A Reputation System to Increase MIX-net Reliability

Roger Dingledine[1], Michael J. Freedman[2], David Hopwood[3], and David Molnar[4]

[1] Reputation Technologies, Inc. <arma@reputation.com>
[2] Massachusetts Institute of Technology <mfreed@mit.edu>
[3] Independent consultant <david.hopwood@zetnet.co.uk>
[4] Harvard University <dmolnar@hcs.harvard.edu>

Abstract. We describe a design for a reputation system that increases the reliability and thus efficiency of remailer services. Our reputation system uses a MIX-net in which MIXes give receipts for intermediate messages. Together with a set of witnesses, these receipts allow senders to verify the correctness of each MIX and prove misbehavior to the witnesses. We suggest a simple model and metric for evaluating the reliability of a MIX-net, and show that our reputation system improves over randomly picking MIX paths while maintaining anonymity.

1 Introduction

Anonymous remailers are the most common method of anonymous e-mail communication. Despite wide use of the current global remailer network, this network is generally considered unreliable. Messages are often dropped, and the newsgroup `alt.privacy.anon-server` contains many examples of a message being sent two or three times in the hope that one instance will reach the destination. This unreliability directly affects the number of people using the remailer network and their traffic patterns, which in turn reduces the anonymity these networks provide.

Several approaches to the problem of unreliability are possible. One approach is to write more reliable software [20]. Another approach is to build MIX protocols that give provable robustness guarantees [4, 9, 16]. Our approach is to build a reputation system to track MIX reliability, and to modify the MIX protocol to support this system. Because users choose paths based on the published scores for each MIX, this reputation system improves both reliability (fewer messages get routed through dead MIXes) and efficiency (the system dynamically rebalances the load based on available reliable resources).

Currently deployed remailer reputation systems (better known as *remailer statistics*) collect data independently and are treated as trusted third parties by client software. Since shutting down a remailer statistics server means that its statistics are no longer available, reliable servers are good targets for adversaries. Furthermore, these statistics often measure secondary properties of the MIX-net servers, such as up-time, which do not necessarily translate to reliability, especially with regard to Byzantine failure.

We present a MIX-net design that allows the sender of a message, along with a collection of weakly trusted third party *witnesses*, to prove that a MIX failed to process a message correctly. We introduce a new set of agents called *scorers*, who tally failure proofs and serve them to client software. Because each failure can be proven to any number of scorers, loss of any individual scorer will be less disruptive than loss of a statistics server in current reputation systems.

We go on to describe a simple metric for evaluating the reliability of a MIX-net or remailer system, and then show that our reputation system improves over randomly picking MIX paths. Our metric is the expected probability of a message successfully being sent through the MIX-net. We specify a simple probability model for a MIX-net and then give an example of how to evaluate our metric on a MIX-net where no reputation system is specified. Then we do the same analysis with a reputation system in place and show that a message's chance of surviving improves.

We stress that this work does *not* provide a fully satisfactory solution to the problem of MIX reliability. In particular, our reliability model is too simple to take into account active adversaries. As such, it is not clear what effect our reputation system will have against an active adversary. Our main goal is to show a reasonable initial reputation system, to stimulate further research on reputation systems and on modelling the increased reliability they provide.

We describe related MIX-nets and current statistics systems in Section 2. In Section 3 we present a model for describing reliability of a MIX-net. Section 4 presents our MIX-net design, describes specifically how to send a message in this MIX-net, and then argues security of the overall network. In Section 5, we describe the design and motivation for the reputation scoring system, and the implications of deploying this system. In Section 6 we introduce the notion of a *reliability metric* and propose such a metric for MIX-nets. We apply this metric to measure reliability in a MIX-net with randomly chosen paths, and then show that a reputation system can improve reliability, in a drastically simplified model. Finally, we close in Section 8 with a discussion of possible directions for future research.

2 Related Work

2.1 MIX-nets

Chaum introduced the concept of a MIX-net for anonymous communications [3]. A MIX-net consists of a series of servers, called MIXes, each of which is associated with a public key. Each MIX receives encrypted messages. These messages are then decrypted, batched, their order permuted, and forwarded on after stripping the sender's name and identifying information. Chaum also proved security of MIXes against a *passive adversary* who can eavesdrop on all communications between MIXes but is unable to observe the permutation inside each MIX. Another way of saying this is to note that the *view* of a passive adversary gives negliglible advantage over guessing in linking messages with their senders and receivers.

Work on MIX-nets is ongoing. Current research directions include "stop-and-go" MIX-nets [10], distributed "flash MIXes" [9] and their weaknesses [4, 16], and hybrid MIXes [17].

Previous work primarily investigates a notion of the *robustness* of MIX-nets in the context of a distributed MIX system [9]. A MIX is considered robust if it survives the failure of any k of n participating servers, for some threshold k. This robustness is all or nothing: either k servers are good and the MIX works, or they are not good and the MIX likely will not work.

Robustness has been achieved primarily via the use of zero-knowledge proofs of correct computation; participants use such proofs to convince each other that they follow the protocol correctly without revealing secret information. Done naïvely, this incurs significant overhead. Jakobsson showed how to use precomputation to reduce the overhead of such a MIX network to about 160 modular multiplications per message per server [9], but the protocol was later found to be flawed [16] by Mitsumo and Kurosawa. They proposed a fix for the protocol, but the efficacy of this fix remains to be evaluated. A different approach was taken by Desmedt and Kurosawa [4], but their technique requires many participating servers. Other work, such as Abe's MIX [1], provides *universal verifiability* in which any observer can determine after the fact whether a MIX cheated or not, but the protocol is still computationally expensive.

Our notion of reliability differs from robustness in that we do not try to ensure that messages are delivered even when some nodes fail. Instead, we focus on improving a sender's long-term odds of choosing a MIX path that avoids failing nodes, and we attempt to quantify this improvement.

We note that the two approaches can be composed: a distributed MIX with robustness guarantees can be considered as a single node with its own reputation in a larger MIX-net.

2.2 Deployed Remailer Systems

The first widespread public implementations of MIXes were produced by the cypherpunks mailing list. These "Type I" *anonymous remailers* were inspired both by theoretical work on MIXes and by the problems surrounding the `anon.penet.fi` service [8]. Hughes wrote the first cypherpunks anonymous remailer [13]; Finney followed closely with a collection of scripts which used Phil Zimmerman's PGP to encrypt and decrypt remailed messages. While these "Type I" remailers did not include crucial MIX features, such as reordering or batching of messages, they provided a better approximation of MIXes than had been generally available. Later, Cottrell implemented the Mixmaster system [6], or "Type II" remailers, which added message padding, message pools, and other MIX features lacking in the cypherpunk remailers. At about the same time, Gulcu and Tsudik introduced the Babel system [7], which also created a practical remailer design (although one that never saw widespread use).

Producing properly formatted remailer messages by hand is a difficult and error-prone task. The program Private Idaho [15], developed by McNamara, was one of the earliest and best known means of automatically formatting remailer messages. The program is no longer maintained, but many different splinter versions exist. Currently, Potato Software's Jack B. Nymble and Reliable remailer software offer a range of features, including the ability to configure user preferences for picking remailer nodes [19].

2.3 Remailer Statistics

After the deployment of the original cypherpunk remailers, the next major step was Levien's development of remailer *statistics pages*. Statistics consist of both remailer capabilities, such as what kinds of encryption the remailer supports, and remailer *up-times*, observed by pinging the machines in question and by sending test messages through each machine or group of machines.

Levien statistics are an example of what we will call a *reputation system*. Section 5 will discuss this in more detail. Reputation systems offer a means of improving the reliability of MIX-nets by allowing users to avoid choosing unreliable MIXes. The Jack B Nymble 2 remailer client [19] allows users to import statistics files and can then pick remailers according to that data. Users can specify minimum reliability scores, decide that a remailer should always or never be used, and specify maximum latency.

2.4 Approaches to MIX-Net Reliability

There are at least three approaches to MIX-net reliability:

1. **Reliability via Protocol** We can build protocols which give specific guarantees of robustness and reliability, and prove theorems about these protocols in suitably specified adversary models. These results may be quite strong, e.g. "this distributed MIX delivers correctly if no more than half of its participating servers are corrupt." The drawback of this approach is that the resulting protocols may be complicated, and efficient protocols with provable properties even more so. Converging on the "right" protocol may be tricky; even a small misstep in a proof may invalidate all robustness results.
2. **Reliability via Reputations** Instead of engineering the MIX-net protocol directly to provide reliability, we can make use of reputations to track MIX performance. In this approach, we specify a set of behaviors which characterize a "good" or "bad" MIX, then rate MIXes in a system according to their behavior. Unlike the reliability via protocol approach, we are unlikely to prove theorems such as "messages are always delivered if a majority of MIXes are good." Instead, the goal of a reputation system is to make delivery "better" without guaranteeing perfection.
3. **Reliability via Economics** We can create economic incentives for MIX nodes to stay reliable, or ensure that an adversary who wishes to compromise reliability must spend a large amount of resources. Currently, Zero-Knowledge Systems is exploring this approach by paying MIX nodes for

consistent reliability and performance in the Freedom network [21]. Another variant on this approach is to introduce the use of payments for each message sent through a MIX.

Reliability via protocol is the most well-studied approach, while reliability via reputations in the form of Levien statistics is the most widely used approach. Our work explores a combination of the first two approaches. We modify the MIX-net protocol to support easy tallying of MIX failures and then specify a reputation system that makes use of this support.

3 MIX Models

In order to speak about MIX-nets, we first must set out a model of what MIX-nets are. We will actually specify two models. The "general" model will be with respect to an adversary which aims at breaking the anonymity of the MIX-net. Then we will introduce a model for an adversary that aims at breaking the reliability of the MIX-net. Unifying these two adversaries is a topic for future work.

3.1 General MIX Model

A MIX-net consists of participating servers called *MIXes*. These MIXes are connected by communication channels. There also exist one or more *senders*, who wish to use the MIX-net to anonymously send messages to one or more *receivers*. To send a message, a sender picks an ordered collection of MIXes called a *MIX path* to stand between it and the receiver. We will also refer to MIXes as "MIX nodes" or simply "nodes."

The MIX-net also has an *adversary*, whose goal is to link messages passing through the MIX-net with their senders and receivers. We will be most concerned with passive adversaries and the view such adversaries have of the MIX-net. A *passive adversary* is an adversary which may eavesdrop on all communication between MIXes, but it may not modify or inject messages. The *view* of a passive adversary is the collection of all messages that the adversary may see; in other words, the complete transcript of all messages passed between MIXes. This is the model of adversary for which Chaum designed the initial MIX-net protocol. An *active adversary*, in contrast, may send messages of its own, block messages, or intercept and change messages.

3.2 MIX Reliability Model

In section 6, we specify a reliability metric for MIX-nets. First, we will specify a suitable model of a MIX-net, which will concern *only* an adversary interested in minimizing the reliability of the MIX-net.

Let *MIXes* be the set of m participants making up the MIX-net. Let the set of senders be S, and the set of receivers be R. Let $Paths \subset MIXes^*$ be the set of valid *MIX paths*, that is, the set of sequences of MIXes allowed as paths by the MIX-net protocol.

Senders first choose a receiver and then choose an ordered sequence of MIXes. Both choices are random variables. For each sender $s \in S$, we define the random variables R_s corresponding to s's choice of receiver given its knowledge about the MIX-net, and $Path_{s,r}$ corresponding to s's choice of MIX paths when sending to receiver r.

The fundamental operation in our MIX-net model is *sending a message*. To send a message, a sender $s \in S$ picks a recipient $r \in R$ according to the distribution R_s, and then picks a MIX path, $path \in Paths$. The sender then sends the message along $path$ to r.

We say a message *survives* the MIX-net if none of the MIXes in $path$ fail. We say that the message is *dropped* by the MIX-net if any of the MIXes along the path fail. The goal of the sender is to minimize the number of dropped messages.

4 A MIX-net With Witnessed Failures

Verification of transactions in cryptographic protocols is commonly supported either by performing the transaction publicly, or by using digitally signed receipts. We use the latter notion to develop a design in which MIXes normally provide a receipt for each message that they receive.

The naïve use of receipts is not sufficient to be able to pinpoint failures. A MIX could refuse to provide a receipt for a message that it was supposed to send to the next MIX because it failed to send it; alternatively, it might have been following the protocol but unable to obtain a receipt from the next hop. We solve this problem as follows: each message has a *deadline* by which it must be sent to the next MIX. A MIX N_i first tries to send a message directly to the next node, N_{i+1}. If N_i has not received a valid receipt by a specified period before the deadline, it enlists several *witnesses*, who will each independently try to send the message and obtain a receipt. For each witness who succeeds in obtaining a valid receipt, the witness sends it back to N_i. Any witness that fails will be convinced that N_{i+1} is unreliable, and provides MIX N_i with a signed *statement* to that effect.

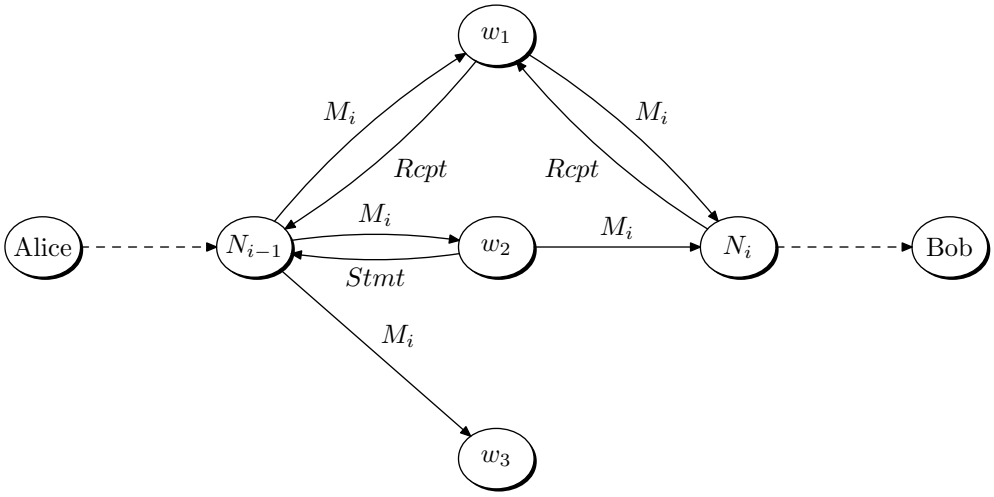

Fig. 1. Message flow example

Thus for each message a MIX sends, it will either have a receipt showing that it processed the message in the time allowed, or statements signed by any witnesses it chooses, asserting that the next MIX did not follow the protocol. We will describe a protocol which uses these receipts and statements to allow the sender of a failed message to demonstrate that a particular MIX on the path failed.

The above argument does not explicitly take into account the possibility that several consecutive MIXes are controlled by an adversary; we cover that in Section 4.5.

Note that the witnesses need not be trusted to preserve anonymity, since the messages sent to them are no more than the view of a passive adversary with complete network access. Therefore, the security proofs that apply to traditional MIX-net protocols still hold: we retain unlinkability between sender and receiver due to the batching and reordering that each MIX performs internally.

In general, a system to verify transactions should provide verification of both successful and failed transactions. However, due to the pseudonymous nature of a MIX-net, successful transactions can easily be

spoofed (via pseudospoofing, as described in Section 5.5) and thus the ability to prove that a transaction succeeded would not provide an accurate evaluation of MIX reliability.

4.1 Cryptographic Algorithms

We require two public key schemes: a public-key encryption scheme, which should be semantically secure under adaptive chosen ciphertext attack (equivalent to IND-CCA2 in the definitions of [2])[1], and a public-key signature scheme, which should be existentially unforgeable under adaptive chosen message attack.

The encryption scheme is modelled as a key pair generation algorithm G_E, a randomized encryption algorithm E, and a deterministic decryption algorithm D. In our notation, we will explicitly include the random value used in the encryption, so $E_i^r(M)$ means the encryption of message M and random value r under the public key of N_i. We assume the usual correctness property that if N_i's key pair is valid (i.e. was generated by G_E), $D_i(E_i^r(M)) = M$ for any plaintext M and random value r.

The signature scheme is modelled as a key pair generation algorithm G_S, a signing algorithm $Sign$, and a verification algorithm Ver. $Sign_i(M)$ is the signature of M under the private key of N_i, and $Ver_i(Sig, M) = 1$ if Sig is a valid signature by N_i on M, or 0 otherwise. If N_i's key pair is valid (i.e. was generated by G_S), then $Ver_i(Sign_i(M), M) = 1$ for any message M.

We also assume that authentic, distinct encryption and verification public keys for each MIX node are known to all parties. Recipients of messages are treated as nodes.

4.2 Overall MIX-net design

Alice wants to send Bob a message anonymously. She chooses a path through the network consisting of $k-1$ MIXes, $N_1 \ldots N_{k-1}$. We denote Alice as N_0 and Bob as N_k. Alice then repeatedly "onion" encrypts her message, and sends the onion to the first MIX in her path. That MIX returns a receipt, processes the onion, and passes the unwrapped-by-one-layer onion to the next MIX in the path, which repeats these steps. If the message does not reach Bob, the transaction has failed. (Section 4.6 shows how to use "end-to-end receipts" to ensure that Alice knows when this has occurred.)

Our system should be able to identify the MIX in the path that caused the transaction to fail. We have two goals:

- Goal 1, **Identify failure**: If N_i fails to pass on a well-formed message within the allowed time to the next node N_{i+1}, then Alice can prove to any third party that N_i was the failing MIX (the completeness property).
- Goal 2, **Reject false claims**: No participant (including Alice) can claim that N_i failed to pass on a well-formed message to N_{i+1} in time, unless it really did fail (the soundness property).

Since these goals involve timing constraints, we first describe the timing model that we use. Then we proceed to show the specifics of constructing and sending a message, and how we can fulfil the completeness and soundness goals.

4.3 Timing Model

We consider time to be split into periods, each of length one time unit, corresponding to batches of messages sent by MIXes. The period numbered t lasts from time $t-1$ to time t. A message received by node N_i in period t will have timestamp t on the corresponding receipt signed by N_i. If N_i is not the

[1] As is common in MIX-net protocols, the encryption scheme should pad plaintexts to a constant length on decryption. The IND-CCA2 notion may need to be modified slightly to take this into account.

final recipient, the message must be sent to the next node N_{i+1} in period $t+1$. That is, the *deadline* for sending it to N_{i+1} is $t+1$.

All parties (MIXes, witnesses, senders of messages, and verifiers of failure claims) have clocks that are loosely synchronized, to within T_ϵ of the global time. Sending a message over a network is assumed to take T_{comm} in the worst case. Therefore, whenever a party needs to send a message by time t, it should attempt to do it before $t - T_\epsilon - T_{comm}$ according to its local clock, because its local clock may be slow, and because it must allow time for the network communication. If a party is expecting a message to be sent to it by time t, then it should allow it to be received as late as $t + T_\epsilon$ according to its local clock, since that clock may be fast. We include these timing considerations in the protocol descriptions.

The protocol requires several system-wide time constants:

- $T_{response}$ is the time that a node is allowed between receiving a message and providing a receipt.
- T_{margin} is the length of the interval before the deadline, during which a node will attempt to send a message via the witnesses, if it has not been able to obtain a receipt directly from the next MIX.
- T_{retain} is the time (number of periods) for which receipts are retained by MIXes. That is, if a MIX is asked for a receipt in period t, it need only provide it if the receipt has timestamp $t - T_{retain}$ or later. This will also determine the length of time for which failure claims can be directly verified.

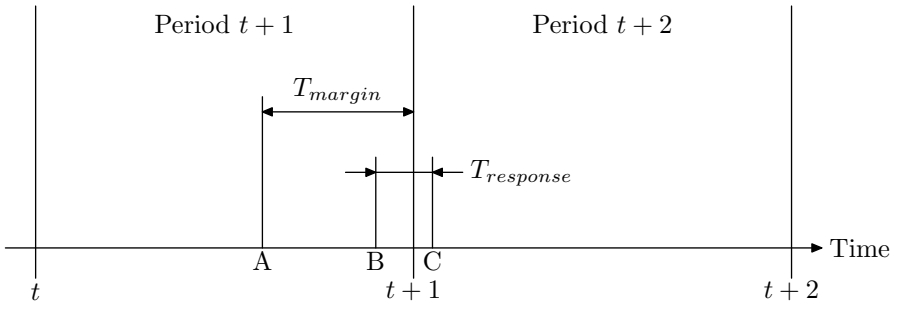

Fig. 2. Timing example

Figure 2 depicts an example time line:

- time A is the point at which N_i stops trying to send to N_{i+1} directly.
- time B is when a witness w tries to contact N_{i+1}.
- time C is the latest time at which N_{i+1} can respond to w with a receipt (this could also be before the deadline t+1).

Only global time is shown in the figure.

Unfortunately the timing constraints, although important to the correctness of the protocol, tend to obscure the description of how it works. You may wish to ignore them on an initial reading.

4.4 Transmitting a Message

This section describes the protocol for transmitting a message from Alice to Bob, using our MIX-net design with verifiable failures.

Procedure `Transmit`($Alice, Bob, Plaintext$):

1. Alice chooses $k-1$ MIXes $N_1, \ldots N_{k-1}$ to form a "MIX path". Let N_0 be Alice, and let N_k be Bob.
2. Alice picks k random seed values $r_1, r_2, \ldots r_k$.
3. Alice creates an initial packet M_1, defined as

$$M_1 = E_1^{r_1}(N_2, E_2^{r_2}(N_3, \ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots))$$

4. Let *now* be the current time according to Alice's local clock, and let $Deadline_1 = \lceil now \rceil$.
5. Try to send M_1 and $Deadline_1$ directly to N_1, waiting for a receipt.
6. If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } M_1, Deadline_1\text{"}) = 1$; if so, stop.
7. If no receipt is returned, set $Deadline_1 := Deadline_1 + 1$, and use the procedure `Hop-send`$(N_1, M_1, Deadline_1)$ below to send M_1 to N_1.

Under normal circumstances (i.e. assuming that all parties follow the protocol), the message will then be processed by $N_1, \ldots N_{k-1}$ as follows:

- N_1 reads M_1 from Alice, and processes it according to the procedure `Hop-receive`$(Alice, N_1, M_1, Deadline_1)$.
- N_1 decrypts M_1 to give (N_2, M_2), where

$$M_2 = E_2^{r_2}(N_3, \ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots)$$

- Let $Deadline_2 = Deadline_1 + 1$.
- N_1 uses the procedure `Hop-send`$(N_2, M_2, Deadline_2)$ to send M_2 to N_2.
- This process is repeated by N_2, which sends M_3 to N_3, and so on for N_3, N_4, …, N_{k-1}.
- Eventually, $E_k^{r_k}(Plaintext)$ is sent to Bob. Bob can decrypt this message, so for this case, the plaintext has successfully been transmitted.

Procedure `Hop-send`($N_{dest}, Message, Deadline$):

1. Try to send $Message$ and $Deadline$ to N_{dest} directly, waiting for a receipt.
2. If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } Message, Deadline\text{"}) = 1$.
3. If a valid receipt is not received before $Deadline - T_{margin} - T_\epsilon$ (by the sending node's local clock),
   (a) Let W be a set of witnesses.
   (b) Send "Witness: $N_{dest}, Message, Deadline$" to each witness $w \in W$ (causing `Witness` to be called on each w). Wait for any w to send back a receipt.
   (c) If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } Message, Deadline\text{"}) = 1$.
   (d) If no valid receipt is received, store any statements returned by the witnesses.

Note that an imposter witness or MIX may send fake receipts to the sender in order to try to confuse it. The sender should ignore any receipts that are not valid. If the sender receives more than one valid receipt, it need only store one (chosen arbitrarily).

Procedure `Hop-receive`($N_{src}, N_{dest}, Message, Deadline$):

1. Let *now* be the current time.
2. If $now > Deadline + T_\epsilon$ or $now < Deadline - 1 - T_\epsilon$, drop the message and respond with an error.
3. Otherwise, decrypt $Message$, and queue for transmission in the next period by `Hop-send`.
4. Send back the receipt $Sign_{dest}(\text{"Receipt: } Message, Deadline\text{"})$ to N_{src}.

Procedure `Witness`$(N_{src}, N_{dest}, Message, Deadline)$**:**

1. Let *now* be the current time.
2. (The witness must be sure that N_{dest} has time to respond:) If $now > Deadline - T_{comm} - T_\epsilon$ or $now < Deadline - 1 + T_\epsilon$, drop the message and respond with an error.
3. Try to send $Message$ to N_{dest} directly, waiting for a receipt.
4. If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } Message, Deadline\text{"}) = 1$. If so, send it back to N_{src}.
5. If a valid receipt is not received before $Deadline + T_{response} + T_\epsilon$, conclude that N_{dest} failed, and send back the statement "Failed: $N_{dest}, Message, Deadline$", signed by the witness, to N_{src}.

4.5 Identifying and Proving Failed Transactions

This section describes the method by which Alice proves the failure of a specific MIX to deliver a message, as well as the protection our protocol offers against a malicious Alice who either sends ill-formed messages or provides false claims.

Alice can prove her claim of failure to any chosen verifier (call him Victor). We will describe the duty of the verifier in our reputation system in Section 5.

Suppose that N_i, where $1 \leq i < k$, is the first node on the path that does not follow the protocol, i.e., it fails to handle a message M_i that N_{i-1} sends to it, within the allowed time. Alice wants to prove to Victor that N_i failed to process this message (which she might discover by a binary search over her MIX path).

Because N_{i-1} behaved according to protocol, it will have a receipt $Rcpt_i$ for the message M_i with deadline $Deadline_i$, where

$$M_i = E_i^{r_i}(N_{i+1}, E_{i+1}^{r_{i+1}}(\ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots))$$

As Alice knows all the random padding values, she can compute the message that N_i is supposed to send to N_{i+1}:

$$(N_{i+1}, M_{i+1}) = D_i(M_i) = (N_{i+1}, E_{i+1}^{r_{i+1}}(\ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots))$$

She suspects that this message was not actually sent to N_{i+1}, so she prepares a claim by obtaining the $Rcpt_i$ from N_{i-1}, and calculating M_{i+1} as above. Then Alice sends Victor "I blame N_i, claim: $r_i, N_{i+1}, M_{i+1}, Deadline_i, Rcpt_i$", which he verifies as follows:

Procedure `Verify-claim`$(N_i, r_i, N_{i+1}, M_{i+1}, Deadline_i, Rcpt_i)$**:**

1. Check that N_i and N_{i+1} refer to valid MIXes, and M_{i+1} is the correct length for an intermediate message.
2. Calculate $M_i = E_i^{r_i}(N_{i+1}, M_{i+1})$, and check that $Ver_i(Rcpt_i, \text{"Receipt: } M_i, Deadline_i\text{"}) = 1$.
3. Let *now* be the current time. If $Deadline_i + 1 < now - T_{retain} + T_\epsilon$, then it is too late for the claim to be verified, since N_i may legitimately have discarded its receipt; the claim is therefore rejected.
4. Send "Receipt request: N_{i+1}, M_{i+1}" to N_i.
5. If "Receipt response: $Rcpt_{i+1}, Timestamp$" is received from N_i within time $T_{response}$, such that $Ver_{i+1}(Rcpt_{i+1}, \text{"Receipt: } M_{i+1}, Timestamp\text{"}) = 1$ and $Timestamp \leq Deadline_i + 1$, then reject the claim.
6. If statements "Failed: $N_{i+1}, M_{i+1}, Timestamp$" signed by a sufficient set of witnesses are received, for some $Timestamp \leq Deadline_i + 1$, conclude that N_i made a reasonable attempt to send the message, and reject the claim. (See Section 4.7 for discussion of trust requirements on witnesses.)
7. Otherwise, conclude that N_i failed – either because it did not process the original message, or because it did not respond to the receipt request.

Proving that a delivery failure results in a proper claim

Sketch N_{i+1} will only give out a receipt for which $Ver_{i+1}(Rcpt_{i+1}, \text{“Receipt: } M_{i+1}, Deadline_i + 1\text{”}) = 1$, if it received the message M_{i+1} by time $Deadline_i + 1$. If it did not receive M_{i+1} by then, and under the assumption that N_{i+1}'s signatures cannot be forged, N_i will not be able to provide such a signature. Thus, Victor will conclude that it failed.

Node N_{i+1} may possibly be controlled by the adversary; in that case, it might provide a receipt on M_{i+1} in order to exonerate N_i, even though M_i was not actually processed. However, Alice can then use this receipt in the same protocol to attempt to prove that N_{i+1} failed. Therefore, the adversary will be able to choose which of the contiguous MIXes it controls can be proven unreliable. However, since there are only $k-2$ other nodes on the path, Alice will be able to prove that some node failed, after at most $k-1$ iterations of the protocol.

Because Alice can always prove that some MIX failed if it did in fact fail, we satisfy Goal 1, being able to identify failures in the MIX-net.

We note that knowledge of the fact that N_i and N_{i+1} are part of Alice's chosen path is *not* part of the view of a passive adversary in a normal MIX-net protocol. The closer N_{i+1} is to the end of the path, the more likely it is that this additional information will allow an adversary to link Alice with Bob (since the remaining path is effectively shortened to $k-(i+1)$ hops). In the worst case N_{i+1} may be Bob, which would reveal directly the fact that Alice was attempting to send a message to him. In order to avoid giving away this information, Alice may send all the messages needed for the failure proof over the MIX-net. When requesting receipts she can include a reply block, so that the node will be able to send back the receipt while Alice remains anonymous.

It may seem circular to rely on the MIX-net to send messages needed for a failure proof, when the goal of the proof protocol is to improve MIX-net reliability. However, since these messages are only used to prove failure and do not convey any other information, it is not critical that all of them are successfully transmitted. Alice can repeat any attempts to obtain receipts and to send the claim message to Victor as often as necessary, using random independent paths for each attempt. This will succeed quickly provided that the probability of a message making it through the MIX-net (from Alice) is not too small.

Proving that false claims are rejected

Sketch We wish to show that no participant can claim that N_i failed to pass on a well-formed message sent by Alice to N_{i+1}, unless it really did fail to send such a message. Without loss of generality, we will consider the adversary to be Alice.

Recall that Alice's claim to Victor is of the form “I blame N_i, claim: $r_i, N_{i+1}, M_{i+1}, Deadline_i, Rcpt_i$”. Victor then calculates $M_i = E_i^{r_i}(N_{i+1}, M_{i+1})$. Decrypting both sides, we obtain $D_i(M_i) = (N_{i+1}, M_{i+1})$, assuming N_i's key pair is valid.[2]

- Suppose Alice caused the message M_i to be sent to N_i by time $Deadline_i$, and then tried to claim that N_i failed. A well-behaving N_i will decrypt M_i to give the next hop and intermediate message (N_{i+1}, M_{i+1}). It will then try to send M_{i+1} to N_{i+1} using `Hop-send`. Either N_i will obtain a receipt for this message (signed by N_{i+1} and having the timestamp $Deadline_i + 1$), that refutes Alice's claim, or it will have signed statements from a sufficient set of witnesses (see Section 4.7) saying that N_{i+1} refused to provide a receipt when it was obliged to do so. In either case, N_i will be exonerated.

[2] We can assume that N_i's public key is valid because it is chosen by N_i, who would have no incentive to generate an invalid key. We omit consideration of an N_i controlled by Alice that purposely chooses an invalid key, because then Alice only damages the reputation of a node she controls.

– Suppose Alice did not cause M_i to be sent to N_i by $Deadline_i$. In order to make a credible claim, Alice needs a receipt $Rcpt_i$ such that $Ver_i(Rcpt_i, \text{"Receipt: } M_i, Deadline_i\text{"}) = 1$ (since Victor will check this). However, if N_i did not receive M_i, it will not have given out any such receipt, and so assuming that N_i's signatures cannot be forged,[3] it will be exonerated.

Another way in which a node could be falsely accused of failure, is if there are "sufficient" witness statements against it as defined in Section 4.7. We assume that this does not occur, because the adversary is not able to subvert witnesses in the core group.

Therefore, we satisfy Goal 2. Note that the above argument covers the case in which Alice attempts to send messages to N_i that either do not decrypt, or decrypt to ill-formed plaintexts, since we have proven that for Alice's claim to be accepted, N_i must have received a message with a well-formed (N_{i+1}, M_{i+1}) pair as the plaintext.

The above discussion does not take into account replay attacks. Many MIX-net protocols require MIX nodes to refuse to pass on a message if it is a *replay*, that is if the MIX had already received that message within a given time period (see [7] for a rationale). We do not want nodes to lose reputation because of this behavior.

Our protocol is able to handle this case because when a MIX receives a replayed message (using `Hop-receive` as normal), it will already have a receipt from the previous time it sent that message (or else witness statements showing that it tried to send it). Steps 5 and 6 of `Verify-claim` show that it can provide the earlier receipt or statements when challenged with a failure claim. We define the length of time for which replays are remembered to be the same as T_{retain}. To make sure that it is able to respond to future claims, whenever a node receives a replayed message, it should record that it needs to store the receipt or statements that were obtained for the earlier message, for a further T_{retain} periods.

4.6 End-to-End Receipts

The protocol described above assumes that Alice has some way of determining whether her message has failed. In some cases, such as posts to a public forum like Usenet, Alice (and any verifier) can easily see whether the message was posted correctly. For the remaining cases, if we use a MIX-net design that supports reply blocks or another method for two-way communication, it is possible for Bob's software to immediately send an *end-to-end receipt* back through the MIX-net to Alice, as soon as he receives a message.

Note that reply blocks used for end-to-end receipts only need to be used once, and the receipt is sent immediately. This allows some of the security problems otherwise associated with reply blocks, pointed out in [7], to be eliminated.

Since a failure can also occur on the return path, we would like Alice to be able to prove these failures as well. That is, if Alice does not get an end-to-end receipt within a specific time, she will be able to prove one of three cases: a MIX on the forward path failed; a MIX on the reply path failed; or Bob failed to send a receipt.

With careful design of message formats, it is possible to make receipt messages indistinguishable from forward messages to nodes on the reply path. In that case, the same protocol used to claim failures in sending forward messages will be applicable to reply messages. That is, the path can be treated as a loop from Alice to Bob, then back to Alice. However, we do not attempt to explain details of message formats in this paper.

[3] We take the position that if N_i's private key has been compromised, it should be considered to have failed, and so it does not matter that an adversary who knows the private key can "prove" that N_i failed to pass on any message.

4.7 Trust Requirements for Witnesses

Dishonest witnesses do not compromise anonymity, but they can compromise reliability. Witnesses could either refuse to give a statement or receipt for a MIX that has failed, or make false statements in order to frame a MIX that has not in fact failed.

If users of the MIX-net were prepared to trust an arbitrary set of witnesses chosen by a sending node, then it would be easy for an adversary to perform the latter attack. Therefore, we suggest defining a core group of witnesses who are relatively widely trusted. If some threshold number of this core group provide statements implying that a node N_{i+1} has failed (and that its predecessor on the path N_i has not), that would be considered *sufficient* for the purposes of `Verify-claim`.

This solution is not ideal, because specifying a fixed (or even slowly changing) group of witnesses runs counter to the goals of a truly decentralised protocol. If the messages sent to this group are also published, though, then it is possible for other parties to duplicate their actions, thus gaining confidence that they are behaving honestly.

5 Reputation Systems

Reputations have been suggested as a means of improving MIX-net reliability [12, 14]. To date, discussions of reputation systems in the context of MIX-nets have been vague. We now specify what we mean by a reputation system in the context of a MIX-net, and investigate a specific design to help solve the reliability problem.

As stated earlier, the major parties in the overall MIX-net system are Alice (the sender), Bob (the recipient), and the MIX nodes. In layering on a reputation system, we add two more agents to the system. *Raters* are entities who make observations about the performance or honesty of MIXes. In our case, the raters are both the sender Alice and any MIXes that make use of the witnesses. *Scorers* are entities who tally observations from raters and make these tallies (or *scores*) available. For convenience and simplicity, we choose to give the scorer the duties of verifier and witness as well.

5.1 Requirements for Our Reputation System

We have a variety of goals for developing a reputation scoring system for a MIX-net. Senders must be able to configure their software to automatically take into account the scores for each MIX; indeed, current client software such as [19] already includes this support. Any user of the MIX-net must be able to contribute *ratings* (observations about MIX behavior). The system must resist attempts of any adversaries to influence scores in a way that does not "reflect reality"; but at the same time, to maintain efficiency, security, and reliability, no single server should be a bottleneck anywhere in the protocol.

The scoring system must be verifiable. That is, scorers need some way to determine the credibility of ratings, and other users need some way to verify that scorers are tallying ratings correctly. The scores themselves must be accurate — clients must be able to draw conclusions from scores that lead to "good" predictions. The overall scoring algorithm must be dynamic, recognizing and reflecting recent trends in MIX performance.

While achieving all of these goals, the system must also maintain the level of anonymity provided by the MIX-net.

5.2 Reputation System Overview

Section 4 outlined a technique by which Alice can confirm the failure of a specific MIX to forward her message, and communicate this knowledge to some third party verifier.

We introduce a set of scorers, each named Sally. Each of these scorers keeps her own database of performance scores for MIXes in the system. When she receives a failure claim as detailed in Section 4, she verifies it, and updates her score database if necessary. Scorers also send test messages to distinguish reliable MIXes (few delivery failures because they actually deliver messages well) from new MIXes (few delivery failures because nobody has sent any messages through them yet). Sally makes available her database of MIX reputations, so that Alice's client software can download it and use it to automatically choose paths. Client software can be configured to only choose MIXes with a minimum number of successful tests; after that, MIXes are rated by the number of verified delivery failures.

If we simply count the number of messages that each MIX drops, an effective attack would be to constantly add new unreliable MIXes. Instead, we describe scores as two numbers: the first is the count of negative ratings as described above, and the second is a "minimum number of positive ratings" requirement, which is a threshold configurable on the client side.

5.3 Tallying negative ratings

As we showed in Section 4.5, each negative rating really does represent a failed delivery. That is, if a valid message onion is sent to N_i, then it really is N_i's fault if he doesn't either deliver the decrypted onion to N_{i+1} within the allowed time window, or get statements from credible witnesses saying that N_{i+1} was down. Anybody who is able to demonstrate that it was a valid onion is either that MIX or the onion's author; the MIX has no incentive to publicize that it failed at a transaction.

Thus, if our assumptions hold, there is no way to spoof negative ratings. Note that an adversary may be able to force negative ratings on a MIX, while goal 2 still holds: if he floods that MIX's incoming bandwidth (either directly or via witnesses), the MIX will no longer be able to sustain the load. However, this point is exactly the point where the MIX is demonstrating the fact that it is unreliable. Causing MIXes to lose reputation in the face of *successful* flooding is consistent with our scoring system goals; after all, the scoring system measures reliability and capabilities, not intent.

5.4 Tallying positive ratings

Positive ratings differ from negative ratings in that they can be easily faked. Specifically, an adversary can make up a set of messages, each using a MIX path entirely owned by him, and then simply generate transcripts for those messages from the same machine that built each onion.

Algorithms that consider the number of positive ratings (either by adding positive and negative ratings, or by taking the ratio of positive ratings to negative ratings) are generally going to be vulnerable to this sort of shilling attack.

However, using only the number of negative ratings to calculate reputation score gives a perfect score to completely untested MIXes. We need some way to confirm that positive ratings actually reflect a MIX's ability to successfully deliver Alice's message in the future.

5.5 Approaches to increase confidence in positive ratings

One approach to making positive ratings more reliable (and thus more meaningful) is to build a graph based on rater credibility such as that employed by Advogato [11]. Similar to the PGP web of trust, this technique builds a directed graph where edge weights represent trust that one vertex places in the other; in our case, each vertex would be a MIX. Calculating maximum flow from some node A in the graph to some node B shows how much trust A places in B, even if A never directly rated B.

The Advogato trust metric aims to reduce the damage that an adversary can cause by *pseudospoofing* – creating a multitude of identities each controlled by that adversary. At first, each of the new nodes

is connected only to itself in the graph. Even after the pseudospoofing nodes begin to form connections with the rest of the graph, there will still be "trust bottlenecks" that limit the amount of trust arriving at any of the pseudospoofing nodes.

Another approach is to treat reputation as a probability: an estimate of the expected outcome of a transaction with that MIX. Scores might simply be the sum of ratings, normalized and weighted by the credibility of raters.

Other designs employ neural networks or data clustering techniques to apply non-linear fitting and optimization systems to the field of reputation. However, such complex solutions may not be easily understood or verified by the users of the reputation score. Moreover, to our knowledge these more statistical approaches to aggregating reputation values have never been studied in the context of adversaries who understand and attempt to exploit the algorithm. Such evaluation techniques which are intended for "clean" environments can introduce serious vulnerabilities and complications, which in turn require rater credibility tracking systems to track which raters are providing "good" data.

We solve the positive rating credibility problem by having each Sally produce positive ratings herself — after all, if Sally sends the test messages herself, she knows that they are unbiased ratings. MIXes that reliably and correctly process her message will earn positive ratings that Sally knows to be accurate.

In order to ensure that positive ratings are accurate, test messages should be indistinguishable from real messages (otherwise, an adversary could behave reliably for test messages, but unreliably for real messages). In addition, nodes should not be able to determine that the sender of a message is a rater; otherwise the MIX could decide to behave well only for a small set of senders and badly for everyone else. This is similar to the problem restaurant critics face when attempting to determine a restaurant's "real" level of service.

It will normally be possible for an adversary to guess whether a message has been received directly from a sender (i.e. the adversary is the first hop on the path, N_1), or if it is being sent to the final recipient (i.e. the adversary is N_{k-1}). Unfortunately, it is difficult to produce simulated messages that are completely indistinguishable from real messages in these cases. We do not have a fully satisfactory solution to this for positive ratings; instead, we rely on negative ratings to expose an adversary that behaves unreliably only when it is the first or last hop.

Sally should expire her memories of transactions after a certain amount of time so that failures in the distant past do not haunt a MIX forever. Similarly, MIXes need to have a threshold of recent successes in order to stay "in the running". Alice configures her client software to choose only among MIXes that have a specified minimum number of positive ratings. Out of this pool, she weights the MIXes that she chooses for her path based on the number of verified delivery failures that have been observed for this MIX.

This system reacts quickly to a decrease in reliability of a MIX. If a MIX has a high reputation, it is likely that there are many users routing messages through it. If it suddenly stops delivering messages, these users will notice these failures, and a series of negative ratings will quickly arrive. This negative feedback process serves to stabilize the system and help the scores reflect reality.

5.6 Redundancy and verifiability of scorers

If Alice keeps a tally herself of her transactions (and she can, by remembering which mails had a corresponding end-to-end receipt), then she can build her own score table in which she is confident of both the positive ratings and the negative ratings. Every so often she might compare her score tables with those of the available Sally's. This process allows Alice to "test the waters" herself and weakens the trust requirements on scorers – Sally is more accountable for her scores because Alice can also compute scores and compare.

Indeed, if claims for dropped messages are published, anybody can verify them. Thus Alice might keep track of negative ratings for a few weeks, then compare with Sally to determine if Sally's scores are actually reflecting all of the negative ratings.

5.7 Implications for Traffic Analysis

The addition of witnesses to the protocol may introduce new attacks. Direct communications between nodes can use link encryption, but encrypting the messages to witnesses would have little benefit (and would be incompatible with publishing these messages, as suggested in Section 4.7). So if an adversary can force the witnesses to be used instead of direct communication, this may weaken the security of the network. On the other hand, link encryption of messages between nodes is not strictly necessary to achieve anonymity in a MIX-net; it is only a "belt-and-braces" measure.

The reputation system introduces new attacks as well. For instance, Eve might gain a high reputation and thus get more traffic routed through her MIX, in order to make traffic analysis easier. In a system without reputations, the way to purchase more traffic to analyze is not so clear; now it is simply a matter of maintaining a reliable MIX. In addition, the adversary now has incentive to degrade or sabotage the performance of other nodes in order to make his relative reputation higher. This kind of attack was described by RProcess as "selective denial of service": the bad guys want traffic to go through their nodes, so they ensure that all other nodes are less reliable [20]. As recent distributed denial of service attacks demonstrate, crippling an Internet host can be easy. Scorers must expire ratings promptly enough to prevent an adversary from easily tarnishing the reputations of all other MIXes.

On the other hand, we may well be making the system *more* anonymous by making it more reliable. Currently, users may have to send a message many different times, through many different paths, before it successfully arrives at its destination. These re-sends of the same message offer more information to traffic analyzers. A better theoretical framework for traffic analysis needs to be established before we can make any quantifiable statements about the implications of the proposed protocol.

6 Quantifying MIX Reliability

We now turn our attention to quantifying MIX reliability in order to measure the improvement in reliability provided by reputation systems. It seems intuitively clear that using Levien-style statistics to pick remailers gives a message a better chance of reaching its destination safely. Why would we need to introduce a formal way of quantifying this information and its result? There are at least three answers to this:

1. To confirm our intuition. Cryptography, especially anonymous systems, is a surprising field.
2. To offer a means to compare two different proposals for MIX-nets based on a quantitative measure of the reliability improvement they provide.
3. To better understand why and how reputation systems work. In particular, to gain insight into what could happen if a reputation system provides incorrect reputations (through stupidity or malice).

We now present a model and metric which will fulfil the first of these goals. The second and third are left for future work.

6.1 A Proposed Reliability Metric

A *reliability metric* is a quantitative measure of the reliability of a MIX-net. We now propose a reliability metric that is the expected value of the probability that a message reaches its intended receiver.

Recall from section 3.2 that to send a message, a sender $s \in S$ picks a recipient $r \in R$ according to the distribution R_s, and then picks a MIX path, $path \in Paths$ according to the distribution $Path_{s,r}$.

For simplicity, we restrict ourselves to a single receiver at a time with MIX paths that are exactly k nodes long; this means that R_s takes on single values $r \in R$ while $Path_{s,r}$ takes on values that are sequences of length k in $Paths$.

We consider an adversary that corrupts MIXes and causes them to fail. Every $mix \in MIXes$ is either *good* or *bad*. Good MIXes work normally (but may fail due to error), and bad MIXes are controlled by an adversary. Good MIXes have probability of failure p_{good} and bad MIXes have probability of failure p_{bad}, where "failure" means that a message fails to be passed on to the next hop. An *adversary* then controls the subset $BAD \subset MIXes$. We note that as stated here, the adversary does not have a memory and does not adapt to previous behavior; both of these are major simplifying assumptions which should be removed in future work.

We use the simple metric that the reliability of the MIX-net is the expected probability of V defined as the event "a message survives the MIX-net" – that is, the probability that none of the nodes on the MIX path picked by the sender fail. We further define V_s for "a message from sender s survives the MIX-net" and $V_{s,r}$ for "a message from sender s to recipient r survives the MIX-net."

We can evaluate the expected probability of V by using the law of total expectation:

$$E(V) = E(E(V_s)|\texttt{sender} = s) = E(E(E(V_{s,r})|R_s = r)|\texttt{sender} = s)$$

That is, we can compute $E(V)$ by first computing the probability of survival for a single message sent from a sender to a receiver, then averaging and summing over all senders and all receivers.

6.2 MIX Reliability With No Reputation System

We now compute the reliability metric on a sample MIX-net with no reputation system whatsoever. We assume that there is no information available on which MIXes are good and which are bad, and so each client picks MIXes randomly (with replacement[4]); that is, for all s and r, $Path_{s,r}$ is a discrete uniform random variable. We make the simplifying assumption that senders choose from all receivers uniformly at random, so for all s, R_s is a discrete uniform random variable. Finally, we set $p_{good} = 0$ and $p_{bad} > 0$; this corresponds to the simplifying assumption that good nodes are always reliable, while bad nodes have some constant probability of failure.

Suppose the adversary controls a randomly chosen subset BAD of the m MIXes, with $|BAD| = q$. Then when picking a MIX path to a receiver, a sender has chance $\frac{q}{m}$ each time it picks a MIX of picking a bad MIX. The probability that each MIX it chooses will fail is $p_{bad} \cdot \frac{q}{m} + p_{good} \cdot (1 - \frac{q}{m}) = p_{bad} \cdot \frac{q}{m}$. Therefore, the chance of picking no MIXes that will fail is $(1 - p_{bad} \cdot \frac{q}{m})^k$, since the sender picks k MIXes. Because a message survives the MIX-net if and only if no MIXes on the path fail, this gives us $E(V_{s,r}) = (1 - p_{bad} \cdot \frac{q}{m})^k$.

Given that every receiver has the same $E(V_{s,r})$ and that receivers and senders are picked uniformly, we also have $E(V) = E(V_s) = (1 - p_{bad} \cdot \frac{q}{m})^k$.

Notice that, assuming $p_{bad} > 0$ and $q > 0$, the reliability of the MIX-net drops off exponentially as the number of hops in the MIX path increases.

6.3 MIX-net Reliability with Reputation Systems

We now show that a reputation system improves the reliability of a MIX-net in our model. The only quantities that change are the distributions $Path_s$. A reputation system will give each sender s some

[4] Picking MIXes with replacement is not an ideal strategy from the point of view of maintaining anonymity, because it gives a slightly increased probability that all or nearly all of the MIXes are controlled by an adversary, compared to picking k distinct MIXes. There is little difference if the number of MIXes m is large, however.

information about which MIXes are good and which are bad. If all goes well, s can then adjust its choice of MIX paths to avoid the bad MIXes.

In order to study this adjustment, we make several simplifications. First, time is divided into two phases called the "observe phase" and the "advice phase." In the observe phase, the raters send test messages, such that each MIX node is queried at least $n \geq 1$ times by an honest rater. The information from these queries is collected and made available by the scorer. Then in the advice phase, we assume that all senders adjust their choice of MIXes based solely on the advice of the scorer.

In the observe phase, we assume that raters make queries independently of each other by sending test messages through the network. If a test message survives, the rater should make a *positive report* for all the MIXes on that path. If the test message is dropped, the rater should determine which MIX failed as described in Section 4.5, and make a *negative report* for that MIX.

After the queries are complete, the scorer makes the reports available to all senders. Each sender restricts consideration to MIXes that have at least a given number of positive reports, and from those picks the MIXes with the least number of negative reports. In our example situation where $p_{good} = 0$, all negative reports will be for bad MIXes. We assume that n positive reports are sufficient to satisfy the minimum requirement, so that all the MIXes will be used.

A single query to an adversary-controlled MIX is enough to establish a negative report and disqualify it from MIX path selection. If n queries are made to each MIX, then the probability of each bad MIX remaining without any negative reports is $(1 - p_{bad})^n$. (However, it can be argued that such a MIX is not really "bad" at that point, since it has not yet failed.)

Recall that we found the chance that a sender picks a good MIX path to be $(1 - p_{bad} \cdot \frac{q}{m})^k$. Now in the "advice phase," instead of q bad MIXes picked from a pool of m, an expected $q \cdot (1 - p_{bad})^n$ bad MIXes will be picked from a pool of $m - q + q \cdot (1 - p_{bad})^n$. Following the analysis in the previous section, we see that $E(V)$ becomes $(1 - \frac{p_{bad} \cdot q \cdot (1 - p_{bad})^n}{m - q + q \cdot (1 - p_{bad})^n})^k$. Provided that at least one bad MIX was eliminated and $q < m$, this is an improvement.

Therefore we have shown in this simplified model that honest reports yield an improvement in MIX-net reliability. Note that the first part of this paper is precisely about how to ensure reports are honest, so that we can assume honest reports at this point. There are several directions we could go from here to make this a more realistic analysis:

- Investigate other reliability metrics that may be more natural or precise.
- Investigate the cases when $p_{good} \neq 0$, and where p_{bad} is different for the first or last hop of a path.
- Investigate non-uniform distributions for each of the random variables involved.
- Investigate a unifying probability of failure p_f for all nodes, instead of having disjoint "good" and "bad" subsets. Furthermore, investigate the general case of some probabilistic distribution for failures with non-zero variance, for system-specific reputation functions.
- Bring the adversary model closer to reality.
- Treat anonymity and reliability simultaneously, with an eye to "how many people use the service to provide cover." One way to model this might be to set the number of users as a function of the reliability of the MIX-net.

7 Network Flow Analysis for MIX-nets

We have just described a simple metric for measuring MIX-net reliability on a per-node basis. In reality, a reputation system provides a feedback mechanism to account for dynamic network reliability. As users engage in the MIX-net protocol and report failures accordingly, reputations are continually revised based on this ongoing accountability mechanism. Therefore, this feedback provides an element of learning for path selection.

From a graph theoretic standpoint, the MIX path selection process can be modelled as a multicommodity unsplittable flow problem. Multicommodity flow generalizes a maxflow problem by allowing multiples sources and sinks, which can transmit arbitrary types of flow, called *commodities*. The problem is complicated in that our flows are *unsplittable* – flows are onion-encrypted messages, sent along paths pre-defined by the sources and thus must be transmitted as integral units.

While finding an exact solution to fractional multicommodity flow is in P, obtaining *integral* solutions is NP-hard. Furthermore, network flow problems generally require global, exact knowledge and centralized control of flows. This is not realistic for a decentralized MIX-net. We must contend with imperfect knowledge of bandwidth (edge capacity) and the costs associated with an edge. We may approximate the cost of an edge (s, t) as equal to its destination's probability of failure $p_f(t)$; still, reputation measurements may lag behind current network state.

We turn to approximation algorithms to solve such multicommodity UFP in *practically* efficient polynomial time. A classic technique, Raghavan's rounding algorithm [18], may be used for this general class of flow problems, provided that maximum demands are logarithmically smaller than the minimum capacity of a path. For client-selected MIX paths, the users themselves could solve individual flow problems, with scorers' reputations only serving as an input to their calculations. This decentralized solving mechanism offers a challenge for traditional multicommodity flow problems. We leave this for future work.

8 Conclusion and Future Directions

We have described a more robust reputation system for remailer networks, based on a MIX-net design that employs receipts for intermediate messages. This reputation system improves the reliability of the overall network over a random MIX path, and is less susceptible to service denial than other remailer reputation solutions such as statistics pages. There are a number of directions for future research:

- The protocol as we have described it is still very rough; actual deployment and testing of the MIX extension and reputation system would let us see how well it really works.
- More analysis of our model and our reliability metric, as described in the previous section, could lead to better reputation systems. Indeed, a parallel model characterizing efficiency of a MIX network might be very enlightening, especially from a network flow optimization viewpoint.
- Can we achieve some further measure of reliability (or resource management) through the use of electronic cash or similar accountability measures? Indeed, the NymIP group [5] has a long list of similar improvements that need to be further analyzed and developed.
- Can we make a reputation system that is both efficient and universally verifiable? Currently, only Alice can claim and prove that a message did not reach its destination. Can we apply zero-knowledge proofs so that Alice does not leak any information about the next hop while claiming failure, yet remain practically efficient? Can we extend this so that anyone can detect a failed MIX node?

This paper provides a foundation for further analysis of MIX-net reliability and reputations. Our model and reputation system are designed to be simple and easily extensible. Much work remains in a wide variety of directions before a reliable, secure, and ubiquitous remailer network can be put in place.

Acknowledgements

We'd like to thank Nick Mathewson and Blake Meike for their help with the reputation system; Dave Andersen and Robert Zeithammer for interesting discussions concerning the network flow problem; Anna Lysyanskaya, Chris Laas, Nick Feamster, and Mark Waldman for various intelligent discussions; and our anonymous reviewers for their many useful comments.

References

1. Masayuki Abe. Universally verifiable MIX with verification work independent of the number of MIX servers. In *Advances in Cryptology - EUROCRYPT 1998, LNCS Vol. 1403*. Springer-Verlag, 1998.
2. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. Extended abstract in *Advances in Cryptology - CRYPTO '98*, LNCS Vol. 1462. Springer-Verlag, 1998. Full version available from <http://www-cse.ucsd.edu/users/mihir/>.
3. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982.
4. Yvo Desmedt and Kaoru Kurosawa. How to break a practical MIX and design a new one. In *Advances in Cryptology - EUROCRYPT 2000, LNCS Vol. 1803*. Springer-Verlag, 2000.
5. John Bashinski et al. The NymIP effort. <http://nymip.velvet.com/>.
6. Electronic Frontiers Georgia (EFGA). Anonymous remailer information. <http://anon.efga.org/Remailers/>.
7. C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Security Symposium - NDSS '96*. IEEE, 1996. <http://cob.isu.edu/sndss/sndss96.html>.
8. J. Helsingius. `anon.penet.fi` press release. <http://www.penet.fi/press-english.html>.
9. Markus Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM, 1999.
10. D. Kesdogan, M. Egner, and T. Büschkes. Stop-and-go MIXes providing probabilistic anonymity in an open system. In *Information Hiding Workshop 1998, LNCS Vol. 1525*. Springer Verlag, 1998.
11. Raph Levien. Advogato's trust metric. <http://www.advogato.org/trust-metric.html>.
12. Tim May. Cyphernomicon. <http://www2.pro-ns.net/~crypto/cyphernomicon.html>.
13. Tim May. Description of early remailer history. E-mail archived at <http://www.inet-one.com/cypherpunks/dir.1996.08.29-1996.09.04/msg00431.html>.
14. Tim May. Description of Levien's pinging service. <http://www2.pro-ns.net/~crypto/chapter8.html>.
15. Joel McNamara. Private Idaho. <http://www.eskimo.com/~joelm/pi.html>.
16. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
17. M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
18. P. Raghavan and C. Thompson. Randomized rounding. *Combinatorica*, 7:365–374, 1987.
19. RProcess. Potato Software. <http://www.skuz.net/potatoware/>.
20. RProcess. Selective denial of service attacks. <http://www.eff.org/pub/Privacy/Anonymity/1999_09_DoS_remail_vuln.html>.
21. Zero Knowledge Systems. Freedom version 2 white papers. <http://www.freedom.net/info/whitepapers/>.