon().

4.54.4.5 int ftime_skew = 0 `[static]`

Our current estimate of our skew, such that we think the current time is closest to time(NULL)+ftime_skew.

Referenced by ftime_definitely_after(), ftime_definitely_before(), ftime_maybe_after(), ftime_maybe_before(), and ftime_set_estimated_skew().

4.54.4.6 int ftime_slop = 60 `[static]`

Tolerance during time comparisons, in seconds.

Referenced by ftime_definitely_after(), ftime_definitely_before(), ftime_maybe_after(), ftime_maybe_before(), and ftime_set_maximum_sloppiness().

4.54.4.7 const char* MONTH_NAMES[] [static]

Initial value:

```
{ "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" }
```

A c-locale array of 3-letter names of months, starting with Jan.

Referenced by format_rfc1123_time(), parse_http_time(), and parse_rfc1123_time().

4.54.4.8 int start_daemon_called = 0 [static]

True iff we've called **start_daemon()** (p. 1252).

Referenced by finish_daemon(), and start_daemon().

4.54.4.9 const char* WEEKDAY_NAMES[] [static]

Initial value:

```
{ "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" }
```

A c-locale array of 3-letter names of weekdays, starting with Sun.

Referenced by format_rfc1123_time().

4.55 util.h File Reference

Headers for **util.c** (p. 1237).

`#include "orconfig.h"`

`#include "torint.h"`

`#include "compat.h"`

`#include <stdio.h>`

`#include <stdlib.h>`

Data Structures

- struct **sized_chunk_t**

Defines

- #define **O_BINARY** 0
- #define **O_TEXT** 0
- #define **tor_assert**(expr)
- #define **DMALLOC_PARAMS**
- #define **DMALLOC_ARGS**
- #define **tor_fragile_assert**()
- #define **tor_free**(p)
- #define **tor_malloc**(size) _tor_malloc(size DMALLOC_ARGS)
- #define **tor_malloc_zero**(size) _tor_malloc_zero(size DMALLOC_ARGS)
- #define **tor_malloc_roundup**(szp) _tor_malloc_roundup(szp DMALLOC_ARGS)
- #define **tor_realloc**(ptr, size) _tor_realloc(ptr, size DMALLOC_ARGS)
- #define **tor_strdup**(s) _tor_strdup(s DMALLOC_ARGS)
- #define **tor_strndup**(s, n) _tor_strndup(s, n DMALLOC_ARGS)
- #define **tor_memdup**(s, n) _tor_memdup(s, n DMALLOC_ARGS)
- #define **STRUCT_OFFSET**(tp, member) ((off_t) (((char∗)&((tp∗)0) → member)-(char∗)0))
- #define **STRUCT_VAR_P**(st, off) ((void∗) (((char∗)(st)) + (off)))
- #define **SUBTYPE_P**(p, subtype, basemember) ((void∗) (((char∗)(p)) - STRUCT_OFFSET(subtype, basemember)))
- #define **bool_eq**(a, b) (!(a)==!(b))
- #define **bool_neq**(a, b) (!(a)!=!(b))
- #define **HEX_CHARACTERS** "0123456789ABCDEFabcdef"
- #define **RFC1123_TIME_LEN** 29
- #define **ISO_TIME_LEN** 19
- #define **OPEN_FLAGS_REPLACE** (O_WRONLY|O_CREAT|O_TRUNC)
- #define **OPEN_FLAGS_APPEND** (O_WRONLY|O_CREAT|O_APPEND)
- #define **RFTS_BIN** 1
- #define **RFTS_IGNORE_MISSING** 2

Typedefs

- typedef struct **open_file_t** **open_file_t**

Enumerations

- enum **file_status_t** { **FN_ERROR**, **FN_NOENT**, **FN_FILE**, **FN_DIR** }
- enum **cpd_check_t** { **CPD_NONE**, **CPD_CREATE**, **CPD_CHECK** }

Functions

- void ∗ **_tor_malloc** (size_t size DMALLOC_PARAMS) ATTR_MALLOC
- void ∗ **_tor_malloc_zero** (size_t size DMALLOC_PARAMS) ATTR_MALLOC
- void ∗ **_tor_malloc_roundup** (size_t ∗size DMALLOC_PARAMS) ATTR_MALLOC
- void ∗ **_tor_realloc** (void ∗ptr, size_t size DMALLOC_PARAMS)
- char ∗ **_tor_strdup** (const char ∗s DMALLOC_PARAMS) ATTR_MALLOC ATTR_NONNULL((1))
- char ∗ **_tor_strndup** (const char ∗s, size_t n DMALLOC_PARAMS) ATTR_MALLOC ATTR_NONNULL((1))
- void ∗ **_tor_memdup** (const void ∗mem, size_t len DMALLOC_PARAMS) ATTR_MALLOC ATTR_NONNULL((1))
- void **_tor_free** (void ∗mem)
- void **tor_log_mallinfo** (int severity)
- double **tor_mathlog** (double d) ATTR_CONST
- long **tor_lround** (double d) ATTR_CONST
- int **tor_log2** (uint64_t u64) ATTR_CONST
- uint64_t **round_to_power_of_2** (uint64_t u64)
- unsigned **round_to_next_multiple_of** (unsigned number, unsigned divisor)
- uint32_t **round_uint32_to_next_multiple_of** (uint32_t number, uint32_t divisor)
- uint64_t **round_uint64_to_next_multiple_of** (uint64_t number, uint64_t divisor)
- void **tor_strlower** (char ∗s) ATTR_NONNULL((1))
- void **tor_strupper** (char ∗s) ATTR_NONNULL((1))
- int **tor_strisprint** (const char ∗s) ATTR_PURE ATTR_NONNULL((1))
- int **tor_strisnonupper** (const char ∗s) ATTR_PURE ATTR_NONNULL((1))
- int **strcmpstart** (const char ∗s1, const char ∗s2) ATTR_PURE ATTR_NONNULL((1
- int int **strcmp_len** (const char ∗s1, const char ∗s2, size_t len) ATTR_PURE ATTR_NONNULL((1
- int int int **strcasecmpstart** (const char ∗s1, const char ∗s2) ATTR_PURE ATTR_NONNULL((1
- int int int int **strcmpend** (const char ∗s1, const char ∗s2) ATTR_PURE ATTR_NONNULL((1
- int int int int int **strcasecmpend** (const char ∗s1, const char ∗s2) ATTR_PURE ATTR_NONNULL((1
- int int int int int int **memcmpstart** (const void ∗mem, size_t memlen, const char ∗prefix) ATTR_PURE
- void **tor_strstrip** (char ∗s, const char ∗strip) ATTR_NONNULL((1
- void long **tor_parse_long** (const char ∗s, int base, long min, long max, int ∗ok, char ∗∗next)
- unsigned long **tor_parse_ulong** (const char ∗s, int base, unsigned long min, unsigned long max, int ∗ok, char ∗∗next)
- double **tor_parse_double** (const char ∗s, double min, double max, int ∗ok, char ∗∗next)
- uint64_t **tor_parse_uint64** (const char ∗s, int base, uint64_t min, uint64_t max, int ∗ok, char ∗∗next)
- const char ∗ **hex_str** (const char ∗from, size_t fromlen) ATTR_NONNULL((1))
- const char ∗ **eat_whitespace** (const char ∗s) ATTR_PURE

- const char * **eat_whitespace_eos** (const char *s, const char *eos) ATTR_PURE
- const char * **eat_whitespace_no_nl** (const char *s) ATTR_PURE
- const char * **eat_whitespace_eos_no_nl** (const char *s, const char *eos) ATTR_PURE
- const char * **find_whitespace** (const char *s) ATTR_PURE
- const char * **find_whitespace_eos** (const char *s, const char *eos) ATTR_PURE
- const char * **find_str_at_start_of_line** (const char *haystack, const char *needle) ATTR_PURE
- int **tor_mem_is_zero** (const char *mem, size_t len) ATTR_PURE
- int **tor_digest_is_zero** (const char *digest) ATTR_PURE
- int **tor_digest256_is_zero** (const char *digest) ATTR_PURE
- char * **esc_for_log** (const char *string) ATTR_MALLOC
- const char * **escaped** (const char *string)
- void **wrap_string** (struct **smartlist_t** *out, const char *string, size_t width, const char *prefix0, const char *prefixRest)
- int **tor_vsscanf** (const char *buf, const char *pattern, va_list ap)
- int **tor_sscanf** (const char *buf, const char *pattern,...)
- int **hex_decode_digit** (char c)
- void **base16_encode** (char *dest, size_t destlen, const char *src, size_t srclen)
- int **base16_decode** (char *dest, size_t destlen, const char *src, size_t srclen)
- double **tv_to_double** (const struct timeval *tv)
- int64_t **tv_to_msec** (const struct timeval *tv)
- int64_t **tv_to_usec** (const struct timeval *tv)
- long **tv_udiff** (const struct timeval *start, const struct timeval *end)
- long **tv_mdiff** (const struct timeval *start, const struct timeval *end)
- time_t **tor_timegm** (struct tm *tm)
- void **format_rfc1123_time** (char *buf, time_t t)
- int **parse_rfc1123_time** (const char *buf, time_t *t)
- void **format_local_iso_time** (char *buf, time_t t)
- void **format_iso_time** (char *buf, time_t t)
- int **parse_iso_time** (const char *buf, time_t *t)
- int **parse_http_time** (const char *buf, struct tm *tm)
- int **format_time_interval** (char *out, size_t out_len, long interval)
- time_t **approx_time** (void)
- void **update_approx_time** (time_t now)
- void **ftime_set_maximum_sloppiness** (int seconds)
- void **ftime_set_estimated_skew** (int seconds)
- int **ftime_maybe_after** (time_t now, time_t when)
- int **ftime_maybe_before** (time_t now, time_t when)
- int **ftime_definitely_after** (time_t now, time_t when)
- int **ftime_definitely_before** (time_t now, time_t when)
- ssize_t **write_all** (int fd, const char *buf, size_t count, int isSocket)
- ssize_t **read_all** (int fd, char *buf, size_t count, int isSocket)
- **file_status_t** **file_status** (const char *filename)
- int **check_private_dir** (const char *dirname, **cpd_check_t** check)
- int **start_writing_to_file** (const char *fname, int open_flags, int mode, **open_file_t** **data_out)
- FILE * **start_writing_to_stdio_file** (const char *fname, int open_flags, int mode, **open_file_t** **data_out)
- FILE * **fdopen_file** (**open_file_t** *file_data)
- int **finish_writing_to_file** (**open_file_t** *file_data)

- int **abort_writing_to_file** (**open_file_t** ∗file_data)
- int **write_str_to_file** (const char ∗fname, const char ∗str, int bin)
- int **write_bytes_to_file** (const char ∗fname, const char ∗str, size_t len, int bin)
- int **write_chunks_to_file** (const char ∗fname, const struct **smartlist_t** ∗chunks, int bin)
- int **append_bytes_to_file** (const char ∗fname, const char ∗str, size_t len, int bin)
- char ∗ **read_file_to_str** (const char ∗filename, int flags, struct stat ∗stat_out) ATTR_MALLOC
- const char ∗ **parse_config_line_from_str** (const char ∗line, char ∗∗key_out, char ∗∗value_out)
- char ∗ **expand_filename** (const char ∗filename)
- struct **smartlist_t** ∗ **tor_listdir** (const char ∗dirname)
- int **path_is_relative** (const char ∗filename) ATTR_PURE
- void **start_daemon** (void)
- void **finish_daemon** (const char ∗desired_cwd)
- void **write_pidfile** (char ∗filename)
- const char ∗ **libor_get_digests** (void)

4.55.1 Detailed Description

Headers for **util.c** (p. 1237).

4.55.2 Define Documentation

4.55.2.1 #define bool_eq(a, b) (!(a)==!(b))

Macro: true if two values have the same boolean value.

Referenced by circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), options_act(), and routerstatus_parse_entry_from_string().

4.55.2.2 #define bool_neq(a, b) (!(a)!=!(b))

Macro: true if two values have different boolean values.

Referenced by directory_remove_invalid().

4.55.2.3 #define HEX_CHARACTERS "0123456789ABCDEFabcdef"

Allowable characters in a hexadecimal string.

Referenced by is_legal_hexdigest().

4.55.2.4 #define RFTS_BIN 1

Flag for read_file_to_str: open the file in binary mode.

Referenced by read_file_to_str(), and router_reload_router_list_impl().

4.55.2.5 #define RFTS_IGNORE_MISSING 2

Flag for read_file_to_str: it's okay if the file doesn't exist.

Referenced by dirserv_load_fingerprint_file(), load_bridge_stats(), microdesc_cache_reload(), read_file_to_str(), rend_service_load_keys(), rep_hist_load_mtbf_data(), router_reload_consensus_networkstatus(), router_reload_router_list_impl(), and trusted_dirs_reload_certs().

4.55.2.6 #define STRUCT_OFFSET(tp, member) ((off_t) (((char*)&((tp*)0) → member)-(char*)0))

Return the offset of **member** within the type **tp**, in bytes

Referenced by add_cell_ewma_to_conn(), mp_pool_new(), pop_first_cell_ewma_from_conn(), purge_expired_resolves(), remove_cell_ewma_from_conn(), and set_expiry().

4.55.2.7 #define STRUCT_VAR_P(st, off) ((void*) (((char*)(st)) + (off)))

Macro: yield a pointer to the field at position **off** within the structure **st**. Example:

```
struct a { int foo; int bar; } x;
off_t bar_offset = STRUCT_OFFSET(struct a, bar) (p. 1266);
int *bar_p = STRUCT_VAR_P(&x, bar_offset) (p. 1266);
*bar_p = 3;
```

Referenced by config_alloc(), config_assign_line(), config_assign_value(), config_dump(), config_free(), get_assigned_option(), and option_clear().

4.55.2.8 #define SUBTYPE_P(p, subtype, basemember) ((void*) (((char*)(p)) - STRUCT_OFFSET(subtype, basemember)))

Macro: yield a pointer to an enclosing structure given a pointer to a substructure at offset **off**. Example:

```
struct base { ... };
struct subtype { int x; struct base b; } x;
struct base *bp = &x.base;
struct *sp = SUBTYPE_P(bp, struct subtype, b) (p. 1266);
```

Referenced by cell_ewma_to_circuit().

4.55.2.9 #define tor_assert(expr)

Value:

```
STMT_BEGIN                                    \
    if (PREDICT_UNLIKELY(!(expr))) {                                \
      log_err(LD_BUG, "%s:%d: %s: Assertion %s failed; aborting.",      \
          _SHORT_FILE_, __LINE__, __func__, #expr);                 \
      fprintf(stderr,"%s:%d %s: Assertion %s failed; aborting.\n",      \
              _SHORT_FILE_, __LINE__, __func__, #expr);             \
```

```
    abort();                                                        \
  } STMT_END
```

Like assert(3), but send assertion failures to the log as well as to stderr.

Referenced by _circuit_mark_for_close(), _connection_free(), _connection_mark_for_close(), _connection_mark_unattached_ap(), _crypto_new_pk_env_rsa(), _crypto_pk_env_get_evp_pkey(), _dirreq_map_put(), _find_by_keyword(), _routerlist_find_elt(), _tor_memdup(), _tor_strdup(), _tor_strndup(), accounting_parse_options(), accounting_set_wakeup_time(), add_answer_to_cache(), add_cell_ewma_to_conn(), add_connection_to_closeable_list(), add_fingerprint_to_dir(), add_nickname_list_to_smartlist(), add_spaces_to_fp(), addr_policy_free(), address_is_in_virtual_range(), addressmap_get_virtual_address(), addressmap_register_virtual_address(), aes_set_key(), alloc_chunk(), any_bridge_descriptors_known(), assert_active_circuits_ok(), assert_addr_policy_ok(), assert_all_pending_dns_resolves_ok(), assert_buf_ok(), assert_circuit_ok(), assert_connection_edge_not_dns_pending(), assert_connection_ok(), assert_cpath_layer_ok(), assert_cpath_ok(), assert_freelist_ok(), assert_resolve_ok(), assign_onionskin_to_cpuworker(), authdir_wants_to_reject_router(), authority_cert_dup(), authority_cert_get_all(), authority_cert_parse_from_string(), base16_encode(), base32_decode(), base32_encode(), base64_decode(), base64_encode(), buf_add_chunk_with_capacity(), buf_find_pos_of_char(), buf_find_string_offset(), buf_pullup(), buf_remove_from_front(), buf_shrink_freelists(), cell_queue_append(), cell_queue_pop(), check_private_dir(), check_signature_token(), choose_good_middle_server(), chunk_grow(), chunk_new_with_alloc_size(), circuit_all_predicted_ports_handled(), circuit_append_new_exit(), circuit_build_times_add_time(), circuit_build_times_calculate_timeout(), circuit_build_times_cdf(), circuit_build_times_count_pretimeouts(), circuit_build_times_generate_sample(), circuit_build_times_initial_alpha(), circuit_build_times_new_consensus_params(), circuit_build_times_parse_state(), circuit_build_times_update_alpha(), circuit_deliver_create_cell(), circuit_detach_stream(), circuit_discard_optional_exit_enclaves(), circuit_extend(), circuit_finish_handshake(), circuit_free(), circuit_get_all_pending_on_or_conn(), circuit_get_best(), circuit_get_by_edge_conn(), circuit_get_by_rend_query_and_purpose(), circuit_get_next_by_pk_and_purpose(), circuit_get_open_circ_or_launch(), circuit_handle_first_hop(), circuit_init_cpath_crypto(), circuit_is_acceptable(), circuit_launch_by_extend_info(), circuit_list_path_impl(), circuit_n_conn_done(), circuit_package_relay_cell(), circuit_receive_relay_cell(), circuit_remove_handled_ports(), circuit_send_next_onion_skin(), circuit_set_n_circid_orconn(), circuit_set_p_circid_orconn(), circuit_set_state(), circuit_truncated(), clean_cell_pool(), client_dns_set_addressmap(), client_dns_set_addressmap_impl(), command_process_create_cell(), command_process_netinfo_cell(), command_process_versions_cell(), compare_tor_addr_to_addr_policy(), compute_consensus_method(), config_assign_value(), config_dump(), config_free(), config_parse_units(), connection_about_to_close_connection(), connection_add(), connection_ap_can_use_exit(), connection_ap_expire_beginning(), connection_ap_handshake_attach_chosen_circuit(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_process_socks(), connection_ap_handshake_rewrite_and_attach(), connection_ap_handshake_send_begin(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_reply(), connection_ap_make_link(), connection_ap_process_end_not_open(), connection_ap_process_natd(), connection_ap_process_transparent(), connection_bucket_refill(), connection_bucket_should_increase(), connection_connect(), connection_control_finished_flushing(), connection_control_process_inbuf(), connection_control_reached_eof(), connection_cpu_finished_flushing(), connection_cpu_process_inbuf(), connection_create_listener(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_routerdesc_failed(), connection_dir_finished_connecting(), connection_dir_finished_flushing(), connection_dir_process_inbuf(), connection_dirserv_add_dir_bytes_to_outbuf(), connection_dirserv_flushed_some(), connection_dns_remove(), connection_edge_end_errno(), connection_edge_finished_connecting(), connection_edge_finished_flushing(), connection_edge_is_rendezvous_stream(), connection_edge_package_raw_inbuf(), connection_edge_process_inbuf(), connection_edge_process_relay_cell(), connection_edge_process_relay_cell_not_open(), connection_edge_send_command(), connection_exit_begin_conn(), connection_expire_held_open(), connection_finished_connecting(), connection_finished_flushing(), connection_flushed_some(), connection_free(), connection_get_by_type_state_rendquery(), connection_handle_listener_read(), connection_handle_read_impl(), connection_handle_write_impl(), connection_is_reading(), connection_is_writing(), connection_link_connections(), connection_or_check_valid_tls_handshake(), connection_or_connect(), connection_or_finished_connecting(), connection_or_finished_flushing(), connection_or_flush_from_first_active_circuit(), connection_or_get_for_extend(), connection_or_nonopen_was_started_here(), connection_or_process_inbuf(), connection_or_remove_from_identity_map(), connection_or_send_destroy(), connection_or_send_versions(), connection_or_set_identity_digest(), connection_or_write_cell_to_buf(), connection_or_write_var_cell_to_buf(), connection_process_inbuf(), connection_proxy_connect(), connection_read_proxy_handshake(), connection_remove(), connection_start_reading(), connection_start_reading_from_linked_conn(), connection_start_writing(), connection_state_is_connecting(), connection_state_is_open(), connection_stop_reading(), connection_stop_reading_from_linked_conn(), connection_stop_writing(), connection_tls_continue_handshake(), consider_hibernation(), control_event_circuit_status(), control_event_stream_status(), cpuworker_main(), crypto_cipher_decrypt(), crypto_cipher_decrypt_init_cipher(), crypto_cipher_decrypt_with_iv(), crypto_cipher_encrypt(), crypto_cipher_encrypt_init_cipher(), crypto_cipher_encrypt_with_iv(), crypto_cipher_generate_key(), crypto_cipher_set_iv(), crypto_cipher_set_key(), crypto_dh_compute_secret(), crypto_dh_free(), crypto_dh_get_bytes(), crypto_dh_get_public(), crypto_digest(), crypto_digest_add_bytes(), crypto_digest_all(), crypto_digest_assign(), crypto_digest_dup(), crypto_digest_get_digest(), crypto_expand_key_material(), crypto_free_cipher_env(), crypto_get_rsa_padding(), crypto_get_rsa_padding_overhead(), crypto_hmac_sha1(), crypto_pk_check_key(), crypto_pk_cmp_keys(), crypto_pk_copy_full(), crypto_pk_dup_key(), crypto_pk_generate_key_with_bits(), crypto_pk_key_is_private(), crypto_pk_keysize(), crypto_pk_private_decrypt(), crypto_pk_private_hybrid_decrypt(), crypto_pk_private_sign(), crypto_pk_public_checksig(), crypto_pk_public_checksig_digest(), crypto_pk_public_encrypt(), crypto_pk_public_hybrid_encrypt(), crypto_pk_read_private_key_from_string(), crypto_pk_read_public_key_from_string(), crypto_pk_write_key_to_string_impl(), crypto_pk_write_private_key_to_filename(), crypto_rand(), crypto_rand_int(), crypto_rand_uint64(), crypto_random_hostname(), decide_to_advertise_dirport(), decode_escaped_string(), decode_hashed_passwords(), digestmap_assert_ok(), digestmap_free(), digestmap_get(), digestmap_iter_get(), digestmap_iter_init(), digestmap_iter_next(), digestmap_iter_next_rmv(), digestmap_remove(), digestmap_set(), digit_to_num(), dir_routerdesc_download_failed(), dir_split_resource_into_fingerprints(), directory_get_from_all_authorities(), directory_get_from_hs_dir(), directory_handle_command(), directory_handle_command_get(), directory_handle_command_post(), directory_initiate_command_rend(), directory_post_to_dirservers(), directory_send_command(), dirserv_add_extrainfo(), dirserv_add_multiple_descriptors(), dirserv_dump_directory_to_string(), dirserv_estimate_data_size(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_networkstatus_v2(), dirserv_get_networkstatus_v2_fingerprints(), dirserv_get_nickname_by_digest(), dirserv_orconn_tls_done(), dirserv_set_cached_networkstatus_v2(), dirvote_add_signatures_to_pending_consensus(), dirvote_add_vote(), dirvote_compute_consensuses(), dirvote_compute_params(), dirvote_format_microdesc_vote_line(), dirvote_get_pending_consensus(), dirvote_get_preferred_voting_intervals(), dirvote_publish_consensus(), dirvote_recalculate_timing(), dns_cancel_pending_resolve(), dns_found_answer(), dns_resolve(), dns_resolve_impl(), dnsserv_close_listener(), dnsserv_configure_listener(), do_list_fingerprint(), download_status_increment_failure(), dump_desc(), dup_onion_keys(), eat_whitespace(), eat_whitespace_eos(), edge_connection_new(), edge_of_accounting_period_containing(), enable_control_logging(), entry_guard_set_status(), entry_guards_compute_status(), entry_guards_prepend_from_config(), entry_is_live(), evdns_log_cb(), evdns_server_callback(), exit_policy_is_general_exit_helper(), exit_policy_remove_redundancies(), expand_filename(), extend_info_dup(), extend_info_from_router(), extrainfo_get_by_descriptor_digest(), extrainfo_insert(), extrainfo_parse_entry_from_string(), fdopen_file(), fetch_from_buf(), fetch_from_buf_http(), fetch_from_buf_socks(), fetch_from_buf_socks_client(), find_dir_signing_key(), find_dl_schedule_and_len(), find_intro_circuit(), find_start_of_next_router_or_extrainfo(), finish_writing_to_file_impl(), flush_buf(), flush_buf_tls(), flush_chunk(), flush_chunk_tls(), format_networkstatus_vote(), format_rfc1123_time(), ftime_set_maximum_sloppiness(), geoip_add_entry(), geoip_get_client_history(), geoip_note_ns_response(), get_assigned_option(), get_identity_key(), get_interface_address6(), get_next_token(), get_onion_key(), get_options(), get_or_state(), get_unique_circ_id_by_conn(), get_user_homedir(), get_voter(), handle_control_authenticate(), handle_control_getconf(), handle_control_postdescriptor(), handle_getinfo_helper(), hex_digest_matches(), hibernate_end(), hid_serv_get_responsible_directories(), HT_HEAD(), HT_PROTOTYPE(), init_cell_pool(), init_dh_param(), init_key_from_file(), init_keys(), is_legal_hexdigest(), is_legal_nickname(), launch_resolve(), launch_test_addresses(), launch_wildcard_check(), learned_bridge_descriptor(), link_apconn_to_circ(), list_pending_downloads(), list_server_status_v1(), list_single_server_status(), load_downloaded_routers(), load_torrc_from_disk(), make_circuit_active_on_conn(), make_circuit_inactive_on_conn(), make_consensus_method_list(), memarea_alloc(), memarea_assert_ok(), memarea_get_stats(), microdesc_cache_rebuild(), microdescs_parse_from_string(), move_buf_to_buf(), mp_pool_new(), munge_extrainfo_into_routerinfo(), networkstatus_add_detached_signatures(), networkstatus_check_consensus_signature(), networkstatus_compute_consensus(), networkstatus_get_detached_signatures(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), new_route_len(), next_circ_on_conn_p(), onion_next_task(), onion_pending_add(), onion_skin_client_handshake(), onion_skin_create(), options_act_reversible(), options_dup(), options_get_datadir_fname2_suffix(), options_init_from_string(), options_validate(), or_state_save(), or_state_set(), orconn_target_get_name(), parse_addr_and_port_range(), parse_addr_port(), parse_bridge_stats_controller(), parse_config_line_from_str(), parse_http_response(), parse_http_time(), parse_port_range(), peek_from_buf(), pick_entry_guards(), policy_expand_private(), policy_summarize(), policy_summary_accept(), policy_summary_add_item(), policy_summary_item_split(), policy_summary_reject(), prev_circ_on_conn_p(), process_pending_task(), purge_expired_resolves(), read_file_to_str(), read_to_buf(), read_to_buf_tls(), read_to_chunk(), read_to_chunk_tls(), realign_pointer(), rectify_client_ciphers(), relay_crypt(), relay_send_command_from_edge(), remove_cell_ewma_from_conn(), remove_obsolete_entry_guards(), rend_cache_lookup_entry(), rend_cache_lookup_v2_desc_as_dir(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_check_authorization(), rend_client_introcirc_has_opened(), rend_client_introduction_acked(), rend_client_lookup_service_authorization(), rend_client_receive_rendezvous(), rend_client_refetch_v2_renddesc(), rend_client_rendcirc_has_opened(), rend_client_send_establish_rendezvous(), rend_client_send_introduction(), rend_config_services(), rend_data_dup(), rend_decrypt_introduction_points(), rend_encode_v2_descriptors(), rend_encrypt_v2_intro_points_basic(), rend_encrypt_v2_intro_points_stealth(), rend_get_service_id(), rend_id_is_in_interval(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_service_descriptor(), rend_parse_v2_service_descriptor(), rend_service_intro_established(), rend_service_intro_has_opened(), rend_service_introduce(), rend_service_relaunch_rendezvous(), rend_service_rendezvous_has_opened(), rend_service_set_connection_addr_port(), rend_services_introduce(), rep_hist_exit_stats_write(), rep_hist_fill_bandwidth_history(), rep_hist_get_bandwidth_lines(), rep_hist_get_predicted_ports(), rep_hist_load_state(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), rep_hist_note_used_port(), resolve_-

my_address(), retry_listeners(), router_add_to_routerlist(), router_choose_random_node(), router_differences_are_cosmetic(), router_dump_router_to_string(), router_get_by_descriptor_digest(), router_get_by_digest(), router_get_by_extrainfo_digest(), router_get_by_hexdigest(), router_get_by_nickname(), router_get_my_descriptor(), router_handles_some_port(), router_load_single_router(), router_nickname_is_in_list(), router_parse_addr_policy(), router_parse_directory(), router_parse_entry_from_string(), router_parse_list_from_string(), router_parse_runningrouters(), router_pick_trusteddirserver(), router_purpose_to_string(), router_rebuild_descriptor(), router_set_networkstatus_v2(), router_set_status(), routerinfo_incompatible_with_extrainfo(), routerlist_assert_ok(), routerlist_descriptors_added(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_remove_old_cached_routers_with_id(), routerlist_replace(), routerset_get_all_routers(), routerset_refresh_countries(), routerset_subtract_routers(), routerset_union(), routerstatus_format_entry(), routerstatus_has_changed(), routerstatus_parse_entry_from_string(), run_connection_housekeeping(), safe_str(), safe_str_client(), scale_active_circuits(), secret_to_key(), send_control_event_string(), send_resolved_cell(), send_resolved_hostname_cell(), set_expiry(), set_log_severity_config(), set_max_file_descriptors(), set_options(), set_streams_blocked_on_circ(), should_use_create_fast_for_circuit(), signed_descriptor_get_body_impl(), smartlist_add_all(), smartlist_bsearch_idx(), smartlist_choose_by_bandwidth(), smartlist_choose_by_bandwidth_weights(), smartlist_del(), smartlist_del_keeporder(), smartlist_ensure_capacity(), smartlist_insert(), smartlist_join_strings2(), smartlist_pop_last(), smartlist_pqueue_assert_ok(), smartlist_pqueue_pop(), smartlist_pqueue_remove(), smartlist_reverse(), smartlist_split_string(), smartlist_string_remove(), spawn_cpuworker(), spawn_exit(), spawn_func(), start_writing_to_file(), strmap_assert_ok(), strmap_free(), strmap_get(), strmap_iter_get(), strmap_iter_init(), strmap_iter_next(), strmap_iter_next_rmv(), strmap_remove(), strmap_set(), switch_id(), TO_CONTROL_CONN(), TO_DIR_CONN(), TO_EDGE_CONN(), TO_OR_CIRCUIT(), TO_OR_CONN(), TO_ORIGIN_CIRCUIT(), tokenize_string(), tor_addr_compare_masked(), tor_addr_copy(), tor_addr_from_ipv4n(), tor_addr_from_ipv6_bytes(), tor_addr_from_sockaddr(), tor_addr_from_str(), tor_addr_is_loopback(), tor_addr_is_null(), tor_addr_is_v4(), tor_addr_lookup(), tor_addr_parse_mask_ports(), tor_addr_port_parse(), tor_addr_to_str(), tor_asprintf(), tor_check_dh_key(), tor_close_socket(), tor_gzip_compress(), tor_gzip_uncompress(), tor_inet_pton(), tor_libevent_initialize(), tor_lockfile_unlock(), tor_main(), tor_memmem(), tor_mmap_file(), tor_mutex_acquire(), tor_mutex_release(), tor_mutex_uninit(), tor_parse_uint64(), tor_strtok_r_impl(), tor_timegm(), tor_tls_context_decref(), tor_tls_context_new(), tor_tls_create_certificate(), tor_tls_free(), tor_tls_get_pending_bytes(), tor_tls_handshake(), tor_tls_is_server(), tor_tls_new(), tor_tls_read(), tor_tls_renegotiate(), tor_tls_set_logged_address(), tor_tls_shutdown(), tor_tls_write(), tor_version_as_new_as(), tor_version_compare(), tor_version_is_obsolete(), tor_version_parse(), tor_version_same_series(), tor_zlib_process(), update_consensus_networkstatus_fetch_time(), update_consensus_router_descriptor_downloads(), update_or_history(), validate_data_directory(), validate_ports_csv(), wrap_string(), write_all(), write_chunks_to_file_impl(), write_configuration_file(), write_escaped_data(), write_http_response_header_impl(), and write_to_buf().

4.55.2.10 #define tor_fragile_assert()

Define this if you want Tor to crash when any problem comes up, so you can get a coredump and track things down.

Referenced by _connection_mark_for_close(), _connection_write_to_buf_impl(), choose_good_exit_server(), choose_random_entry(), circuit_detach_stream(), circuit_launch_by_extend_info(), command_process_cell(), command_process_var_cell(), conn_read_callback(), conn_write_callback(), connection_about_to_close_connection(), connection_ap_get_original_destination(), connection_buckets_decrement(), connection_close_immediate(), connection_dir_finished_flushing(), connection_edge_end(), connection_edge_finished_flushing(), connection_edge_process_inbuf(), connection_finished_connecting(), connection_finished_flushing(), connection_free(), connection_handle_read_impl(), connection_or_finished_flushing(), connection_process_inbuf(), connection_proxy_connect(), connection_reached_eof(), connection_read_proxy_handshake(), crypto_digest_add_bytes(), crypto_digest_algorithm_get_name(), crypto_digest_get_digest(), dns_cancel_pending_resolve(), dns_resolve_impl(), evdns_get_orig_address(), getinfo_helper_events(), networkstatus_get_flavor_name(), purge_expired_resolves(), rend_process_relay_cell(), router_get_consensus_status_by_nickname(), smartlist_choose_by_bandwidth(), smartlist_choose_by_bandwidth_weights(), tor_addr_compare_masked(), tor_addr_hash(), tor_addr_is_loopback(), tor_mutex_acquire(), tor_mutex_init(), tor_mutex_release(), tor_mutex_uninit(), and unescape_string().

4.55.2.11 #define tor_free(p)

Value:

```
STMT_BEGIN                                 \
    if (PREDICT_LIKELY((p)!=NULL)) {                          \
      free(p);                                                \
      (p)=NULL;                                               \
    }                                                         \
  STMT_END
```

Release memory allocated by tor_malloc, tor_realloc, tor_strdup, etc. Unlike the free() function, **tor_free()** (p. 1271) will still work on NULL pointers, and it sets the pointer value to NULL after freeing it.

This is a macro. If you need a function pointer to release memory from tor_malloc(), use **_tor_free()** (p. 1241).

Referenced by _connection_free(), _free_cached_resolve(), _free_link_history(), _openssl_dynlock_destroy_cb(), _remove_old_client_helper(), _tor_free(), accounting_parse_options(), add_fingerprint_to_dir(), add_networkstatus_to_cache(), add_nickname_list_to_smartlist(), add_temp_log(), addr_policy_free(), addressmap_ent_free(), addressmap_register(), addressmap_register_virtual_address(), addressmap_rewrite(), addressmap_rewrite_reverse(), addressmap_virtaddress_ent_free(), addressmap_virtaddress_remove(), aes_free_cipher(), alloc_http_authenticator(), assign_onionskin_to_cpuworker(), authority_cert_free(), authority_cert_parse_from_string(), base32_decode(), buf_free(), buf_shrink_freelists(), cached_dir_decref(), cell_queue_clear(), check_nickname_list(), check_private_dir(), check_signature_token(), choose_good_exit_server_general(), chunk_free_unchecked(), circuit_all_predicted_ports_handled(), circuit_build_times_mode(), circuit_build_times_new_consensus_params(), circuit_build_times_parse_state(), circuit_build_times_update_state(), circuit_discard_optional_exit_enclaves(), circuit_free(), circuit_free_cpath_node(), circuit_get_open_circ_or_launch(), circuit_list_path_impl(), circuit_log_path(), circuit_n_conn_done(), circuit_remove_handled_ports(), clean_accepted_intros(), clear_bridge_list(), clear_cached_dir(), clear_geoip_db(), clear_pending_onions(), clear_trackexithost_mappings(), client_check_address_changed(), client_dns_set_reverse_addressmap(), client_likes_consensus(), config_assign(), config_assign_line(), config_assign_value(), config_dump(), config_free(), config_free_all(), config_free_lines(), config_get_lines(), config_register_addressmaps(), configure_nameservers(), connection_ap_handshake_attach_circuit(), connection_ap_handshake_send_resolve(), connection_ap_handshake_socks_resolved(), connection_ap_process_end_not_open(), connection_dir_client_reached_eof(), connection_dir_download_cert_failed(), connection_dir_download_networkstatus_failed(), connection_dirserv_add_microdescs_to_outbuf(), connection_dirserv_add_networkstatus_bytes_to_outbuf(), connection_dirserv_add_servers_to_outbuf(), connection_dns_remove(), connection_exit_begin_conn(), connection_free_all(), connection_or_check_valid_tls_handshake(), connection_or_init_conn_from_address(), connection_proxy_connect(), connection_read_https_proxy_response(), connection_read_proxy_handshake(), connection_unregister_events(), control_event_circuit_status(), control_event_descriptors_changed(), control_event_logmsg(), control_event_networkstatus_changed_helper(), control_event_or_authdir_new_descriptor(), control_event_stream_status(), control_setconf_helper(), cpuworker_main(), create_inet_sockaddr(), crypto_dh_compute_secret(), crypto_dh_free(), crypto_dh_new(), crypto_expand_key_material(), crypto_free_cipher_env(), crypto_free_digest_env(), crypto_free_pk_env(), crypto_global_cleanup(), crypto_pk_asn1_decode(), crypto_pk_asn1_encode(), crypto_pk_get_digest(), crypto_pk_private_hybrid_decrypt(), crypto_pk_public_checksig_digest(), crypto_pk_public_hybrid_encrypt(), crypto_pk_read_private_key_from_filename(), crypto_pk_write_private_key_to_filename(), crypto_random_hostname(), decode_hashed_passwords(), digestmap_free(), digestmap_iter_next_rmv(), digestmap_remove(), digestset_free(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_clean_last_hid_serv_requests(), directory_get_consensus_url(), directory_handle_command(), directory_handle_command_get(), directory_handle_command_post(), directory_post_to_dirservers(), directory_post_to_hs_dir(), directory_send_command(), dirserv_add_descriptor(), dirserv_clear_old_networkstatuses(), dirserv_dump_directory_to_string(), dirserv_free_fingerprint_list(), dirserv_generate_networkstatus_vote_obj(), dirserv_get_networkstatus_v2(), dirserv_get_routerdesc_fingerprints(), dirserv_get_routerdescs(), dirserv_get_status_impl(), dirserv_load_fingerprint_file(), dirserv_regenerate_directory(), dirserv_remove_old_statuses(), dirvote_act(), dirvote_add_vote(), dirvote_clear_pending_consensuses(), dirvote_clear_votes(), dirvote_compute_consensuses(), dirvote_compute_params(), dirvote_create_microdescriptor(), dirvote_free_all(), dirvote_perform_vote(), dns_cancel_pending_resolve(), dns_found_answer(), dns_free_all(), dns_reset(), dns_reset_correctness_checks(), dns_resolve(), dnsserv_launch_request(), dnsserv_resolved(), document_signature_free(), dump_desc(), entry_guard_free(), entry_guards_parse_state(), entry_guards_prepend_from_config(), esc_router_info(), escaped(), evdns_callback(), evdns_log_cb(), evdns_server_callback(), evdns_wildcard_check_callback(), expand_filename(), extend_info_free(), extrainfo_dump_to_string(), extrainfo_free(), fast_client_handshake(), fast_server_handshake(), fetch_from_buf_socks(), file_status(), find_torrc_filename(), finish_writing_to_file_impl(), format_networkstatus_vote(), format_versions_list(), free_or_history(), generate_runningrouters(), geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), geoip_free_all(), geoip_get_client_history(), geoip_get_dirreq_history(), geoip_get_request_history(), geoip_load_file(), get_assigned_option(), get_default_nickname(), getargs_helper(), getinfo_helper_config(), getinfo_helper_dir(), getinfo_helper_entry_guards(), getinfo_helper_events(), getinfo_helper_networkstatus(), handle_control_attachstream(), handle_control_authenticate(), handle_control_closecircuit(), handle_control_closestream(), handle_control_extendcircuit(), handle_control_getconf(), handle_control_getinfo(), handle_control_loadconf(), handle_control_mapaddress(), handle_control_postdescriptor(), handle_control_protocolinfo(), handle_control_redirectstream(), handle_control_resolve(), handle_control_setcircuitpurpose(), handle_control_setevents(), handle_control_signal(), handle_control_usefeature(), HT_HEAD(), http_set_address_origin(), init_cookie_authentication(), init_keys(), initiate_descriptor_downloads(), launch_direct_bridge_descriptor_fetch(), launch_resolve(), launch_test_addresses(), launch_wildcard_check(), list_getinfo_options(), list_server_status_v1(), list_v3_auth_ids(), load_authority_keyset(), load_bridge_stats(), load_stats_file(), load_torrc_from_disk(), log_cert_lifetime(), log_credential_status(), log_entry_guards(), log_free(), log_set_application_name(), logs_free_all(), logv(), lookup_last_hid_serv_request(), make_consensus_method_list(), measured_bw_line_parse(), memarea_clear_freelist(), memarea_-

drop_all(), microdesc_cache_rebuild(), microdesc_cache_reload(), microdesc_free(), microdesc_free_all(), munge_extrainfo_into_routerinfo(), networkstatus_check_document_signature(), networkstatus_compute_consensus(), networkstatus_dump_bridge_status_to_file(), networkstatus_format_signatures(), networkstatus_free_all(), networkstatus_get_detached_signatures(), networkstatus_note_certs_arrived(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_set_current_consensus(), networkstatus_v2_free(), networkstatus_v2_list_clean(), networkstatus_vote_free(), normalize_data_directory(), ns_detached_signatures_free(), onion_pending_add(), onion_pending_remove(), onion_skin_client_handshake(), onion_skin_server_handshake(), onionskin_answer(), option_clear(), option_reset(), options_act(), options_act_reversible(), options_dup(), options_init_from_string(), options_init_from_torrc(), options_init_logs(), options_validate(), or_handshake_state_free(), or_state_load(), or_state_save(), or_state_set(), parse_addr_and_port_range(), parse_addr_policy(), parse_addr_port(), parse_bridge_line(), parse_bridge_stats_controller(), parse_dir_server_line(), parse_http_response(), parse_iso_time(), parse_log_severity_config(), parse_port_config(), parse_rfc1123_time(), policy_summarize(), predicted_ports_free(), purge_expired_resolves(), read_bandwidth_usage(), read_file_to_str(), relay_send_command_from_edge(), remove_obsolete_entry_guards(), rend_authorized_client_free(), rend_cache_entry_free(), rend_cache_store(), rend_cache_store_v2_desc_as_client(), rend_cache_store_v2_desc_as_dir(), rend_config_services(), rend_data_free(), rend_decrypt_introduction_points(), rend_desc_v2_is_parsable(), rend_encode_v2_descriptors(), rend_encode_v2_intro_points(), rend_encoded_v2_service_descriptor_free(), rend_encrypt_v2_intro_points_basic(), rend_encrypt_v2_intro_points_stealth(), rend_intro_point_free(), rend_parse_service_authorization(), rend_parse_v2_service_descriptor(), rend_service_authorization_free(), rend_service_descriptor_free(), rend_service_free(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_free_all(), rep_hist_get_predicted_ports(), rep_hist_get_router_stability_doc(), rep_hist_load_mtbf_data(), rep_hist_load_state(), rep_hist_record_mtbf_data(), rep_hist_update_state(), rep_history_clean(), retry_listeners(), rotate_onion_key(), router_append_dirobj_signature(), router_dump_router_to_string(), router_free_all(), router_load_routers_from_string(), router_nickname_is_in_list(), router_rebuild_descriptor(), router_rebuild_store(), router_reload_consensus_networkstatus(), router_reload_router_list_impl(), router_reload_v2_networkstatus(), router_reset_warnings(), router_set_networkstatus_v2(), router_upload_dir_desc_to_dirservers(), routerinfo_free(), routerinfo_incompatible_with_extrainfo(), routerlist_free(), routerlist_free_all(), routerlist_remove_old_cached_routers_with_id(), routerlist_reset_warnings(), routerset_free(), routerset_parse(), routerset_union(), routerstatus_format_entry(), routerstatus_free(), secret_to_key(), set_cached_dir(), signed_desc_append_to_journal(), signed_descriptor_free(), smartlist_choose_by_bandwidth(), smartlist_choose_by_bandwidth_weights(), smartlist_free(), smartlist_string_remove(), spawn_cpuworker(), start_writing_to_file(), strmap_free(), strmap_get_lc(), strmap_iter_next_rmv(), strmap_remove(), strmap_remove_lc(), strmap_set_lc(), tell_controller_about_resolved_result(), tor_addr_from_str(), tor_addr_parse_mask_ports(), tor_addr_port_parse(), tor_event_free(), tor_free_all(), tor_gzip_compress(), tor_gzip_uncompress(), tor_listdir(), tor_lockfile_unlock(), tor_munmap_file(), tor_mutex_free(), tor_pthread_helper_fn(), tor_tls_client_is_using_v2_ciphers(), tor_tls_context_decref(), tor_tls_context_new(), tor_tls_free(), tor_tls_free_all(), tor_tls_new(), tor_tls_set_logged_address(), tor_vasprintf(), tor_version_is_obsolete(), tor_zlib_free(), tor_zlib_new(), trusted_dir_server_free(), trusted_dirs_load_certs_from_string(), trusted_dirs_reload_certs(), try_locking(), unescape_string(), var_cell_free(), vote_routerstatus_free(), and write_configuration_file().

4.55.3 Enumeration Type Documentation

4.55.3.1 enum cpd_check_t

Possible behaviors for **check_private_dir()** (p. 1244) on encountering a nonexistent directory; see that function's documentation for details.

4.55.3.2 enum file_status_t

Return values from **file_status()** (p. 1246); see that function's documentation for details.

4.55.4 Function Documentation

4.55.4.1 void _tor_free (void * *mem*)

Helper for places that need to take a function pointer to the right spelling of "free()".

References tor_free.

Referenced by dir_split_resource_into_fingerprint_pairs(), dirserv_free_fingerprint_list(), dns_reset_correctness_checks(), download_status_map_update_from_v2_networkstatus(), entry_guards_parse_state(), networkstatus_free_all(), ns_detached_signatures_free(), rend_service_free(), routerstatus_list_update_named_server_map(), smartlist_uniq_digests(), smartlist_uniq_digests256(), smartlist_uniq_strings(), sort_version_list(), and tor_main().

4.55.4.2 void* _tor_malloc (size_t size *DMALLOC_PARAMS*)

Allocate a chunk of **size** bytes of memory, and return a pointer to result. On error, log and terminate the process. (Same as malloc(size), but never returns NULL.)

file and **line** are used if dmalloc is enabled, and ignored otherwise.

References LD_MM.

Referenced by _tor_malloc_roundup(), _tor_malloc_zero(), _tor_memdup(), _tor_strndup(), and tor_main().

4.55.4.3 void* _tor_malloc_zero (size_t size *DMALLOC_PARAMS*)

Allocate a chunk of **size** bytes of memory, fill the memory with zero bytes, and return a pointer to the result. Log and terminate the process on error. (Same as calloc(size,1), but never returns NULL.)

References _tor_malloc().

4.55.4.4 void* _tor_memdup (const void * *mem*, size_t len *DMALLOC_PARAMS*)

Allocate a chunk of **len** bytes, with the same contents as the **len** bytes starting at **mem**.

References _tor_malloc(), and tor_assert.

4.55.4.5 void∗ _tor_realloc (void ∗ *ptr*, size_t size *DMALLOC_PARAMS*)

Change the size of the memory block pointed to by **ptr** to **size** bytes long; return the new memory block. On error, log and terminate. (Like realloc(ptr,size), but never returns NULL.)

References LD_MM.

Referenced by tor_main().

4.55.4.6 char∗ _tor_strdup (const char ∗s *DMALLOC_PARAMS*)

Return a newly allocated copy of the NUL-terminated string s. On error, log and terminate. (Like strdup(s), but never returns NULL.)

References LD_MM, and tor_assert.

4.55.4.7 char∗ _tor_strndup (const char ∗ *s*, size_t n *DMALLOC_PARAMS*)

Allocate and return a new string containing the first **n** characters of **s**. If **s** is longer than **n** characters, only the first **n** are copied. The result is always NUL-terminated. (Like strndup(s,n), but never returns NULL.)

References _tor_malloc(), and tor_assert.

4.55.4.8 int abort_writing_to_file (open_file_t ∗ *file_data*)

Finish writing to **file_data**: close the file handle, free memory as needed, and if using a temporary file, delete it.

References finish_writing_to_file_impl().

Referenced by geoip_dirreq_stats_write(), geoip_entry_stats_write(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_record_mtbf_data(), start_writing_to_stdio_file(), and write_chunks_to_file_impl().

4.55.4.9 int append_bytes_to_file (const char ∗ *fname*, const char ∗ *str*, size_t *len*, int *bin*)

As write_bytes_to_file, but if the file already exists, append the bytes to the end of the file instead of overwriting it.

References smartlist_add(), smartlist_create(), smartlist_free(), and write_chunks_to_file_impl().

Referenced by signed_desc_append_to_journal().

4.55.4.10 time_t approx_time (void)

Return a cached estimate of the current time from when **update_approx_time()** (p. 1258) was last called. This is a hack to avoid calling time(NULL) on critical paths: please do not even think of calling it anywhere else.

References cached_approx_time.

Referenced by append_cell_to_circuit_queue(), cell_ewma_get_tick(), circuit_build_times_needs_circuits_now(), circuit_build_times_network_check_live(), circuit_build_-

times_network_is_live(), connection_consider_empty_read_buckets(), connection_consider_empty_write_buckets(), connection_handle_read_impl(), connection_handle_write_impl(), connection_or_flushed_some(), connection_or_get_for_extend(), connection_or_write_cell_to_buf(), connection_or_write_var_cell_to_buf(), connection_read_to_buf(), origin_circuit_new(), and relay_send_command_from_edge().

4.55.4.11 int base16_decode (char ∗ *dest*, size_t *destlen*, const char ∗ *src*, size_t *srclen*)

Given a hexadecimal string of **srclen** bytes in **src**, decode it and store the result in the **destlen**-byte buffer at **dest**. Return 0 on success, -1 on failure.

References _hex_decode_digit().

Referenced by add_fingerprint_to_dir(), authority_cert_parse_from_string(), circuit_get_open_circ_or_launch(), client_likes_consensus(), decode_hashed_passwords(), dir_networkstatus_download_failed(), dir_routerdesc_download_failed(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), dirserv_load_fingerprint_file(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), handle_control_authenticate(), hex_digest_matches(), hexdigest_to_digest(), measured_bw_line_parse(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_bridge_line(), parse_dir_server_line(), rep_hist_load_mtbf_data(), router_fingerprint_is_me(), router_get_by_nickname(), router_get_consensus_status_by_nickname(), router_parse_entry_from_string(), router_reload_v2_networkstatus(), routerset_parse(), and tor_version_parse().

4.55.4.12 void base16_encode (char ∗ *dest*, size_t *destlen*, const char ∗ *src*, size_t *srclen*)

Encode the **srclen** bytes at **src** in a NUL-terminated, uppercase hexadecimal string; store it in the **destlen**-byte buffer **dest**.

References tor_assert.

Referenced by circuit_list_path_impl(), connection_ap_make_link(), connection_dir_client_reached_eof(), connection_or_check_valid_tls_handshake(), connection_or_init_conn_from_address(), control_event_guard(), crypto_pk_get_fingerprint(), directory_get_consensus_url(), dirserv_get_name_status(), dirserv_get_networkstatus_v2(), dirvote_fetch_missing_votes(), do_hash_password(), entry_guard_set_status(), entry_guards_update_state(), extrainfo_dump_to_string(), fetch_bridge_descriptors(), format_networkstatus_vote(), getinfo_helper_entry_guards(), getinfo_helper_misc(), hex_str(), initiate_descriptor_downloads(), list_single_server_status(), networkstatus_add_detached_signatures(), networkstatus_compute_consensus(), networkstatus_format_signatures(), networkstatus_get_cache_filename(), networkstatus_get_detached_signatures(), orconn_target_get_name(), remove_dead_entry_guards(), remove_obsolete_entry_guards(), rend_mid_establish_rendezvous(), rend_mid_rendezvous(), rend_parse_introduction_points(), rend_service_introduce(), rend_service_launch_establish_intro(), rend_service_rendezvous_has_opened(), rep_hist_dump_stats(), rep_hist_record_mtbf_data(), router_dump_router_to_string(), router_get_verbose_nickname(), router_load_extrainfo_from_string(), router_load_routers_from_string(), router_rebuild_descriptor(), router_set_networkstatus_v2(), routerstatus_format_entry(), routerstatus_get_verbose_nickname(), and update_v2_networkstatus_cache_downloads().

4.55.4.13 int check_private_dir (const char * *dirname*, cpd_check_t *check*)

Check whether dirname exists and is private. If yes return 0. If it does not exist, and check==CPD_CREATE is set, try to create it and return 0 on success. If it does not exist, and check==CPD_CHECK, and we think we can create it, return 0. Else return -1.

References clean_name_for_stat(), LD_FS, LD_GENERAL, tor_assert, and tor_free.

Referenced by geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), init_keys(), load_bridge_stats(), options_act_reversible(), rend_service_load_keys(), rep_hist_buffer_stats_write(), and rep_hist_exit_stats_write().

4.55.4.14 const char* eat_whitespace (const char * *s*)

Return a pointer to the first char of s that is not whitespace and not a comment, or to the terminating NUL if no such character exists.

References tor_assert.

Referenced by authority_cert_parse_from_string(), config_parse_units(), get_token_arguments(), parse_log_severity_config(), rend_parse_client_keys(), router_parse_addr_policy_item_from_string(), router_parse_addr_policy_private(), tor_addr_port_parse(), tor_version_as_new_as(), and tor_version_parse().

4.55.4.15 const char* eat_whitespace_eos (const char * *s*, const char * *eos*)

Return a pointer to the first char of s that is not whitespace and not a comment, or to the terminating NUL if no such character exists.

References tor_assert.

Referenced by find_start_of_next_microdesc(), find_start_of_next_router_or_extrainfo(), get_next_token(), microdescs_parse_from_string(), and tokenize_string().

4.55.4.16 const char* eat_whitespace_eos_no_nl (const char * *s*, const char * *eos*)

As eat_whitespace_no_nl, but stop at **eos** whether we have found a non-whitespace character or not.

Referenced by get_next_token().

4.55.4.17 const char* eat_whitespace_no_nl (const char * *s*)

Return a pointer to the first char of s that is not a space or a tab or a \r, or to the terminating NUL if no such character exists.

Referenced by parse_bridge_stats_controller(), and parse_http_url().

4.55.4.18 char* esc_for_log (const char * *s*)

Allocate and return a new string representing the contents of **s**, surrounded by quotes and using standard C escapes.

Generally, we use this for logging values that come in over the network to keep them from tricking users, and for sending certain values to the controller.

We trust values from the resolver, OS, configuration file, and command line to not be maliciously ill-formed. We validate incoming routerdescs and SOCKS requests and addresses from BEGIN cells as they're parsed; afterwards, we trust them as non-malicious.

References tor_snprintf().

Referenced by control_event_logmsg(), dirserv_get_status_impl(), esc_router_info(), escaped(), evdns_callback(), get_assigned_option(), handle_control_protocolinfo(), parse_addr_port(), parse_iso_time(), and parse_rfc1123_time().

4.55.4.19 const char∗ escaped (const char ∗ *s*)

Allocate and return a new string representing the contents of **s**, surrounded by quotes and using standard C escapes.

THIS FUNCTION IS NOT REENTRANT. Don't call it from outside the main thread. Also, each call invalidates the last-returned value, so don't try log_warn(LD_GENERAL, "%s %s", escaped(a), escaped(b));

References esc_for_log(), and tor_free.

Referenced by add_fingerprint_to_dir(), authority_cert_parse_from_string(), circuit_build_failed(), command_process_create_cell(), connection_ap_process_natd(), connection_dir_client_reached_eof(), connection_read_https_proxy_response(), connection_read_proxy_handshake(), dir_networkstatus_download_failed(), dir_routerdesc_download_failed(), dir_split_resource_into_fingerprint_pairs(), dir_split_resource_into_fingerprints(), directory_handle_command(), dirserv_add_multiple_descriptors(), dirserv_get_status_impl(), dirserv_read_measured_bandwidths(), entry_guards_parse_state(), escaped_safe_str(), escaped_safe_str_client(), evdns_log_cb(), extrainfo_parse_entry_from_string(), fetch_from_buf_socks(), geoip_parse_entry(), handle_control_protocolinfo(), http_set_address_origin(), init_cookie_authentication(), load_downloaded_routers(), measured_bw_line_parse(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_addr_and_port_range(), parse_addr_port(), parse_http_response(), parse_port_config(), parse_port_range(), remove_obsolete_entry_guards(), rend_parse_client_keys(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rend_service_introduce(), rend_service_load_keys(), rep_hist_load_mtbf_data(), router_dump_router_to_string(), router_get_consensus_status_by_nickname(), router_new_address_suggestion(), router_parse_addr_policy(), router_parse_addr_policy_item_from_string(), router_parse_entry_from_string(), router_rebuild_descriptor(), routerstatus_parse_entry_from_string(), signed_descriptor_get_body_impl(), tor_addr_parse_mask_ports(), and tor_free_all().

4.55.4.20 char∗ expand_filename (const char ∗ *filename*)

Expand any homedir prefix on **filename**; return a newly allocated string.

References get_user_homedir(), LD_CONFIG, strcmpend(), tor_assert, tor_free, and tor_snprintf().

Referenced by find_torrc_filename(), and normalize_data_directory().

4.55.4.21 FILE∗ fdopen_file (open_file_t ∗ *file_ data*)

Given **file_data** from **start_writing_to_file()** (p. 1252), return a stdio FILE∗ that can be used to write to the same file. The caller should not mix stdio calls with non-stdio calls.

References open_file_t::binary, open_file_t::fd, open_file_t::filename, LD_FS, open_file_t::stdio_file, and tor_assert.

Referenced by start_writing_to_stdio_file().

4.55.4.22 file_status_t file_status (const char ∗ *fname*)

Return FN_ERROR if filename can't be read, FN_NOENT if it doesn't exist, FN_FILE if it is a regular file, or FN_DIR if it's a directory. On FN_ERROR, sets errno.

References clean_name_for_stat(), and tor_free.

Referenced by dirserv_clear_old_networkstatuses(), find_torrc_filename(), init_key_from_file(), init_keys(), load_stats_file(), load_torrc_from_disk(), networkstatus_v2_list_clean(), or_state_load(), replace_file(), rotate_onion_key(), router_reload_router_list_impl(), and write_configuration_file().

4.55.4.23 const char∗ find_str_at_start_of_line (const char ∗ *haystack*, const char ∗ *needle*)

Return the first occurrence of **needle** in **haystack** that occurs at the start of a line (that is, at the beginning of **haystack** or immediately after a newline). Return NULL if no such string is found.

Referenced by parse_bridge_stats_controller().

4.55.4.24 const char∗ find_whitespace (const char ∗ *s*)

Return a pointer to the first char of s that is whitespace or #, or to the terminating NUL if no such character exists.

Referenced by get_token_arguments(), parse_http_url(), tor_version_as_new_as(), tor_version_parse(), and version_from_platform().

4.55.4.25 const char∗ find_whitespace_eos (const char ∗ *s*, const char ∗ *eos*)

As find_whitespace, but stop at **eos** whether we have found a whitespace or not.

Referenced by get_next_token().

4.55.4.26 void finish_daemon (const char ∗ *desired_ cwd*)

Finish putting the process into daemon mode: drop standard fds, and tell the parent process to exit. (Note: it's safe to call this more than once: calls after the first are ignored. Calls start_daemon first if it hasn't been called already.)

References daemon_filedes, finish_daemon_called, LD_GENERAL, start_daemon(), and start_daemon_called.

Referenced by options_act().

4.55.4.27 int finish_writing_to_file (open_file_t * *file_data*)

Finish writing to **file_data**: close the file handle, free memory as needed, and if using a temporary file, replace the original file with the temporary file.

References finish_writing_to_file_impl().

Referenced by geoip_dirreq_stats_write(), geoip_entry_stats_write(), microdesc_cache_rebuild(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_record_mtbf_data(), and write_chunks_to_file_impl().

4.55.4.28 void format_iso_time (char * *buf*, time_t *t*)

Set **buf** to the ISO8601 encoding of the GMT value of **t**. The buffer must be at least ISO_TIME_LEN+1 bytes long.

Referenced by accounting_set_wakeup_time(), addressmap_get_mappings(), control_event_address_mapped(), dirserv_add_multiple_descriptors(), dirserv_dump_directory_to_string(), dirvote_add_vote(), dirvote_recalculate_timing(), entry_guards_update_state(), extrainfo_dump_to_string(), format_networkstatus_vote(), generate_runningrouters(), generate_v2_networkstatus_opinion(), geoip_bridge_stats_write(), geoip_dirreq_stats_write(), geoip_entry_stats_write(), get_assigned_option(), getinfo_helper_accounting(), getinfo_helper_entry_guards(), hibernate_end_time_elapsed(), load_downloaded_routers(), networkstatus_compute_consensus(), networkstatus_get_detached_signatures(), networkstatus_set_current_consensus(), parse_bridge_stats_controller(), read_bandwidth_usage(), rend_encode_v2_descriptors(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), rep_hist_format_router_status(), rep_hist_get_bandwidth_lines(), rep_hist_get_router_stability_doc(), rep_hist_record_mtbf_data(), router_dump_router_to_string(), router_set_networkstatus_v2(), and routerstatus_format_entry().

4.55.4.29 void format_local_iso_time (char * *buf*, time_t *t*)

Set **buf** to the ISO8601 encoding of the local value of **t**. The buffer must be at least ISO_TIME_LEN+1 bytes long.

(ISO8601 format is 2006-10-29 10:57:20)

Referenced by accounting_set_wakeup_time(), consider_hibernation(), control_event_address_mapped(), or_state_save(), remove_dead_entry_guards(), remove_file_if_very_old(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), and update_consensus_networkstatus_fetch_time().

4.55.4.30 void format_rfc1123_time (char * *buf*, time_t *t*)

Set **buf** to the RFC1123 encoding of the GMT value of **t**. The buffer must be at least RFC1123_TIME_LEN+1 bytes long.

(RFC1123 format is Fri, 29 Sep 2006 15:54:20 GMT)

References MONTH_NAMES, tor_assert, and WEEKDAY_NAMES.

Referenced by directory_send_command(), and write_http_response_header_impl().

4.55.4.31 int format_time_interval (char * *out*, size_t *out_len*, long *interval*)

Given an **interval** in seconds, try to write it to the **out_len**-byte buffer in **out** in a human-readable form. Return 0 on success, -1 on failure.

References tor_snprintf().

Referenced by command_process_netinfo_cell(), connection_dir_client_reached_eof(), networkstatus_set_current_consensus(), and router_set_networkstatus_v2().

4.55.4.32 int ftime_definitely_after (time_t *now*, time_t *when*)

Return true if we think that **now** is definitely after **when**.

References ftime_skew, and ftime_slop.

Referenced by authority_certs_fetch_missing(), and trusted_dirs_remove_old_certs().

4.55.4.33 int ftime_definitely_before (time_t *now*, time_t *when*)

Return true if we think that **now** is definitely before **when**.

References ftime_skew, and ftime_slop.

Referenced by networkstatus_set_current_consensus().

4.55.4.34 int ftime_maybe_after (time_t *now*, time_t *when*)

Return true iff we think that **now** might be after **when**.

References ftime_skew, and ftime_slop.

4.55.4.35 int ftime_maybe_before (time_t *now*, time_t *when*)

Return true iff we think that **now** might be before **when**.

References ftime_skew, and ftime_slop.

4.55.4.36 void ftime_set_estimated_skew (int *seconds*)

Set the amount by which we believe our system clock to differ from real time.

References ftime_skew.

4.55.4.37 void ftime_set_maximum_sloppiness (int *seconds*)

Set the largest amount of sloppiness we'll allow in fuzzy time comparisons.

References ftime_slop, and tor_assert.

4.55.4.38 int hex_decode_digit (char *c*)

Helper: given a hex digit, return its value, or -1 if it isn't hex.

References _hex_decode_digit().

Referenced by tor_addr_parse_reverse_lookup_name(), and unescape_string().

4.55.4.39 const char∗ hex_str (const char ∗ *from*, size_t *fromlen*)

Return a pointer to a NUL-terminated hexadecimal string encoding the first **fromlen** bytes of **from**. (fromlen must be <= 32.) The result does not need to be deallocated, but repeated calls to hex_str will trash old results.

References base16_encode().

Referenced by authority_certs_fetch_missing(), dirvote_act(), dirvote_add_vote(), handle_control_attachstream(), list_v3_auth_ids(), networkstatus_check_consensus_signature(), rep_hist_note_router_reachable(), rep_hist_note_router_unreachable(), router_add_to_routerlist(), router_load_routers_from_string(), router_parse_directory(), and trusted_dirs_load_certs_from_string().

4.55.4.40 int int int int int int memcmpstart (const void ∗ *mem*, size_t *memlen*, const char ∗ *prefix*)

Compare the value of the string **prefix** with the start of the **memlen**-byte memory chunk at **mem**. Return as for strcmp.

[As memcmp(mem, prefix, strlen(prefix)) but returns -1 if memlen is less than strlen(prefix).]

Referenced by rend_decrypt_introduction_points(), and rend_parse_introduction_points().

4.55.4.41 const char∗ parse_config_line_from_str (const char ∗ *line*, char ∗∗ *key_out*, char ∗∗ *value_out*)

Given a string containing part of a configuration file or similar format, advance past comments and whitespace and try to parse a single line. If we parse a line successfully, set ∗**key_out** to a new string holding the key portion and ∗**value_out** to a new string holding the value portion of the line, and return a pointer to the start of the next line. If we run out of data, return a pointer to the end of the string. If we encounter an error, return NULL.

References tor_assert, and unescape_string().

Referenced by config_get_lines().

4.55.4.42 int parse_http_time (const char ∗ *date*, struct tm ∗ *tm*)

Given a **date** in one of the three formats allowed by HTTP (ugh), parse it into **tm**. Return 0 on success, negative on failure.

References MONTH_NAMES, tor_assert, and tor_sscanf().

Referenced by directory_handle_command_get().

4.55.4.43 int parse_iso_time (const char ∗ *cp*, time_t ∗ *t*)

Given an ISO-formatted UTC time value (after the epoch) in **cp**, parse it and store its value in ∗**t**. Return 0 on success, -1 on failure. Ignore extraneous stuff in **cp** separated by whitespace from the end of the time string.

References esc_for_log(), LD_GENERAL, tor_free, tor_sscanf(), and tor_timegm().

Referenced by authority_cert_parse_from_string(), config_assign_value(), entry_guards_parse_state(), extrainfo_parse_entry_from_string(), load_stats_file(), microdescs_parse_from_string(), networkstatus_parse_detached_signatures(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_bridge_stats_controller(), parse_possibly_bad_iso_time(), rend_parse_v2_service_descriptor(), rep_hist_load_mtbf_data(), router_parse_directory(), router_parse_entry_from_string(), router_parse_runningrouters(), and routerstatus_parse_entry_from_string().

4.55.4.44 int parse_rfc1123_time (const char * *buf*, time_t * *t*)

Parse the RFC1123 encoding of some time (in GMT) from **buf**, and store the result in ***t**.

Return 0 on success, -1 on failure.

References esc_for_log(), LD_GENERAL, MONTH_NAMES, tor_free, tor_sscanf(), and tor_timegm().

Referenced by parse_http_response().

4.55.4.45 int path_is_relative (const char * *filename*)

Return true iff **filename** is a relative path.

Referenced by options_validate().

4.55.4.46 ssize_t read_all (int *fd*, char * *buf*, size_t *count*, int *isSocket*)

Read from **fd** to **buf**, until we get **count** bytes or reach the end of the file. **isSocket** must be 1 if fd was returned by socket() or accept(), and 0 if fd was returned by open(). Return the number of bytes read, or -1 on error. Only use if fd is a blocking fd.

Referenced by cpuworker_main(), crypto_seed_rng(), and read_file_to_str().

4.55.4.47 char* read_file_to_str (const char * *filename*, int *flags*, struct stat * *stat_out*)

Read the contents of **filename** into a newly allocated string; return the string on success or NULL on failure.

If **stat_out** is provided, store the result of stat()ing the file into **stat_out**.

If **flags** & RFTS_BIN, open the file in binary mode. If **flags** & RFTS_IGNORE_MISSING, don't warn if the file doesn't exist.

References LD_FS, read_all(), RFTS_BIN, RFTS_IGNORE_MISSING, tor_assert, tor_free, and tor_strstrip().

Referenced by crypto_pk_read_private_key_from_filename(), dirserv_load_fingerprint_file(), init_keys(), load_authority_keyset(), load_bridge_stats(), load_stats_file(), load_torrc_from_disk(), microdesc_cache_reload(), options_act(), or_state_load(), rend_service_load_keys(), rep_hist_load_mtbf_data(), router_reload_consensus_networkstatus(), router_reload_router_list_impl(), router_reload_v2_networkstatus(), trusted_dirs_reload_certs(), and write_configuration_file().

4.55.4.48 unsigned round_to_next_multiple_of (unsigned *number*, unsigned *divisor*)

Return the lowest x such that x is at least **number**, and x modulo **divisor** == 0.

Referenced by geoip_get_client_history(), and geoip_get_request_history().

4.55.4.49 uint64_t round_to_power_of_2 (uint64_t *u64*)

Return the power of 2 closest to **u64**.

References tor_log2().

Referenced by mp_pool_new().

4.55.4.50 uint32_t round_uint32_to_next_multiple_of (uint32_t *number*, uint32_t *divisor*)

Return the lowest x such that x is at least **number**, and x modulo **divisor** == 0.

Referenced by geoip_dirreq_stats_write(), geoip_get_dirreq_history(), and rep_hist_exit_stats_write().

4.55.4.51 uint64_t round_uint64_to_next_multiple_of (uint64_t *number*, uint64_t *divisor*)

Return the lowest x such that x is at least **number**, and x modulo **divisor** == 0.

Referenced by rep_hist_exit_stats_write().

4.55.4.52 void start_daemon (void)

Start putting the process into daemon mode: fork and drop all resources except standard fds. The parent process never returns, but stays around until finish_daemon is called. (Note: it's safe to call this more than once: calls after the first are ignored.)

References daemon_filedes, LD_GENERAL, set_main_thread(), and start_daemon_called.

Referenced by finish_daemon(), and options_act_reversible().

4.55.4.53 int start_writing_to_file (const char * *fname*, int *open_flags*, int *mode*, open_file_t ** *data_out*)

Try to start writing to the file in **fname**, passing the flags **open_flags** to the open() syscall, creating the file (if needed) with access value **mode**. If the O_APPEND flag is set, we append to the original file. Otherwise, we open a new temporary file in the same directory, and either replace the original or remove the temporary file when we're done.

Return the fd for the newly opened file, and store working data in ***data_out**. The caller should not close the fd manually: instead, call **finish_writing_to_file()** (p. 1247) or **abort_writing_to_file()** (p. 1242). Returns -1 on failure.

NOTE: When not appending, the flags O_CREAT and O_TRUNC are treated as true and the flag O_EXCL is treated as false.

NOTE: Ordinarily, O_APPEND means "seek to the end of the file before each write()". We don't do that.

References open_file_t::binary, open_file_t::fd, open_file_t::filename, LD_FS, LD_GENERAL, open_file_t::rename_on_close, open_file_t::tempname, tor_assert, tor_fd_seekend(), tor_free, and tor_snprintf().

Referenced by start_writing_to_stdio_file(), and write_chunks_to_file_impl().

4.55.4.54 FILE∗ start_writing_to_stdio_file (const char ∗ *fname*, int *open_flags*, int *mode*, open_file_t ∗∗ *data_out*)

Combines start_writing_to_file with **fdopen_file()** (p. 1246): arguments are as for start_writing_to_file, but

References abort_writing_to_file(), fdopen_file(), and start_writing_to_file().

Referenced by geoip_dirreq_stats_write(), geoip_entry_stats_write(), microdesc_cache_rebuild(), rend_service_load_keys(), rep_hist_buffer_stats_write(), rep_hist_exit_stats_write(), and rep_hist_record_mtbf_data().

4.55.4.55 int tor_digest256_is_zero (const char ∗ *digest*)

Return true iff the DIGEST256_LEN bytes in digest are all zero.

References DIGEST256_LEN, and tor_mem_is_zero().

4.55.4.56 int tor_digest_is_zero (const char ∗ *digest*)

Return true iff the DIGEST_LEN bytes in digest are all zero.

References DIGEST_LEN, and tor_mem_is_zero().

Referenced by _connection_free(), circuit_extend(), circuit_get_all_pending_on_or_conn(), circuit_is_acceptable(), circuit_n_conn_done(), connection_about_to_close_connection(), connection_ap_fail_onehop(), connection_free(), connection_or_check_valid_tls_handshake(), connection_or_remove_from_identity_map(), connection_or_set_identity_digest(), connection_unlink(), fetch_bridge_descriptors(), format_networkstatus_vote(), handle_control_attachstream(), list_v3_auth_ids(), lookup_cached_dir_by_fp(), networkstatus_compute_consensus(), networkstatus_parse_vote_from_string(), orconn_target_get_name(), routerinfo_get_configured_bridge(), routerlist_insert(), routerlist_insert_old(), routerlist_remove(), routerlist_remove_old(), routerlist_replace(), and update_extrainfo_downloads().

4.55.4.57 struct smartlist_t∗ tor_listdir (const char ∗ *dirname*) `[read]`

Return a new list containing the filenames in the directory **dirname**. Return NULL on error or if **dirname** is not a directory.

References LD_FS, smartlist_add(), smartlist_create(), tor_free, and tor_snprintf().

Referenced by router_reload_v2_networkstatus().

4.55.4.58 int tor_log2 (uint64_t *u64*)

Returns floor(log2(u64)). If u64 is 0, (incorrectly) returns 0.

Referenced by digestset_new(), and round_to_power_of_2().

4.55.4.59 void tor_log_mallinfo (int *severity*)

Call the platform malloc info function, and dump the results to the log at level **severity**. If no such function exists, do nothing.

References LD_MM, and tor_log().

Referenced by dumpmemusage().

4.55.4.60 long tor_lround (double *d*)

Return the long integer closest to d. We define this wrapper here so that not all users of math.h need to use the right incancations to get the c99 functions.

Referenced by circuit_build_times_add_timeout_worker(), circuit_build_times_generate_sample(), circuit_build_times_network_check_changed(), circuit_build_times_network_check_live(), and circuit_build_times_set_timeout().

4.55.4.61 double tor_mathlog (double *d*)

Returns the natural logarithm of d base 2. We define this wrapper here so as to make it easier not to conflict with Tor's log() macro.

Referenced by circuit_build_times_initial_alpha(), and circuit_build_times_update_alpha().

4.55.4.62 int tor_mem_is_zero (const char * *mem*, size_t *len*)

Return true iff the 'len' bytes at 'mem' are all zero.

Referenced by check_sockaddr(), get_link_history(), get_or_history(), networkstatus_add_detached_signatures(), networkstatus_get_detached_signatures(), networkstatus_parse_detached_signatures(), rend_cache_store_v2_desc_as_client(), tor_digest256_is_zero(), and tor_digest_is_zero().

4.55.4.63 double tor_parse_double (const char * *s*, double *min*, double *max*, int * *ok*, char ** *next*)

As **tor_parse_long()** (p. 1255), but return a double.

Referenced by config_parse_units().

4.55.4.64 void long tor_parse_long (const char * *s*, int *base*, long *min*, long *max*, int * *ok*, char ** *next*)

Extract a long from the start of s, in the given numeric base. If there is unconverted data and next is provided, set *next to the first unconverted character. An error has occurred if no characters are converted; or if there are unconverted characters and next is NULL; or if the parsed value is not between min and max. When no error occurs, return the parsed value and set *ok (if provided) to 1. When an error occurs, return 0 and set *ok (if provided) to 0.

Referenced by _cmp_int_strings(), accounting_parse_options(), compute_consensus_method(), config_assign_value(), connection_ap_process_natd(), dirvote_compute_params(), handle_control_protocolinfo(), measured_bw_line_parse(), networkstatus_parse_vote_from_string(), networkstatus_v2_parse_from_string(), parse_addr_port(), parse_dir_server_line(), parse_port_config(), parse_port_range(), parse_possibly_bad_iso_time(), rend_parse_introduction_points(), rend_parse_v2_service_descriptor(), rep_hist_load_mtbf_data(), router_parse_entry_from_string(), routerstatus_parse_entry_from_string(), and tor_addr_port_parse().

4.55.4.65 uint64_t tor_parse_uint64 (const char * *s*, int *base*, uint64_t *min*, uint64_t *max*, int * *ok*, char ** *next*)

As tor_parse_long, but return a uint64_t. Only base 10 is guaranteed to work for now.

References tor_assert.

Referenced by config_parse_units(), get_stream(), and rep_hist_load_state().

4.55.4.66 unsigned long tor_parse_ulong (const char * *s*, int *base*, unsigned long *min*, unsigned long *max*, int * *ok*, char ** *next*)

As **tor_parse_long()** (p. 1255), but return an unsigned long.

Referenced by circuit_build_times_parse_state(), dirserv_read_measured_bandwidths(), get_circ(), handle_control_attachstream(), handle_control_closestream(), handle_control_redirectstream(), and routerstatus_parse_entry_from_string().

4.55.4.67 int tor_sscanf (const char * *buf*, const char * *pattern*, *...*)

Minimal sscanf replacement: parse **buf** according to **pattern** and store the results in the corresponding argument fields. Differs from sscanf in that it: Only handles u and Ns. Does not handle arbitrarily long widths. u does not consume any space. Is locale-independent. Returns -1 on malformed patterns.

(As with other locale-independent functions, we need this to parse data that is in ASCII without worrying that the C library's locale-handling will make miscellaneous characters look like numbers, spaces, and so on.)

References tor_vsscanf().

Referenced by parse_http_response(), parse_http_time(), parse_iso_time(), parse_rfc1123_time(), tor_inet_aton(), and tor_inet_pton().

4.55.4.68 int tor_strisnonupper (const char * *s*)

Return 1 if no character in **s** is uppercase, else return 0.

Referenced by assert_resolve_ok().

4.55.4.69 int tor_strisprint (const char * *s*)

Return 1 if every character in **s** is printable, else return 0.

Referenced by fetch_from_buf_socks().

4.55.4.70 void tor_strlower (char * *s*)

Convert all alphabetic characters in the nul-terminated string **s** to lowercase.

Referenced by clear_trackexithost_mappings(), connection_ap_handshake_rewrite_and_attach(), connection_exit_begin_conn(), dns_resolve_impl(), geoip_add_entry(), get_default_nickname(), handle_control_mapaddress(), router_parse_addr_policy_item_from_string(), routerset_get_countryname(), strmap_get_lc(), strmap_remove_lc(), and strmap_set_lc().

4.55.4.71 void tor_strupper (char * *s*)

Convert all alphabetic characters in the nul-terminated string **s** to lowercase.

4.55.4.72 time_t tor_timegm (struct tm * *tm*)

Return a time_t given a struct tm. The result is given in GMT, and does not account for leap seconds.

References days_per_month, IS_LEAPYEAR, LD_BUG, n_leapdays(), and tor_assert.

Referenced by directory_handle_command_get(), dirvote_get_start_of_next_interval(), parse_iso_time(), and parse_rfc1123_time().

4.55.4.73 int tor_vsscanf (const char * *buf*, const char * *pattern*, va_list *ap*)

Locale-independent, minimal, no-surprises scanf variant, accepting only a restricted pattern format. For more info on what it supports, see **tor_sscanf()** (p. 1256) documentation.

References digit_to_num(), scan_string(), and scan_unsigned().

Referenced by tor_sscanf().

4.55.4.74 long tv_mdiff (const struct timeval * *start*, const struct timeval * *end*)

Return the number of milliseconds elapsed between *start and *end.

References LD_GENERAL.

Referenced by circuit_send_next_onion_skin(), and geoip_get_dirreq_history().

4.55.4.75 double tv_to_double (const struct timeval * *tv*)

Converts struct timeval to a double value. Preserves microsecond precision, but just barely. Error is approx +/- 0.1 usec when dealing with epoch values.

4.55.4.76 int64_t tv_to_msec (const struct timeval * *tv*)

Converts timeval to milliseconds.

4.55.4.77 int64_t tv_to_usec (const struct timeval ∗ *tv*)

Converts timeval to microseconds.

4.55.4.78 long tv_udiff (const struct timeval ∗ *start*, const struct timeval ∗ *end*)

Return the number of microseconds elapsed between ∗start and ∗end.

References LD_GENERAL.

4.55.4.79 void update_approx_time (time_t *now*)

Update the cached estimate of the current time. This function SHOULD be called once per second, and MUST be called before the first call to get_approx_time.

References cached_approx_time.

Referenced by do_main_loop(), second_elapsed_callback(), and tor_main().

4.55.4.80 void wrap_string (smartlist_t ∗ *out*, const char ∗ *string*, size_t *width*, const char ∗ *prefix0*, const char ∗ *prefixRest*)

Rudimentary string wrapping code: given a un-wrapped **string** (no newlines!), break the string into newline-terminated lines of no more than **width** characters long (not counting newline) and insert them into **out** in order. Precede the first line with prefix0, and subsequent lines with prefixRest.

References smartlist_add(), and tor_assert.

4.55.4.81 ssize_t write_all (int *fd*, const char ∗ *buf*, size_t *count*, int *isSocket*)

Write **count** bytes from **buf** to **fd**. **isSocket** must be 1 if fd was returned by socket() or accept(), and 0 if fd was returned by open(). Return the number of bytes written, or -1 on error. Only use if fd is a blocking fd.

References tor_assert.

Referenced by cpuworker_main(), log_tor_version(), logv(), and write_chunks_to_file_impl().

4.55.4.82 int write_bytes_to_file (const char ∗ *fname*, const char ∗ *str*, size_t *len*, int *bin*)

As write_str_to_file, but does not assume a NUL-terminated string. Instead, we write **len** bytes, starting at **str**.

References smartlist_add(), smartlist_create(), smartlist_free(), and write_chunks_to_file_impl().

Referenced by init_cookie_authentication(), and write_str_to_file().

4.55.4.83 void write_pidfile (char ∗ *filename*)

Write the current process ID, followed by NL, into **filename**.

References LD_FS.

Referenced by options_act().

4.55.4.84 int write_str_to_file (const char * *fname*, const char * *str*, int *bin*)

Create a file named **fname** with the contents **str**. Overwrite the previous **fname** if possible. Return 0 on success, -1 on failure.

This function replaces the old file atomically, if possible. This function, and all other functions in **util.c** (p. 1237) that create files, create them with mode 0600.

References LD_BUG, and write_bytes_to_file().

Referenced by add_networkstatus_to_cache(), crypto_pk_write_private_key_to_filename(), dump_desc(), geoip_bridge_stats_write(), init_keys(), microdesc_cache_rebuild(), networkstatus_dump_bridge_status_to_file(), networkstatus_set_current_consensus(), or_state_save(), rend_service_load_keys(), and write_configuration_file().