The Loopix Anonymity System

Ania M. Piotrowska and Jamie Hayes, ***University College London;*** **Tariq Elahi,** ***KU Leuven;***
Sebastian Meiser and George Danezis, ***University College London***

https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska

This paper is included in the Proceedings of the
26th USENIX Security Symposium

August 16–18, 2017 • Vancouver, BC, Canada

ISBN 978-1-931971-40-9

Open access to the Proceedings of the
26th USENIX Security Symposium
is sponsored by USENIX

The Loopix Anonymity System

Ania M. Piotrowska
University College London

Jamie Hayes
University College London

Tariq Elahi
KU Leuven

Sebastian Meiser
University College London

George Danezis
University College London

Abstract

We present *Loopix*, a low-latency anonymous communication system that provides bi-directional 'third-party' sender and receiver anonymity and unobservability. Loopix leverages cover traffic and *Poisson mixing*—brief independent message delays—to provide anonymity and to achieve traffic analysis resistance against, including but not limited to, a global network adversary. Mixes and clients self-monitor and protect against active attacks via self-injected loops of traffic. The traffic loops also serve as cover traffic to provide stronger anonymity and a measure of sender and receiver unobservability. Loopix is instantiated as a network of Poisson mix nodes in a stratified topology with a low number of links, which serve to further concentrate cover traffic. Service providers mediate access in and out of the network to facilitate accounting and off-line message reception.

We provide a theoretical analysis of the Poisson mixing strategy as well as an empirical evaluation of the anonymity provided by the protocol and a functional implementation that we analyze in terms of scalability by running it on AWS EC2. We show that mix nodes in Loopix can handle upwards of 300 messages per second, at a small delay overhead of less than 1.5 *ms* on top of the delays introduced into messages to provide security. Overall message latency is on the order of seconds – which is relatively low for a mix-system. Furthermore, many mix nodes can be securely added to the stratified topology to scale throughput without sacrificing anonymity.

1 Introduction

In traditional communication security, the confidentiality of messages is protected through encryption, but this exposes meta-data, such as who is sending messages to whom, to network eavesdroppers. As illustrated by recent leaks of extensive mass surveillance programs[1], exposing such meta-data leads to significant privacy risks.

Since 2004, Tor [20], a practical manifestation of circuit-based onion routing, has become the most popular anonymous communication tool, with systems such as Herd [33], Riposte [11], HORNET [10] and Vuvuzela [46] extending and strengthening this paradigm. In contrast, message-based architectures, based on mix networks, have become unfashionable due to perceived higher latencies, that cannot accommodate real-time communications. However, unless cover traffic is employed, onion routing is susceptible to traffic analysis attacks [7] by an adversary that can monitor network links between nodes. Recent revelations suggest that capabilities of large intelligence agencies approach that of global passive observers—the most powerful form of this type of adversary.

It is not sufficient to provide strong anonymity against such an adversary while providing low-latency communication. A successful system additionally needs to resist powerful active attacks and use an efficient, yet secure way of transmitting messages. Moreover, the system needs to be scalable to a large number of clients, which makes classical approaches based on synchronized rounds infeasible.

For this reason we reexamine and reinvent mix-based architectures, in the form of the Loopix anonymity system. Loopix is resists powerful adversaries who are capable of observing all communications and performing active attacks. We demonstrate that such a mix architecture can support low-latency communications that can tolerate small delays, at the cost of using some extra bandwidth for cover traffic. Message delay and the ratio of cover to real traffic can all be flexibly traded-off against each other to offer resistance to traffic analysis. Loopix provides *'third-party'* anonymity, namely it hides the sender-receiver relationships from third parties, but

[1] See EFF's guide at `https://www.eff.org/files/2014/05/29/unnecessary_and_disproportionate.pdf`

senders and recipients can identify one another. This simplifies the design of the system, prevents abuse, and provides security guarantees against powerful active adversaries performing $(n-1)$ attacks [41].

Loopix provides anonymity for private email or instant messaging applications. For this reason, we adopt and leverage an architecture by which users of Loopix are associated with service providers that mediate their access to a stratified anonymity system. Such providers are only semi-trusted[2], and are largely present to maintain accounting, enforce rate limiting, and ensure messages sent to off-line users can be retrieved at a later time. To provide maximal flexibility, Loopix only guarantees unreliable datagram transmission and is carried over UDP. Reliable transport is left to the application as an end-to-end concern [39].

Contributions. In this paper we make the following contributions:

- We introduce Loopix, a new message-based anonymous communication system. It allows for a tunable trade-off between latency and genuine and cover traffic volume to foil traffic analysis.
- As a building block of Loopix we present the *Poisson Mix*, and provide novel theorems about its properties and ways to analyze it as a pool-mix. Poisson mixing does not require synchronized rounds, can be used for low-latency anonymous communication, and provides resistance to traffic analysis.
- We analyze the Loopix system against a strong, global passive adversary. Moreover, we show that Loopix provides resistance against active attacks, such as trickling and flooding. We also present a methodology to empirically estimate the security provided by particular mix topologies and other security parameter values.
- We provide a full implementation of Loopix and measure its performance and scalability in a cloud hosting environment.

Outline. The remainder of this paper is organized as follows. In Section 2, we present a brief, high-level overview of Loopix and define the security goals and threat model. In Section 3, we detail the design of Loopix and describe Poisson mixes, upon which Loopix is based and introduce their properties. In Section 4, we present the analysis of Loopix's security properties and discuss the resistance against traffic analysis and active attacks. In Section 5, we discuss the implementation of Loopix and evaluate its performance. In Section 6, we survey related works and compare Loopix with recent designs of anonymity systems. In Section 7, we discuss remaining open problems and possible future work. Finally, we conclude in Section 8.

[2]Details about the threat model are in Section 2.3

2 Model and Goals

In this section, we first outline the design of Loopix. Then we discuss the security goals and types of adversaries that Loopix guarantees users' privacy against.

2.1 High-level overview

Loopix is a mix network [8] based architecture allowing *users*, distinguished as *senders* and *receivers*, to route messages anonymously to each other using an infrastructure of *mix* servers, acting as relays. These mix servers are arranged in a stratified topology [21] to ensure both horizontal scalability and a sparse topology that concentrates traffic on a few links [13]. In a stratified topology, mixes are arranged in a fixed number of layers. Each mix, at any given time, is assigned to one specific layer. Each mix in layer i is connected with every mix in layers $i-1$ and $i+1$. Each user is allowed to access the Loopix network through their association with a *provider*, a special type of mix server. Each provider has a long-term relationship with its users and may authenticate them, potentially bill them, or discontinue their access to the network. Each provider is connected to each mix in the first layer, in order to inject packets into the mix network, and also to every mix in the last layer, to receive egress packets. The provider not only serves as an access point, but also stores users' incoming messages. In contrast to previous anonymous messaging designs [46, 11], Loopix does not operate in deterministic rounds, but runs as a continuous system. This means that incoming messages can be retrieved at any time, hence users do not have to worry about lost messages when they are off-line. Additionally, Loopix uses the Poisson mixing technique that is based on the independent delaying of messages, which makes the timings of packets unlinkable. This approach does not require the synchronization of client-provider rounds and does not degrade the usability of the system for temporarily off-line clients. Moreover, Loopix introduces different types of cover traffic to foil de-anonymization attacks.

2.2 Threat Model

Loopix assumes sophisticated, strategic, and well-resourced adversaries concerned with linking users to their communications and/or their communication partner(s). As such, Loopix considers adversaries with three distinct *capabilities*, that are described next.

Firstly, a *global passive adversary* (GPA) is able to observe all network traffic between users and providers and between mix servers. This adversary is able to observe the entire network infrastructure, launch network attacks such as BGP re-routing [4], or conduct indirect observa-

	GPA	Corrupt mixes	Corrupt provider	Insider
Sender-Recipient Third-Party Unobservability	✓	✓	✓	✓
Sender online unobservability	✓	✓	✓	•
Sender anonymity	✓	✓	✓	✓
Receiver unobservability	✓	✓	✗	•
Receiver anonymity	✓	✓	✗	•

Table 1: The summary of security properties of the Loopix system in face of different threats. For the insider column we write • to denote that this concept doesn't apply to the respective notion.

tions such as load monitoring and off-path attacks [25]. Thus, the GPA is an abstraction that represents many different classes of adversaries able to observe some or all information between network nodes.

Secondly, the adversary has the ability to observe all of the internal state of some corrupted or malicious mix relays. The adversary may inject, drop, or delay messages. She also has access to, and leverages, all secrets of those compromised parties. Furthermore, such corrupted nodes may deviate from the protocol, or inject malformed messages. A variation of this ability is where the mix relay is also the provider node meaning that the adversary additionally knows the mapping between clients and their mailboxes. When we say that a provider node is *corrupt*, we restrict that node to being honest but curious. In Loopix, we assume that a fraction of mix/provider relays can be corrupted or are operated by the adversary.

Finally, the adversary has the ability to participate in the Loopix system as a compromised user, who may also deviate from the protocol. We assume that the adversary can control a limited number of such users—effectively excluding Sybil attacks [22] from the Loopix threat model—since we assume that *honest providers* are able to ensure that at least a large fraction of their users base are genuine users faithfully following all Loopix protocols. Thus, the fraction of users controlled by the adversary may be capped to a small known fraction of the user base. We further assume that the adversary is able to control a compromised user in a conversation with an honest user, and become a *conversation insider*.

An adversary is always assumed to have the GPA capability, but other additional capabilities depend on the adversary. We evaluate the security of Loopix in reference to these capabilities.

2.3 Security Goals

The Loopix system aims to provide the following security properties against both passive and active attacks—including end-to-end correlation and $(n-1)$ attacks. These properties are inspired by the formal definitions from AnoA [3]. All security notions assume a strong adversary with information on all users, with up to one bit of uncertainty. In the following we write $\{S \to R\}$ to denote a communication from the sender S to the receiver R, $\{S \to\}$ to denote that there is a communication from S to any receiver and $\{S \not\to\}$ to denote that there is no communication from S to any receiver (S may still send cover messages). Analogously, we write $\{\to R\}$ to denote that there is a communication from any sender to the receiver R and $\{\not\to R\}$ to denote that there is no communication from any sender to R (however, R may still receive cover messages).

Sender-Receiver Third-party Unlinkability. The senders and receivers should be unlinkable by any unauthorized party. Thus, we consider an adversary that wants to infer whether two users are communicating. We define *sender-receiver third party unlinkability* as the inability of the adversary to distinguish whether $\{S_1 \to R_1, S_2 \to R_2\}$ or $\{S_1 \to R_2, S_2 \to R_1\}$ for any online honest senders S_1, S_2 and honest receivers R_1, R_2 of the adversary's choice.

Loopix provides strong sender-receiver third-party unlinkability against the GPA even in collaboration with corrupt mix nodes. We refer to Section 4.1.3 for our analysis of the unlinkability provided by individual mix nodes, Section 4.3 for a quantitative analysis of the sender-receiver third-party unlinkability of Loopix against the GPA and honest-but-curious mix nodes, and Section 4.2 for our discussion on malicious mixes performing active attacks.

Sender online unobservability. Whether or not senders are communicating should be hidden from an unauthorized party. We define *sender online unobservability* as the inability of an adversary to decide whether a specific sender S is communicating with any receiver $\{S \to\}$ or not $\{S \not\to\}$, for any concurrently online honest sender S of the adversary's choice.

Loopix provides strong sender online unobservability against the GPA and even against a *corrupt provider*. We refer to Section 4.1.2 for our analysis of the latter.

Note, that sender online unobservability directly implies the notion of *sender anonymity* where the adversary tries to distinguish between two possible senders communicating with a target receiver. Formally, $\{S_1 \to R, S_2 \not\to\}$ or $\{S_1 \not\to, S_2 \to R\}$ for any concurrently online

honest senders S_1 and S_2 and any receiver of the adversary's choice. Loopix provides sender anonymity even in light of a conversation insider, i.e., against a corrupt receiver.

Receiver unobservability. Whether or not receivers are communicating should be hidden from an unauthorized party. We define *receiver unobservability* as the inability of an adversary to decide whether any sender is communicating with a specific receiver R $\{\rightarrow R\}$ or not $\{\not\rightarrow R\}$, for any online or offline honest receiver R of the adversary's choice.

Loopix provides strong receiver unobservability against the GPA, under the condition of an *honest provider*. We show in Section 4.1.2 how an honest provider assists the receiver in hiding received messages from third party observers.

Note, that receiver unobservability directly implies the notion of *receiver anonymity* where the adversary tries to distinguish between two possible receivers in communication with a target sender. Formally, $\{S \rightarrow R_1, \not\rightarrow R_2\}$ or $\{\not\rightarrow R_1, S \rightarrow R_2\}$ for any concurrently online honest sender S and any two honest receivers R_1, R_2 of the adversary's choice. [3]

Non-Goals. Loopix provides anonymous unreliable datagram transmission and facilities replying to sent messages (through add-ons). This choice allows for flexible traffic management, cover traffic, and traffic shaping. On the downside, higher-level applications using Loopix need to take care of reliable end-to-end transmission and session management. We leave the detailed study of those mechanisms as future work.

The provider-based architecture supported by Loopix aims to enable managed access to the network, anonymous blacklisting to combat abuse [27], and payments for differential access to the network [2]. However, we do not discuss these aspects of Loopix in this work, and concentrate instead on the core anonymity features and security properties described above.

3 The Loopix Architecture

In this section we describe the Loopix system in detail—Figure 1 provides an overview. We also introduce the notation used further in the paper, summarized in Table 2.

3.1 System Setup

The Loopix system consists of a set of mix nodes, N, and providers, P. We consider a population of U users communicating through Loopix, each of which can act as *sender* and *receiver*, denoted by indices S_i, R_i, where $i \in \{1, \ldots, U\}$ respectively. Each entity of the Loopix infrastructure has its unique public-private key pair (sk, pk). In order for a *sender* S_i, with a key pair (sk_{S_i}, pk_{S_i}), to send a message to a *receiver* R_j, with a key pair (sk_{R_j}, pk_{R_j}), the sender needs to know the receiver's Loopix *network location*, i.e., the IP address of the user's provider and an identifier of the user, as well as the public encryption key pk_{R_j}. Since it is out of scope for this work, we will assume this information can be made available through a privacy-friendly lookup or introduction system for initiating secure connections [32].

Symbol	Description
N	Mix nodes
P	Providers
λ_L	Loop traffic rate (user)
λ_D	Drop cover traffic rate (user)
λ_P	Payload traffic rate (user)
l	Path length (user)
μ	The mean delay at mix M_i
λ_M	Loop traffic rate (mix)

Table 2: Summary of notation

3.2 Format, Paths and Cover Traffic

Message packet format. All messages in Loopix are *end-to-end encrypted* and encapsulated into packets to be processed by the mix network. We use the Sphinx packet design [16], to ensure that intermediate mixes learn no additional information beyond some routing information. All messages are padded to the same length, which hides the path length and the relay position and guarantees unlinkability at each hop of the messages' journey over the network. The Sphinx packet format allows for detection of tagging attacks and replay attacks.

Each message wrapped into the Sphinx packet consists of a concatenation of two separate parts: a header, carrying the layered encryption of meta-data for each hop, and the encrypted payload, which allows for confidential message exchange. The header provides each mix server on the path with confidential meta-data, which is necessary to verify packet integrity and correctly process the packet. The structure of the header consists of (I) a single element of a cyclic group that is re-randomized at each hop, (II) an onion-encrypted vector, with each layer containing the routing information for one hop, and (III) the message authentication code $\mathsf{MAC_i}$, which allows header integrity checking. The payload is encrypted using the LIONESS cipher [1], which guarantees that in case the adversary modifies the payload in transit, any informa-

[3] If the receiver's provider is honest, Loopix provides a form of receiver anonymity even in light of a conversation insider: a corrupt sender that only knows the pseudonym of a receiver cannot learn which honest client of a provider is behind the pseudonym.

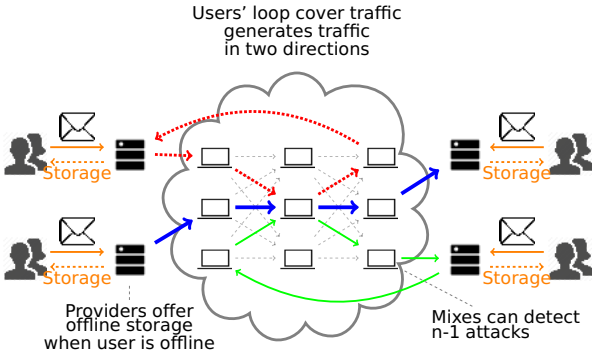

Figure 1: The Loopix Architecture. Clients pass the messages to the providers, which are responsible for injecting traffic into the network. The received messages are stored in individual inboxes and retrieved by clients when they are online.

tion contained in it becomes irrecoverable. Thanks to the message authentication code in the header and the LIONESS encryption the Sphinx packet format thus allows for detection of tagging attacks.

Sphinx packet generation: The sender, given the public keys of the recipient and the nodes in the path, computes the sequence of shared secrets and blinded group elements. Next, the sender encrypts with the derived secret keys the vector of routing information and corresponding message authentication codes. The sender concatenates the computed header and onion-encrypted payload encapsulating confidential message to send to the recipient.

Sphinx packet processing: Each node after receiving the packet proceeds as follows. First, it computes a shared key using the group element included in the packet header and its private key. Next, using the computed shared key, the node validates the integrity of the packet by computing the hash of the encrypted routing information vector and comparing it with the received MAC. If the MAC is correct, the node, using the obtained key, strips off a single layer of encryption from the routing information and payload. The decryption operation returns the routing commands and a new packet, which should be forwarded to the next hop.

We extend the Sphinx packet format to carry additional routing commands in the header to each intermediate relay, including a delay and additional flags.

Path selection. As opposed to circuit-based onion routing, in Loopix the communication path for every single message is chosen independently, even between the same pair of users.

Messages are routed through l layers of mix nodes, assembled in a stratified topology [13, 21]. Each mix node is connected only with all the mix nodes from adjacent layers. This ensures that few links are used, and those few links are well covered in traffic; stratified topologies mix well in few layers [21]. Providers act as the first and last layer of mix servers.

Preparing message for sending. To send a message, the sender generates a random path, as described above. For each hop in the path the sender samples a delay from an exponential distribution with parameter μ, and includes it in the vector of routing commends, together with any other auxiliary information, to the corresponding relay. Given the message, recipient, path and routing commends the client encapsulates them into a Sphinx packet format.

Sending messages and cover traffic. Users and mix servers continuously generate a bed of *real* and *cover traffic* that is injected into the network. Our design guarantees that all outgoing traffic sent by users can by modeled by a Poisson process.

To send a message, a user packages their message into a mix packet and places it into their *buffer*—a first-in-first-out (FIFO) queue that stores all the messages scheduled to be sent.

Each sender periodically checks, following the exponential distribution with parameter $\frac{1}{\lambda_P}$, whether there is any scheduled message to be sent in their buffer. If there is a scheduled message, the sender pops this message from the buffer queue and sends it, otherwise a *drop* cover message is generated (in the same manner as a regular message) and sent (depicted as the four middle blue, solid arrows in Figure 1). Cover messages are routed through the sender's provider and a chain of mix nodes to a random destination provider. The destination provider detects the message is cover based on the special drop flag encapsulated into the packet header, and drops it. Thus, regardless of whether a user actually wants to send a message or not, there is always a stream of messages being sent according to a Poisson process $Pois(\lambda_P)$.

Moreover, independently from the above, all users emit separate streams of special indistinguishable types of *cover messages*, which also follow a Poisson process. The first type of cover messages are Poisson distributed *loops* emitted at rate λ_L. These are routed through the network and *looped back* to the senders (the upper four red arrows in Figure 1), by specifying the sending user as the recipient. These *"loops"* inspire the system's name. Users also inject a separate stream of *drop* cover messages, defined before, following the Poisson distribution $Pois(\lambda_D)$. Additionally, each user sends a stream of *pull* requests at a fixed frequency to its *provider* in order to retrieve received messages, described in Section 3.2.

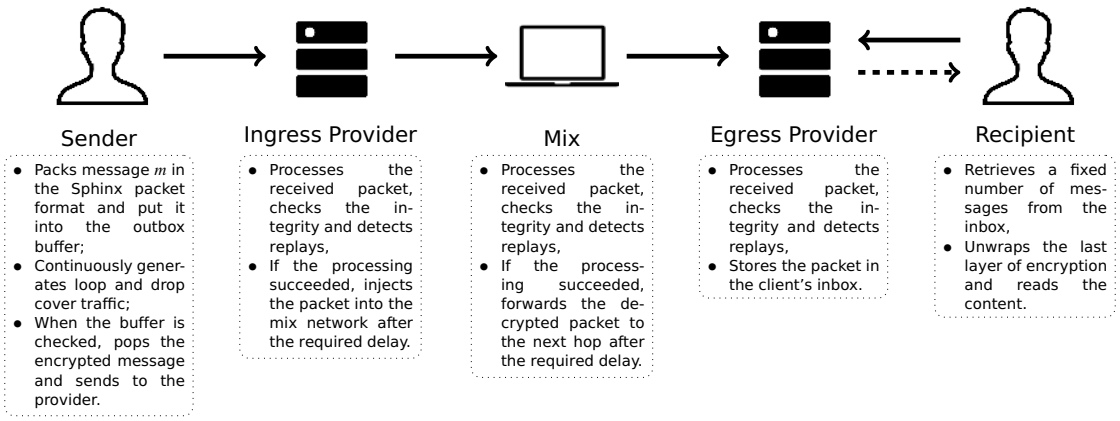

Figure 2: Sending a single message between two users using the Loopix system. For simplicity, we present the mix network as a single mix; however, all mixes in the network perform the same operations. The mail client, besides sending the messages, generates constant streams of loop and drop cover traffic, independently of the user activity. The dotted line depicts retrieving of messages.

Each mix also injects its own *loop* cover traffic, drawn from a Poisson process with rate $Pois(\lambda_M)$, into the network. Mix servers inject mix packets that are looped through a path, made up of a subset of other mix servers and one randomly selected *provider*, back to the sending mix server, creating a second type of *"loop"*. This loop originates and ends in a mix server (shown as the lower four green arrows in Figure 1). In Section 4 we examine how these *loops* and the *drop* cover messages help protect against passive and active attacks.

Processing messages. Upon receiving a packet, each node, i.e., each mix and provider, performs the operation of processing the Sphinx packet. While processing the packet, the server recomputes the shared secret and checks the MAC's correctness. If this integrity test fails, the packet is dropped. Otherwise, the unwrapping function returns the *replay detection tag* and the vector of *routing commands*, as well the *new packet*. The vector of *routing commands* includes, among others, the *routing flag*, the *address* of the next hop and the *delay*. After unwrapping the packet, the node checks whether the returned *replay detection tag* has been already seen and if so, drops the packet. This allows for detection and protection against *replay attacks*. Otherwise, the node saves the tag in a data structure that stores previously observed tags. Next, it checks whether the *routing flag* is set to Relay or Dest. The Dest flag means that the received message is a loop message transferred back to the node. In the case of the Relay flag, we consider two scenarios depending on whether the processing node is a mix or a provider. In the case of a mix, the decrypted *new packet* is send to the next hop, specified by *address*, after the *delay* has elapsed. In the case of a provider, the new packet is either forwarded as before or saved in the inbox of one of the provider's clients specified by the *address*.

Message storing and retrieving. Providers do not forward the incoming mix packets to users but instead buffer them in clients' inboxes. Users, when online, *poll* providers or register their online status to download a fixed subset of stored messages, allowing for the reception of the off-line messages. Recall that cover loops are generated by users and traverse through the network and come back to the sender. Cover loops serve as a cover set of *outgoing* and *incoming* real messages. Whenever a user requests messages, their provider responds with a constant number of messages, which includes their cover loop messages and real messages. If the inbox of a particular user contains fewer messages than this constant number, the provider generates and sends dummy messages to the sender up to that number.

3.3 The Poisson Mix Strategy

Loopix leverages cover traffic to resist traffic analysis while still achieving low- to mid-latency. To this end Loopix employs a mixing strategy that we call a *Poisson Mix*, to foil observers from learning about the correspondences between input and output messages. The Poisson Mix is a simplification of the Stop-and-go mix strategy [29]. A similar strategy has been used to model traffic in onion routing servers [12]. In contrast, recall that in Loopix each message is source routed through an independent route in the network.

The Poisson Mix functions as follows: mix servers listen for the incoming mix packets and received messages are checked for duplication and decoded using the mix

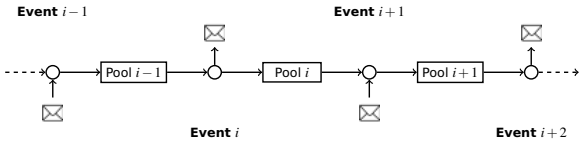

Figure 3: The Poisson Mix strategy mapped to a Pool mix strategy. Each single message sending or receiving event leads to a new pool of messages that are exchangeable and indistinguishable with respect to their departure times.

node's private keys. The detected duplicates are dropped. Next, the mix node extracts a subsequent mix packet. Decoded mix packets are not forwarded immediately, but each of them is delayed according to a source predetermined delay d_i. Honest clients chose these delays, independently for each hop, from an exponential distribution with a parameter μ that is assumed to be public and the same for all mix nodes. This parameter determines how long the message is queued in the mix. Thus, the end-to-end latency of the messages depends on the selected parameter μ.

Mathematical model of a Poisson Mix. Honest clients and mixes generate drop cover traffic, loop traffic, and messaging traffic following a Poisson process. Aggregating Poisson processes results in a Poisson process with the sum of their rates, therefore we may model the streams of traffic received by a Poisson mix as a Poisson process. It is the superposition of traffic streams from multiple clients. It has a rate λ_n depending on the number of clients and the number of mix nodes.

Since this input process is a Poisson process and each message is independently delayed using an exponential distribution with parameter μ, the Poisson Mix may be modeled as an $M/M/\infty$ queuing system – for which we have a number of well known theorems [5]. We know that output stream of messages is also a Poisson process with the parameter λ_n as the the input process. We can also derive the distribution of the number of messages within a Poisson Mix in a *steady state* [34]. By the *steady state* we mean the state of the system in which all entities have already generated and processed messages for some reasonable period of time. By the convergence of the system to the equilibrium, this guarantees that the observed traffic closely follows the assumed distribution.

Lemma 1. *The mean number of messages in the Poisson Mix with input Poisson process Pois*(λ) *and exponential delay parameter* μ *at a steady state follows the Poisson distribution Pois*(λ/μ).

These characteristics, which give the Poisson Mix its name, allow us to calculate the mean number of messages *perfectly* mixed together at any time, as well as the probability that the number of messages falls below or above certain thresholds.

The Poisson Mix, under the assumption that it approximates an $M/M/\infty$ queue is a stochastic variant of a pool mixing strategy [42]. Conceptually, every message sent or received leads to a pool within which messages are indistinguishable due to the memoryless property of the exponential delay distribution.

Lemma 2 (Memoryless property [34])**.** *For an exponential random variable X with parameter* μ *holds* $\Pr[X > s+t|X > t] = \Pr[X > s]$.

Intuitively, any two messages in the same pool are emitted next with equal probability – no matter how long they have been waiting. As illustrated in Figure 3, the receiving event $i-1$ leads to a pool of messages $i-1$, until the sending event i. From the perspective of the adversary observing all inputs and outputs, all messages in the pool $i-1$ are indistinguishable from each other. Only the presence of those messages in the pool is necessary to characterize the hidden state of the mix (not their delay so far). Relating the Poisson mix to a pool mix allows us to compute easily and exactly both the entropy metric for the anonymity it provides [40] within a trace (used in Section 4.1.3). It also allows us to compute the likelihood that an emitted message was any specific input message used in our security evaluation.

Synchronous variant of Loopix. While Loopix operates asynchronously by design, we now consider a synchronous Loopix variant that operates in discrete rounds and thus cannot use the exponential mixing strategy, where delays attached to the packets are drawn from a continuous distribution. However, note that in a single round of the synchronous system the mixes gather packets - thus creating pools of packets - which are then flushed following the mixing strategy. All the messages gathered in the pool during a single round are indistinguishable from each other. Hence, since we have shown earlier that the Poisson mix can be modeled as a pool mix, the security analysis of mixing we present next can be applied both in the asynchronous and synchronous design.

4 Analysis of Loopix security properties

In this section we present the analytical and experimental evaluation of the security of Loopix and argue its resistance to traffic analysis and active attacks.

4.1 Passive attack resistance

4.1.1 Message Indistinguishability

Loopix relies on the Sphinx packet format [16] to provide bitwise unlinkability of incoming and outgoing messages from a mix server; it does not leak information about the number of hops a single message has traversed or the total path length; and it is resistant to tagging attacks.

For Loopix, we make minor modifications to Sphinx to allow auxiliary meta-information to be passed to different mix servers. Since all the auxiliary information is encapsulated into the header of the packet in the same manner as any meta-information was encapsulated in the Sphinx design, the security properties are unchanged. An external adversary and a corrupt intermediate mix node or a corrupt provider will not be able to distinguish *real* messages from *cover* messages of any type. Thus, the GPA observing the network cannot infer any information about the type of the transmitted messages, and intermediate nodes cannot distinguish real messages, drop cover messages or loops of clients and other nodes from each other. Providers are able to distinguish *drop* cover message destined for them from other messages, since they learn the *drop flag* attached in the header of the packet. Each mix node learns the delay chosen by clients for this particular mix node, but all delays are chosen independently from each other.

4.1.2 Client-Provider unobservability

In this section, we argue the *sender and receiver unobservability* against different adversaries in our threat model. Users emit payload messages following a Poisson distribution with parameter λ_P. All messages scheduled for sending by the user are placed within a first-in-first-out buffer. According to a Poisson process, a single message is popped out of the buffer and sent, or a drop cover message is sent in case the buffer is empty. Thus, from an adversarial perspective, there is always traffic emitted modeled by $Pois(\lambda_P)$. Since clients send also streams of cover traffic messages with rates λ_L for loops and λ_D for drop cover messages, the traffic sent by the client follows $Pois(\lambda_P + \lambda_L + \lambda_D)$. Thus, we achieve perfect *sender unobservability*, since the adversary cannot tell whether a genuine message or a drop cover message is sent.

When clients query providers for received messages, the providers always send a constant number of messages to the client. If the number of messages in client's inbox is smaller than a constant threshold, the provider generates additional dummy messages. Thus, the adversary observing the client-provider connection, as presented on Figure 4, cannot learn how many messages were in the user's inbox. Note that, as long as the providers are honest, the protection and *receiver unobservability* is perfect and the adversary cannot learn any information about the inbox and outbox of any client.

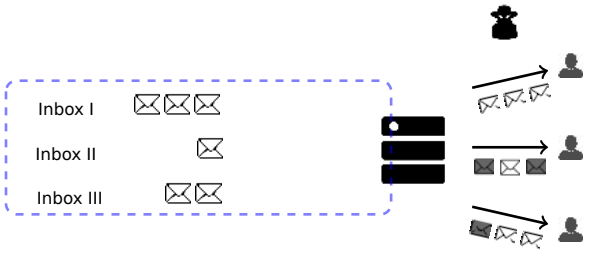

Figure 4: Provider stores messages destined for assigned clients in a particular inbox. When users pull messages from the mix node, the provider generates cover messages to guarantee that the adversary cannot learn how many messages are in the users inbox. The messages from the inbox and dummies are indistinguishable.

Corrupt providers: We distinguish the sender's and recipient's providers by calling them the ingress and egress providers respectively. If the *ingress provider* is compromised, all security properties of the Loopix system are still preserved, since the *ingress provider* observes a rate of traffic shaped by the Poisson distribution coming from the client and cannot distinguish whether the received packets carry real, loop or drop messages.

If the *egress provider* is malicious it can reveal to the adversary whether a particular client is receiving messages or not since the provider is responsible for managing the clients' inboxes. However, even an egress provider is still uncertain whether a received message is genuine or the result of a client loop – this cannot be determined from their bit pattern alone. Further statistical attacks may be possible, and we leave quantifying the exact information leakage against this threat model as future work. Thus, Loopix does not guarantee perfect *receiver unobservability* in the presence of a corrupted egress provider.

4.1.3 Poisson mix security

We first show that a single honest Poisson mix provides a measure of *sender-receiver unlinkability*. From the properties of Poisson mix, we know that the number of messages in the mix server at a steady state depends on the ratio of the incoming traffic (λ) and the delay parameter (μ) (from Section 3.3). The number of messages in each mix node at any time will on average be $\frac{\lambda}{\mu}$. However, an adversary observing the messages flowing into and out of a single mix node could estimate the exact number of

messages within a mix with better accuracy – hindered only by the mix loop cover traffic.

We first consider, conservatively, the case where a mix node is not generating any loops and the adversary can count the exact number of messages in the mix. Let us define $o_{n,k,l}$ as an adversary A observing a mix in which n messages arrive and are mixed together. The adversary then observes an outgoing set of $n-k$ messages and can infer that there are now $k<n$ messages in the mix. Next, l additional messages arrive at the mix before any message leaves, and the pool now mixes $k+l$ messages. The adversary then observes exactly one outgoing message m and tries to correlate it with any of the $n+l$ messages which she has observed arriving at the mix node.

The following lemma is based on the memoryless property of the Poisson mix. It provides an upper bound on the probability that the adversary A correctly links the outgoing message m with one of the previously observed arrivals in observation $o_{n,k,l}$.

Theorem 1. *Let m_1 be any of the initial n messages in the mix node in scenario $o_{n,k,l}$, and let m_2 be any of the l messages that arrive later. Then*

$$Pr(m=m_1)=\frac{k}{n(l+k)}, \tag{1}$$

$$Pr(m=m_2)=\frac{1}{l+k}. \tag{2}$$

Note that the last l messages that arrived at the mix node have equal probabilities of being the outgoing message m, independently of their arrival times. Thus, the arrival and departure times of the messages cannot be correlated, and the adversary learns no additional information by observing the timings. Note that $\frac{1}{l+k}$ is an upper bound on the probability that the adversary A correctly links the outgoing message to an incoming message. Thus, continuous observation of a Poisson mix leaks no additional information other than the number of messages present in the mix. We leverage those results for a single Poisson Mix to simulate the information propagated withing a the whole network observed by the adversary (c.f. Section 4.3).

We quantify the anonymity of messages in the mix node empirically, using an information theory based metric introduced in [40, 18]. We record the traffic flow for a single mix node and compute the distribution of probabilities that the outgoing message is the adversary's target message. Given this distribution we compute the value of Shannon entropy (see Appendix A), a measure of unlinkability of incoming to outgoing messages. We compute this using the simpy package in Python. All data points are averaged over 50 simulations.

Figure 5 depicts the change of entropy against an increasing rate of incoming mix traffic λ. We simulate the

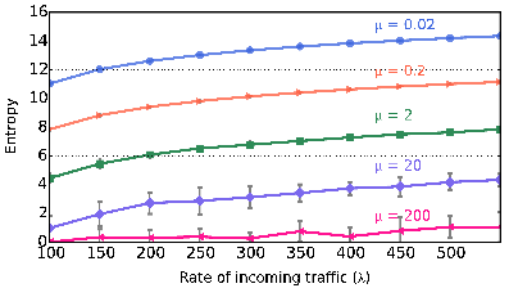

Figure 5: Entropy versus the changing rate of the incoming traffic for different delays with mean $\frac{1}{\mu}$. In order to measure the entropy we run a simulation of traffic arriving at a single Loopix mix node.

dependency between entropy and traffic rate for different mix delay parameter μ by recording the traffic flow and changing state of the mix node's pool. As expected, we observe that for a fixed delay, the entropy increases when the rate of traffic increases. Higher delay also results in an increase in entropy, denoting a larger potential anonymity set, since more messages are mixed together.

In case the mix node emits loop cover traffic, the adversary with observation $o_{n,k,l}$, tries to estimate the probability that the observed outgoing message is a particular *target* message she observed coming into the mix node. An outgoing message can be either input message or a loop message generated by the mix node – resulting in additional uncertainty for the adversary.

Theorem 2. *Let m_1 be any of the initial n messages in the mix node in scenario $o_{n,k,l}$, and let m_2 be any of the l messages that arrive later. Let λ_M denote the rate at which mix node generates loop cover traffic. Then,*

$$\Pr(m=m_1)=\frac{k}{n}\cdot\frac{\mu}{(l+k)\mu+\lambda_M},$$

$$\Pr(m=m_2)=\frac{\mu}{(l+k)\mu+\lambda_M}.$$

We refer to Appendix A for the proof. We conclude that the loops generated by the mix node obfuscate the adversary's view and decrease the probability of successfully linking input and output of the mix node. In Section 4.2 we show that those types of loops also protect against active attacks.

4.2 Active-attack Resistance

Lemma 1 gives the direct relationship between the expected number of messages in a mix node, the rate of incoming traffic, and the delay induced on a message while transiting through a mix. By increasing the rate of cover traffic, λ_D and λ_L, users can collectively maintain strong

anonymity with low message delay. However, once the volume of real communication traffic λ_P increases, users can tune down the rate of cover traffic in comparison to the real traffic, while maintaining a small delay and be confident their messages are mixed with a sufficient number of messages.

In the previous section, we analyze the security properties of Loopix when the adversary observes the state of a single mix node and the traffic flowing through it. We show, that the adversary's advantage is bounded due to the indistinguishability of messages and the memoryless property of the Poisson mixing strategy. We now investigate how Loopix can protect users' communications against active adversaries conducting the $(n-1)$ attack.

4.2.1 Active attacks

We consider an attack at a mix node where an adversary blocks all but a target message from entering in order to follow the target message when it exits the mix node. This is referred to as an *(n-1) attack* [41].

A mix node needs to distinguish between an active attack and loop messages dropped due to congestion. We assume that each mix node chooses some public parameter r, which is a fraction of the number of loops that are expected to return. If the mix node does not see this fraction of loops returning they alter their behavior. In extremis such a mix could refuse to emit any messages – but this would escalate this attack to full denial-of-service. A gentler approach involves generating more cover traffic on outgoing links [17].

To attempt an *(n-1) attack*, the adversary could simply block all incoming messages to the mix node except for a target message. The Loopix mix node can notice that the self-loops are not returning and deduce it is under attack. Therefore, an adversary that wants to perform a stealthy attack has to be judicious when blocking messages, to ensure that a fraction r of loops return to the mix node, i.e. the adversary must distinguish loop cover traffic from other types of traffic. However, traffic generated by mix loops is indistinguishable from other network traffic and they cannot do this better than by chance. Therefore given a threshold $r = \frac{\lambda_M}{s}, s \in \mathbb{R}_{>1}$ of expected returning loops when a mix observes fewer returning it deploys appropriate countermeasures.

We analyze this strategy: since the adversary cannot distinguish loops from other traffic the adversary can do no better than block traffic uniformly such that a fraction $R = \frac{\lambda}{s} = \frac{\lambda_R + \lambda_M}{s}$ enter the mix, where λ_R is the rate of incoming traffic that is not the mix node's loops. If we assume a steady state, the target message can expect to be mixed with $\frac{\lambda_R}{s \cdot \mu}$ messages that entered this mix, and $\frac{\lambda_M}{\mu}$ loop messages generated at the mix node. Thus, the probability of correctly blocking a sufficient number of messages entering the mix node so as not to alter the behavior of the mix is:

$$\Pr(x = \text{target}) = \frac{1}{\lambda_R/s \cdot \mu + \lambda_M/\mu} = \frac{s\mu}{s\lambda_M + \lambda_R}$$

Due to the stratified topology, providers are able to distinguish mix loop messages sent from other traffic, since they are unique in not being routed to or from a client. This is not a substantial attack vector since mix loop messages are evenly distributed among all providers, of which a small fraction are corrupt and providers do not learn which mix node sent the loop to target it.

4.3 End-to-End Anonymity Evaluation

We evaluate the *sender-receiver third-party unlinkability* of the full Loopix system through an empirical analysis of the propagation of messages in the network. Our key metric is the *expected difference in likelihood* that a message leaving the last mix node is sent from one sender in comparison to another sender. Given two probabilities $p_0 = \Pr[S_0]$ and $p_1 = \Pr[S_1]$ that the message was sent by senders S_0 and S_1, respectively, we calculate

$$\varepsilon = |\log(p_0/p_1)|. \tag{3}$$

To approximate the probabilities p_0 and p_1, we proceed as follows. We simulate $U = 100$ senders that generate and send messages (both payload and cover messages) with a rate $\lambda = 2$. Among them are two challenge senders S_0 and S_1 that send payload messages at a constant rate, i.e, they add one messages to their sending buffer every time unit.

Whenever a challenge sender S_0 or S_1 sends a payload message from its buffer, we tag the message with a label S_0 or S_1, respectively. All other messages, including messages from the remaining 98 clients and the cover messages of S_0 and S_1 are unlabeled. At every mix we track the probability that an outgoing message is labeled S_0 or S_1, depending on the messages that entered the mix node and the number of messages that already left the mix node, as in Theorem 1. Thus, messages leaving a mix node carry a probability distribution over labels S_0, S_1, or 'unlabeled'. Corrupt mix nodes, assign to outgoing messages their input distributions. The probabilities naturally add up to 1. For example, a message leaving a mix can be labeled as $\{S_0 : 12\%, S_1 : 15\%, \text{unlabeled} : 73\%\}$.

In a burn-in phase of 2500 time units, the 98 senders without S_0 or S_1 communicate. Then we start the two challenge senders and then simulate the network for another 100 time units, before we compute the *expected difference in likelihood* metric. We pick a final mix node and using probabilities of labels S_0 and S_1 for any message in the pool we calculate ε as in Equation (3).

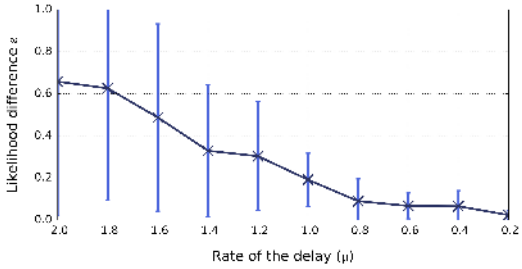

Figure 6: Likelihood difference ε depending on the delay parameter μ of mix nodes. We use $\lambda = 2$, a topology of 3 layers with 3 nodes per layer and no corruption.

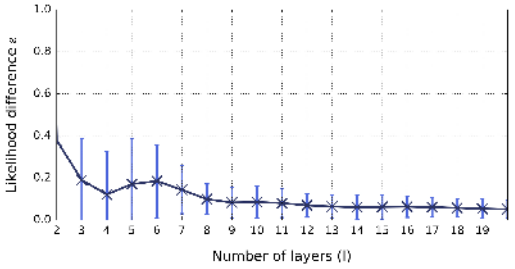

Figure 7: Likelihood difference ε depending on the number of layers of mix nodes with 3 mix nodes per layer. We use $\lambda = 2$, $\mu = 1$, and no corruption.

This is a conservative approximation: we tell the adversary which of the messages leaving senders S_0 and S_1 are payload messages; and we do not consider mix or client loop messages confusing them. [4] However, when we calculate our anonymity metric at a mix node we assume this mix node to be honest.

4.3.1 Results

We compare our metric for different parameters: depending on the delay parameter μ, the number of layers in our topology l and the percentage of corrupt mix nodes in the network. All simulations are averaged over 100 repetitions and the error bars are the standard deviation.

Delay. Increasing the average delay (by decreasing parameter μ) with respect to the rate of message sending λ immediately increases anonymity (decreases ε) (Figure 6). For $\mu = 2.0$ and $\lambda/\mu = 1$, Loopix still provides a weak form of anonymity. As this fraction increases, the log likelihood ratio grow closer and closer to zero. We consider values $\lambda/\mu \geq 2$ to be a good choice in terms of anonymity.

Number of layers. By increasing the number of layers of mix nodes, we can further strengthen the anonymity of Loopix users. As expected, using only one or two layers of mix nodes leads to high values of adversary advantage ε. For a increasing number of layers, ε approaches zero (Figure 7). We consider a number of 3 or more layers to be a good choice. We believe the bump between 5–8 layers is due to messages not reaching latter layers within 100 time units. Results from experiments with increased duration do not display such a bump.

[4] The soundness of our simplification can be seen by the fact that we could tell the adversary which messages are loops and the adversary could thus ignore them. This is equivalent to removing them, as an adversary could also simulate loop messages.

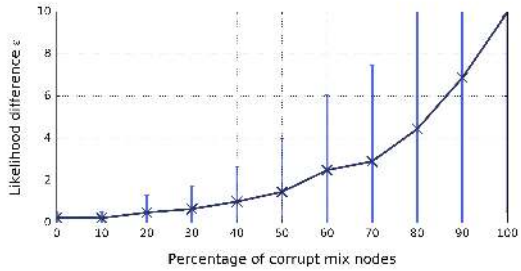

Figure 8: Likelihood difference ε depending on the percentage of (passively) corrupted mix nodes. We use $\lambda = 2$, $\mu = 1$ and a topology of 3 layers with 3 nodes per layer.

Corruption. Finally, we analyze the impact that corrupt mix nodes have on the adversary advantage ε (Figure 8). We assume that the adversary randomly corrupts mix nodes. Naturally, the advantage ε increases with the percentage of corrupt mix nodes in the network. In a real-world deployment we do not expect a large fraction of mix nodes to be corrupt. While the adversary may be able to observe the entire network, to control a large number of nodes would be more costly.

5 Performance Evaluation

Implementation. We implement the Loopix system prototype in 4000 lines of `Python 2.7` code for *mix nodes*, *providers* and *clients*, including unit-tests, deployment, and orchestration code. Loopix source code is available under an open-source license[5]. We use the `Twisted 15.5.0` network library for networking; as well as the `Sphinx` mix packet format[6] and the cryptographic tools from the *petlib*[7] library. We modify `Sphinx` to use `NIST/SEGS-p224` curves and to accommodate additional information inside the packet, including the delay

[5] `https://github.com/UCL-InfoSec/loopix`
[6] http://sphinxmix.readthedocs.io/en/latest/
[7] http://petlib.readthedocs.org

for each hop and auxiliary flags. We also optimize the Sphinx implementation leading to processing times per packet of less than $1ms$.

The most computationally expensive part of Loopix is messages processing and packaging, which involves cryptographic operations. Thus, we implement Loopix as a multi-thread system, with cryptographic processing happening in a thread pool separated from the rest of the operations in the main thread loop. To recover from congestion we implement active queue management based on a PID controller and we drop messages when the size of the queue reaches a (high) threshold.

Experimental Setup. We present an experimental performance evaluation of the Loopix system running on the `AWS EC2` platform. All mix nodes and providers run as separate instances. Mix nodes are deployed on `m4.4xlarge` instances running `EC2 Linux` on $2.3\,GHz$ machines with $64\,GB$ RAM memory. Providers, since they handle more traffic, storage and operations, are deployed on `m4.16xlarge` instances with $256\,GB$ RAM. We select large instances to ensure that the providers are not the bottleneck of the bandwidth transfer, even when users send messages at a high rate. This reflects real-world deployments where providers are expected to be well-resourced. We also run one `m4.16xlarge` instance supporting 500 clients. We only show results for 500 clients, due to limitations of our experimental hardware setup such as ports and memory. A real world deployment of Loopix would scale to a larger client base. We believe that our empirical analysis is a more accurate assessment of real-world performance than those reported by other works, e.g. [45, 46], which depend on simplish extrapolation. In order to measure the system performance, we run six mix nodes, arranged in a stratified topology with three layers, each layer composed of two mix nodes. Additionally, we run four providers, each serving approximately 125 clients. The delays of all the messages are drawn from an exponential distribution with parameter μ, which is the same for all mix servers in the network. All measurements are taken from network traffic dumps using `tcpdump`.

Bandwidth. First, we evaluate the increase of bandwidth of mix nodes by measuring the rate at which a single mix node processes messages, for an increasing overall rate at which users send messages.

We set up the fixed delay parameter $\mu = 1000$ (s.t. the average delay is $1\,ms$). We have 500 clients actively sending messages at rate λ each, which is the sum of payload, loop and drop rates, i.e., $Pois(\lambda) = Pois(\lambda_L + \lambda_D + \lambda_P)$. We start our simulation with parameters $\lambda_L = \lambda_D = 1$ and $\lambda_P = 3$ messages per minute for a single client. Mix nodes send loop cover traffic at rate starting from $\lambda_M = 1$. Next, we periodically increase each Poisson rate by another 2 messages per minute. Each packet sent through the network has a size of a few kilobytes only, but this size is a parameter that can, of course, be increased to fit the needs of a particular application.

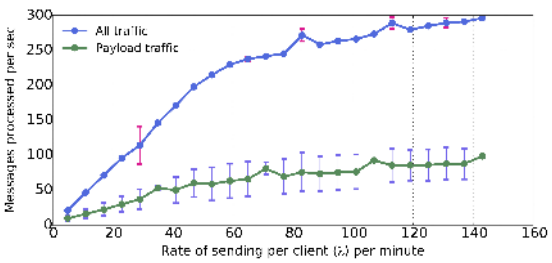

Figure 9: Overall bandwidth and good throughput per second for a single mix node.

In order to measure the overall bandwidth, i.e. the number of all messages processed by a single mix node, we use the network packet analyzer `tcpdump`. Since real and cover message packets are indistinguishable, we measure the good throughput by encapsulating an additional, temporary, *typeFlag* in the packet header for this evaluation, which leaks to the mix the message type—real or cover—and is recorded. Knowing the parameters λ_P, λ_L, and λ_D the adversary can try to estimate how many messages on average in the outgoing stream are real, loop or drop messages. However, the average estimation does not give the adversary any significant information, since the outgoing traffic may contain various numbers of each type of message which an adversary is not able to distinguish between.

Figure 9 illustrates the number of total messages and the number of payload messages that are processed by a single mix node versus the overall sending rate λ of a single user. We observe that the bandwidth of the mix node increases linearly until it reaches around 225 messages per second. After that point the performance of the mix node stabilizes and we observe a much smaller growth. We highlight that the amount of *real* traffic in the network depends on the parameter λ_P within λ. A client may chose to tune up the rate of real messages sent, by tuning down the rate of loops and drop messages – at the potential loss of security in case less cover traffic is present in the system overall. Thus, depending on the size of the honest user population in Loopix, we can increase the rate of goodput.

Latency Overhead & Scalability. End-to-end latency overhead is the cost of routing and decoding relayed messages, without any additional artificial delays. We run

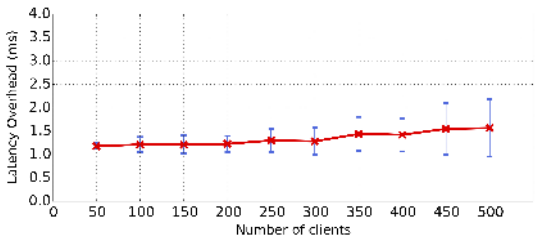

Figure 10: Latency overhead of the system where 50 to 500 users simultaneously send traffic at rates $\lambda_P = \lambda_L = \lambda_D = 10$ per minute and mix nodes generate loop cover traffic at rate $\lambda_M = 10$ per minute. We assume that there is not additional delay added to the messages by the senders.

simulations to measure its sensitivity to the number of users participating in the system.

We measure the time needed to process a single packet by a mix node, which is approximately $0.6\,ms$. This cost is dominated by the scalar multiplication of an elliptic curve point and symmetric cryptographic operations. For the end-to-end measurement, we run Loopix with a setup where all users have the same rates of sending real and cover messages, such that $\lambda_P = \lambda_D = \lambda_L = 10$ messages per minute and mix servers generate loops at rate $\lambda_M = 10$ messages per minute. All clients set a delay of 0.0 seconds for all the hops of their messages – to ensure we only measure the system overhead, not the artificial mixing delay.

Figure 10 shows that increasing the number of online clients, from 50 to 500, raises the latency overhead by only $0.37\,ms$. The variance of the processing delay increases with the amount of traffic in the network, but more clients do not significantly influence the average latency overhead. Neither the computational power of clients nor mix servers nor the communication between them seem to become bottlenecks for these rates. Those results show that the increasing number of users in the network does not lead to any bottleneck for our parameters. The measurements presented here are for a network of 6 mix nodes, however we can increase the system capacity by adding more servers. Thus, Loopix scales well for an increasing number of users.

We also investigate how increasing the delays through Poisson Mixing with $\mu = 2$ affects the end-to-end latency of messages. We measure this latency through timing mix heartbeat messages traversing the system. Figure 11 illustrates that when the mean delay $1/\mu$ sec. is higher than the processing time ($\sim 1\,ms - 2\,ms$), the end-to-end latency is determined by this delay, and follows the Gamma distribution with parameter being the sum of the exponential distribution parameter over the number of servers on the path. The good fit to a gamma distribution provides evidence that the implementation of Loopix is faithful to the queuing theory models our analysis assumes.

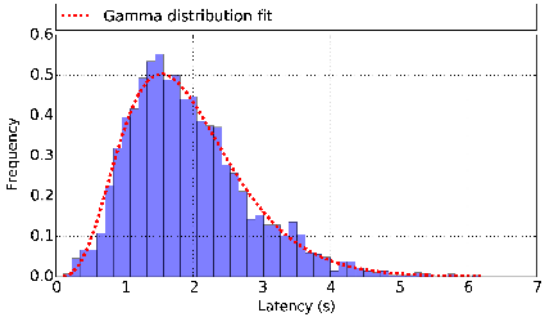

Figure 11: End-to-end latency histogram measured through timing mix node loops. We run 500 users actively communicating via Loopix at rates $\lambda_P = \lambda_L = \lambda_D = 60$ per minute and $\lambda_M = 60$ per minute. The delay for each hop is drawn from $Exp(2)$. The latency of the message is determined by the assigned delay and fits the Gamma distribution with mean 1.93 and standard deviation 0.87.

6 Related Work

All anonymous communication designs share the common goal of hiding users' communication patterns from adversaries. Simultaneously minimizing latency and communication overhead while still providing high anonymity is challenging. We survey other anonymous systems and compare them with Loopix (a summary is provided in Table 3).

Early designs. Designs based on Chaum's mixes [8] can support both high and low latency communication; all sharing the basic principles of mixing and layered encryption. Mixmaster [35] supports sender anonymity using messages encryption but does not ensure receiver anonymity. Mixminion [15] uses fixed sized messages and supports anonymous replies and ensures forward anonymity using link encryption between nodes. As a defense against traffic analysis, but at the cost of high-latencies, both designs delay incoming messages by collecting them in a pool that is flushed every t seconds (if a fixed message threshold is reached).

In contrast, Onion routing [26] was developed for low-latency anonymous communication. Similar to mix designs, each packet is encrypted in layers, and is decrypted by a chain of authorized onion routers. Tor [20], the most popular low-latency anonymity system, is an overlay network of onion routers. Tor protects against sender-receiver message linking against a partially global adversary and ensures perfect forward secrecy, integrity of the

messages, and congestion control. However, Tor is vulnerable to traffic analysis attacks, if an adversary can observe the ingress and egress points of the network. A great number of works have studied how mix networks and onion routing leak information, and how better design such systems [36, 38, 44, 48].

Recent designs. Vuvuzela [46] protects against both passive and active adversaries as long as there is one honest mix node. Since Vuvuzela operates in rounds, offline users lose the ability to receive messages and all messages must traverse a single chain of relay servers. Loopix does not operate in rounds, thus the end-to-end latency can be significantly smaller than in Vuvuzela, depending on the delay parameter the senders choose. Moreover, Loopix allows off-line users to receive messages and uses parallel mix nodes to improve the scalability of the network.

Stadium [45] and AnonPop [24] refine Vuvuzela; both operating in rounds making the routing of messages dependent on the dynamics of others. Stadium is scalable, but it lacks offline storage, whereas AnonPop does provide offline message storage. Loopix also provides both properties, and because it operates continuously avoids user synchronization issues. Additionally, Loopix, in comparison to AnonPop, protects against active attacks.

Riposte [11] is based on a *write* PIR scheme in which users write their messages into a database, without revealing the row into which they wrote to the database server. Riposte enjoys low communication-overhead and protects against traffic analysis and denial of service attacks, but requires long epochs and a small number of clients writing into the database simultaneously. In contrast to Loopix, it is suitable for high-latency applications.

Dissent [9], based on DC-networks [9], offers resilience against a GPA and some active attacks, but at significantly higher delays and scales to only several thousand clients.

Riffle [31] introduces a new verifiable shuffle technique to achieve sender anonymity. Using PIR, Riffle guarantees receiver anonymity in the presence of an active adversary, as well as both sender and receiver anonymity, but it cannot support a large user base. Riffle also utilizes rounds protect traffic analysis attacks. Riffle is not designed for Internet-scale anonymous communication, like Loopix, but for supporting intra-group communication.

Finally, Atom [30] combines a number of novel techniques to provide mid-latency communication, strong protection against passive adversaries and uses zero knowledge proofs between servers to resist active attacks. Performance scales horizontally, however latency comparisons between Loopix and Atom are difficult due to the dependence on pre-computation in Atom. Unlike Loopix, Atom is designed for latency tolerant unidirectional anonymous communication applications with only sender anonymity in mind.

7 Discussion & Future Work

As shown in Section 4.1, the security of Loopix heavily depends on the ratio of the rate of traffic sent through the network and the mean delay at every mix node. Optimization of this ratio is application dependent. For applications with small number of messages and delay tolerance, a small amount of cover traffic can guarantee security.

Loopix achieves its stated security and performance goals. However, there are many other facets of the design space that have been left for future work. For instance, reliable message delivery, session management, and flow control while avoiding inherent risks, such as statistical disclosure attacks [14], are all fruitful avenues of pursuit.

We also leave the analysis of replies to messages as future work. Loopix currently allows two methods if the receiver does not already know the sender a priori: we either attach the address of the sender to each message payload, or provide a single-use anonymous reply block [15, 16], which enables different use-cases.

The Loopix architecture deliberately relies on established providers to connect to and authenticate end-users. This architecture brings a number of potential benefits, such as resistance to Sybil attacks, enabling anonymous blacklisting [27] and payment gateways [2] to mitigate flooding attacks and other abuses of the system, and privacy preserving measurements [23, 28] about client and network trends and the security stance of the system. All of this analysis is left for future work.

It is also apparent that an efficient and secure private lookup system, one that can deliver network state and keying information to its users, is necessary to support modern anonymous communications. Proposals of stand-alone 'presence' systems such as DP5 [6] and MP3 [37] provide efficient lookup methods, however, we anticipate that tight integration between the lookup and anonymity systems may bring mutual performance and security benefits, which is another avenue for future work.

8 Conclusion

The Loopix mix system explores the design space frontiers of low-latency mixing. We balance cover traffic and message delays to achieve a tunable trade-off between real traffic and cover traffic, and between latency and good anonymity. Low-latency incentivizes early adopters to use the system, as they benefit from good

	Low Latency	Low Communication Overhead	Scalable Deployment	Asynchronous Messaging†	Active Attack Resistant	Offline Storage*	Resistance to GPA
Loopix	✓	✓	✓	✓	✓	✓	✓
Dissent [47]	✗	✗	✗	✗	✓	✗	✓
Vuvuzela [46]	✗	✗	✓	✗	✓	✗	✓
Stadium [45]	✗	✓	✓	✗	✓	✗	✓
Riposte [11]	✗	✗	✓	✗	✓	✗	✓
Atom [30]	✗	✓	✓	✗	✓	✗	✓
Riffle [31]	✓	✓	✗	✗	✓	✗	✓
AnonPoP [24]	✗	✓	✓	✗	✗	✓	✓
Tor [20]	✓	✓	✓	✓	✗	✗	✗

Table 3: Comparison of popular anonymous communication systems. By *, we mean if the design intentionally incorporates provisions for delivery of messages when a user is offline, perhaps for a long period of time. By †, we mean that the system operates continuously and does not depend on synchronized rounds for its security properties and users do not need to coordinate to communicate together.

performance. Moreover, the cover traffic introduced by both clients and mix servers provides security in the presence of a smaller user-base size. In turn this promotes growth in the user-base leading on one hand to greater security [19], and on the other a tuning down of cover traffic over time.

Loopix is the first system to combine a number of best-of-breed techniques: we provide definitions inspired by AnoA [3] for our security properties; improve the analysis of simplified variants of stop-and-go-mixing as a Poisson mix [29]; we use restricted topologies to promote good mixing [21]; we deploy modern active attack mitigations based on loops [17]; and we use modified modern cryptographic packet formats, such as Sphinx [16], for low information leakage. Our design, security and performance analysis, and empirical evaluation shows they work well together to provide strong security guarantees.

The result of composing these different techniques – previously explored as separate and abstract design options – is a design that is strong against global network level adversaries without the very high-latencies traditionally associated with mix systems [35, 15]. Thus, Loopix revitalizes message-based mix systems and makes them competitive once more against onion routing [26] based solutions that have dominated the field of anonymity research since Tor [20] was proposed in 2004.

Acknowledgments In memory of Len Sassaman. We thank Claudia Diaz and Mary Maller for the helpful discussions. This work was supported by NSERC through a Postdoctoral Fellowship Award, the Research Council KU Leuven: C16/15/058, the European Commission through H2020-DS-2014-653497 PANORAMIX, the EPSRC Grant EP/M013-286/1, and the UK Government Communications Headquarters (GCHQ), as part of University College London's status as a recognised Academic Centre of Excellence in Cyber Security Research.

References

[1] ANDERSON, R., AND BIHAM, E. Two practical and provably secure block ciphers: Bear and lion. In *Fast Software Encryption* (1996), Springer, pp. 113–120.

[2] ANDROULAKI, E., RAYKOVA, M., SRIVATSAN, S., STAVROU, A., AND BELLOVIN, S. M. PAR: Payment for Anonymous Routing. In *Privacy Enhancing Technologies, 8th International Symposium, PETS 2008, Leuven, Belgium, July 23-25, 2008, Proceedings* (2008), pp. 219–236.

[3] BACKES, M., KATE, A., MANOHARAN, P., MEISER, S., AND MOHAMMADI, E. AnoA: A Framework for Analyzing Anonymous Communication Protocols. In *Computer Security Foundations Symposium (CSF), 2013 IEEE 26th* (2013), IEEE, pp. 163–178.

[4] BALLANI, H., FRANCIS, P., AND ZHANG, X. A study of prefix hijacking and interception in the Internet. In *ACM SIGCOMM Computer Communication Review* (2007), vol. 37, ACM, pp. 265–276.

[5] BOLCH, G., GREINER, S., DE MEER, H., AND TRIVEDI, K. S. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[6] BORISOV, N., DANEZIS, G., AND GOLDBERG, I. DP5: A private presence service. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 4–24.

[7] CAI, X., ZHANG, X. C., JOSHI, B., AND JOHNSON, R. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 605–616.

[8] CHAUM, D. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM 24*, 2 (1981), 84–88.

[9] CHAUM, D. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology 1*, 1 (1988), 65–75.

[10] CHEN, C., ASONI, D. E., BARRERA, D., DANEZIS, G., AND PERRIG, A. HORNET: High-speed Onion Routing at the Network Layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015* (2015), pp. 1441–1454.

[11] CORRIGAN-GIBBS, H., BONEH, D., AND MAZIÈRES, D. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 321–338.

[12] DANEZIS, G. The Traffic Analysis of Continuous-Time Mixes. In *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004*, pp. 35–50.

[13] DANEZIS, G. Mix-networks with restricted routes. In *International Workshop on Privacy Enhancing Technologies* (2003), Springer, pp. 1–17.

[14] DANEZIS, G. Statistical disclosure attacks. In *Security and Privacy in the Age of Uncertainty*. Springer, 2003, pp. 421–426.

[15] DANEZIS, G., DINGLEDINE, R., AND MATHEWSON, N. Mixminion: Design of a type III anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on* (2003), IEEE, pp. 2–15.

[16] DANEZIS, G., AND GOLDBERG, I. Sphinx: A compact and provably secure mix format. In *Security and Privacy, 2009 30th IEEE Symposium on* (2009), IEEE, pp. 269–282.

[17] DANEZIS, G., AND SASSAMAN, L. Heartbeat traffic to counter (n-1) attacks: red-green-black mixes. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society* (2003), ACM, pp. 89–93.

[18] DIAZ, C., SEYS, S., CLAESSENS, J., AND PRENEEL, B. Towards measuring anonymity. In *International Workshop on Privacy Enhancing Technologies* (2002), Springer, pp. 54–68.

[19] DINGLEDINE, R., AND MATHEWSON, N. June 2006. anonymity loves company: Usability and the network effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*.

[20] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume* (2004).

[21] DINGLEDINE, R., SHMATIKOV, V., AND SYVERSON, P. Synchronous batching: From cascades to free routes. In *International Workshop on Privacy Enhancing Technologies* (2004), Springer, pp. 186–206.

[22] DOUCEUR, J. R. The sybil attack. In *International Workshop on Peer-to-Peer Systems* (2002), Springer, pp. 251–260.

[23] ELAHI, T., DANEZIS, G., AND GOLDBERG, I. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, Arizona, November 3-7, 2014* (2016), pp. 1068–1079.

[24] GELERNTER, N., HERZBERG, A., AND LEIBOWITZ, H. Two cents for strong anonymity: The anonymous post-office protocol. *Proceedings on Privacy Enhancing Technologies 2* (2016), 1–20.

[25] GILAD, Y., AND HERZBERG, A. Spying in the dark: TCP and Tor traffic analysis. In *International Symposium on Privacy Enhancing Technologies Symposium* (2012), Springer, pp. 100–119.

[26] GOLDSCHLAG, D., REED, M., AND SYVERSON, P. Onion routing. *Communications of the ACM 42*, 2 (1999), 39–41.

[27] HENRY, R., AND GOLDBERG, I. Thinking inside the BLAC box: smarter protocols for faster anonymous blacklisting. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (2013), ACM, pp. 71–82.

[28] JANSEN, R., AND JOHNSON, A. Safely Measuring Tor. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016* (2016), pp. 1553–1567.

[29] KESDOGAN, D., EGNER, J., AND BÜSCHKES, R. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *International Workshop on Information Hiding* (1998), Springer, pp. 83–98.

[30] KWON, A., CORRIGAN-GIBBS, H., DEVADAS, S., AND FORD, B. Atom: Scalable Anonymity Resistant to Traffic Analysis. *CoRR abs/1612.07841* (2016).

[31] KWON, Y. H. *Riffle: An efficient communication system with strong anonymity*. PhD thesis, Massachusetts Institute of Technology, 2015.

[32] LAZAR, D., AND ZELDOVICH, N. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proceedings of the 12th Symposium on Operating Systems Design and Implementation (OSDI), Savannah, GA* (2016).

[33] LE BLOND, S., CHOFFNES, D., CALDWELL, W., DRUSCHEL, P., AND MERRITT, N. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 639–652.

[34] MITZENMACHER, M., AND UPFAL, E. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[35] MÖLLER, U., COTTRELL, L., PALFRADER, P., AND SASSAMAN, L. Mixmaster Protocol-Version 2. Draft. *July, available at: www. abditum. com/mixmaster-spec. txt* (2003).

[36] NIPANE, N., DACOSTA, I., AND TRAYNOR, P. Mix-in-place anonymous networking using secure function evaluation. In *Proceedings of the 27th Annual Computer Security Applications Conference* (2011), ACM, pp. 63–72.

[37] PARHI, R., SCHLIEP, M., AND HOPPER, N. MP3: A More Efficient Private Presence Protocol. *arXiv preprint arXiv:1609.02987* (2016).

[38] REBOLLO-MONEDERO, D., PARRA-ARNAU, J., FORNÉ, J., AND DIAZ, C. Optimizing the design parameters of threshold pool mixes for anonymity and delay. *Computer networks 67* (2014), 180–200.

[39] SALTZER, J. H., REED, D. P., AND CLARK, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS) 2*, 4 (1984), 277–288.

[40] SERJANTOV, A., AND DANEZIS, G. Towards an information theoretic metric for anonymity. In *International Workshop on Privacy Enhancing Technologies* (2002), Springer, pp. 41–53.

[41] SERJANTOV, A., DINGLEDINE, R., AND SYVERSON, P. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding* (2002), Springer, pp. 36–52.

[42] SERJANTOV, A., AND NEWMAN, R. E. On the anonymity of timed pool mixes. In *Security and Privacy in the Age of Uncertainty*. Springer, 2003, pp. 427–434.

[43] SHANNON, C. E. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review 5*, 1 (2001), 3–55.

[44] SHMATIKOV, V., AND WANG, M.-H. Timing analysis in low-latency mix networks: Attacks and defenses. *Computer Security–ESORICS 2006* (2006), 18–33.

[45] TYAGI, N., GILAD, Y., ZAHARIA, M., AND ZELDOVICH, N. Stadium: A Distributed Metadata-Private Messaging System. Cryptology ePrint Archive, Report 2016/943, 2016. `http://eprint.iacr.org/2016/943`.

[46] VAN DEN HOOFF, J., LAZAR, D., ZAHARIA, M., AND ZELDOVICH, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), ACM, pp. 137–152.

[47] WOLINSKY, D. I., CORRIGAN-GIBBS, H., FORD, B., AND JOHNSON, A. Dissent in numbers: Making strong anonymity scale. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (2012), pp. 179–182.

[48] ZHU, Y., FU, X., BETTATI, R., AND ZHAO, W. Anonymity analysis of mix networks against flow-correlation attacks. In *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, vol. 3, IEEE, pp. 5–pp.

A Appendix

A.1 Incremental Computation of the Entropy Metric

Let X be a discrete random variable over the finite set $\mathscr{X}$ with probability mass function $p(x) = \Pr(X = x)$. The Shannon entropy $H(X)$ [43] of a discrete random variable X is defined as

$$H(X) = -\sum_{x\in\mathscr{X}} p(x)\log p(x). \tag{4}$$

Let $o_{n,k,l}$ be an observation as defined in Section 4.1.3 for a pool at time t. We note that any outgoing message will have a distribution over being linked with past input messages, and the entropy H_t of this distribution is our anonymity metric. H_t can be computed incrementally given the size of the pool l (from previous mix rounds) and the entropy H_{t-1} of the messages in this previous pool, and the number of messages k received since a message was last sent:

$$\begin{aligned} H_t =& H\left(\left\{\frac{k}{k+l},\frac{l}{k+l}\right\}\right) \\ &+ \frac{k}{k+l}\log k + \frac{l}{k+l}H_{t-1}, \end{aligned} \tag{5}$$

for any $t > 0$ and $H_0 = 0$. Thus for sequential observations we can incrementally compute the entropy metric for each outgoing message, without remembering the full history of the arrivals and departures at the Poisson mix. We use this method to compute the entropy metric illustrated in Figure 5.

A.2 Proof of Theorem 2

Let us assume, that in mix node M_i there are n' messages at a given moment, among which is a target message m_t. Each message has a delay d_i drawn from the exponential distribution with parameter μ. The mix node generates loops with distribution $Pois(\lambda_M)$. The adversary observes an outgoing message m and wants to quantify whether this outgoing message is her target message. The adversary knows, that the output of the mix node can be either one of the messages inside the mix or its loop cover message. Thus, for any message m_t, the following holds

$$\Pr[m = m_t] = \Pr[m \neq loop]\cdot\Pr[m = m_t | m \neq loop] \tag{6}$$

We note that the next message m is a loop if and only if the next loop message is sent before any of the messages within the mix, i.e., if the sampled time for the next loop message is smaller than any of the remaining delays of all messages within the mix. We now leverage the memoryless property of the exponential distribution to model the remaining delays of all n' messages in the mix as fresh random samples from the same exponential distribution.

$$\begin{aligned} &\Pr[m \neq loop] = 1 - \Pr[m = loop] \\ &= 1 - \Pr[X < d_1 \wedge X < d_2 \wedge \ldots \wedge X < d_{n'}] \\ &= 1 - \Pr[X < min\{d_1, d_2, \ldots d_{n'}\}] \end{aligned} \tag{7}$$

We know, that $d_i \sim Exp(\mu)$ for all $i \in \{1,\ldots,n'\}$ and $X \sim Exp(\lambda_M)$. Moreover, we know that the minimum of n independent exponential random variables with rate μ is an exponential random variable with parameter $\sum_i^{n'}\mu$. Since all the delays d_i are independent exponential variables with the same parameter, we have for $Y = \min\{d_1, d_2, \ldots d_{n'}\}$, $Y \sim Exp(n'\mu)$. Thus, we obtain the following continuation of Equation (7).

$$\begin{aligned} &\Pr[m \neq loop] = 1 - \Pr[X < Y] \\ &= 1 - \int_0^\infty \Pr[X < Y | X = x]\Pr[X = x]\,dx \\ &= 1 - \int_0^\infty \Pr[x < Y]\,\lambda_M e^{-\lambda_M x}dx \\ &= 1 - \int_0^\infty e^{-n'\mu x}\lambda_M e^{-\lambda_M x}dx \\ &= 1 - \frac{\lambda_M}{\lambda_M + n\mu} \\ &= \frac{n'\mu}{n'\mu + \lambda_M} \end{aligned} \tag{8}$$

Since the probability to send a loop depends only on the *number* of messages in a mix, but not on which messages are in the mix, this probability is independent of the probability from Theorem 1. Theorem 2 follows directly by combining Theorem 1 and Equation (8), with $n' = k + l$. We get for messages m_1 that previously were in the mix,

$$\begin{aligned} \Pr[m = m_1] &= \Pr[m \neq loop]\cdot\Pr[m = m_1 | m \neq loop] \\ &= \frac{(k+l)\mu}{(k+l)\mu + \lambda_M}\cdot\frac{k}{n(k+l)} \\ &= \frac{k}{n}\cdot\frac{\mu}{(k+l)\mu + \lambda_M}. \end{aligned}$$

Analogously, we get for m_2,

$$\begin{aligned}\Pr[m = m_2] &= \Pr[m \neq loop] \cdot \Pr[m = m_2 | m \neq loop] \\ &= \frac{(k+l)\mu}{(k+l)\mu + \lambda_M} \cdot \frac{1}{k+l} \\ &= \frac{\mu}{(k+l)\mu + \lambda_M}.\end{aligned}$$

This concludes the proof.