

Tor Hidden Service Performance Analysis

Roger Dingledine, Karsten Loesing, Steven J. Murdoch,
and Christian Wilms*

June 15, 2008

Abstract

This document describes a performance analysis of setting up and accessing a Tor Hidden Service. The objective is to spot performance bottlenecks in the hidden service protocol and suggest which parts should be changed in order to improve the overall performance of Tor Hidden Services.

1 Motivation

Tor Hidden Services allow users to set up anonymous information services, like websites, that can only be accessed through the Tor network and are protected against identification of the host that runs the services. The most critical limitations of Tor Hidden Services are the time it takes until a hidden service is registered in the network and the latency of contact establishment when accessed by a user. Due to design issues in the original Tor protocol, the connection to a new hidden service can take several minutes, which leads most users to give up before the connection has been established. Using Tor Hidden Services for direct interactive user-to-user communication (e.g. messaging) is nearly impossible due to the high latency of hidden service circuit setup.

This document describes measurements of setting up and accessing a Tor Hidden Service. The first part of these measurements consists of an *external view* of a user experiencing the delay in publishing a hidden service, establishing a connection to an existing hidden service, sending and receiving messages, and the durability of an established connection. The intention of the second part of these measurements is to obtain an *internal view* of the delay that is introduced by single substeps of connection establishing. The following questions shall be answered:

Service Publication How long does it take from starting a Tor process that is configured to provide a hidden service until the hidden service is advertised in the network and accessible by clients?

*Please direct questions and comments either to tor-assistants@torproject.org or or-dev@freehaven.net.

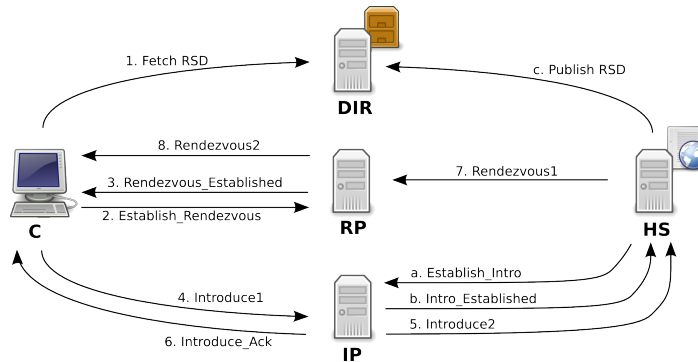


Figure 1: Overview of hidden service establishment and access

Connection Establishment How long does it take for a client to open a connection to an existing hidden service? As this includes multiple substeps, there are two distinct measurements to answer this question: one measuring the overall connection establishment time and one measuring the 11 internal substeps that are involved in this process.

Message Transfer What is the delay of transferring messages from client to server and back over an established connection?

Connection Durability How long do connections persist before breaking?

2 Hidden Service Protocol

Before going into the details of measuring performance of setting up and accessing a hidden service, a brief description of the protocol is given. This is required for understanding the timings of internal processes and explaining possible delays. Figure 1 shows an overview of exchanged messages.

Consider a user called Bob who wants to offer a location-hidden service. All Bob needs to do to establish a hidden service is to set up the actual service, e.g. a website, and start a Tor client, which is configured to provide a hidden service. At startup the Tor client builds circuits to three randomly chosen Tor relays, which will act as *introduction points* for the service. As soon as a circuit is built the hidden server sends an `ESTABLISH_INTRO` cell over the circuit, containing the public key of the hidden server. The introduction point acknowledges by sending an `INTRO_ESTABLISHED` cell. After receiving acknowledgments the hidden server builds the *rendezvous service descriptor*. This descriptor includes contact information of the established introduction points and the hidden service’s public key. The hidden server uploads the descriptor and the hash value of the public key as a unique service identifier, the onion address, to the hidden service directory.

If a user called Alice wants to access a hidden service, she must have learned the onion address before out-of-band. First the Tor client needs to know the rendezvous service descriptor to contact the hidden server. To retrieve the descriptor including the contact information of the introduction points the client sends a request to the hidden service directory, using an anonymous 3-hop circuit. After receiving the rendezvous service descriptor the Tor client randomly picks one of the introduction points it finds in the descriptor and builds a circuit to the introduction point. To accelerate the circuit building, so-called *cannibalization* can be used. That means, that an existing 3-hop circuit is extended to a desired relay, in this case the introduction point. Obviously this only works, if there is a usable circuit available to be cannibalized. Otherwise this circuit first needs to be built hop by hop.

Simultaneously Alice's Tor client selects another relay to act as *rendezvous point*. To establish a circuit to the rendezvous point, again cannibalization is used. But this time no 3-hop circuit is extended to the desired relay, but the third hop of an existing 3-hop circuit is chosen as rendezvous point. The advantage is that the rendezvous circuit is immediately built and Alice's Tor client can send an `ESTABLISH_RENDEZVOUS` cell over the circuit. This cell contains a random rendezvous cookie for identification purposes. The rendezvous point saves the rendezvous cookie and acknowledges its new functionality by replying with a `RENDEZVOUS_ESTABLISHED` cell.

If the introduction circuit is successfully opened and the rendezvous point has replied with the acknowledgment cell, Alice's client sends an `INTRODUCE1` cell over the introduction circuit. This cell contains contact information of the rendezvous point and the rendezvous cookie. The introduction point receives the cell and forwards the content in an `INTRODUCE2` cell to Bob's Tor client. Afterwards it sends an acknowledgment back to the client, the `INTRODUCE_ACK` cell. The hidden service uses the contact information found in the `INTRODUCE2` cell to contact the rendezvous point. First it builds a circuit to the relay acting as rendezvous point using cannibalization again. In this case an existing 3-hop circuit is extended to the rendezvous point. When the circuit is open, a `RENDEZVOUS1` cell is sent down the circuit, including the rendezvous cookie. Upon receiving the `RENDEZVOUS1` cell the rendezvous point uses the rendezvous cookie to find the matching client circuit and connects it with the one it just received the cell over. At last the rendezvous point sends a `RENDEZVOUS2` cell to the client to notify it that a connection to the hidden service has been successfully established and is now ready to transfer user data.

3 Measurement Setup

3.1 External View Measurements

The first set of measurements aims at the user experience of a hidden service provider or client requesting an existing hidden service. The setup consists of two Tor clients, one of them being the *server-side Tor process* that is configured

to provide a hidden service and the other one being the *client-side Tor process* that is used to access the hidden service. 15 minutes after starting the two Tor processes, two more processes are started: a *server process* that acts as hidden server and listens for requests from the server-side Tor process and a *client process* that uses the client-side Tor process to establish a connection to the hidden service. The client process, after establishing the connection to the server process via Tor, sends numbered echo messages every minute until the connection breaks (or four hours elapse). The server process echoes all received messages back to the client. Both Tor processes as well as client and server process run on the same physical machine and log their actions to local files.

Service Publication The process of service publication can best be analyzed by evaluating Tor's log statements:

1. Jun 03 20:50:02.100 [notice] Tor 0.2.1.0-alpha-dev (r14739) opening new log file.
2. Jun 03 20:50:11.151 [notice] We now have enough directory information to build circuits.
3. Jun 03 20:50:12.697 [info] rend_services_introduce(): Giving up on sabotage as intro point for stuptdu2qait65zm.
4. Jun 03 20:50:18.633 [info] rend_service_intro_established(): Received INTRO_ESTABLISHED cell on circuit 1560 for service stuptdu2qait65zm
5. Jun 03 20:51:18.997 [info] upload_service_descriptor(): Sending publish request for hidden service stuptdu2qait65zm
6. Jun 03 20:51:22.878 [info] connection_dir_client_reached_eof(): Uploaded rendezvous descriptor (status 200 ("Service descriptor stored"))

From these log statements the following times can be calculated by subtracting the appropriate timestamps:

Overall Service Publication The only time that is relevant from a user perspective is the delay between starting Tor (1) until a hidden service is available for clients (6); here 80.778 seconds

Service Establishment A certain part of the overall service publication time is required by Tor for downloading enough directory information to build circuits before being able to setup a hidden service. However, the focus here is to speed up the hidden service related parts of Tor. Therefore, the *service establishment time* focuses only on the time between Tor knows enough directory information (2) until uploading the first hidden service descriptor (6); here: 71.727 seconds

The remaining events of giving up on an introduction point candidate (3), successful establishment of an introduction point (4) and initiating upload of a rendezvous service descriptor (5) are used to further examine individual components of service establishment times.

Connection Establishment The connection time (from a user perspective) is the time that it takes to open a socket connection on client-side. In the client process this is performed by a single line of code (with a previous initialization to use the client-side Tor process as SOCKS proxy):

```
mySocket.connect(hiddenServiceAddress);
```

Connection establishment time is measured by firing a log event immediately before and after this code line and subtracting the first timestamp from the second. The corresponding log statements (created by the client and server processes, not by Tor) are: (connection time: 37.679 seconds)

```
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=21:05:03.158, source="clientapp",
  type=CLIENT_OPENING_CONNECTION, message="Client opening
  connection." from source clientapp!
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=21:05:40.837, source="clientapp",
  type=CLIENT_OPENED_CONNECTION, message="Client opened
  connection." from source clientapp!
```

A more in-depth measurement of the multitude of substeps involved in connection establishment is described below in Section 3.2.

Message Transfer The client process sends a numbered message to the server process every minute and logs the sending locally. Upon receiving the message, the server logs receipt and replies immediately. As soon as the client receives the reply, it also logs receipt. The corresponding log statements for the first message exchange with message number 0 are: (request time: 5.559 seconds, response time 3.576 seconds)

```
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=21:05:40.838, source="clientapp",
  type=CLIENT_SENDING_REQUEST, message="Client sending request
  0." from source clientapp!
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=21:05:46.397, source="serverapp",
  type=SERVER_RECEIVING_REQUEST_SENDING_REPLY, message="Server
  receiving request 0." from source serverapp!
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=21:05:49.973, source="clientapp",
  type=CLIENT_REPLY_RECEIVED, message="Client receiving
  response 0." from source clientapp!
```

Connection Durability If the client process observes a connection break before the test duration of four hours elapses, it logs this event locally. The difference between the timestamp of having established the connection and the timestamp of the connection break is considered as the connection durability. If there is no such log event, the last message receipt time is used instead. All times are truncated at 3:45 hours. The relevant log statements are: (connection duration: 3:23:53.992 hours)

```
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=21:05:40.837, source="clientapp",
  type=CLIENT_OPENED_CONNECTION, message="Client opened
  connection." from source clientapp!
FINE event.hidservmeas observeEvent Observed event EventImpl:
  occurrenceTime=00:29:34.829, source="clientapp",
  type=CLIENT_CLOSING_CONNECTION, message="Client is no longer
  connected." from source clientapp!
```

3.2 Internal View Measurements

A second measurement environment is set up to measure the multiple substeps necessary to access a hidden service. Tor clients generate log statements during runtime, which are used to gain insights, when certain internal events occur. To have access to all log files needed, the most important roles in the process of connection establishment are set up especially for the measurements. This includes the two roles mentioned in the first measurement environment, a client-side Tor process and a server-side Tor process, providing the hidden service. In addition to that, two more Tor processes configured as relays are started on the same machine, acting as introduction point and rendezvous point. All other Tor relays involved in the process, constituting the circuits between the above-mentioned roles, are selected from the global Tor network during runtime using the regular Tor path-selection algorithm.

Special configuration of client-side and server-side processes is necessary to make them select the specified relays as introduction and rendezvous point, on the one hand using the regular Tor configuration options, on the other hand changing the Tor source code to achieve the desired behavior. The hidden service process and the introduction and rendezvous point processes are started 10 minutes prior to the beginning of the measurements. This short time is possible because the Tor version used for the measurements does not utilize the V3 directory protocol, which would require more time until all information is available in the directories. Then dedicated client processes are launched every 25 minutes, which try to access the hidden service 75 seconds after their launch to let them build circuits previously.

When setting up the measurement environment, a bug was discovered considering the `RendNodes` configuration option. Using this configuration option the user can suggest specific Tor relays to be selected as rendezvous node by providing a list of nicknames or identifiers. Since the rendezvous point is usu-

ally established using cannibalization, there must be an existing circuit available with the desired rendezvous point at the end. Otherwise the configuration is simply ignored, i.e. a new circuit to the rendezvous point is not build. This bug was introduced with the configuration option RendNodes itself in version 0.0.6pre1 that was released on April 8, 2004.

Rendezvous Descriptor Round-trip Time The time for fetching the rendezvous service descriptor from a hidden service directory is measured as round-trip time, because the directory server is not part of the measurement environment, and therefore its log files cannot be accessed. This time may also include opening a circuit to contact the directory server, if there is no appropriate circuit available. If opening the circuit fails, even multiple attempts may constitute the value. The value is the difference of the timestamps in the following log statements in the log file of the client-side process.

```
Apr 23 20:21:24.456 [info] rend_client_refetch_renddesc():
    Fetching rendezvous descriptor for service xpw51cjsag7u6l6w
Apr 23 20:21:37.905 [info] connection_dir_client_reached_eof():
    Received rendezvous descriptor (size 402, status 200 ("OK"))
```

Open Rendezvous Circuit In the next step the client opens a circuit to a rendezvous point. More precisely, what is measured here is not the actual time to open the rendezvous circuit, but the time until a usable circuit is open after receiving the rendezvous descriptor. If there is a problem with the first rendezvous circuit and a second one must be opened, both together constitute this value.

The rendezvous circuit is usually open immediately, because an existing 3-hop circuit is selected and the last node is declared as rendezvous point. If the circuit was already open at the time of selection, the rendezvous circuit is open, too.

```
Apr 23 20:21:37.905 [info] connection_dir_client_reached_eof():
    Received rendezvous descriptor (size 402, status 200 ("OK"))
Apr 23 20:21:37.906 [info] rend_client_rendcirc_has_opened():
    rendcirc is open
```

Open Introduction Circuit In parallel to opening the rendezvous circuit, an introduction circuit is opened. The measured time is not necessarily equal to the opening time of a single circuit, but may consist of multiple attempts, if a circuit cannot be built or a cannibalized circuit cannot be extended to the introduction point.

```
Apr 23 20:21:37.905 [info] connection_dir_client_reached_eof():
    Received rendezvous descriptor (size 402, status 200 ("OK"))
Apr 23 20:21:38.240 [info] rend_client_introcirc_has_opened():
    introcirc is open
```

Rendezvous Establishment The ESTABLISH_RENDEZVOUS cell is sent from the client to the rendezvous point to make the rendezvous cookie known. While the first statement occurs in the client log file the second appears in the rendezvous point log file.

```
Apr 23 20:21:37.906 [info]
  rend_client_send_establish_rendezvous(): Sending an
  ESTABLISH_RENDEZVOUS cell
Apr 23 20:21:38.494 [notice]
  Received an ESTABLISH_RENDEZVOUS request on circuit 3307
```

Rendezvous Acknowledgment The rendezvous point acknowledges with a RENDEZVOUS_ESTABLISHED cell, that it sends immediately after receiving the ESTABLISH_RENDEZVOUS cell. Therefore the latter is used as starting time for this value.

```
Apr 23 20:21:38.494 [notice]
  Received an ESTABLISH_RENDEZVOUS request on circuit 3307
Apr 23 20:21:38.811 [info] rend_client_rendezvous_acked():
  Got rendezvous ack. This circuit is now ready for rendezvous.
```

Contacting Introduction Point After the rendezvous circuit is established and the router has acknowledged to act as rendezvous point, the introduction point can be contacted. To do so an INTRODUCE1 cell is sent down the introduction circuit. Again the two following statements occur in the log files of two processes, the first in the client's log and the second in the log file of the introduction point.

```
Apr 23 20:21:39.078 [info] rend_client_send_introduction():
  Sending an INTRODUCE1 cell
Apr 23 20:21:39.305 [notice]
  Received an INTRODUCE1 request on circuit 49661
```

Forwarding Request to Hidden Service The introduction point forwards the client request to the hidden service relay, immediately after receiving it. The introduction point uses the circuit that was built during establishing the hidden service and since then kept open.

```
Apr 23 20:21:39.305 [notice]
  Received an INTRODUCE1 request on circuit 49661
Apr 23 20:21:39.857 [info] rend_service_introduce():
  Received INTRODUCE2 cell for service "xpw5lcjsag7u6l6w"
  on circ 64317.
```

Introduction Acknowledgment After successfully forwarding the introduction request to the hidden service, the introduction point acknowledges this back to the client with an INTRODUCE_ACK cell.

```
Apr 23 20:21:39.305 [notice]
  Received an INTRODUCE1 request on circuit 49661
Apr 23 20:21:39.469 [info] rend_client_introduction_acked():
  Received ack. Telling rend circ...
```

Open Rendezvous Circuit on Server-side In the INTRODUCE2 cell the hidden server finds the contact information of the rendezvous point, together with the rendezvous cookie. Then it extends a circuit to the rendezvous point. Again, the time to open the rendezvous circuit on server-side does not need to be constituted by a single circuit cannibalization or build attempt.

```
Apr 23 20:21:39.873 [info] rend_service_introduce():
  Accepted intro; launching circuit
  to "$094C0337F5A03A9D62B35358DDD9F3A8E48FF23B"
  (cookie 4815A117) for service xpw5lcjsag7u6l6w.
Apr 23 20:21:44.042 [info] rend_service_rendezvous_has_opened():
  Done building circuit 64529 to rendezvous with cookie 4815A117
  for service xpw5lcjsag7u6l6w
```

Rendezvous Confirmation After building the circuit to the rendezvous point, a RENDEZVOUS1 cell is sent down the circuit.

```
Apr 23 20:21:44.042 [info] rend_service_rendezvous_has_opened():
  Done building circuit 64529 to rendezvous with cookie 4815A117
  for service xpw5lcjsag7u6l6w
Apr 23 20:21:45.680 [notice]
  Got request for rendezvous from circuit 26085 to cookie
  4815A117.
```

Forwarding Rendezvous Confirmation to Client The rendezvous point forwards the content of the RENDEZVOUS1 cell to the client immediately after receiving a RENDEZVOUS2 cell and connects the circuits of client and server. When the client receives this cell the connection to the hidden service is open and user data can be transferred via the rendezvous point.

```
Apr 23 20:21:45.680 [notice]
  Got request for rendezvous from circuit 26085 to cookie
  4815A117.
Apr 23 20:21:46.791 [info] rend_client_receive_rendezvous():
  Got RENDEZVOUS2 cell from hidden service.
```

4 Results

The *external view* measurements were performed using Tor version 0.2.1.0-alpha-dev (r14739) between June 1, 2:50pm (starting time of first test run) and June 5, 9:35am (starting time of last test run), resulting in a total of 1,090 data samples. A tarball with all log files is available online.¹

During evaluation it has turned out that there is a bug in Tor that leads to a delay in service publication (see below for details). This made it necessary to perform a second set of measurements between June 12, 9:22pm and June 13, 3:31pm with Tor version 0.2.1.0-alpha-dev (r15153). This time a new test was started every single minute instead of every five minutes and tests were aborted after uploading the first rendezvous service descriptor. The resulting 1,090 data samples are also available for download.²

The *internal view* measurements started on April 22 at 4:00pm and finished on May 13, 10:20pm, resulting in a total of 1,200 data samples. The log files are also online.³ All roles used Tor version 0.2.0.7-alpha.

4.1 Service Publication

Figure 2 shows the overall service publication times as a user experiences the process of starting up Tor until a hidden service is available for clients. From the 1,090 measured publication times, the 1% highest values (698, 901, 1214, 1220, 1222, 1224, 1228, 1241, 1290, 3073, and 3685 seconds) were considered as outliers and thereby discarded in the statistical analysis. (Nevertheless, especially outliers need to be analyzed separately, because they are accountable for a substantial amount of variance.)

However, these values still contain initialization times, i.e. the time that Tor needs to download enough directory information to build circuits. Figure 3 contains only service establishment times that are directly related to establishing a hidden service. This time only the top two value had to be discarded as outliers (362 and 3665 seconds). These times will later be used to measure improvements of hidden service publication.

Too Small Minimum Value The first observation is that the minimum value of 12.85 seconds is—albeit being a good result—lower than it ought to be. Actually, there is a 30-second delay meaning that a descriptor is only uploaded if it is unchanged for at least 30 seconds. Hence, the descriptor was uploaded prematurely in the considered case. In the whole set of 1,090 samples, there are four service establishment times below 30 seconds (13, 18, 27, and 28 seconds).

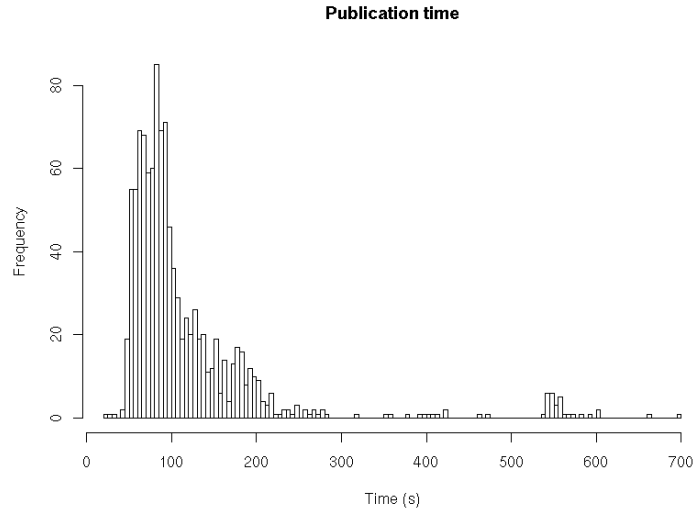
It turned out that the reason for this is a minor bug in the code which was in Tor since version 0.0.9pre6 released on November 15, 2004. This bug is now fixed in SVN revision r15113⁴ and released in Tor 0.2.1.1-alpha on June 13,

¹<http://freehaven.net/~karsten/hidserv/perfdata-2008-06-01.tar.gz>

²<http://freehaven.net/~karsten/hidserv/perfdata-2008-06-12.tar.gz>

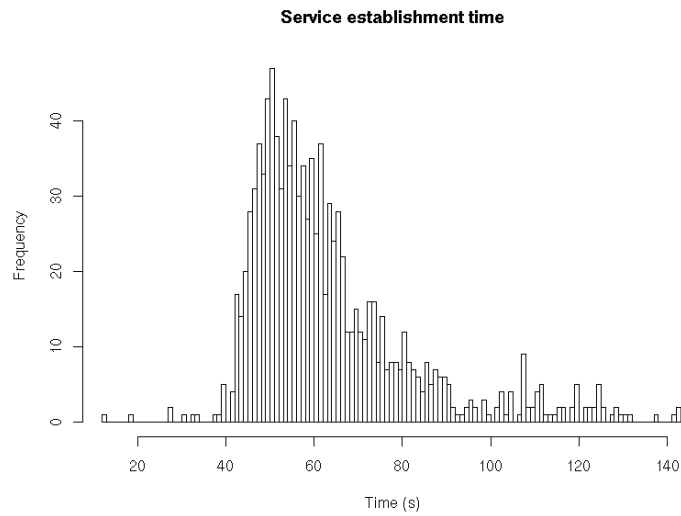
³<http://freehaven.net/~karsten/hidserv/perfdata-2008-04-22.tar.bz2>

⁴<http://archives.seul.org/or/cvs/Jun-2008/msg00231.html>



Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	StdDev
22.85	69.91	89.68	118.10	129.20	698.10	93.77

Figure 2: Overall service publication times



Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	StdDev
12.85	50.59	58.27	63.25	69.73	142.20	18.67

Figure 3: Service establishment times

2008. This is not meant as confirmation for the usefulness of the 30-second delay, but only to make the implementation consistent with the specification.

Slow Decrease of Frequencies Figure 3 shows an unexpectedly large number of service establishment times of 90 seconds or greater. The histogram exhibits a decline of establishment times from their maximum at 50 seconds to a very low frequency of values around 90 seconds. But there is still a significant number of test cases with establishment times greater than 90.

An in-depth analysis of the log files has revealed an even more severe bug. While setting up a hidden service, some valid introduction circuits were overlooked and abandoned. This leads to random delay in establishing introduction points and publishing a descriptor. This bug was introduced with Tor version 0.2.0.14-alpha released on December 23, 2007. It is now fixed in SVN revision r15149⁵ and included in Tor 0.2.1.1-alpha and 0.2.0.28-rc which were both released on June 13, 2008.

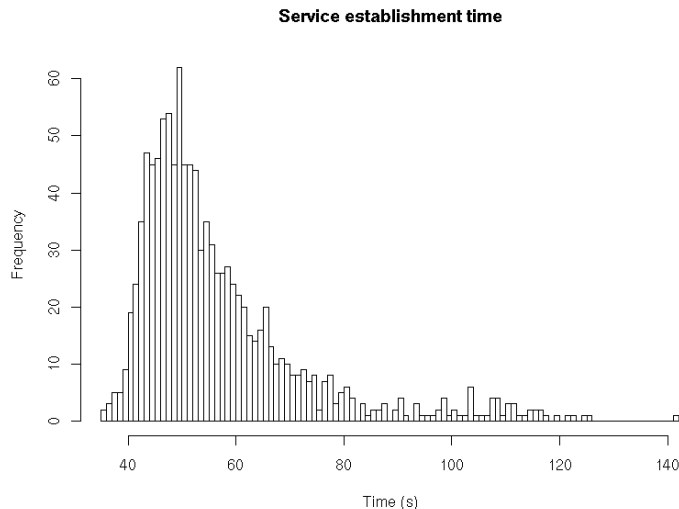
This bugfix made it necessary to perform a second set of measurements with special focus on service publication. Figure 4 shows service establishment times with a bugfixed Tor version. Again, three values were discarded as outliers (3665, 3680, and 10931 seconds). The histogram features a clearly smoother decline of frequencies than the histogram of the first measurements. The mean has decreased to 56.73 (-6.52), median to 51.99 (-6.28), and standard deviation to 15.82 (-2.85).

Outliers The outlier values of 3665, 3680, and 10931 seconds in the second set of measurements using the bugfixed Tor version appear to be unbelievably high for a service establishment time. They further obey a certain pattern, as the conversion to hours clarifies: 1:00:04, 1:00:36, and 3:01:26 hours.

A closer look into the log files reveals that the delay occurs in all three cases after Tor decided to upload a descriptor and up to the moment at which it is stored on the directory servers. The explanation for this behavior is that Tor attempts to upload a descriptor, fails, and waits for the next regular descriptor republication one hour later to perform a second upload attempt. As a solution to avoid these outliers, a significantly shorter timeout or exception handling routine for failed descriptor uploads appears useful. This should improve performance of service publication by greatly reducing its variation.

Exceeding the 30-Seconds Delay The current logic for publishing a rendezvous service descriptor is to wait for 30 seconds after the last change to the set of introduction points. Though, in some test cases these 30 seconds were exceeded: In one test case, there is a 34 seconds gap inbetween two introduction establishments, but without an attempt to upload a descriptor. In another case, a descriptor is not uploaded until 41 seconds after an introduction point establishment.

⁵<http://archives.seul.org/or/cvs/Jun-2008/msg00268.html>



Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	StdDev
35.55	46.73	51.99	56.73	61.42	141.60	15.82

Figure 4: Service establishment times with bugfixed Tor version

An inspection of the log files revealed that the abandoning of an introduction point candidate resets the 30-seconds counter, too, even though it would have no effect on the uploaded descriptor. Hence, given that the 30-seconds delay is a useful means to estimate when a descriptor is stable, the 30-seconds counter should only be reset when an actual change to the descriptor to be uploaded occurs.

Initial Upload of Empty Descriptors There is another effect of the current behavior to reset the 30-seconds counter when abandoning an introduction point candidate: In very rare cases the 30-seconds counter is initialized by such an abandoning event, but then Tor does not succeed in establishing an introduction point within the next 30 seconds. The result is the upload of an empty rendezvous service descriptor, i.e. a descriptor containing zero introduction points. Under certain circumstances this might be considered an improvement for clients if the most recent descriptor originates from a previous run of Tor and is entirely wrong. However, in most cases it would be sufficient and even more useful to wait for the construction of a non-empty descriptor before uploading.

Failing Introduction Point Establishments In some cases Tor establishes up to 13 introduction points before uploading the first descriptor. It appears rather unlikely, that 10 out of 13 circuit establishment fail. In fact, it turned out that there is another major bug in Tor that completely ignores introduction points when they were established by cannibalization of an existing circuit.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Fetch rend. desc.	0.138	1.419	3.595	8.144	7.975	119.600
Open rend. circ.	0.000	0.000	0.001	2.316	0.001	85.880
Est. rend.	0.009	0.109	0.267	0.926	0.773	56.100
Rend. ack.	0.004	0.071	0.203	0.736	0.898	32.770
Open intro. circ.	0.035	0.624	1.763	7.048	4.304	100.900
Intro1	0.003	0.168	0.466	1.121	1.046	51.740
IntroAck	0.017	0.214	0.765	1.402	1.471	24.150
Intro2	0.013	0.244	0.571	1.405	1.155	37.260
Server rend. circ.	0.095	1.741	3.139	5.711	6.343	48.530
Rend1	0.021	0.552	1.118	2.461	2.138	37.140
Rend2	0.003	0.091	0.227	0.711	0.796	26.230
Round trip time	2.479	11.450	18.210	27.370	33.220	181.800

Figure 5: Statistical data of all substeps

Even though there may in general be other reasons for failing establishment of a circuit, this bug was the reason for all 10 failed establishments in the considered case. This bug was introduced in Tor with version 0.2.0.14-alpha released on December 23, 2007. It will probably be fixed in the upcoming Tor versions 0.2.0.29(-rc) and 0.2.1.2-alpha. This should significantly speed up and reduce variance in service publication time.

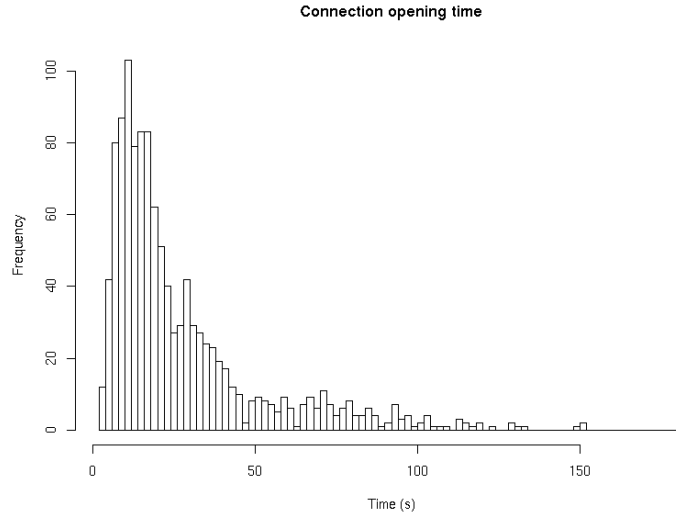
Usefulness of 30-Seconds Delay The 30-seconds delay before publishing a descriptor unsurprisingly leads to a lower bound of service establishment time of 30 seconds. However, these 30 seconds originally emerged from a guess of the developers. There is no proof that a shorter period might also suffice to upload a stable descriptor, i.e. one that does not have to be replaced shortly afterwards. The other way around it has not yet been evaluated whether a longer delay might significantly improve stability.

One of the next steps in the attempt to speed up service publication is to measure successful introduction establishments during the first half hour (not stopping at the first descriptor publication). From these times one can determine the number of necessary descriptor publications for different delays than 30 seconds. Further, it might make sense to adapt the delay depending on the number of previously published descriptors or the extent of changes to the last published descriptor.

4.2 Connection Establishment

After reviewing and analyzing all data of the different substeps resulting from the internal measurements, only a selection is presented here. A statistical summary of all steps is shown in Table 5.

In 96.33% of all access attempts the connection to the hidden service could be established. The other attempts failed because of a number of different



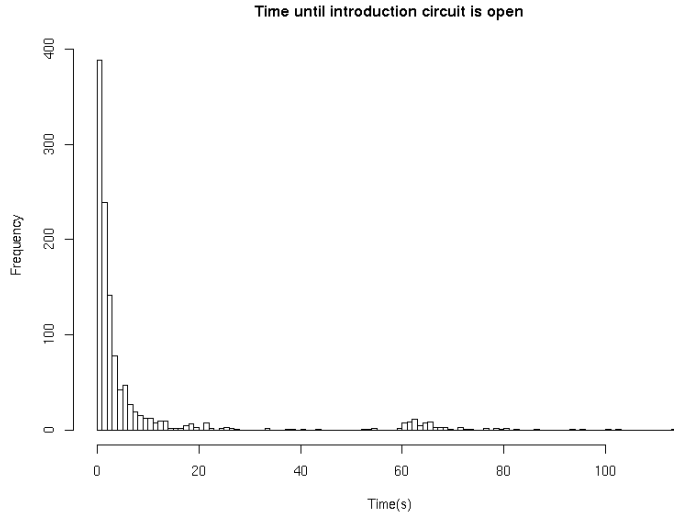
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	StdDev
2.479	11.450	18.210	27.370	33.220	181.800	24.832

Figure 6: Histogram of connection opening times

reasons. The rendezvous descriptor could not be retrieved from the directory because building the circuit failed. Introduction or rendezvous circuits could not be opened at all on client side. The same is possible for the rendezvous circuit on server side. In a small number of cases cells were not forwarded on introduction and rendezvous points.

Overall Establishment Time Figure 6 shows the distribution of user experienced connection establishment time of hidden services. An analysis of the reasons for higher values brings up that not a single step is responsible for delays, but many different steps, with nearly each having the potential to delay the process for tens of seconds. This includes circuit establishment as well as message transfers.

Circuit Cannibalization Figure 7 shows that in most cases the introduction circuit is open within seconds. For the values around 60 seconds the circuit could not be cannibalized successfully and after a timeout of one minute a second attempt is started. The reasons for failing cannibalization can be that only the actual extension, i.e. connection to the introduction point fails or that a circuit was chosen for extension, that was not open yet. That means, that also the first to third hop can be responsible for the failure. If no appropriate circuit is available, neither open nor still being built, a completely new circuit is opened, which also takes longer than extending an existing circuit. Since the measurements were performed with Tor version 0.2.0.7-alpha, there is no



Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	StdDev
0.035	0.636	1.793	7.784	4.642	114.400	17.532

Figure 7: Histogram of times until introduction circuit is open

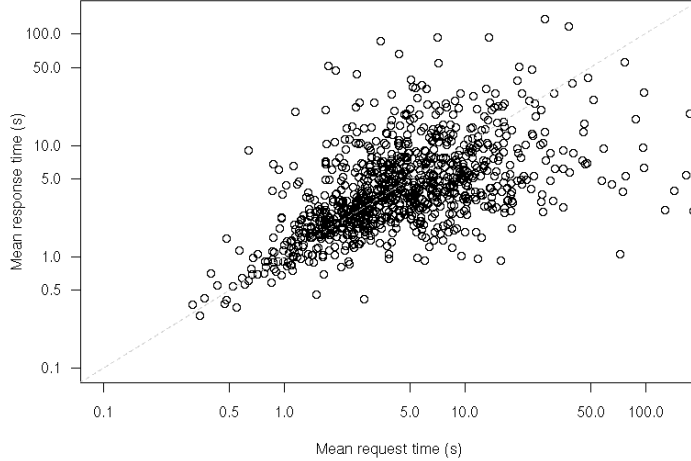
connection to the bug discussed in the previous section, which was introduced in Tor 0.2.0.14-alpha.

The timeout of 60 seconds seems too long. It is the same timeout that applies for general circuit establishment, but in this case only an extension by one hop takes place. So for cannibalization a lower timeout values might improve the situation. Further investigations are necessary, to find out how lower bandwidth connections of clients affect circuit building and extension times. This can give an answer, if the general circuit building timeout should also be reduced. The whole hidden service access currently has a timeout of 120 seconds after receiving the rendezvous descriptor. Considering intermediate substeps like receiving the INTRODUCE_ACK cell for timeouts should also be investigated.

4.3 Message Transfer

Figure 8 contains a plot of mean request times versus mean response times for each test. Data points on the diagonal represent tests with equal mean request and mean response time. Most points seem clustered around the line. However, there is a tendency for requests to be longer than responses which is also supported by mean and median.

Mean request or response times of 100 seconds or more are almost unacceptable for most applications. A look into the log statements confirmed that these mean values result from normal connections and not from short-lived connections with few, very high message transfer times. Further, in some cases mean



	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Mean request time	0.312	2.302	4.154	7.596	8.234	143.3
Mean response time	0.298	2.184	3.682	6.560	6.911	135.0

Figure 8: Mean request and response times

request times are up to two orders of magnitude higher than mean response times and vice versa.

At the moment, message transfer times raise more questions than can be answered. However, these questions are not directly related to hidden services, but also apply to regular Tor circuits. There are no hints so far that delays in message transfer times are specific to hidden services. A solution to improve message transfer times would probably require changes to Tor’s general path selection algorithm. Therefore, message transfer times will not be further considered in this attempt to speed up hidden services. The presented data may nevertheless be helpful for later attempts to speed up Tor message transfer times in general.

4.4 Connection Durability

Figure 9 shows connection durations. The artificial upper limit of 13,500 seconds (3:45 hours) comes from the maximum allowed test time of four hours after a connection has been established. The unexpectedly high number of 650 out of 962 (67.57%) established connections did not break within the first 3:45 hours. Considering the set of connections that broke before 3:45 hours elapsed, there is no noticeable point of time at which connections break, but connection breaks seem to be equally distributed over the whole time interval.

Hidden service connections appear to be quite stable, so that there is no

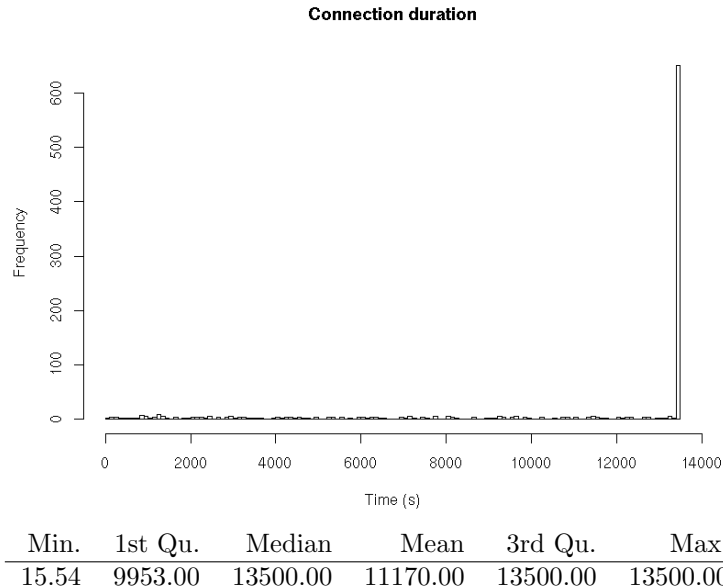


Figure 9: Connection durations

need to put special focus on it in the attempt to improve the hidden service protocol. Furthermore, in the measurement setup the requested ports were chosen randomly and were very likely *not* included in the set of long-lived ports (which are by default: 21, 22, 706, 1863, 5050, 5190, 5222, 5223, 6667, 6697, 8300). For connections to these ports Tor selects only high-uptime nodes to reduce the chance that a node will go down before the connection is finished. As a consequence, e.g. instant messaging connections probably exhibit even higher connection stability than measured here.

5 Discussion

The analysis of setting up and accessing a hidden service has revealed a couple of new insights for potential performance improvements that shall be summarized here. First, a few *bugs* could be spotted of which some have already been fixed in the course of this analysis:

Premature Descriptor Upload In very rare situations new hidden service descriptors were published earlier than 30 seconds after the last change to the service, although the current thinking is that a hidden service descriptor that's been stable for 30 seconds is worth publishing. This minor bug was in the code since Tor version 0.0.9pre6 released on November 15, 2004. This bug is fixed in Tor version 0.2.1.1-alpha released on June 13, 2008.

Abandoning Valid Introduction Points While setting up a hidden service, some valid introduction circuits were overlooked and abandoned. This might be the reason for the long delay in making a hidden service available. This major bug was introduced with Tor version 0.2.0.14-alpha released on December 23, 2007. It is now fixed and included in Tor 0.2.1.1-alpha and 0.2.0.28-rc which were both released on June 13, 2008.

Ignoring Cannibalized Introduction Points When establishing a hidden service, introduction points that originate from cannibalized circuits are completely ignored and not included in rendezvous service descriptors. This might be another major reason for delay in making a hidden service available. This bug was introduced in Tor with version 0.2.0.14-alpha released on December 23, 2007. It will probably be fixed in the upcoming Tor versions 0.2.0.29(-rc) and 0.2.1.2-alpha.

Disregarding Predefined Set of Rendezvous Points The RendNode bug can either be solved by building a circuit to the rendezvous point when the rendezvous circuit is needed, or by building such a circuit preemptively during startup. In the latter case the circuit could be easily cannibalized, while the former scenario reduces performance, because we have to wait until the new circuit is built.

The analysis further revealed a couple of *possible improvements* for the phases of making a hidden service available and for establishing a connection:

Descriptor Upload Failures The current logic to upload rendezvous service descriptors does not handle failures in a reasonable way. In case of a failure, Tor waits for a solid hour before making the next attempt. There should either be a smaller timeout or an individual handling of failures per directory.

Inaccuracies in Descriptor Upload Logic The logic to decide whether a descriptor should be uploaded needs a reworking. Currently, unrelated events like giving up on an introduction point candidate resets an internal 30-seconds timer. The result is an unexpected exceedance of the timer. Another weird effect is the uploading of descriptors without any introduction points contained in them.

Descriptor Upload Timing The choice to wait for 30 seconds for a service to have a stable set of introduction points is rather arbitrary. An analysis of typical delays in establishing introduction points might help to apply a more suitable algorithm here.

Increase Count of Internal Circuits The number of preemptively built internal circuits for later cannibalization should be increased. Really popular hidden services require more than two internal circuits in the pool to answer multiple client requests at the same time. This scenario was not

yet analyzed, but will probably exhibit even worse performance as measured here. The number of preemptively built internal circuits should be a function of connection requests in the past to adapt to changing needs. Furthermore, an increased number of internal circuits on client side would allow clients to establish connections to more than one hidden service at a time.

Build More Introduction Circuits When establishing introduction points, a hidden service could launch 5 instead of 3 introduction circuits at the same time and use only the first 3 that could be established. The remaining two circuits could still be used for other purposes afterwards.

Parallel Connections to Introduction Points A client could attempt to establish two introduction circuits to two different introduction points simultaneously and only use the first that succeeds. The slower circuit could still be used for another purpose. However, there is a possible anonymity issue here that needs to be taken under consideration, because the circuit can now be linked to one of the hidden service's introduction points. Further, it needs to be evaluated whether the extension of two circuits at the same time has a negative effect on clients with low bandwidth.

Replacement for 1-Second Circuit Creation Loop The current implementation of a client accessing a hidden service makes use of a 1-second loop that checks whether circuits have been successfully established. This results in an unnecessary delay of up to one second or 0.5 seconds in average when the client could already proceed in connection establishment. This can be avoided by processing the establishment of a circuit immediately.

There are areas which are candidates for performance improvements, but which are rather vague at the moment and require a relevant amount of effort for *further investigation* before they can be realized. These items will be considered in the course of this project:

Rendezvous Protocol Simplifications Øverlier and Syverson proposed two simplified rendezvous protocols.⁶ Their first protocol aims at using a single circuit on client-side to contain the rendezvous point and connect to an introduction point. The second protocol goes the extra mile to unite the roles of introduction point and rendezvous point and save another circuit. It can be assumed that these simplifications improve connection establishment times. However, they have yet unclear effects on anonymity, given that they are implemented without the authors' proposed valet node approach which requires a major redesign of hidden services.

Distributed Hidden Service Directory Current measurements have been performed using the central hidden service directory only. In the near fu-

⁶Lasse Øverlier and Paul Syverson, Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services, <http://www.freehaven.net/anonbib/#overlier-pet2007>

ture the central directory will be replaced by a distributed directory consisting of a subset of all Tor relays. This design is already deployed and in an experimental state. As soon as a reasonable number of distributed hidden service directory nodes are deployed, there should be further measurements to compare performance with the current directory design. It might turn out that more sophisticated means for load balancing between distributed directory nodes are necessary than are currently implemented.

Grand Scaling Plan There are other attempts to make Tor more useful for low-bandwidth clients.⁷ Part of this project might be that clients do not download the complete set of router descriptors, but request descriptors during the process of circuit establishment. Alas this would slow down circuit creation even more, so that hidden services should even try harder to avoid on-demand circuit creation.

Low-Bandwith Measurements For some improvement suggestions, e.g. reducing timeouts, the effect on clients with low bandwidth is yet unclear. Future measurements should therefore include clients and possibly hidden servers on low-bandwidth Internet connections.

At last, there are few approaches that could have a positive effect on the performance of Tor Hidden Services, but which require substantial changes to the core parts of Tor. That brings them *out of the scope* of this project that aims at improving hidden services only. Anyway, these ideas are at least worth mentioning for later attempts to improve Tor performance in general:

Circuit Establishment Timeout A Tor client could keep track of how long it takes to establish a circuit, and discard circuits that took too long to establish. The justification is that a circuit that took long to build will probably take long to transport messages. This approach should considerably increase all parts of hidden services, too.

Allow Failing Circuit Extensions The current rationale of Tor is to discard a complete circuit once an extend attempt has failed. The main reason is to inhibit that an adversary can control the set of relays to which a client can build circuits and prevent extensions to other nodes than the adversary's. A relaxation of this logic by allowing, e.g., 3 extend failures per hop might result in a significant speeding up of hidden services, that depend on fast circuit extension to a great extent.

Better Load Balancing on Relays The reason why some circuits are orders of magnitude slower than others might result from the fact that single relays are overloaded. A better path selection scheme based on a relay's current load might lead to better load balancing and thereby lower delay in message transmission.

⁷<https://www.torproject.org/projects/lowbandwidth>

Prioritize Low-Volume Circuits Tor relays currently handle all incoming cells on a first-come-first-served basis. A better approach than this is to give higher priority to circuits that have not sent as many cells lately. This would especially prioritize introduction and rendezvous circuits and thereby accelerate connection establishment.