Mixed Messages: Anonymity through Forwarding

Quinn Carmack
University of Maryland
qcarmack@terpmail.umd.edu

Kevin Goldberg
University of Maryland
kevinhg@terpmail.umd.edu

Akshay Trivedi
University of Maryland
aktriv@terpmail.umd.edu

ABSTRACT

Metadata privacy refers to the protection of information in a secure message channel beyond the content of the messages, such as the sender, the receiver, and the activity over time. This often requires techniques such as routing through mixnets, which are networks of routers that shuffle packets to obscure a message's sender and receiver, and transmitting cover traffic, which is noise meant to be indistinguishable from real encrypted messages to conceal a user's activity. Protocols like Tor use mixnets to prevent a client's IP address from being linked to its traffic, but do not use cover traffic, and are thus susceptible to side-channel attacks such as traffic correlation analysis. Although adding cover traffic can mitigate such vulnerabilities, it comes at the cost of increased bandwidth usage.

We introduce a key insight: in order to defend against many kinds of traffic analysis, it is sufficient for each node's upload rate to remain independent of the amount of data the node needs to send. One approach is for the node to fix a constant rate of upload into the network, filling any unused upload capacity with cover traffic. However, rather than using cover traffic, a node can meet its upload quota by uploading other kinds of data that are indistinguishable from cover traffic to an eavesdropper yet provide a useful function to the node or the network. Replacing cover traffic with packet forwarding, probes of the network, and reliability tests of nodes and service providers, our protocol is able to achieve strong metadata privacy guarantees without significantly wasting bandwidth or increasing latency.

Our protocol implements an anonymous overlay on top of the IP network, offering a connectionless, probabilistically reliable unicast abstraction. It exposes anonymous send and receive primitives that allow users to interact with each other or with participating service providers who can provide any compatible service while limiting metadata leakage. As a concrete application, we instantiate a distributed messaging service that enables users to maintain consistent identities across devices and supports both synchronous and asynchronous message delivery.

1 INTRODUCTION

1.1 Background

The landscape of metadata-private communication has evolved significantly in recent years, especially with advancements in mix networks—first introduced by Chaum [4]—and their integration with a broader set of privacy-enhancing tools. Emerging systems increasingly incorporate support for multi-device communication, anonymous storage, and private information retrieval [8], enabling richer applications without compromising user privacy. Additionally, verifiable shuffling techniques enhance the trustworthiness of mixing operations, post-quantum cryptographic primitives [10] prepare systems for future threats, differential privacy guarantees have been upgraded to cryptographic privacy guarantees[11], and secure hardware (e.g., enclaves) is being explored to protect sensitive operations against compromised software stacks. Researchers have also investigated novel mix networks (mixnet) topologies that optimize for latency, throughput, or anonymity under different adversarial models [1, 6, 9, 10, 12–14].

1.2 Problem

Despite these advances, there are no widely used modern applications that provide unlinkability—obscuring sender-receiver relationships from observers—rendering users vulnerable to the power of global adversaries capable of large-scale traffic analysis attacks. Entities such as ISPs, cloud providers, or surveillance agencies often control or observe wide ranges internet infrastructure, allowing them to monitor traffic patterns across many users and mount active attacks. Even when content is encrypted and identities are hidden, such adversaries can infer communication relationships through timing, volume, and flow correlation techniques. Furthermore, real-world consequential actions are routinely taken by such agencies based on metadata leakage—including investigations, censorship, targeted surveillance, and even lethal drone strikes—underscoring the crucial need for robust metadata protection.

To defend against these threats, systems must incorporate techniques such as cover traffic, constant-rate transmission, semi-trusted servers, and cryptographically-secure packet construction [5, 15], often at the cost of additional latency or bandwidth. Balancing these tradeoffs while providing practical usability remains an open and active area of research.

2 OVERVIEW

In this work, we implement a fully decentralized peer-to-peer mixnet for anonymous communication. Our design

uses the Sphinx packet format, which provides compact, cryptographically strong onion encryption and replay protection, all within a uniform packet size [5]. The network operates over UDP, allowing low-latency and connectionless communication between nodes.

A key feature of our system is that *all clients act as mix nodes*. Each client forwards traffic on behalf of others, ensuring that message sending activity is indistinguishable from routine forwarding. This uniform behavior increases the anonymity set for every participant and eliminates the need for designated infrastructure nodes.

2.1 Anonymous Communication API

The core functionality exposed to applications is the following:

`sendanon(socket, destination, data, params)`
Sends an anonymous message to the specified destination through a path of mixnet hops. Parameters may include path length, redundancy coefficient, service provider set, and bandwidth constraints.

`recvanon(socket, callback, params)`
Registers a passive listener that invokes the callback when a new anonymous message is received. Parameters may include filtering policies, message type, or an erasure code recovery failure handler. Optionally sets an out parameter to a SURB object if one is provided.

A *socket* is required for anonymous networking and the corresponding socket operations are as follows:

`socketanon(params)`
Initializes an anonymous communication context. This includes loading cryptographic keys, configuring parameters such as available bandwidth, cover traffic preferences, and message buffer settings. This begins passive usage of the network, where metadata-private sending is possible, but the node is not yet publicly routable for forwarding nor visible as "online" to peers. We refer to this operational state as *passive mode* where a node does not register in the PKI, does not forward traffic, and is not visible to other participants, but can still send and receive anonymous messages via SURBs or storage providers. Returns a socket-like handle used in subsequent operations.

`bindanon(socket, bandwidth, pkparams)`
Joins the mixnet as a participating node. This step advertises the node's availability by registering in the public key infrastructure (PKI), publishing declared bandwidth and a public encryption key. Once bound, the node may be selected for forwarding traffic and can receive direct messages from peers.

`unbindanon(socket)`
Deregisters the node from the public key infrastructure (PKI), removing it from the set of publicly visible and reachable nodes and disabling forwarding responsibilities. This transitions the node back into passive mode 4.2.1 without tearing down the socket or erasing local state. The node can still send messages and receive replies via SURBs.

`closeanon(socket)`
Gracefully shuts down the anonymous socket. This clears local state and deletes ephemeral keys. If the socket has not been explicitly unbound, this operation implicitly performs the equivalent of `unbindanon()`.

2.2 Goals

Many current systems today rely on dummy traffic or cover messages to improve anonymity by making the traffic pattern more uniform and adding noise to any statistical analysis done by adversaries. However, dummy messages are pure overhead—they do not carry useful payload—so they reduce effective network capacity. Our main goal is to show that client forwarding helps to reduce the need for cover traffic without losing any privacy. Furthermore, we aim to show that we can use cover traffic for useful purposes, such as probing the network to assess user and network behavior.

Though we may fall short in this first version, we ultimately aim to achieve a system design that provides the following **security properties** in the face of our threat model:

- **Confidentiality**: Only the intended recipient can decrypt and read the message content.
- **Integrity**: Any tampering with a message is detectable by the recipient.
- **Authenticity**: The receiver can verify that a message was sent by the claimed sender.
- **Sender Unobservability**: Observers cannot tell whether a node is sending a real message or only cover/forwarding traffic.
- **Receiver Unobservability**: Observers cannot tell whether a node is receiving a real message.
- **Sender Anonymity**: The sender of a message is indistinguishable from other possible senders.
- **Receiver Anonymity**: The recipient of a message is hidden from observers and intermediaries.
- **Sender-Receiver Unlinkability**: No entity can link the sender of a message to its recipient.
- **Forwarding Deniability**: Nodes that forward messages cannot be proven to have authored them.
- **Forward Secrecy**: Compromise of long-term keys does not compromise the confidentiality of past messages.

- **Post-Compromise Security**: The system regains its security guarantees after a temporary compromise.

Further, we aim to achieve the following **performance properties**:

- **Reliable Delivery**: Messages are delivered with high probability despite potential loss, churn, or reordering.
- **Bandwidth Efficiency**: The system minimizes bandwidth overhead relative to actual message content.
- **Latency Awareness**: The protocol supports configurable tradeoffs between latency and anonymity.
- **Scalability**: The system can accommodate a growing number of users, messages, and nodes without significant degradation in performance or anonymity.

2.3 Our Main Contributions

- *A decoupled abstraction for anonymous communication:* We introduce a connectionless, anonymous communication interface that is distinct from metadata-private messaging semantics. While prior work focuses on building full messaging systems, none, to our knowledge, isolate and expose the underlying anonymous networking primitives. We believe this framing can help systematize existing designs and unify them under a clear, reusable abstraction layer.
- *A fully P2P, connectionless anonymous overlay:* We propose a fully peer-to-peer (P2P), connectionless overlay network for anonymous communication. To the best of our knowledge, no existing system employs a completely decentralized and connectionless infrastructure to achieve metadata privacy.
- *Reliability via erasure coding instead of ARQ:* We use erasure codes to provide reliability without introducing the recognizable timing patterns and vulnerabilities of traditional ARQ (Automatic Repeat reQuest) protocols. Our approach avoids kernel-level acknowledgments, retransmissions, and timing feedback, eliminating ARQ-induced metadata leakage. Rateless codes also simplify sender and receiver logic to a fire-and-forget model and enable straightforward replay protection: replays are simply invalid. Additionally, the use of erasure coding enables trust to be distributed across both routing nodes and storage providers.
- *Repurposing cover traffic for utility:* We show how cover traffic can be reused productively for tasks such as probing the network, evaluating node and service provider behavior, and forwarding messages—thereby extracting value from traffic that would otherwise waste bandwidth and lower overall network capacity.
- *An incentive-compatible anonymity ecosystem:* We design an incentive structure in which nodes are rewarded for forwarding traffic, aligning individual self-interest with the collective goal of stronger anonymity and high network capacity. By forwarding messages, a node not only expands its own anonymity set but also contributes to the anonymity of others.

2.4 Threat model

We assume the same threat model as many other metadata privacy papers [1, 10, 12]. There is a globally active adversary that can monitor all links in the network, and cut off any set of links in the network at any time. These behaviors model the abilities of ISPs in the real world. Additionally, we assume the adversary is protocol-aware, and is strategically acting in order to compromise the privacy of the protocol. The adversary may also control some fraction of users in the network, although the adversary must not control all or a significant majority of the users in the network. In particular, as the proportion of users compromised by a single adversary or colluding group of adversaries approaches 1, the privacy guarantees provided by the protocol become weaker.

3 RELATED WORK

Low-latency anonymous communication systems fall into two broad categories: decentralized peer-to-peer (P2P) mixnets and stratified mixnets based on trusted infrastructure. Early P2P systems like Crowds [13] obfuscate senders by probabilistically forwarding messages among a group of peers, while Tarzan [9] generalizes this to IP-layer onion routing with dummy traffic on all links to resist traffic analysis. MorphMix [14] scales circuit construction to millions of nodes, tolerating churn with dynamic peer discovery. Mix Rings [3] route messages through cyclic topologies to optimize throughput. These designs avoid relying on centralized infrastructure but often assume only local views of the network, which can be exploited by adversaries in epistemic attacks—maliciously manipulating partial knowledge to partition or isolate target nodes. In contrast, I2P [**?**] is a mature UDP-based mixnet using garlic routing over a Kademlia DHT, achieving full decentralization without trusted directories.

The more common client-server systems most often use layered mixnets, as in Loopix [12], which is a stratified architecture where server mixes are arranged in layers. Loopix clients send messages through Poisson-delayed mixes via trusted service providers. Loopix provides sender and receiver unobservability under a global passive adversary with ~1–2s latency, but requires centralized access points.

Other recent contributions aim to decouple anonymity from online presence. Groove [1] implements a parallel mixnet

with “oblivious delegation,” allowing clients to send asynchronously via untrusted providers. Each user sends one message per epoch, yielding predictable bandwidth usage but requiring coordination and incurring ~30s latency for large populations. Across all systems, cover traffic and routing policy are central to unlinkability.

In general, P2P networks scale more easily than client-server models that consolidate trust in servers that must each be able to handle thousands of clients. As such, our design follows the P2P tradition.

3.1 Groove

Groove provides sender-receiver anonymity through a round-based mixnet design, where each user (or the service provider, on their behalf) transmits a fixed number of messages per round, determined by a `MAX_BUDDIES` parameter. To maintain a constant outgoing traffic rate, users fill unused message slots with cover traffic. This uniformity prevents traffic analysis based on upload spikes or variable sending behavior.

Receiver anonymity is achieved through the use of dropoff points and an "oblivious fetch" mechanism. Rather than sending messages directly to recipients, they are shuffled at a sequence of mix nodes, from which intended recipients retrieve their messages anonymously.

Groove employs onion routing for hop-by-hop encryption and relies on the assumption that at least one node along each path is honest to prevent replay attacks. While the protocol offers strong anonymity guarantees, its design is relatively heavy-handed: it requires synchronized rounds, constant traffic padding from all users, and substantial bandwidth overhead, making it resource-intensive and less suitable for constrained environments.

Ultimately, Groove’s anonymity hinges on traffic regularity. By enforcing constant-rate message sending and saturating the mixnet with cover traffic, it thwarts attempts to correlate users based on timing or volume patterns. However, this comes at the cost of efficiency and latency on the order of ~30s.

3.2 Loopix

Loopix is a low-latency metadata private communication framework using a stratified mixnet topology. In Loopix, users can use “loops” which are routes from a user to themself in order to protect against active attacks and provide additional cover traffic. Loopix uses Poisson mixing at mixnet servers to circumvent the need for synchronized rounds while also having predictable latencies and providing anonymity guarantees.

However, Loopix separates mixnet servers from clients. This prevents forwarded traffic from being a potential candidate for a client’s cover traffic.

3.3 Echomix

Echomix describes the architectural design of Katzenpost. Echomix incorporates post-quantum cryptographic primitives layered on top of the Sphinx packet format, enhancing future-proof security. Unlike Groove [1], where the service provider acts as the client’s entry point into the mixnet, Echomix positions the client’s provider on the opposite side of the mixnet, thereby reducing the amount of metadata observable by the provider. Echomix introduces an “echo” mechanism, conceptually similar to the loops used in Loopix, but with a key distinction: each echo targets a fixed recipient, who replies using a Single-Use Reply Block (SURB) to maintain anonymity.

Table 1: Summary of Notation

Symbol	Meaning
P	The set of all online nodes in the network
N	Number of online users in the network
i	A specific client
PK_i	Public encryption key of a user
IP_i	IP address and port of a user
B_i	Constant, advertised message transmission rate of client
u_i	Rate of user-generated messages by user i
h	Maximum number of hops supported by the network
ρ	Leniency factor: approximates $\frac{\max \Sigma u_i}{\text{avg } \Sigma u_i}$ (see 4.2.3)

4 DESIGN

We consider a network of users, where each user i is identified by a public key PK_i. Only the corresponding user possesses the matching private key SK_i. To communicate, a sender must first obtain the recipient’s public key through an out-of-band channel. Our protocol enables metadata-private delivery of messages using the recipient’s public key.

In order to minimize a user’s susceptibility to traffic analysis-based attacks, we find that their device’s rate of upload into the network should be independent of both their own activity (e.g., how often they send messages) and that of their peers. In our design, all nodes transmit messages at a constant bandwidth. This constant-rate design ensures that the adversary’s posterior belief about any message’s origin, conditioned on all observations, remains statistically indistinguishable from their prior.

Clients emit four categories of messages: (1) user-generated messages to peers, (2) service requests to infrastructure nodes with attached SURBs, (3) loop messages for sensing network security, and (4) outgoing forwarding traffic on behalf of other users.

Clients receive four analogous categories: (1) synchronous messages from peers, (2) service provider responses via

SURBs, (3) incoming forwarding packets, and (4) returning loops.

Incoming traffic, while less controllable by the client, still requires careful handling. The distribution of expected forwarding load for each node is publicly computable from the PKI. Over time, the sampling of forwarding nodes—weighted by advertised bandwidths—can be approximated by a Poisson process, due to many trials with low individual selection probability. Detectable deviations from this distribution may reveal client behavior, motivating the need for even rate shaping of user and peer-driven incoming traffic.

Each message at a node is scheduled for transmission after a delay sampled from an exponential distribution. The memoryless property of the exponential distribution, as described in [10, 12], enables continuous-time mixing without the need for synchronized rounds or threshold-based batching, thereby reducing latency.

While the precise queuing mechanics are covered in prior work, we make the following observations. The expected delay for a packet sampled from an $\text{Exp}(\lambda)$ distribution is $1/\lambda$. Given a fixed output bandwidth b_i (in packets per second) at node i, the expected number of other packets that will be transmitted before a given packet is approximately b_i/λ. This quantifies the extent to which a packet is mixed with others at each hop.

Furthermore, due to the fan-out at each hop—where packets are forwarded independently along randomly chosen paths—the anonymity set grows approximately exponentially with the number of hops. This compounding effect is key to achieving strong mixing properties in our low-latency setting.

4.1 Public Key Directory

We publish a Public Key Infrastructure (PKI) to distribute global information pertaining on all users in the network which we call the directory. The PKI is represented as a set of tuples:

$$\text{PKI} = \{(\text{IP}_i, B_i, \text{PK}_i) \mid i \in \mathcal{P}\}$$

Each tuple corresponds to a user $i \in \mathcal{P}$ in the network and contains:

(1) *Send Bandwidth* B_i. By publishing the bandwidth capacity, other users will know the weight by which to probabilistically select the user for forwarding packets. With sufficient conforming users in the network, the percentage of the network's data that traverse through this user will approach the proportional network bandwidth for this user.
(2) *Public encryption key* PK_i. This is used to encrypt an ephemeral symmetric key for onion encryption and decryption at each hop.
(3) *Network address* IP_i. The IP address and port at which to reach a node.

As in Echomix [10] the security of our system relies on a consistent, accurate view of the network across all users. We employ a similar decentralized protocol by which multiple directory authorities reach consensus and sign the PKI document each round. As the system scales, the document and directory authorities can be partitioned to divide up responsibility for the PKI document by using hashes of public keys.

When coming online or going offline, a user notifies a the directory authorities responsible for its record. We do not discuss the details of the PKI implementation further as we consider the problem of distributed, untrusted storage to be a different research question.

4.2 Sending Messages

Since metadata privacy is maintained by the constant upload bandwidth, each user must ensure that it uploads the constant amount as advertised in the public ledger. If a user has messages to send, then it will send those as per the bandwidth rate. To ensure that messages do not become lost in the network, we take advantage of fragmentation via rateless erasure codes– in particular, Raptor codes [16]. Each message is encoded into different Raptor code chunks, where each chunk is routed independently throughout the network. Raptor codes have linear time encryption and decryption functions which helps to avoid added latency.

Messages are sent directly to the destination user with a unique message ID generated by a pseudorandom number generator, used for reconstruction of the erasure codes. Obviously, this unique message ID is the same across all message erasure code chunks, and they are not encoded as part of the chunk. They are tacked on after the erasure code chunk so that a single received chunk can afilliated with a message ID. Lastly, the entire message, including the message chunk and the message ID, is cryptographically encoded using the receiver's public key which can only be decrypted with the receiver's secret key.

Furthermore, messages are onion encrypted to ensure that each user can only decrypt its onion layer [7]. This way, each hop is blind of the destination user and the sender. We denote the onion route to be the path, where the path has h hops, including the final destination user. Hops in the onion route are chosen at random as per the advertised bandwidth of each user, proportional to the total network bandwidth. That is, any given user i in the network is chosen to be a hop with probability $b_i/\sum b_j$, where $\sum b_j$ is the total advertised network bandwidth.

To ensure that message chunks are indistinguishable from others, we must ensure all packets are the same size. To

Algorithm 1 Bandwidth-Weighted Path Construction

Require: Public Key Infrastructure PKI containing user set $\mathcal{P} = \{P_1, \ldots, P_N\}$ with declared bandwidths $\{b_1, \ldots, b_N\}$
Require: Number of hops h
Ensure: Path $\mathcal{N} = \{P_{i_1}, \ldots, P_{i_h}\}$, where each P_{i_k} is selected with probability proportional to b_{i_k}

$W \leftarrow \sum_{j=1}^{N} b_j$ ▷ Total bandwidth
for $k \leftarrow 1$ to h **do**
  Compute weights: $w_j \leftarrow b_j / W$ for all j
  Sample i_k from categorical distribution with probabilities $\{w_1, \ldots, w_N\}$ without replacement
  Add P_{i_k} to $\mathcal{N}$
end for
return $\mathcal{N}$

ensure that all message chunks are the same size even after onion encryption/decryption, we use Sphinx packets [5]. Sphinx are cryptographically secure and indistinguishable after a single onion-layer decryption. Sphinx is also *key-private* meaning ciphertexts do not reveal which public key was used to encrypt them. The ciphertext is computationally indistinguishable from a ciphertext encrypted to anyone else such that no one (not even the recipient) can determine who a ciphertext was encrypted for unless they try to decrypt it.

However, if the user does not have any messages to send, an attacker could detect a drop in the outgoing bandwidth and realize the user is no longer communicating. To remedy this, we introduce the following protocol [2].

Algorithm 2 Bandwidth Allocation and Forwarding

Require: Bandwidth b_i, user messages, forward data, parameters ρ, h, b_{loop}, L
Ensure: Bandwidth-conforming output traffic scheduled

if user has messages to send **then**
  Schedule up to $\frac{\rho \cdot b_i}{h}$ bandwidth of new data
end if
Schedule L loops ▷ Costs $L \cdot b_{loop}$ bandwidth
$fwd\_budget \leftarrow b_i - \frac{\rho \cdot b_i}{h} - L \cdot b_{loop}$
if forward data $\geq$ *fwd_budget* **then**
  Forward *fwd_budget* of forward data
  Randomly drop the remaining forward data
else
  Forward all available forward data
  Fill remaining bandwidth with loop traffic
end if

If the user has messages to send, it sends up to $\rho \cdot b_i / h$ in bandwidth capacity at max, where ρ is a leniency constant. If all users in the network were to send messages, then this would match the total network bandwidth capacity since each message stays in the network for h hops. However, since messages can be dropped and not all users are necessarily sending messages at once, users can send a bit more than the theoretical maximum by introducing the leniency constant ρ. This leniency constant should be empirically determined, increasing if it appears that the maximum network useful usage is below the limit, and decreased if packets are regularly getting dropped.

After sending any messages that the user has available, it must pad the advertised bandwidth to ensure that adversaries can not detect any changes in outgoing links by sending loops. Loops can be used for a variety of things, but they are mainly used for testing other users in the network. Like Groove [1], this is a form of cover traffic to conceal whether a user is sending messages. However, unlike Groove, this chaff is not useless, as loops can be used for purposes as outlined later.

After sending L number of loops that each take up b_{loop} bandwidth, there is $b_i - \frac{\rho \cdot b_i}{h} - L \cdot b_{loop}$ outgoing bandwidth left for the user. The user then tries to use as much of this remaining bandwidth for forwarding messages. Just as this user randomly chooses other users in the network to forward messages, this user also will be randomly selected forwarding messages. If there are too many messages to forward, then messages will be randomly dropped to ensure the outgoing bandwidth remains constant. If there is not sufficient incoming forward bandwidth to fill up the outgoing bandwidth, then loops are used to pad out the rest.

To maintain fairness between forwarding traffic, we impose an exponential delay, similar to Loopix [12]. This exponential delay factor is λ, and the function is imposed on all forwarded traffic. This ensures that an attacker cannot correlate the incoming traffic packets with outgoing packets, as they will not necessarily be in the same order from the exponential time delay function. These forwarded packets will be scheduled to be sent as per the randomly picked value from the exponential delay function.

4.2.1 Passive participation. Our design can seamlessly support a typical client-server mixnet usage model for users who do not wish to forward traffic or expose their IP address. These clients are not routable by peers unless they explicitly initiate contact via the PKI directory and share their current IP address.

Such clients can still maintain privacy by sending at a constant rate using only user messages and loop traffic as cover. This form of passive participation provides plausible deniability without contributing to network forwarding.

The bandwidth demands of passive clients could be offset by long-lived servers dedicated to mixing or forwarding, as commonly done in many mixnet designs. This “dark-mode”

operation is especially useful when a client suspects it is under active surveillance or attack.

While we acknowledge that passive mode may be appropriate in certain scenarios, we maintain that active forwarding generally improves a node's privacy. Therefore, we expect passive operation to remain the exception rather than the norm.

4.2.2 Timeouts. As outlined later, receivers will send an ACK back to the sending user to let them know the message has been received. If the sender does not receive an ACK after $T_{timeout}$ time, then the sender will retry by sending the entire message again, broken up with new erasure codes with the κ overhead. By sending new erasure codes, this leaves the receiver with the possibility to reconstruct the message if it did receive some erasure code message chunks, but not enough for reconstruction.

If after three attempts no ACK has been received, the client is assumed to be offline. At this point, messages are sent to the service provider. More details about the server and offline communication is outlined in 4.6.1.

4.2.3 Convergence of Network Rates. Our send protocol divides a users upload bandwidth into multiple regions based on factors such as h (maximum number of hops of a packet) ρ (leniency factor). These constants are motivated by the following analysis of what the network should look like under ideal conditions.

For example, suppose every user continuously sends u_i bytes of original data into the network, using a random route of h hops. Suppose every user uses the same probability distribution to choose each node in the path, based on the nodes' relative network capacities. By linearity of expectation, the amount each node will be expected to forward (denote this f_i) will be proportional to its share of the probability distribution, and hence its network capacity. In this model, any data a node transmits is either original or forwarded. So denoting the transmission rate of each node with B_i, we have $B_i = u_i + f_i$. Since B_i and f_i are proportional to the nodes network capacity, u_i must be as well. Additionally, each byte sent will cause a byte to be sent from h nodes in the network, once for each hop that byte takes. So, the sum of sum of all nodes' transmission rates is is $\sum_{i \in \mathcal{P}} B_i = h \cdot \sum_{i \in \mathcal{P}} u_i$. Together with the fact that B_i is proportional to u_i, this implies that $B_i = h \cdot u_i$. Hence, when traffic is amortized across al nodes, a node is expected to use a proportion $\frac{1}{h}$ of its transmission rate for its own original traffic.

However, in the real world, nodes may have a variable rate of sending original data, i.e. u_i may change for each node over time. Since we fix B_i for metadata privacy and the analysis above depends only on the sum $\sum u_i$, we allow some nodes to send more than their $\frac{1}{h} \cdot B_i$ of original traffic, in the hope that there are other nodes which are using less than their $\frac{1}{h} \cdot B_i$. We call this factor the leniency factor, $\rho > 1$, and hence a node sends at most $\frac{\rho \cdot B_i}{h}$. In the case where too many node send above $\frac{1}{h} \cdot B_i$, we expect that the amount of traffic will be more than the network can support, and reliability will decrease without compromising security.

4.3 Receiving Messages

When a user comes online, the client will prioritize loading messages from conversations in the order that they are viewed in the user interface. Additionally the client will check for new messages from all friends with a single request to the appropriate storage providers. The client will slowly draw messages from the service provider to avoid correlation with its login to the network. For more privacy or for faster downloads of large files, a user can begin downloading messages in passive participation mode before coming online by publishing their entry to the PKI.

Any incoming messages whose final destination is the user are synchronous messages intended for the user. In our system, synchronous messages may or may not reveal the sender's identity or public key. They also may or may not contain a SURB for anonymous replies. All messages are decrypted with the user's private key. Each message sent has a unique message ID which is used for erasure code reconstruction. After sufficient message blocks have been received with a particular message ID, the receiver can recover the message [16]. Once the message has been recovered, if the sender is an existing peer a single ACK is sent. If the message is not from a peer and there is a SURB provided, it can be used for reply. If insufficient packets have been received for reconstruction and the message header proves the sender is a peer, a NACK will be sent.

4.4 Testing the Network

A potential concern with permitting anyone to join the mixnet is that malicious actors can mount a denial-of-service (DOS) attack by joining the mixnet as a node and dropping many or all packets intended for forwarding. So, it would be useful for a user to be able to test whether a node is actually forwarding packets, so that the user can bias their route selection algorithm towards more reliable nodes.

Since nodes have no information about the metadata of a packet they are asked to forward, a node cannot selectively forward some packets with lower probability and others with higher probability, so a node can only drop packets with a fixed probability (or a probability based on extrinsic factors, like time). In particular, a node cannot distinguish between "real" user messages and cover traffic in the form of a loop from a user to itself.

Therefore, a user can "test" the reliability of a node (or group of nodes) by routing a message in a loop through the

nodes from the user to itself. Since a malicious node can change its reliability over time, the user should do this often, and should trust more recent tests. Since nodes cannot distinguish between these loop messages and actual messages from the user to a recipient, the reliability observed for these loop messages should reflect the reliability of sending to an arbitrary recipient.

4.5 Erasure Codes

Our protocol intentionally drops valid messages to enhance privacy. To ensure reliable delivery despite this, we analyze the probability that a given message successfully reaches its destination and apply forward error-correcting erasure codes accordingly. This success probability informs our choice of how many rateless-encoded blocks to transmit initially.

Rateless codes are particularly suitable in this setting, as they allow the sender to generate an unbounded number of encoded symbols. If the recipient is unable to recover the original message from the initial transmission, additional blocks can be streamed as needed until successful decoding is possible.

We will show that it is highly unlikely for all encoded blocks to be lost, and we can therefore base the decision to retransmit additional check blocks on the reception of a single negative acknowledgment (NACK) from the recipient.

4.5.1 Probability model. We denote by λ_i^{in} the incoming message rate to node i, which consists of three types of traffic: real messages from peers or service providers, messages to be forwarded on behalf of other nodes, and self-originated loop messages. Each node forwards as much as it can but drops the rest, so: $\lambda_i^{\text{in}} = \lambda_i^{\text{out}} + \lambda_i^{\text{drop}}$, meaning messages entering node i equal those successfully forwarded out plus those dropped. We define drop probability $p_i = \lambda_i^{\text{drop}}/\lambda_i^{\text{in}}$ for node i, and success rate $\rho_i = 1 - p_i$ as the fraction of packets forwarded successfully. Because routes are selected at random, each node's p_i may depend on congestion. We assume nodes are chosen such that on average $\lambda_i^{\text{in}} \approx B_i$.

For a given message traversing h hops through nodes with success probabilities $\rho_1, \rho_2, \ldots, \rho_h$, the end-to-end success probability for a single packet is $\prod_{j=1}^{h} \rho_j$. In a simplified homogeneous model, assume each hop has the same drop probability p, then a single message is delivered with probability $q = (1-p)^h$. For example, if $p = 0.05$ (5% drop per hop) and $h = 3$ hops, then $q = (0.95)^3 \approx 0.857$ (about 85.7% chance the message survives all 3 hops).

To improve reliability, we introduce erasure coding over each message. A message is split into k pieces and encoded into n packets such that any k out of the n packets suffice to reconstruct the original message. This can be achieved with a systematic (n, k) erasure code like Reed–Solomon or with a fountain code (e.g. RaptorQ). The ratio $n/k > 1$ represents redundancy overhead. A maximum distance separable (MDS) code (like Reed–Solomon) can recover from any $n-k$ packet losses (i.e. up to $(n-k)/n$ fraction dropped) with certainty. Modern fountain codes can achieve probabilistic MDS properties with very low overhead; for example, a RaptorQ code can often recover the message from just $1.05k$ packets (only 5% extra) with high probability.

Let us denote by P_{succ} the message delivery success probability (i.e. the probability that the recipient obtains enough fragments to reconstruct the message). Given independent drop events, this is the probability that at least k out of n coded packets survive their multi-hop routes. We can express this using the tail of a binomial distribution. If each packet has success probability q (as defined above for a single packet through the mixnet), then:

$$P_{\text{succ}} = \Pr[X \geq k] = \sum_{i=k}^{n} \binom{n}{i} q^i (1-q)^{n-i}$$

where $X \sim \text{Binomial}(n, q)$ is the number of packets that arrive successfully.

4.5.2 High delivery probability. Require a minimum message success probability, e.g. $P_{\text{succ}} \geq 0.95$ (95%). This will be the target reliability level for the system. The code type and (k, n) can thus be tuned to meet this goal. Ideally, we want just enough redundancy to meet the target success probability as excessive redundancy would consume bandwidth unnecessarily.

We choose the redundancy n such that the expected number of surviving packets $E[X] = nq$ is greater than the threshold k by a comfortable margin. If n is chosen so that $qn = k+\Delta$ for some safety margin Δ, the chance that fewer than k packets arrive is extremely small. In summary, the mixnet can deliver and reconstruct messages with high probability by combining path diversity and erasure coding.

4.5.3 Low probability of no arrivals. We show that it is feasible to rely on infrequent negative acknowledgments (NACKs) instead of acknowledgments (ACKs), based on the observation that the probability of total packet loss is exceedingly low. Specifically, the probability that *none* of the n packets arrive is:

$$\Pr[X = 0] = (1-q)^n$$

This probability decays exponentially with n for any fixed success probability $q > 0$, and becomes negligible even with modest redundancy. For example, with $q = 0.8$ and $n = 4$, the total loss probability is only 0.0016 (or 0.16%). This implies that in nearly all cases, at least one packet reaches the receiver.

As a result, receivers can detect message presence and issue a NACK only if recovery fails. This approach minimizes

upstream signaling and reduces overhead, while maintaining high reliability.

4.6 Service Providers

Service providers (SPs) participate in the network by responding to anonymous client requests received through the overlay network. To maintain metadata privacy, given that SPs do not add delays for mixing, we propose that each incoming request is matched with exactly one outgoing response, thereby ensuring that traffic rates in and out of the SP remain uniform and do not leak usage patterns. This symmetry can be achieved by attaching Single-Use Reply Blocks (SURBs) to requests, which enable SPs to return acknowledgments or service responses anonymously.

While we want to emphasize that many types of services can be supported, we focus on one particularly useful application: asynchronous message storage for offline communication. Here, clients issue anonymous `PUT` and `GET` requests to a key-value store interface, mapping message identifiers to encrypted message payloads.

4.6.1 Offline Communication. Although the protocol interface only provides support for online communication, offline messaging can be layered on top with minimal overhead. When a sender fails to receive acknowledgments after three transmission attempts, it assumes the recipient is offline. In such cases, the sender stores the message at a designated service provider acting as a temporary message depot.

We borrow the BACAP scheme (Blinding-and-Capability scheme) from [10] where messages are stored at an evolving shared secret key. In our protocol each erasure code block is store at a different secret key to obscure file sizes. The keyspace is large enough so that collisions are highly improbable and only the intended recipient can retrieve their message.

To preserve anonymity, messages should be uploaded to storage providers at a slow and regular rate, avoiding spikes that could correlate a user's offline status with increased storage usage. Since sender identities are hidden, an increase in storage provider traffic does not reveal which users are actively sending.

When a user rejoins the network, they contact service providers where their friends are likely to have stored messages. Message downloads from these providers should also occur slowly and sporadically to prevent correlation with the user's return to the network. In addition, the returning user can notify all online senders who previously left messages, indicating that they are back online. These notifications prompt senders to resume direct communication rather than relying on the storage intermediary. If acknowledgments to these notifications are not received within three attempts, the notification need not be redirected to a service provider, as the intended recipient is also considered offline.

5 PRIVACY ANALYSIS

In analyzing the security properties of our design we consider the different types of threats posed both from outside and inside the system: global traffic analysis, active network attacks, malicious clients, and malicious service providers.

5.0.1 Flooding for denial-of-service. An attacker can flood a user with traffic, consuming a large proportion of their available forwarding bandwidth f_{out}. By composing onion routes through nodes controlled by the adversary immediately before and after the target, the attacker can with high probability differentiate the forwarded messages and infer user-originated messages. This attack is possible because f_{out} is inversely correlated with u_i (rate of user-generated messages), so by influencing f_{out} the attacker can obtain information about u_i. For example, if u_i is high, then the client will forward messages at a lesser rate, so the attacker will expect to see less of their messages pass through the client.

A defense of this is realized through our routing PKI. We know the approximate distribution of expected sources of forwarded messages as well as destinations. This can be used to detect a malicious client injecting their nodes into routes more often than statistically probable.

Our security analysis has not yet included the *prevention* of DOS attacks and this remains open for further work. However, we believe that the nature of a P2P network makes our service much more resilient to DOS attacks due to the decentralization of the service infrastructure.

5.0.2 "n-1 attack". The n-1 attack relies on an active adversary being able to (undetectably) control all inputs to a node and observe all outputs. The adversary sends the node $n-1$ messages and leaves the n-th message as one supplied by the honest user under observation. The attacker then correlates the input and output message patterns to deanonymize the user's message by effectively crowding all of its anonymity set.

As [10] notes, any attack, like the n-1 attack, that involves disruption of the network can be detected via loops. When under attack, the node will detect the failed return delivery of these periodic loop messages. Without incoming loop traffic, we cannot guarantee anonymity to the same degree, so the client can switch into a high-alert mode, removing themselves from the public ledger, suspending forwarding, and using heavy cover traffic to obscure messages. We discuss this mode further in 4.2.1. significantly strengthen resistance to targeted de-anonymization attempts.

5.0.3 Dishonest nodes. A node may deviate from the protocol by not correctly implement the exponential distribution mixing delay specified in the continuous Poisson mixing strategy. For instance, a node could decrypt a packet and immediately forward it, bypassing the delay. Though the packet's contents may appear cryptographically different, through timing analysis the incoming and outgoing packet can be linked.

However, as long as there is at least one honest node on the message path, the packet will be mixed with others, building its anonymity set. For every additional honest node, the anonymity set grows exponentially. Assuming a proportion of malicious nodes m and a path of h hops, the expected number of honest nodes on the path is $(1-m)h$. By adjusting h, the user can tune the level of desired anonymity.

5.0.4 Dishonest service providers. A service provider (SP) may deviate from its declared behavior, such as failing to store or return messages. In the context of offline message storage, this risk can be mitigated by distributing erasure-coded message blocks across multiple independent providers. As long as the recipient can successfully retrieve at least one block, they can detect message loss and assume SP misbehavior or network failure.

If a provider is unresponsive or returns invalid data, the recipient can inform the sender of the failure. The sender and recipient may then coordinate to switch to a different, presumably honest, SP. This approach enables fault detection and resilience without relying on any single provider for availability or integrity.

5.0.5 Replay and tagging attacks. Any Sphinx packet or SURB can, in principle, be duplicated and re-injected into the mixnet an arbitrary number of times. This can result in a traffic surge toward the recipient, which may be visible to a large-scale adversary, uncloaking the sender. Additionally, a tagging attack is one in which a malicious node subtly modifies a packet to mark it, allowing the attacker to trace it as it moves through the network. In order to prevent these vectors of attack, the Sphinx packet format includes both a replay prevention mechanism and a modification detection mechanism using message authentication codes (MACs) [5].

Honest nodes maintain a cache of the recently seen headers (or their hashes) for replay prevention and verify the MAC to ensure packet integrity. Any packet with a duplicate tag or incorrect MAC is discarded. Furthermore, it may be possible to infer that an honest node would not forward such a packet. If that inference is cryptographically valid, the sender can be flagged as potentially malicious and further traffic from it may be rate-limited or dropped.

5.0.6 Joining and leaving the network. A key metadata privacy concern arises when a user joins (or leaves) the network: if another user's download rate suddenly increases (or decreases), this surge can potentially be correlated with the newly online (or offline) user. More generally, any join or leave event may induce observable traffic pattern changes—either directly to the user or indirectly via their peers or associated service providers.

To mitigate this, we introduce the `bindanon` and `unbindanon` primitives, which decouple public key infrastructure (PKI) visibility from active network participation. This separation allows users to appear offline or anonymous within the PKI, while still maintaining passive or limited engagement with the network. By reducing the visibility of user presence or re-entry in the global directory, these primitives provide additional protection against adversaries attempting correlation or intersection attacks based on PKI observations and traffic changes.

6 FURTHER WORK

6.0.1 Random bridges. In scenarios where users wish to conceal their IP address from the rest of the network, we propose the use of ephemeral volunteer relays—similar to the Snowflake system in Tor—which act as anonymous first-hop "bridges" [2]. These bridges serve as intermediaries between the user and the public network, effectively masking the client's origin.

Such a system would be particularly useful in adversarial settings where censors may blocklist known IP addresses published in the PKI directory. From a security standpoint, bridges can be modeled similarly to ISPs: they observe all ingress and egress traffic and can potentially mount active attacks, including man-in-the-middle attempts. Careful cryptographic design and protocol-level defenses must be employed to prevent bridges from compromising anonymity or traffic integrity.

6.0.2 Client simulation and network dynamics. To better understand the performance and security characteristics of our protocol under realistic conditions, we plan to simulate network dynamics including user churn, varying bandwidth capacities, and adversarial behaviors. In parallel, we aim to implement a full reference client to evaluate protocol correctness, latency, and message delivery success under different network topologies and adversarial models.

6.0.3 Microservices-based architecture. We believe the field would benefit from the modularization of anonymous communication systems into a set of composable microservices. By decoupling core functions—such as path selection, packet mixing, SURB management, and message scheduling—we enable flexible deployments that can be tailored to diverse

application requirements and threat models. A microservices-based architecture would also support experimentation with alternative mixing strategies, topologies, packet formats, and attack mitigation mechanisms, fostering more robust and extensible privacy-preserving communication systems.

REFERENCES

[1] Ludovic Barman, Moshe Kol, David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2022. Groove: Flexible Metadata-Private Messaging. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 735–750. https://www.usenix.org/conference/osdi22/presentation/barman

[2] Cecylia Bocovich, Arlo Breault, David Fifield, Serene, and Xiaokang Wang. 2024. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *Proceedings of the 33rd USENIX Conference on Security Symposium (SEC '24)*. USENIX Association, USA, Article 148, 18 pages.

[3] Matthew Burnside and Angelos D. Keromytis. 2006. Low Latency Anonymity with Mix Rings. In *Information Security*, Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 32–45.

[4] David L. Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (Feb. 1981), 84–90. https://doi.org/10.1145/358549.358563

[5] George Danezis and Ian Goldberg. 2009. Sphinx: A Compact and Provably Secure Mix Format. In *2009 30th IEEE Symposium on Security and Privacy*. 269–282. https://doi.org/10.1109/SP.2009.15

[6] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2024. Divide and Funnel: A Scaling Technique for Mix-Networks. In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*. 49–64. https://doi.org/10.1109/CSF61375.2024.00031

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: the second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13 (SSYM'04)*. USENIX Association, USA, 21.

[8] Saba Eskandarian, Henry Corrigan-Gibbs, Matei A. Zaharia, and Dan Boneh. 2019. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. *ArXiv* abs/1911.09215 (2019). https://api.semanticscholar.org/CorpusID:208201930

[9] Michael J. Freedman and Robert Morris. 2002. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*. Association for Computing Machinery, New York, NY, USA, 193–206. https://doi.org/10.1145/586110.586137

[10] Ewa J. Infeld, David Stainton, Leif Ryge, and Threebit Hacker. 2025. Echomix: a Strong Anonymity System with Messaging. *arXiv preprint arXiv:2501.02933* (2025). https://arxiv.org/abs/2501.02933

[11] Albert Kwon, David Lu, and Srinivas Devadas. 2020. XRD: Scalable Messaging System with Cryptographic Privacy. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 759–776. https://www.usenix.org/conference/nsdi20/presentation/kwon

[12] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, Canada, 1199–1216. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska

[13] Michael K. Reiter and Aviel D. Rubin. 1998. Crowds: anonymity for Web transactions. *ACM Trans. Inf. Syst. Secur.* 1, 1 (Nov. 1998), 66–92. https://doi.org/10.1145/290163.290168

[14] Marc Rennhard and Bernhard Plattner. 2002. Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society (WPES '02)*. Association for Computing Machinery, New York, NY, USA, 91–102. https://doi.org/10.1145/644527.644537

[15] Philip Scherer, Christiane Weis, and Thorsten Strufe. 2024. Provable Security for the Onion Routing and Mix Network Packet Format Sphinx. *Proceedings on Privacy Enhancing Technologies* 2024 (10 2024), 755–783. https://doi.org/10.56553/popets-2024-0140

[16] A. Shokrollahi. 2006. Raptor codes. *IEEE Transactions on Information Theory* 52, 6 (2006), 2551–2567. https://doi.org/10.1109/TIT.2006.874390