Reliable MIX Cascade Networks through Reputation

Roger Dingledine[1] and Paul Syverson[2]

[1] The Free Haven Project (arma@mit.edu)
[2] Naval Research Lab (syverson@itd.nrl.navy.mil)

Abstract. We describe a MIX cascade protocol and a reputation system that together increase the reliability of a network of MIX cascades. In our protocol, MIX nodes periodically generate a communally random seed that, along with their reputations, determines cascade 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.

Keywords: anonymity, reputation, peer-to-peer, communal randomness

1 Introduction

Practical anonymous communication systems require high reliability. Reliability can lead to efficiency because routes are more likely to succeed. Reliability can also improve anonymity because senders need to resend fewer messages, and because a reliable system draws more users and thereby increases anonymity sets. Past approaches to increasing remailer reliability have included writing more reliable software [26], building MIX protocols that give provable robustness guarantees [6, 13, 21], 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:

– *The 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 create 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 trying to do traffic analysis can get more traffic by gaining a high reputation.* We protect against such adversaries by choosing from a pool of "acceptable" MIX nodes, and building cascades so we can bound 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 electronic commerce, where transactions need to be efficient, reliable — and, frequently, anonymous. 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. Our reputation system gives users a more accurate picture of which nodes are currently up, allowing them to choose routes more reliably and to know what level of protection they're getting.

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 [20], 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. Nodes that misbehave by incorrectly reporting cascade failure thus 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 examine 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 group 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 reordering inside each MIX.

One type of MIX hierarchy is a cascade. In a cascade network, users choose from a set of fixed paths through the MIX-net. Cascades can provide greater anonymity against a large adversary, 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 [3].

Current research on MIX-nets includes stop-and-go MIX-nets [16], distributed flash MIXes [13], and hybrid MIXes [23]. MIX cascade research includes real-time MIXes [15] 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 [13]. 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 [13], but the protocol was later found to be flawed [21] by Mitomo 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. Neff recently made further efficiency improvements to universally verifiable mixing [22].

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 Approaches to MIX-net Reliability

MIX-net protocols can give specific guarantees of robustness. Under 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." Such protocols are often complicated and inefficient.

Levien's statistics pages [19] present another approach. They 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.

Instead of engineering the MIX-net protocol directly to provide reliability, we make use of reputations to track MIX performance. In this approach, we specify a set of behaviors that characterize a functioning or failed MIX. We are not likely to prove strong theorems — the goal of reputation is to make the system "better" without guaranteeing perfection. Like Levien's reputation system for free-route MIX networks, our published reputations enable users to find better routes. Further, our reputation system works behind the scenes to build more reliable cascades — a process that may even be entirely transparent to the users.

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 into cascades, so that cascades reflect recent changes in reliability, and so nodes in failed cascades can get back in a working cascade. We choose to rearrange *all* nodes, not just those from failed cascades; 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 (e.g., one day). By $T-a-b$, each participant sends a 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 node N a signed receipt $\texttt{sign}(CS, [commitment, timestamp])$. 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. With both receipts, N can prove that he should be included in the next cascade configuration.

Commitment from N: $\texttt{sign}(N, [N, \texttt{IP}, \texttt{port}, \texttt{bandwidthpledge}, \texttt{tsbc}(\texttt{rand}_N)])$.

Reveal from N: $\texttt{sign}(N, [N, \texttt{IP}, \texttt{port}, \texttt{bandwidthpledge}, \texttt{rand}_N])$

Here "$\texttt{tsbc}(\texttt{rand}_N)$" is N's temporarily-secret bit-commitment to his random value (to be explained below, in Section 4.1). The CS then builds 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 the four buckets is formed into a set of cascades according to the algorithm described in Section 5, using the unpredictable communal value.

The communal value can be used as a seed to a PRNG, so the amount of randomness required from each participant is quite small. If N is worried that CS will ignore his commitment or reveal (not provide a receipt), he can use a certified mail delivery system [24, 28] to convince other people that CS is misbehaving. Alternatively, we can build our own certified delivery system by pre-assigning a set of witnesses (perhaps fifteen) from the previous T. A threshold of these witnesses is sufficient to prove misbehavior. As long as the adversary does not control too high a percentage of nodes then we can trust this system (cf., Section 5 for discussion of how we limit this).

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.

4.1 Communal Randomness via Temporarily Secret Commitments

Communal randomness, or more precisely a communally determined unpredictable value, is obtained by collecting random values from participating MIXes. These are kept secret until 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 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 [11], the committed unpredictability comes from a long "delaying" calculation on the result of a (fast) combination of the inputs from participants. Another similar scheme is given in [27]. Both [4] and [27] give commitment schemes that are individual, as here; [4] also describes a zero-knowledge proof of a well-formed commitment. [12] further expands and develops the work in [11] and [27].

We follow the individual commitment approach of [12, 27] 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. Commitments take the form $\mathtt{tsbc}(\mathtt{rand}_N) = \langle \mathtt{enc}(K, \mathtt{rand}_N), w(K) \rangle$, the encryption of $\texttt{rand}_N$ with K followed by a TSBC function used to commit to K. [25] presents a function such that the committer N can quickly and easily calculate $w(K)$ as well as $\texttt{enc}(K, \texttt{rand}_N)$. 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 [11]. 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 [9] 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 [12, 27]). 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 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, we use the delaying calculation to uncover those not revealed. If all such revealed values correspond to well-formed TSBC entries, the result 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. [12, 27] 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 cascademates, 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 that minimizes impact from this attack.

Since bad nodes can position themselves anywhere (reputation-wise) to increase the chance of winning a whole cascade, the optimal strategy to protect anonymity chooses nodes for each cascade entirely at random. But we also want to increase the cost of reliability breaking, so that the adversary can only affect cascades likely to be highly desired for use if he runs reliable nodes himself; simply getting nodes into the system should have less impact. We thus combine these approaches and begin choosing cascade nodes randomly (using the random seed from the method of Section 4) from an adequately large set of nodes, but still of the highest possible reputation. (See Section 5.1 for details.)

While the reputation system reduces the impact of an adversary that merely gets nodes accepted into the system, the creeping death attack allows a resourceful adversary to quickly move up on the reputation spectrum. An adversary with many nodes can still succeed at breaking reliability and anonymity. We prevent our adversary from creating a multitude of identities and flooding the system with them (an attack known as *pseudospoofing*) with an identity-based barrier to entry: a web of trust like Advogato [17]. A web of trust allows us to limit the number of nodes an adversary can get certified. [17] gives a proof that the

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

number of bad nodes accepted by the web is limited by the number of honest members that might assign trust to the adversary (*confused* nodes) — not the number of nodes the adversary creates. If we pick the seeds of the web carefully and make some assumptions about the number and position of confused nodes, we can bound the fraction of bad nodes in the system.

Alice should certify Bob based on whether she believes him to be trustworthy (to be a real person with good intentions). If she certifies based on expected performance, the adversary can simply run convincingly reliable nodes. Certification aims only to bound the total percentage of adversary-owned nodes in the system.

On the other hand, we might ask Alice to not certify Bob if she believes he might be unreliable. However, this imposes a greater 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 event). Instead, the reliability of an admitted node will be determined by the reputation system. Nodes that are the most reliable over time will have the highest reputations.

5.1 Building Cascades

It is tempting to believe that some alternative reputation system or cascade algorithm can reduce or simplify the problem introduced by creeping death. For instance, we might punish nodes incrementally more as they have more failures on record. But this is exactly the problem — honest nodes *do* fail more often than adversary nodes. For any pattern that we look for, our adversary can arrange it so honest nodes fit that pattern better or more often than bad nodes.

Consider cascades with only two nodes. In cascades with one good and one bad, the bad node can hurt only one good node, so by construction bad nodes do equal damage to good nodes. But even this system falls prey to the creeping death. Since a cascade can only fail if one of its nodes writes the "We failed" certificate, the both-bad case will *never* fail (the nodes simply never write the certificate), whereas a both-good case will sometimes legitimately fail. Thus every both-bad cascade can stop functioning immediately, yet the bad reputations increase faster than the good reputations over time.

While we cannot easily reduce or prevent creeping death, we can choose cascades to produce acceptable and predictable risk and reliability despite creeping death. Reasonable anonymity protection may require chaining cascades into longer paths.

We order the nodes by reputation, and choose nodes for the first cascade randomly from within a pool of nodes at the top of the reputation spectrum. (Randomness is obtained from the seed chosen in the method of Section 4.1.) Next, add to the pool enough next-highest reputation nodes to maintain its size and pick another cascade at random. This continues until the last cascade for which an adequate pool size can be maintained. At that point, the remaining nodes are formed into cascades at random.

How do we decide this pool size? Assume the following notation:

- p = fraction of nodes that are bad,
- s = scare factor: acceptable probability of adversary-controlled path,
- r = range: size of the pool from which nodes are chosen for a single cascade,
- l = length of a single cascade,
- c = chain length: number of cascades chained together,

For determining the range, we assume a worst case for adversary distribution, because of creeping death. That is, we always assume the adversary resides entirely in the pool of nodes from which we're picking a cascade. This means

$$\left(\frac{p}{r}\right)^{lc} = s \quad \text{and so} \quad r = \frac{p}{s^{\frac{1}{lc}}}$$

For example, suppose that cascades are of length 4, there are not more than 20% bad nodes altogether, and it is acceptable that one of every hundred thousand paths (cascade chains) is completely bad — meaning messages through it are compromised. Also assume that we chain three cascades to reduce the odds that all nodes traversed are bad.[2] Then

$$r = \frac{.2}{(10^{-5})^{\frac{1}{12}}} = 0.522$$

Thus the first cascade must come from nodes in the top 52.2% of the reputation spectrum. The next cascade must be chosen from the same pool, minus the four nodes of the first cascade and plus the four next highest reputation nodes. Once the lowest reputation node has been added to the choice pool, the remaining nodes are just chosen at random until all the cascades are formed.

A chained cascade path is the same as a single long cascade of length lc with respect to the odds that all nodes in it are bad, but they are not the same in general. When a chained cascade fails, only the offending subcascade is removed from the system for that period and its nodes decremented in reputation. Also, users may choose to chain cascades or not, may not always choose to chain in the same way, or may choose a longer or shorter chain. A user may choose not to chain because of the computational overhead, the latency, etc. This choice will afford him improvements in those areas, but at an increased risk to anonymity. Users should be made aware of the risks. Our system allows an easy explicit presentation of the relative risks and tradeoffs. It also allows us to adjust these at the system level. For example, if we wish to reduce r, we can weaken s, or adjust l or the recommended c. Of course if p is high enough, s strong enough, and we limit lc to some practical bound, r may exceed 1 and no network is feasible. Or r may be close enough to 1 to render the reputation system largely moot — but at least we can calculate this and react accordingly.

[2] A path length of 12 would be absurd with current remailers; but the whole point of this design is to improve reliability so long paths are feasible.

6 The Cascade Protocol, or, When to Fail Your Cascade

Opportunies for misbehavior in cascades fall into three classes:

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.

Each MIX can test its cascade by sending and receiving messages using ordinary-looking external addresses — but spoofing or maintaining plausible external addresses is hard. Instead, we protect against this "selectively process the test messages" attack by relaying traffic through other nodes in the cascade and allowing them to undetectably insert test messages. All l nodes in the cascade (typically l might be 4 or 5) accept $\frac{1}{l}$ of the total traffic, and deliver the messages to the head of the cascade. 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.

Because all the messages to the head come from other nodes in the cascade, these nodes can insert indistinguishable test messages into the batches. If a 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. Thus our protocol detects misbehavior at the entry point (Goal 1).

In the naive delivery 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. To prevent the tail from selectively dropping messages based on destination, nodes address some of their test messages to previous recipients. Thus, the tail must deliver even to a user not known to be running a node. (This reliability increase must be balanced with possible spam abuse.) A more efficient design assumes a PKI which includes all recipients. In this case, we shortcut the need for broadcasting when 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).

A dishonest head can publish a correct batch snapshot but replace its (or a conspirator's) 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 after the batch snapshot is published. (Goal 2).

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. Each node from the first cascade chooses a random node in the second cascade and attempts delivery. Nodes may verify that their message is included in the next cascade's batch, and claim misbehavior if not. With this modification, the head of a cascade must be able to detect duplicates in the batch; however, since all nodes 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 messages through another cascade, some nodes need to explicitly send external test messages to other cascades and verify their delivery.

Test messages using real addresses help foil time-based intersection attacks. In a standard MIX network, an adversary with information about what users are active at what times can quickly narrow down the set of suspects based on when traffic is seen. An active adversary works even faster by knocking out suspects until traffic stops. Because cascade nodes send messages too, an address might get mail at other times as well.

Users worried about profiling should send each message through a different cascade, so an adversary who owns a few cascades cannot read all messages. Users worried about message linkability should send all messages through one cascade: a single compromised cascade can reveal linkability.

A decentralized algorithm would allow users to keep a similar anonymity set across cascade reconfigurations, further blocking time-based intersection attacks. On the other hand, an adversary targetting a specific user benefits from this predictable behavior. We leave this as future work.

Plaintext messages are distinguishable and so are less reliable. Further, since test messages with convincing plaintext are hard to write, nodes are unlikely to address tests to a recipient without a known public key. Optionally, outside users could contribute convincing plaintext messages to be used as test messages by a node; that way it could send a plaintext test message to the recipient and verify its 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 the remaining decryptions 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, he knows T is blameless. If not, he 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 an honest 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 recipients are known to all parties. Without this PKI, the exit node can forge a signed receipt from the recipient. But since we don't need to link keys to external identities, a sender Alice can include Bob's signature verification key in her message, allowing any cascade node to verify Bob's receipt in a decentralized fashion.

6.2 Capacity-attacking Adversary

Since nodes can refuse incoming messages by falsely 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, as well as guaranteeing that the batch provides a minimum level of anonymity. Nodes should pad if they don't have a full batch fraction; by failing the cascade if the amount of traffic coming in from the previous hop is outside suitable thresholds, 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 delivering 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 period. Thus our design frustrates a *capacity-breaking adversary* (a special case of the reliability-breaking adversary).

6.3 Resource Management and Reputation Servers

Because the highest-reputation cascade cannot process all of the traffic, cascades publish available capacity information, including the expected wait or available quality of service for messages. Users compare reputation and available QoS from each cascade, thus balancing load across the cascades.

A group of redundant reputation servers (RS) serve current node state. Nodes give each RS hourly *heartbeat* updates, and deliver failure messages immediately. Because each node signs and timestamps each certificate, an RS can at most fail to provide the newest certificate. Each RS works with the others to ensure correct data, perhaps by successively signing certificate bundles. Senders download the entire bundle of certificates if it is small enough, else they must query through the MIX-net or query via Private Information Retrieval [18] to privately download a random subset. Otherwise, the adversary could use that information to aid intersection attacks.

Actual deployment is still a complex problem; we leave this as future work.

7 Attacks and Defenses

Attacks on Anonymity

Have enough nodes to own an entire cascade. By using a web of trust, building cascades from a large enough pool of reliable nodes, and suggesting a safe minimum chain length, we control 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 $\frac{1}{l}$ 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 pad with dummy messages when not sending traffic, and to continue using similar anonymity sets across cascade configurations, we can further complicate intersection attacks. However, a complete solution to the intersection attack remains an open problem.

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

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

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

Attacks on Capacity and Reliability

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 [8, 10, 14].

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).

Flood the CS with commits. We only consider commits from nodes which have been certified in the web of trust, and we only need to actually use those commitments from relatively high-reputation nodes.

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.

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

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 good nodes lose more reputation than bad nodes) 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 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:

– Better approaches to generating convincing 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?
– Adapting this design to free-route MIX networks would allow it to be tested with a wider deployment in the current remailer system.

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; Geoff Goodell and Marc Waldman for edits; 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. In *Designing Privacy Enhancing Technologies, LNCS Vol. 2009*, pages 115 – 129. Springer-Verlag, 2000.
3. Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *Designing Privacy Enhancing Technologies, LNCS Vol. 2009*, pages 30 – 45. Springer-Verlag, 2000.
4. Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology – CRYPTO 2000, LNCS Vol. 1880*, pages 236–254. Springer-Verlag, 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. Roger Dingledine, Michael J. Freedman, and David Molnar. Accountability. In *Peer-to-peer: Harnessing the Benefits of a Disruptive Technology.* O'Reilly and Associates, 2001.

9. 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.
10. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. Technical Report CS95-20, Weizmann Institute, 1995. A preliminary version appeared in *Crypto '92*, pp. 129–147.
11. David M. Goldschlag and Stuart G. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In *Financial Cryptography, FC'98, LNCS Vol. 1465*, pages 214–226. Springer-Verlag, 1998.
12. David M. Goldschlag, Stuart G. Stubblebine, and Paul F. Syverson. Temporarily hidden bit commitment and lottery applications. Submitted for journal publication.
13. Markus Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM, 1999.
14. Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99)*. Kluwer, September 1999.
15. 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.
16. 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.
17. Raph Levien. Advogato's trust metric. <http://www.advogato.org/trust-metric.html>.
18. Tal Malkin. *Private Information Retrieval*. PhD thesis, MIT, 2000. see <http://www.toc.lcs.mit.edu/~tal/>.
19. Tim May. Description of Levien's pinging service. <http://www2.pro-ns.net/~crypto/chapter8.html>.
20. Jim McCoy. Re: DC-net implementation via reputation capital. <http://www.privacy.nb.ca/cryptography/archives/coderpunks/new/1998-10/0114.html>.
21. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
22. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In P. Samarati, editor, *8th ACM Conference on Computer and Communications Security (CCS-8)*, pages 116–125. ACM Press, November 2001.
23. M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
24. James Riordan and Bruce Schneier. A certified e-mail protocol with no trusted third party. *13th Annual Computer Security Applications Conference*, December 1998.
25. 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.
26. RProcess. Selective denial of service attacks. <http://www.eff.org/pub/Privacy/Anonymity/1999_09_DoS_remail_vuln.html>.
27. Paul Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Computer Security Foundations Workshop (CSFW11)*, pages 2–13, Rockport Massachusetts, June 1998. IEEE CS Press.
28. Zhou and Gollmann. Certified electronic mail. In *ESORICS: European Symposium on Research in Computer Security, LNCS Vol. 1146*. Springer-Verlag, 1996.