A Reputation System to Increase MIX Cascade Reliability

Roger Dingledine[1] and Paul Syverson[2]

[1] Reputation Technologies, Inc. (arma@reputation.com)

[2] Naval Research Lab (syverson@itd.nrl.navy.mil)

Abstract. We describe a MIX cascade protocol with a reputation system to increase the reliability of a network of MIX cascades. MIX nodes choose a bandwidth level to contribute, and each day generate a communally random seed to determine their configuration. Nodes send test messages to monitor their cascades. Senders can also demonstrate message decryptions to convince honest cascade members that a cascade is misbehaving. By allowing any node to declare the failure of its own cascade, we eliminate the need for global trusted witnesses.

1 Introduction

Increasing the reliability of an anonymous communication system gives better efficiency and stronger anonymity. The system becomes more efficient because senders use routes which are more likely to succeed, and provides stronger anonymity both because senders need to resend fewer messages, and because a more reliable system draws more users and thereby increases anonymity sets.

Past approaches to increasing remailer reliability have included writing more reliable software [23], building MIX protocols that give provable robustness guarantees [6, 11, 17], and building a reputation system to let users choose paths based on the published scores for each node [7].

The reputation system described in [7] uses a MIX-net in which nodes give receipts for intermediate messages. These receipts, together with a set of witnesses, allow senders to verify the correctness of each node and prove misbehavior to the witnesses. Here we investigate and solve two problems from that design:

- *Its mechanism for verifying a failure claim requires a set of global witnesses, a threshold of which need to be involved in confirming every failure claim. These global witnesses represent both a trust bottleneck and a communications bottleneck.* Our new reputation system avoids these bottlenecks by making each node a witness to its own cascade.
- *An adversary wanting to do traffic analysis can gain a high reputation to get more traffic.* We protect against such adversaries by choosing from a pool of "acceptable" MIX nodes, and building cascades so that we can argue bounds on the probability that an adversary will control an entire cascade.

2 Overview and Threat Model

We aim to improve the reliability of anonymous communication systems, enabling their practical use in fields such as financial cryptography where transactions need to be efficient and reliable. We order MIX nodes into cascades, where each cascade presents a fixed path through the network. This approach allows us to decentralize the use of witnesses to detect failure as introduced in [7], since each node can witness the traffic through its own cascade. Further, using cascades allows us to better resist traffic analysis from a pervasive adversary, since sending traffic through fixed paths makes intersection attacks more difficult.

Specifically, we aim to defend against two adversary goals. An *anonymity-breaking* adversary tries to discover linkability between sender and receiver; to identify the sender or receiver of a given message; or to trace a sender forward (or a receiver backward) to any messages. A *reliability-breaking* adversary tries to deny service to users. We assume that our adversary can passively watch all traffic, and can delay, modify, or insert some messages. We also assume that the adversary has compromised some fraction of the participating MIXes.

Section 4 shows the protocol by which MIXes build themselves into cascades in a public and verifiable way, and also describes the process of generating *communal randomness* so MIXes don't need to trust a central authority to build the cascades. We periodically rebuild cascades to reflect changes in reliability.

Inspired by a comment by Jim McCoy about using reputation capital to regulate participants in DC-nets [16], we use a very simple reputation system. Any member of a cascade can declare its own cascade to have failed. Nodes in a successful cascade each gain one reputation point, whereas all nodes in a failed cascade lose one point. By reporting cascade failure, misbehaving nodes damage their own reputations too. We describe this reputation system in Section 5, including some proposals for limiting the fraction of adversary-controlled nodes, a discussion of some pitfalls introduced by our simple reputation system, and our technique for building cascades to reduce the chance that an adversary will control an entire cascade.

Section 6 describes our modified MIX cascade protocol, and also specifies how MIXes can decide when their cascade has failed. In Section 7, we describe a variety of attacks on the system and how well our design withstands these attacks. Finally, we close in Section 8 with a discussion of possible directions for future research.

3 Related Work

3.1 MIX-nets

Chaum introduced the concept of a MIX-net for anonymous communications [5]. A MIX-net consists of a series of servers, called MIXes (or MIX nodes), each of which is associated with a public key. Each MIX receives encrypted messages, which are then decrypted, batched, reordered, stripped of the sender's name and identifying information, and forwarded on. 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.

One type of MIX hierarchy is a cascade. In a cascade network, all users use one or a few fixed paths through the MIX-net. Cascades can provide greater anonymity against a large adversary [3], because in a free-route system an adversary who owns many of the MIXes can use intersection attacks to dramatically reduce the set of possible senders or receivers for a given message.

Current research directions on MIX-nets include stop-and-go MIX-nets [13], distributed flash MIXes [11], and hybrid MIXes [18]. MIX cascade research includes real-time MIXes [12] and web MIXes [2].

3.2 Robustness and reliability in MIX-nets

Previous work primarily investigates the *robustness* of MIX-nets in the context of a distributed MIX system [11]. 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 zero-knowledge proofs of correct computation. Jakobsson showed how to use precomputation to reduce the overhead of such a MIX network to about 160 modular multiplications per message per server [11], but the protocol was later found to be flawed [17] by Mitsumo and Kurosawa. Desmedt and Kurosawa's alternate approach [6] requires many participating servers. Abe's MIX [1] provides *universal verifiability* in which any observer can determine after the fact whether a MIX cheated, but the protocol is still computationally expensive.

Reliability differs from robustness in that we do not try to ensure that messages are delivered even when some nodes fail. Dingledine et al's reputation system for free-route MIX-net reliability [7] aims instead to improve a sender's long-term odds of choosing a MIX path that avoids failing nodes. This work similarly attempts to provide more reliable long-term service by identifying reliable MIXes and building cascades from them.

We note that reliability and robustness can be composed: a cascade or distributed MIX with robustness guarantees can be considered as a single node with its own reputation in a larger MIX-net.

3.3 Remailer Statistics

Levien's statistics pages [15] track 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. Such reputation systems improve the reliability of MIX-nets by allowing users to avoid choosing unreliable MIXes. The Jack B Nymble 2 remailer client [22] 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.

Whereas a reputation system in a free-route MIX environment primarily enables users to choose better routes as above, a reputation system for a MIX cascade system instead acts behind the scenes to build the cascades in a reliable manner — this process may even be entirely transparent to the users.

3.4 Approaches to MIX-Net Reliability

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," yet the resulting protocols may be complicated and inefficient.

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. Unlike the above approach, we are not likely to prove strong theorems. The goal of reputation is to make the system "better" without guaranteeing perfection.

Reliability via protocol is the most well-studied approach, while reliability via reputations in the form of Levien statistics is the most widely used. Our work combines the two approaches: we modify the MIX-net protocol to support easier detection of MIX failures and then specify a suitable reputation system.

4 How to randomly self-build cascades

We periodically rearrange the nodes in each cascade, both so the cascades reflect recent changes in reliability, and so nodes in failed cascades can get back in a working cascade. We rearrange *all* the nodes, not just the ones from cascades that failed; otherwise reliable nodes will concentrate in stable cascades and unreliable nodes in unstable ones, making it difficult for new good nodes to gain reputation.

Cascades rebuild with period T. By $T-a-b$, each participant sends his sealed commitment to the Configuration Server (CS). At $T-a-b$, the CS publishes the set of commitments. By $T-b$, participants should reveal to the CS. At T, the CS publishes the set of reveals, along with the configuration of cascades for that round. Observers can verify that the CS followed the configuration scheme and used the contribution from each node.

The CS gives each p a receipt $[commitment, timestamp]_{K_{CS}^{-1}}$ which p can use as proof that he should be included in the next cascade configuration. If p is worried that CS will ignore his commitment (not provide a receipt), he can use a certified mail delivery system [20, 25] to convince other people that CS is misbehaving. The period from $T-a-b$ to $T-b$ should be long enough that anyone within reasonable clock skew can verify that the commitment phase has truly closed. Adequate time must be allowed for nodes to submit their secrets and to make use of a certified delivery service if their submissions are not accepted and receipts provided by the CS in a timely manner. The CS also provides a receipt for the reveal, and again p can use certified mail if desired.

Commitment from p: $[p, \texttt{IP}, \texttt{port}, \texttt{bandwidthpledge}, \texttt{tsbc}(\texttt{rand}(p))]_{K_p^{-1}}$.

Reveal from p: $[p, \texttt{IP}, \texttt{port}, \texttt{rand}(p)]_{K_p^{-1}}$

Here "$\texttt{tsbc}(\texttt{rand}(p))$" is p's temporarily-secret bit-commitment to his random value. The CS will then build a new set of cascades for that T, arranged according to an unpredictable value communally generated by the participating mixes. Thus, no one can control the configuration of cascades. Bandwidth might be divided into four categories: 10Kb/s, 100Kb/s, 1Mb/s, 10Mb/s. Participants choose exactly one of these values for their bandwidth pledge, and each of those four buckets will be formed into cascades according to an algorithm described in Section 5 and using the unpredictable communal value.

The set of cascades is determined by the communal value that is publicly verifiably committed when the CS signs and posts the commitments and their revealed values. Because a node could still control the resulting communal value by choosing whether to reveal its value, the commitment is done such that if it is not revealed, the CS can uncover it after a predictable amount of computation. Any committed value that is not revealed by $T - b$ should be computed by the CS to be revealed at T.

The communal value can be used as a seed to a PRNG, so the amount of randomness required from each participant is quite small.

4.1 Communal Randomness via Temporarily Secret Commitments

Communal randomness, or more precisely communally determined unpredictable value, is obtained by collecting random values from participating MIXes. These are kept secret until after everyone has committed so that no one can predict the result of combining them. The committed values can be uncovered without help from the committers if necessary so that no one can alter the communal result in a predictable way by failing to reveal what she committed. On the other hand, not all of the committed random values can be uncovered by an adversary in time for him to craft a commitment that will yield a predictable result.

Various approaches to this capability have been published. In [9], the committed unpredictability comes from a long "delaying" calculation on the result of a (fast) combination of the inputs from participants. [24] provides an alternate solution. In [4], commitments are individual as here; that paper also describes a zero-knowledge proof of a well-formed commitment. [10] further expands and develops the work in [9] and [24].

We follow the individual commitment approach of [10, 24] because it is both shortcut computable and easily commitment-verifiable. Anyone possessing a secret (in this case the committer) can compute the committed result quickly. Once the committed value is revealed, anyone can easily check that this is the committed value. Commitment take the form $\mathtt{tsbc}(\mathtt{rand}(p)) = \langle \{\mathtt{rand}(p)\}_K, w'(K) \rangle$, the encryption of $\mathtt{rand}(p)$ with K followed by a TSBC function used to commit to K. [21] presents a function such that the committer p can quickly and easily calculate $w'(K)$ as well as $\{\mathtt{rand}(p)\}_K$. Once K has been revealed (or calculated) for each of the entries, all the keys can be published by the CS. Anyone can quickly verify the communal unpredictable value by performing the encryptions and computing their amalgamation.

Neither shortcut computability nor easy commitment-verifiability hold for the collective commitment that first appeared in [9]. The application of commitments to communal unpredictability is not the central focus of [4] and is only described briefly. It is also only described for two parties producing a random bit; thus only one commitment from one party is needed to run the protocol. With multiple parties committing to many bits, malleability of the timed-commitments [8] may become a factor if the revealed values are simply XORed together to produce the result. The well-formedness proofs may ultimately play a role in the non-malleability of the commitments since this is similar to how non-malleability of commitments is proved; we leave this as future work. Instead, we avoid the issue of malleability by ensuring that the amalgamation of the revealed values is not affected by correlations among those revealed values. For example, if the revealed values are concatenated in the order the commitments were posted and then hashed with a well-designed hash function, the result should be unpredictable if even one of the committed values is unpredictable.

We could make use of the well-formedness proofs in [4] to prevent nodes from submitting malformed commitments without detection, but the computational overhead is not necessary. Whether a commitment is malformed can be confirmed after a predictable amount of computation. Once it is known to be malformed, the result can either be discarded or it can be used as if it had been well-formed (details in [10, 24]). Because inputs come from nodes with a vested interest in maintaining reputation, we can use the reputation system to ensure that malformed commitments are rare. All nodes must commit to participate in the network, but to minimize the number of malformed inputs or committed but not overtly revealed inputs, only commitments from high reputation nodes will contribute to the communal unpredictable value. We might use inputs from the top quarter or half of the nodes (all bandwidths), to make sure we have enough inputs to make collusion or compromise of the result unlikely. Any node that commits but does not reveal or puts in a malformed commitment loses reputation. To minimize ongoing problems from a misbehaving node, the random input from that node is not used for a fixed number of subsequent rounds.

If all nodes contributing to the communal value have revealed secret values that can be verified as committed during the entry phase, these can be combined by whatever means we use to yield a random result, e.g., concatenation and hashing as above. If not all the values have been revealed, a delaying calculation must be used to uncover those not revealed. If all such revealed values correspond to well-formed TSBC entries, the whole thing still remains easily commitment verifiable. If any of them are malformed, the communally determined value will only be verifiable by the corresponding delaying calculation (with parallel or probabilistic speedup, once it has been done initially, cf. [10, 24] for details).

5 Reputation System and Cascade Configuration

We would like to develop as simple a scoring system as possible—simple systems are easier to analyze for security, and they allow the users to more easily understand the implications of a given score. Our system decrements the reputation of all nodes in a failed cascade, and increments the reputation of all nodes in a successful cascade. Thus we don't have to worry about pinpointing the cause of failure. The hope is that reputations will give a general sense of the reliability of nodes over the long term.

However, because a node's behavior affects the reputation of its cascade-mates, this introduces a new attack on the reputation system. When a cascade fails that has fewer bad nodes than good nodes, it does more damage to the overall reputation of good nodes than bad. Since bad nodes can intentionally fail a cascade, they can exploit this vulnerability to gradually reduce the relative reputation of the good nodes.[1] The bad nodes "creep upward" on the reputation spectrum, eroding the reputation of nodes above them—our visual simulations led us to refer to this attack as the *creeping death*.

Because the bad nodes can gain reputation faster, they can eventually get to any point on the reputation scale. Rather than complicating the reputation system to prevent our adversary from performing the creeping death attack, we intentionally keep it simple and develop an algorithm for building cascades which minimizes impact from this attack. In particular, we have three goals for cascade configuration:

- Neutralize effect of creeping death. *Heuristic: choose nodes evenly from the whole space of acceptable nodes.*
- Increase cost of anonymity-breaking attack. *Heuristic: choose nodes from the high-reputation end of the spectrum.*
- Increase cost of reliability-breaking attack. *Heuristic: never choose nodes predictably from any small fraction of the spectrum.*

Since bad nodes can position themselves anywhere (reputation-wise) to increase the chance of winning a whole cascade, the optimal response in terms of protecting anonymity is to choose nodes for each cascade entirely at random—thus the adversary's ability to position himself doesn't help. But since it's cheaper

[1] 'Good' and 'bad' refer to nodes that are honest or that are part of the adversary, respectively. Sometimes a bad node may be reliable to compromise anonymity. Thus 'good' and 'bad' do not necessarily correspond to 'reliable' and 'unreliable'.

to fund a node only a little bit (to a small reputation), the adversary ought to make as many cheap nodes as possible, increasing his chances of winning a cascade. This motivates a new algorithm: divide the set of participants into quadrants by reputation (if there are 4 nodes in a cascade), and build cascades one node from each quadrant. In order to win a cascade, the adversary now needs to have a good chance of getting picked from each of the four quadrants. Yet, an adversary who can get a foothold on many different cascades can break reliability. Eg, if he runs a quarter of the participants and he places them all at the lowest (cheapest) quadrant, he gets a node in every single cascade. To make this harder, we modify the cascade configuration algorithm: pick one from the top quarter, and pick three from the other three quarters.

We need some mechanism for limiting the fraction of adversary-controlled participants in the MIX network, and for making sure that we don't use unreliable nodes in our cascades; we discuss this issue more thoroughly in Section 5.1. Section 5.2 considers the issue of the creeping death attack in more detail. We go on in Section 5.3 to discuss the logistics of how to retrieve and update reputations from a centralized or distributed reputation server, as well as the types of reputation that we need to remember for the MIX network.

5.1 Pseudospoofing and consistently unreliable nodes

To prevent our adversary from creating a multitude of identities and flooding the system with them (an attack known as *pseudospoofing*), we place new or unreliable nodes in a probationary phase. Such nodes do not participate in the live cascades (those used by actual users). Nodes in this phase effectively perform a “proof of bandwidth” to demonstrate that they can reliably pass bits.

We originally intended the proof of bandwidth phase (coupled with a proof of computation phase) to force an adversary to spend resources for each node he signs up, thus limiting the number of nodes that he can afford to get past the probationary period. In other words, while we might start out with an arbitrary fraction of adversary-owned nodes, we hoped to end up with a known bound on that fraction. We abandoned that route for two reasons:

– A large or skilled adversary can easily match resources with a group of volunteers, particularly if the adversary can compromise other machines around the Internet and commandeer their resources.
– The adversary can cheat at any of our decentralized proof of bandwidth schemes by skimping on all the bandwidth between his own nodes. If he controls n consecutive nodes in a cascade, he need only do work for one of them (or two, if his nodes are scattered rather than on the same network). If he owns all the nodes in a cascade, he does no work at all. An adversary who owns most of the nodes in the system will often be able to pass proof of bandwidth tests without spending any bandwidth.

Instead, we use an identity-based barrier to entry: a web of trust like PGP [19] or Advogato [14]. A web of trust does not require any controlling or monitoring authority, and still allows us to limit the number of nodes an adversary can get certified. If we pick the seeds of the web carefully and make some assumptions about the number and location in the web of honest members that might assign trust to the adversary, we can bound the fraction of bad nodes that can get into the system.

Alice should certify Bob based on whether she believes him to be trustworthy (to be a real person, to genuinely intend to follow the MIX-net protocol, etc). If she might certify Bob based on whether she believes he will run a reliable node, the adversary can simply run convincingly reliable nodes and watch the certifications roll in. Yet if Alice's certification has no component of how reliable she thinks Bob is, she could easily be certifying people who mean well but turn their computer off each night when they go to bed. If we use our intended cascade building algorithm (pick one from the top quarter, three from the rest), then the unreliable nodes at the bottom of the reputation spectrum will get mixed into every cascade, dramatically reducing the number of available stable cascades.

We could ask Alice to not certify Bob if she believes he might be unreliable, but this imposes more burden on Alice, and also doesn't account for the fact that Bob's behavior can change over time (whereas certification is a one-time thing).

By combining the web of trust model with our probationary phase, we can ensure that people who pass the probationary phase are likely to be running reliable nodes. We divide all the participants into the live cascades and the probationary cascades, by placing a "bar" on the reputation spectrum.

We could place the bar at some fixed minimum reputation, so all nodes with reputations under a certain value r (e.g., five) are matched with each other and process only dummy messages. Thus any node which demonstrates its reliability gets used for the live cascades. In this case, a reliability-breaking adversary can get above the bar, fail half the time, and always be assured of staying above the bar.

We could counter this by dividing such that some fraction L of them (eg 20%) are below the bar, except we make sure that all nodes with reputations less than r are also below the bar. Because the bar is a moving target, our reliability-breaking adversary can at his most damaging match the performance of the nodes around him—if he does less, then he will eventually slip below the bar. Placing the bar higher than all low-reputation participants guarantees us some minimum level of reliability. This approach "wastes" the people who are above reputation r but still below the bar; in particular, if the reputations of participants generally trend upward, new participants may have to do months of good service before they get above the bar. Also, absent a significant reliability-breaking adversary, as it becomes less worth it to try to reach the rising bar, those that have been performing reliably for some time will suddenly find themselves in the bottom twenty percent. If they get frustrated and leave, the whole reputation system could quickly crash.

We could counter that by having only the last so many days count towards reputation, e.g., thirty. This effectively puts an upper bound on reputation. While some features of creeping death are complicated thereby, reliably providing service for years matters no more than providing it for thirty days. Having such a short memory for node performance makes it easier to degrade the reputation of established nodes in particular and the reliability of the system in general.

Our suggested solution is to use both cumulative and short-term reputations. The cumulative reputation builds as previously described. To participate as a live cascade, a node must have a cumulative reputation R of, e.g., five. These can build indefinitely; thus over time it will become increasingly expensive for an adversary to affect a long-term reputation.

A node must also have an adequate short-term reputation S measured both over the sliding window of the past W days and the past $A < W$ active days in that period. For example, if $S = 5$, $A = 15$, and $W = 50$, in order to avoid going back to probation the node must be active for at least 5 of the last 50 days and essentially fail no more than a third of the active days within the last 50. Days and reputation are calculated over both probationary and live activity.

Probationary proof-of-bandwidth is done in cascades with other nodes that are on probation from within the same reputation quadrant (or the highest reputation nodes from below to complete a partial cascade). Thus new probationary nodes have limited effect on the probationary status of high reputation nodes that have been temporarily inactive or recently buggy.

Finally, since we've added a web of trust system to reduce damage from pseudospoofing, can we remove our reputation system and let the web of trust do all the work? There are two reasons for using a reputation system separate from the web of trust:

- A separate reputation system requires less effort and attention from human operators. Operators don't need to certify based on expected reliability, but rather only on perceived honesty. Because unreliable nodes aren't used for live cascades, operators don't have to worry about revoking certification for later poor reliability.

– By choosing one member of each cascade from the high end of the reputation spectrum, we are making it harder for an adversary to get all the nodes in a cascade. Such cascades provide stronger anonymity. Without a reputation system, we would have no mechanism for comparing and choosing the higher-reputation nodes.

5.2 More on Creeping Death

We may not be able to prevent creeping death without assuming a very low percentage of bad nodes, c. less than ten percent. Space limitations preclude describing various alternative reputation systems, cascade algorithms, cascade sizes, etc. that we considered to counter it.

Another feature of creeping death is not just whether the bad nodes can position themselves at whatever reputation level they want but whether they can prevent cumulative reputations from rising, effectively capping the long-term performance of honest nodes.

Without going into details, using only cumulative reputations it seems that if more than 21% of the nodes are bad, the bad nodes can get in cascades with good nodes often enough to cause them to fail at least half the time, preventing their reputations from growing. However, because of the combined cumulative and short-term reputation system we introduced above, the situation is slightly better. For example, if a live bad node can only fail a third of the time it is active or it will return to probation, the percentage of nodes required to prevent a rise in cumulative reputation is much higher.

5.3 how to manage reputations, types of reputation

6 The cascade protocol, or, When you should fail your cascade

Detecting misbehavior in cascades can be broken into three steps:

1. Entry point: Incoming messages might not be accepted.
2. Inside the cascade: Messages might be replaced with dummy messages.
3. Exit point: Messages might not be delivered.

We would like to allow each MIX to test its cascade by sending and receiving messages using ordinary-looking external addresses, but reliably generating plausible external addresses seems hard. Without that, it seems that such testing will fall to the "selectively process the test messages" attack. We solve the problem by doing such testing directly from the nodes themselves.

The cascade has a node called the head (where the cascade starts stripping layers of encryption), and a node called the tail (the last node to strip layers of encryption). All n nodes in the cascade (typically n might be 4 or 5) accept $\frac{1}{n}$ of the total traffic, and deliver the messages to the head. The head publishes a snapshot of the batch (a set of hashes of each message) as he processes it.

A sender Alice can ask for the snapshot to verify that her message got into the batch. If not, she concludes that either the head or the node she used was dishonest, and goes to a different node or cascade. As an optimization, nodes that accept messages can give Alice a receipt if they accept her message. If her message does not make it into the batch, Alice can broadcast the message and the receipt to the other nodes in the cascade; an honest cascade member will determine that the receipt should have been honored and fail the cascade. Thus our protocol can detect misbehavior at the entry point (Goal 1).

Because all the messages coming in to the head are coming from other nodes in the cascade, the other nodes can insert indistinguishable test messages into the batches. If the test message does not make it to the tail, its sender fails the cascade. Since other nodes can't tell which messages are test messages, dishonest nodes risk being caught if they replace even one message with a dummy message.

In the naive design, the tail delivers messages and also broadcasts them to the other nodes in the cascade. Every node attempts delivery. Since the tail can't tell who wrote a test message, he must deliver every message to every node in the cascade or risk failing the cascade. A more efficient design assumes a reasonable PKI for all recipients. In this case, we can shortcut the need for broadcasting to the other nodes if the original delivery attempt produces a signed receipt; we discuss this more in Section 6.1. By delivering outgoing traffic to all cascade members, our protocol detects misbehavior at the exit point (Goal 3).

If the head is dishonest or conspires with another node in the cascade, the head can publish a correct batch snapshot but replace its portion of messages with dummy messages. Because it knows its portion contains no test messages, all of those messages will be undetectably lost. We solve this by supporting external test messages as well. Alice might become suspicious because the cascade accepts messages but doesn't deliver them; she can send a test message, wait a while, and then reveal to everybody how the message should have decrypted. If at least one node in the cascade is honest, he will agree that he didn't see the message and fail the cascade. Thus we detect misbehavior inside the cascade (Goal 3).

Senders may want to chain cascades for stronger anonymity. To make chaining cascades more robust, nodes consider delivery to a second cascade as a special case. We can specify the entire cascade rather than a single node as the next hop in the chain. Thus each node from the first cascade chooses a random node in the second cascade and attempts delivery. Nodes could even verify that their message made it into the in-pool of the next cascade, and claim misbehavior if not. With this modification, the head of a cascade must be able to detect duplicates in the batch; however, since the head must already detect duplicates to foil replays, this presents no extra burden.

Since senders exposing a faulty cascade have no reason to chain their test message through another cascade, some nodes need to explicitly send external test messages to other cascades and verify their delivery.

6.1 Delivery receipts

Message recipients can give the tail a receipt when he delivers a message. The tail first attempts to get a receipt, which he can use to prove that he delivered the message. If he does not get a receipt (e.g., because the destination address refuses to provide a receipt or does not exist), he broadcasts the message to the other members of the cascade, who try to deliver. In any case they now know that he followed the protocol.

An unhappy sender Alice can contact some cascade node N and claim "T didn't deliver my message", along with a demonstration of unwrapping the remaining onion between N and T. If N remembers what he passed to $N + 1$, Alice can show N what it should have looked like when it got to T.

If N has already heard from T about its attempt to deliver the message, it knows T is blameless. If not, it can query T for a receipt — if T has no receipt, the cascade failed (either the message never got to T, or he did not deliver it).

Delivery receipts detect misbehavior as long as one of the nodes in the cascade is honest. If the honest node is the tail T and the message makes it that far, then the message will be delivered (cases 1 and 2 handle if the message doesn't make it that far). If T is bad, either he delivers the message to N and it gets delivered, or he does not and Alice can convince N that the cascade is misbehaving.

If a recipient is not configured to return a receipt, the delivery still gets through — in this case, the message gets broadcast to the other cascade members, and each of them attempts delivery. In a sense, the use of receipts is just a bandwidth optimization.

Most related work assumes public keys for message recipients are known to all parties. Without this PKI, the exit node can forge a signed receipt from the recipient. Our protocol works without a widespread PKI, but works better with one.

6.2 Capacity-attacking adversary

Since nodes can falsely refuse incoming messages by claiming to be full, the number of messages processed by a cascade is (at least) proportional to the number of honest nodes in that cascade. By inserting indistinguishable test messages into its own batches, each node verifies that the rest of the cascade is successfully decrypting and passing on that fraction of its bandwidth promise. By failing the cascade if the amount of traffic coming in from the previous hop is outside suitable thresholds (nodes should pad if they don't have a full batch fraction), each node verifies that the rest of the cascade is spending (even if wasting) its entire bandwidth promise.

By not accepting any messages and then not delivered the corresponding dummy traffic, bad nodes can spend slightly less bandwidth than good nodes. But if those bad nodes are delivering messages for the honest cascade members, they are doing no more damage than if they simply had not signed up that day. Thus our design frustrates a *capacity-breaking adversary* (a special case of the reliability-breaking adversary).

6.3 How users pick a cascade to use

Because cascades are chosen by picking nodes from the same reputation classes, all cascades will have "equivalent" reputation. Senders should simply pick a random cascade for their message. This level of load-balancing ought to be sufficient for normal conditions; but if most of the cascades are full, it will be difficult to find the ones that still have space available.

The simple solution to this involves querying cascades for available space. Alice queries a random cascade, and it either returns a valid ticket for the next time slice or it refuses. If Alice gets a ticket, she redeems it and uses that cascade. If not, she tries a new one. Thus Alice doesn't reveal her message to anybody except the cascade she actually chooses; on the other hand, the ticket system as specified introduces new DoS attacks from requesting tickets but not sending messages and from giving tickets but not honoring messages.

The second approach is more push than pull. Cascades can publish available capacity information, or even the expected wait or available quality of service for a given message on that cascade. Users compare the QoS available from each cascade, thus balancing load across the cascades.

7 Attacks and traffic analysis implications

Attacks on Anonymity

Have enough nodes to own an entire cascade. By using a web of trust and building cascades from the entire spectrum of reliable nodes, we reduce the chance that this attack will succeed.

Gain high reputation to read more traffic. Similarly, our cascade building algorithm blocks this attack.

Replay attack, message delaying, etc. We rely on the standard defenses offered by MIX-net protocols.

Trickle attack. If one node in the cascade is honest, at least $\frac{1}{n}$ of the traffic will be legitimate in every batch.

Intersection attack. Using MIX cascades rather than free routes helps to defend against intersection attacks from very large adversaries [3]. By encouraging users to continue using similar anonymity sets across cascade configurations, we can further complicate intersection attacks. However, they remain a very real way of breaking anonymity in our system.

Influence cascade configuration externally. Our algorithm for generating communally random uncertainty resists individuals and groups, as detailed in Section 4.

Compromise cascade configuration server. Because the output of the CS is publicly verifiable, incorrect behavior can be detected immediately.

Knock down uncompromised cascades to get more traffic. While the adversary can knock down other cascades, the low chance of owning an entire cascade limits the success of this attack.

Attacks on Reliability

Join a cascade, then fail. The reputation system makes this a losing strategy in the long run — for each time a node fails as part of a live cascade, that node must previously have provided good service for an entire day.

Selectively process only test messages. Our protocol addresses this possibility in Section 6.

Attacks on Capacity

Flood nodes with messages. If this becomes a problem, we can integrate a ticket service such as that in [2], limiting the number of messages a given identity can generate for each batch. Other solutions include proofs of work and other micropayment schemes.

Knock down many cascades. We assume that our adversary is not strong enough to knock down all or most of the cascades in the system. If he only knocks down some, service continues as normal.

Block commitments to the Configuration Server. If the adversary launches a denial of service attack against the CS, the participants can together use a decentralized algorithm to simulate the CS (all of its operations are public and verifiable).

Refuse commitments at the Configuration Server. Because we allow certified delivery to the CS, refusing commitments can be detected by any observer.

Refuse incoming messages as a cascade member. This attack can work to reduce capacity; but the node is still spending most of its pledged bandwidth on correctly processing messages from other nodes, as well as generating dummy traffic in place of the refused traffic. This attack is very inefficient.

Attacks on Reputations

Beat the web of trust. The security of the web of trust is perhaps the most critical assumption of our system. If an adversary can get a lot of nodes certified without spending effort for each one, he can start widespread attacks on anonymity, reliability, capacity, and reputations. On the other hand, getting just a few extra nodes certified does not buy him much.

Internal selective DoS — creeping death. The adversary can use the creeping death attack (fail the cascade if the resulting reputation loss is favorable) to gain any position in the reputation spectrum. Our cascade building algorithm makes this technique ineffective at breaking anonymity; further, it may prove very expensive in terms of resources to move a large number of nodes to the top of the reputation spectrum.

External selective DoS — knock down the high-reputation cascades. Our "all for one and one for all" approach to reputation makes selective denial of service even easier than in [7]. Previously, to knock down a reliable node you needed to successfully flood it or cause it to not process messages correctly. Now you simply have to locate and knock down the weakest member of its cascade. However, this vulnerability is acceptable: removing one cascade does not deny service to the system as a whole, and as above it does not get you much closer to breaking anonymity.

8 Conclusion and Future Directions

We have described a protocol for improving reliability of anonymous communication networks, based on a MIX cascade design and a simple reputation system. There are a number of directions for future research:

- A working proof of bandwidth algorithm would allow us to reduce our reliance on the web of trust.

– Better approaches to generating convincing origins and destinations for dummy traffic, or reducing the bandwidth overhead of the current approaches, would make the overall design more reasonable.
– Improved cascade configuration algorithms would allow us to provide stronger anonymity and reliability.
– More research on the scope and characteristics of the creeping death attack might give insight on how to defeat it.
– More analysis on the attack-resistance of our reputation system might yield stronger proofs or a better design. For instance, can we guarantee bounds on work performed by the adversary under various models?

We probably want a good final sentence here.

Acknowledgements

We thank Daniel Barkalow and Nick Mathewson for intelligent discussions about the reputation systems and cascade configuration; Ian Goldberg for thoughts on cascade configuration; Paul van Oorschot for a helpful conversation about hash functions; and David Molnar for early discussions about the design and for help with Section 3.

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. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable internet access. Proceedings of the Workshop on Design Issues in Anonymity and Unobservability 2001.

3. Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. Proceedings of the Workshop on Design Issues in Anonymity and Unobservability 2001.

4. Dan Boneh and Moni Naor. Timed commitments. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, pages 236–254. Springer-Verlag, LNCS 1880, 2000.

5. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982.

6. 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.

7. Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. A Reputation System to Increase MIX-net Reliability. Proceedings of the Information Hiding Workshop 2001. Also available from <http://www.freehaven.net/papers.html>.

8. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *23rd ACM Symposium on the Theory of Computing (STOC)*, pages 542–552, 1991. Full version available from the authors.

9. David M. Goldschlag and Stuart G. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In R. Hirschfeld, editor, *Financial Cryptography, FC'98*, pages 214–226. Springer-Verlag, LNCS 1465, 1998.

10. David M. Goldschlag, Stuart G. Stubblebine, and Paul F. Syverson. Temporarily hidden bit commitment and lottery applications. Submitted for journal publica-

tion.

11. Markus Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM, 1999.

12. Anja Jerichow, Jan Müller, Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. Real-Time Mixes: A bandwidth-efficient anonymity protocol. IEEE Journal on Selected Areas in Communications 1998.

13. 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.

14. Raph Levien. Advogato's trust metric. <http://www.advogato.org/trust-metric.html>.

15. Tim May. Description of Levien's pinging service. <http://www2.pro-ns.net/~crypto/chapter8.html>.

16. Jim McCoy. Re: DC-net implementation via reputation capital. <http://www.privacy.nb.ca/cryptography/archives/coderpunks/new/1998-10/0114.html>.

17. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.

18. M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.

19. PGP FAQ. http://www.faqs.org/faqs/pgp-faq/.

20. James Riordan and Bruce Schneier. A certified e-mail protocol with no trusted third party. *13th Annual Computer Security Applications Conference*, December 1998.

21. Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. MIT LCS technical memo MIT/LCS/TR-684, February 1996.

22. RProcess. Potato Software. <http://www.skuz.net/potatoware/>.

23. RProcess. Selective denial of service attacks. <http://www.eff.org/pub/Privacy/Anonymity/1999_09_DoS_remail_vuln.html>.

24. Paul Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In Simon Foley, editor, *Computer Security Foundations Workshop (CSFW11)*, pages 2–13, Rockport Massachusetts, June 1998. IEEE CS Press.

25. Zhou and Gollmann. Certified electronic mail. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.