Empirical measurements of the Nym mix network

Corentin Junod

IC School EPFL
Security and Privacy Engineering Laboratory

Master Thesis

Spring 2023

Responsible
Prof. Carmela Troncoso
EPFL

Supervisor
Ania M. Piotrowska
Nym Technologies SA

Abstract

The internet was not originally designed to prioritize privacy. Even with encrypted content, the network layer still exposes significant information about individuals through the leakage of metadata.

To address these concerns, several overlay networks like VPNs and onion routing have emerged, each employing different techniques to mitigate these risks. Among them, the Nym network stands out as a mix network that relies on the packet mixing technique. This approach prevents adversaries from linking incoming and outgoing packets based on traffic analysis, offering robust privacy protection even against global adversaries. However, one notable downside of such networks is the trade-off in speed due to the implementation of time-consuming privacy-preserving mechanisms.

In this thesis, we present a comprehensive empirical examination of the latencies experienced by users in the Nym network during their communication. We analyse how various components and processes impact the latency of the transferred data.

Furthermore, we closely investigate the influence of the nodes' geolocation on the latency and compare the default path selection algorithm, which randomly selects nodes, with optimized mechanisms based on inter-node distance. Specifically, we propose a new path selection algorithm that improves the network latency.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Ania Piotrowska, for her devotion and guidance throughout this thesis, which would not have been possible without her support. She perfectly managed to channel and crystallize the incoming ideas and bring her deep expertise in mixnets, even in the exhilarating and fast-paced environment that is Nym.

I'm also extremely grateful to the whole SPRING team at EPFL, who greeted me with an exceptionally warm welcome and shared their true commitment, striving to preserve and enhance privacy during times when it is constantly threatened. Special thanks to Prof. Carmela Troncoso, who I had the chance to meet as an inspiring and vibrant professor, who passes on her passion for computer security and privacy to her students, including myself.

Finally, my warm and heartfelt thanks to my family's unconditional, infallible and loving support.

Contents

List of Figures

Chapter 1

Introduction

From its inception, the Internet was not primarily designed with privacy as a fundamental aspect. Originally conceived as a military project, it rapidly expanded throughout the seventies and eighties without incorporating sufficient safeguards against eavesdropping by adversaries. The introduction of HTTPS in the mid-nineties marked a significant milestone by offering encryption for the communication's content through a secure layer added to the HTTP protocol. This encryption has now become commonplace and widely adopted across numerous services.

Despite the presence of strong end-to-end encryption, an eavesdropping adversary can still gather metadata from a connection. This metadata includes network location, transmission time, duration, volume, and more. The exposure of such information poses a significant risk as it can potentially deanonymize users and jeopardize their privacy, particularly over an extended period. For instance, a regular connection to a specific bank website may disclose the ownership of an account with that bank, thus revealing sensitive personal information. Stewart Baker, former NSA General Counsel from 1992 to 1994, once emphasized the significance of metadata, stating "*Metadata absolutely tells you everything about somebody's life. If you have enough metadata you don't really need content.*"[1] This statement underscores the critical role metadata plays in providing comprehensive insights into an individual's life. He suggests that with a sufficient amount, the actual content becomes less necessary for understanding and analysing a person's activities and behaviour.

With growing interest from various entities in processing private data, the incentives to employ such intrusive techniques continue to escalate. This poses a significant threat to anonymity and personal privacy, ultimately paving the way for potential consequences such as censorship, propaganda dissemination, and even retaliatory actions. The erosion of privacy and anonymity in the digital sphere raises concerns about the potential misuse

[1] Alan Rusbridger "The Snowden Leaks and the Public", November 2013

and abuse of sensitive information by these actors. In recent years, a series of revelations, initiated by figures like Edward Snowden, have brought to light the active surveillance and extensive data collection carried out by intelligence agencies. These agencies, often working in collaboration across countries and with corporations, have established wide-ranging data-sharing agreements. The emergence of various scandals has further demonstrated that adversaries now possess global capabilities, enabling them to engage in widespread surveillance of communications. Their ability to collude, exchange information, and amass unprecedented volumes of traffic has become evident.[2][3]

As the usage of HTTPS and TLS becomes increasingly widespread, eavesdroppers primarily rely on metadata to compromise privacy. Over the past few decades, several endeavours have been undertaken to develop mechanisms for concealing metadata. These range from trusted-party solutions like VPNs, growing in popularity during recent years, to decentralized technologies such as Tor [17], Nym [9], Orchid [6], and I2P [27]. These efforts encompass diverse threat models and offer varying levels of privacy guarantees.

In this work, we focalize our attention on the Nym [9] network, a recent implementation of mixnets founded in 2019 by the homonymous company Nym[4]. We are not aware of any prior research regarding measurements and evaluation of the Nym mainnet, and we consider that a better understanding of this cutting-edge technology can help to improve the network and mixnets as a whole.

Thesis Outline: In Chapter 2, we discuss various technologies designed to conceal metadata and explore the trade-offs associated with each. In Chapter 3, we describe in detail the design of the Nym mix network, which is a key focus of this work. In Chapter 4, we present an empirical analysis of the latencies imposed by the Nym network. In Chapter 5, we demonstrate a new path selection algorithm that reduces the network latency, and we analyse its tradeoffs. In Chapter 6, we discuss the remaining questions and outline potential avenues for future studies on this topic. Finally, we conclude in Chapter 7.

[2]The Maximator program: https://www.economist.com/europe/2020/05/28/a-beery-european-spy-club-is-revealed

[3]The Echelon program: https://www.europarl.europa.eu/doceo/document/A-5-2001-0264_EN.html

[4]Nym: https://nymtech.net/

Chapter 2

Background in Anonymous communications

Over the past two decades, many technologies have been proposed to hide metadata at the network layer and protect privacy. In this chapter, we provide an overview of their most renowned methodologies, detailing their characteristics, advantages, and drawbacks.

2.1 VPNs

Virtual Private Networks (VPNs) are encrypted connections between two hosts over the internet, such that the client is considered in the same subnetwork as the VPN provider. This provider can then forward the client's connection further into the network, acting as a relay on behalf of the user. From the receiver's point of view, the connection appears to originate from the VPN provider, hiding the client's IP address.

Such networks provide sender anonymity by masking the true origin of the connection. They also provide sender and receiver unlinkability against local eavesdroppers spying on a single link. An adversary spying between the sender and the VPN relay knows that the sender is using a VPN, but does not learn the receiver. Symmetrically, an adversary spying between the relay and the receiver knows the receiver, but not the sender.

While VPNs are effective in safeguarding against local passive adversaries, they may not provide adequate protection in the following scenarios:

- If the adversary spies on both links simultaneously (between the sender and the relay, and between the relay and the receiver). In such a case, the adversary can perform a traffic correlation attack [19] and link the sender and the receiver.

- If the VPN provider is malicious or leaks information. The relay constitutes a single trusted point of failure. The provider itself knows how

the packets are transferred, and, moreover, can disclose or leak this information. This has proven to be a concern over multiple existing VPN providers in the past [30].

- Fingerprinting attacks. As the packets are simply wrapped into a VPN stream, an attacker can learn and match the observed streams, even if they are encrypted [21][5].

2.2 Decentralized VPNs (dVPNs)

Decentralized VPNs (dVPNs) operate under the same underlying assumptions as VPNs, but tackle the concern of relying on a single trusted party to obfuscate the users' activities. This is achieved by having each device acting both as a sender and a VPN relay. When connecting, clients select another device running the dVPN service and use it as their proxy.

Despite the fact that dVPNs distribute trust among many peers, the existing solutions still allow an adversary to correlate the input and output traffic of any given relay based on timing analysis, and fingerprinting is still possible even if packets are routed via multi-hop paths.

Finally, each relay is still fully aware of the link between sender and receiver for its traffic (if not multi-hop), it is unclear if trusting one random unknown relay is better than trusting one corporate service provider. There is a wide variety of dVPN technologies, that differ in their precise implementation and guarantees. Some of them are single-relayed [34] [6], while others implement multiple-hop paths by design [32] or as an optional feature [1].

2.3 Onion routing & Tor

Onion routing was first proposed in 1996 by David M. Goldschlag, Michael G. Reed and Paul F Syverson [20]. Such overlay networks route the traffic through a series of intermediate relays. The sender selects a path made of nodes belonging to the onion network. For each hop in the path, the message is encrypted using the node's key. As a result, the message is encrypted multiple times, with one encryption layer per node. The message is then sent through the network. When reaching a node, it "peels off" its encryption layer. The last hop on the path peels off the last onion layer, leading to the initial message which is then transmitted to the receiver.

Similarly to VPNs, such onion networks provide sender anonymity and protect against local adversaries by providing unlinkability between senders and receivers. Moreover, they solve the single point of trust mentioned earlier, as each relay only knows the preceding and next hops on the path. Specifically, the first node (called *entry node*) learns the user IP and the middle node; the middle node knows the first and last nodes, but nothing

about the sender nor the receiver; and the last node (called *exit node*) learns the receiver but not the sender. Overall, not a single node is able to link the sender and the receiver.

By looking at any connection in the network, a local adversary is unable to link the traffic back.

- Looking at the link between the sender and the first hop, an adversary knows the sender's IP, and that the sender communicates with an onion network, but doesn't learn the true receiver.
- Looking at a link inside the network (from one hop to another), an adversary does not learn any information regarding the sender or the receiver.
- Looking at the link between the last node and the receiver, an adversary learns the receiver but does not learn the true sender.

As described, onion routing, unlike VPNs, encompasses active adversaries within its threat model. A single malicious node in the path cannot link the sender and receiver. Unlinkability remains intact as long as at least one non-malicious node is present on the path. Furthermore, as the network comprises numerous globally distributed nodes, it is assumed to be exceedingly challenging to spy on multiple links or compromise all the nodes along a given path.

2.3.1 Tor

The most famous onion routing implementation is Tor [17], which routes the traffic through three Tor nodes before reaching its destination. This network is currently fully deployed, used by millions of users, and actively studied.[1] It is also good to note that, in addition to the original onion routing protocol, the Tor project provides a dedicated browser with built-in protections against website fingerprinting, which is out of the scope of this work. Similar analyses as the one we conduct in this work have been executed with Tor [13] along with considering multiple path selection algorithms [35], some of them similar to the one presented in this work [8].

Even if onion routing, and specifically Tor, works under a strong threat model, global adversaries are not considered. An attacker able to observe both the entry and the exit traffic on Tor can perform an end-to-end correlation attack and link back the sender to the receiver [26]. Over the years, Tor has been subject to many attacks, most of them relying on some kind of traffic analysis [4][29]. Furthermore, Tor does not provide unobservability i.e. an adversary looking at the traffic from a client knows if this client is communicating simply by looking if packets are exchanged. Finally, Tor

[1] Tor metrics: https://metrics.torproject.org/userstats-relay-country.html

is not fully decentralized. The list of nodes is fetched from a register of directory authorities [2] which are manually selected from a "*we-know-you-and-have-had-many-beers-with-you*" metric [3]. This list is hard-coded inside the Tor client. Compromising the nodes in that list may severely harm the network.

During the past two decades, Tor has been subject to many publications regarding its security and privacy [18].

2.4 Mix networks

Mix networks, or mixnets, were first introduced by Chaum [7] in the early eighties and, hence, are older than the concept of onion routing. They were initially invented to provide an anonymous mailing system.

Like onion routing, the packets are encrypted over multiple layers and routed through a series of nodes. However, in mixnets, the packet forwarding process differs from a first-in, first-out order. Instead, the nodes batch multiple packets together, shuffle them randomly, and then forward the batch to the next destination. Each packet is also padded using a random string so that each message has exactly the same size. Therefore, the network observer cannot trace the packets based on their size, timing or binary representation. This addresses the correlation attacks performed on Tor.

Mixnets assume a global adversary that can spy on any link in the network. The key security properties that they aim to provide are (1) *unlinkability* from an external global adversary point of view, and (2) *sender anonymity*. These properties remain intact even when the adversary controls all nodes except one along the path. The presence of at least one honest node ensures that the in-

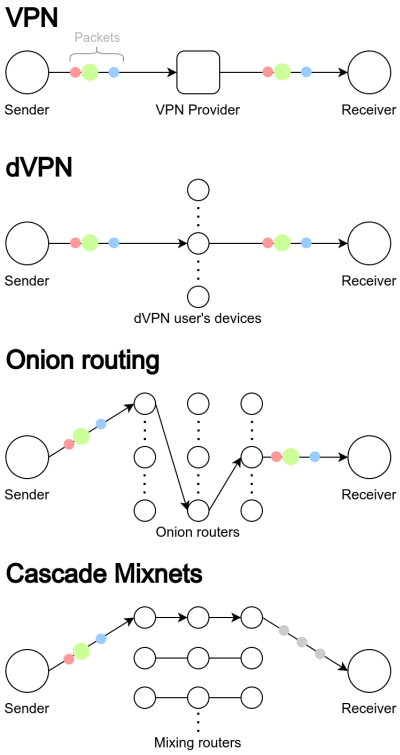

Figure 2.1: Different types of anonymous communication networks. Packets in grey are indistinguishable from each other.

[2] Tor authorities: https://metrics.torproject.org/rs.html#search/flag:authority

[3] Blogpost about Tor authorities: https://blog.torproject.org/introducing-bastet-our-new-directory-authority/

put and output messages are unlinkable, and belong to an anonymity set equivalent to the size of the batch.

In the original design, the mix node gathers a batch of packets before mixing them. This implies a latency on the network proportional to the anonymity set. A bigger anonymity set involves larger batches leading to higher latencies. The proposed topology is a cascade mixnet, where paths are established in advance and clients, upon sending a packet, select one of the available routes. This scheme scales poorly [14], mainly because enlarging the network means adding more cascades. As they are disjointed paths they partition the network, and each of them provides a suboptimal anonymity set bounded by the number of users communicating along that path. Many other network topologies have been proposed since to improve privacy and performances [11] [14].

The original design also targeted a mailing system that sends plain text messages. Achieving unlinkable packets for a general-purpose communication system was out of the scope of the work. In particular, the same message encrypted with the same keys must not be repeated, as it would lead to the exact same packets. An adversary observing the network can then correlate those packets and undo the mixing step.

Finally, as in onion routing, mixnets do not provide sender unobservability. Assuming a global adversary spying on the network over extended periods of time, and a client recurrently connecting to the same receiver, the adversary knows that the receiver is among the set of nodes receiving traffic. The longer the adversary looks at the network, the more limited the set of potential receivers becomes.

2.5 The Nym Network

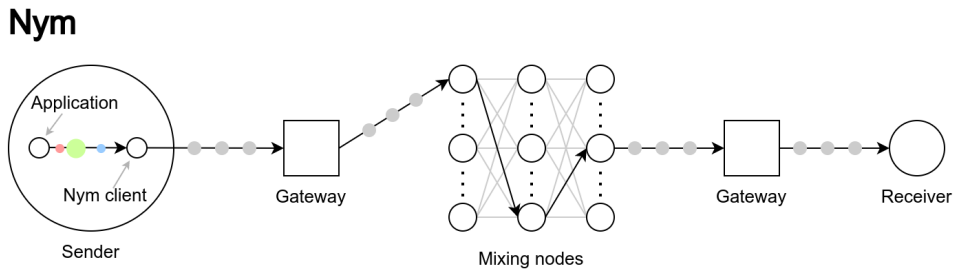

Figure 2.2: In the Nym mixnet, every packet leaving the client is indistinguishable from the others

The Nym network [9] is a novel type of mixnet, which aims to provide a general-purpose mixnet to the public, in the same way, Tor is for onion

routing.[4] The Nym network works as a traditional mixnet under the same threat model, with some enhancements and stronger assumptions.

Contrary to the original mixnet design, the Nym network [9] does not process packets by batches. When the sender creates and encrypts packets, it chooses a delay for each hop in the network. The mixing nodes, upon handling a packet, wait for the corresponding delay before sending it further. The delays are sampled from an exponential distribution and, due to its memoryless property, the time a packet leaves a node is not correlated to the time of arrival. This implies that the anonymity set of a given node at a given time t is the set of all packets having entered the node and not having left it until t. Using this technique significantly decreases the network latency.

The Nym network uses the Sphinx packet format [12] to layer encrypt the packets. Sphinx is a cryptographically secured format that guarantees bitwise unlinkability between packets after each node processing. Unobservability is addressed by adding cover traffic sent by the client. When the client is running, it schedules the sending time of the next message following a Poisson point process. If there is no message to be sent, a dummy loop cover message is generated and sent to the client itself through the network. This message will be discarded by the client upon reception. The Poisson process guarantees that each sent event occurs independently, therefore an adversary spying on the connection is unable to tell if a client is actively communicating.
The network is also fully decentralized, all the information required for the network to operate is stored in a blockchain run by nodes called *Validators*. They ensure the right number of mixnodes are actively mixing traffic and scale the network according to the number of users.

The topology of the Nym mixnet diverges from the original proposal. Mixnodes are scattered into three layers, assigned randomly by the validators. This assignment is renewed at the beginning of each epoch, which currently lasts one hour. Upon sending a packet, the client computes the path by selecting one random node from each layer, in order. Consequently, the anonymity set is larger than with a cascade design. Each packet is given a different path and the privacy scales with the total number of users communicating using the network.

Prior to this work, measurements have been made using a mixnet simulator [31] to empirically define the Nym mixnet parameters [24].

[4] Nym website: https://nymtech.net/

Chapter 3

The Nym Network

In this chapter, we provide an in-depth exploration of the Nym infrastructure, offering a comprehensive description of its primary components.

3.1 The Nym building blocks

Before discussing the network architecture itself, we examine the fundamental underlying technologies employed by the Nym network.

3.1.1 The Sphinx packet format

Sphinx [12] is a cryptographic packet format designed to facilitate data routing in multi-hop anonymous communication networks. It provides key security properties:

- *Bitwise unlinkability:* At each hop, a layer of encryption is stripped away from each packet. The input and output of this process remain bitwise unlinkable, meaning that an adversary cannot establish a correlation between an output packet and its corresponding input packet based on their binary patterns.

- *Integrity:* This packet format is resistant to tampering attacks, like data injection or replay attacks.

- *Hidden routing information:* In the packet header, all metadata, except for the information regarding the predecessor and subsequent node of a specific link, is encrypted. This includes concealing the number of hops the message has traversed thus far and obscuring the actual count of mixes along the message's path.

- *Constant size:* All the packets are padded to be the same size, which makes them unrecognizable based on that metric.

Overall, the Sphinx packet format completely blinds a packet's content and its metadata, making it impossible for an adversary to link an output packet to an input packet at any mixnode.

Sphinx also implements single-use reply blocks (SURB) providing a way for the receiver to reply while keeping the sender anonymous. A SURB is generated by the sender and contains (1) a Sphinx header comprising a path back to the sender, (2) a key to cypher the reply's payload, (3) the address of the first mixnode in the reply path. Upon reception, the receiver cyphers its reply payload with the provided key, appends the ciphered reply to the provided Sphinx header, and sends the resulting packet to the indicated first mixnode. This generates a Sphinx packet indistinguishable from any other while keeping the receiver's anonymity.

SURBs allow for a convenient way to implement acknowledgements in the Nym network. With each message, the sender generates the corresponding acknowledgement and embeds it in the message payload. When acquiring the packet, the receiving gateway extracts and sends back the acknowledgement.

In the current Nym implementation, the default Sphinx packet size is 2048 bytes. Optionally, a user may configure their Nym client to use larger-sized packets of up to 32768 bytes. The acknowledgement inside messages, added with a public key for the Diffie-Hellman exchange, counts for an overhead of 437 bytes, which need to be subtracted from the packet size to get the amount of usable data carried by a packet.

3.1.2 The zk-nyms credentials

The zk-nyms credentials, an extension of the Coconut scheme [33], are an attributes-based credential scheme providing powerful cryptographic capabilities, in particular :

- *Blind issuance:* It allows a user to make an issuer blindly sign some attributes in the credential without revealing their value, such that the issuer and the verifier are not able to identify the attributes. The proof that attributes hold the right value is done using zero knowledge.
- *Threshold issuance:* zk-nyms allows multiple issuers to sign the credentials, the threshold issuance property allows a credential to be valid if a given number of issuers have signed. This property decentralizes the credential system, avoiding relying on a single authority.
- *Selective attribute disclosure:* When showing a credential, the user decides which attributes should be revealed and which should be kept secret. The verifier only sees the disclosed attributes, empowering the user to only show the required attributes in their credentials and nothing more. Coupled with the re-randomization property, the attributes become unlinkable.

- *Re-randomization:* The property allows a credential to be randomized, which means the credential will keep its attributes, their values, and the signature, but will be unrecognizable from its previous state or any other credential with the same attributes and values. This prevents the verifier from recognizing the same credential shown multiple times.

The zk-nym credentials integrate well with a blockchain, which perfectly fits Nym's requirements. Nym aims at using [1] these credentials to prove the right of clients to use the mixnet. Clients can buy NYM tokens and later spend them to acquire zk-nyms granting access to the network. Nodes in the network are rewarded for their work using the same tokens, creating an incentive to contribute to the network.

3.1.3 NYM tokens and the Nym blockchain

NYM Tokens : The NYM tokens are a virtual utility coin maintained by the validators using the Nym blockchain. These tokens mainly serve as a reward for nodes, scaling based on the amount of mixing done.

To compute the wages, measurement packets, indistinguishable from other packets, are generated by the clients and the mixnodes. The generation is based on a publicly verifiable random function disclosed after each period, to ensure nodes' rectitude. At the end of every epoch, the nodes commit to the messages they forwarded during that epoch, and then publicly reveal which measurement messages they sent through the network. Everyone can then witness how many measurement packets each node forwarded correctly, and the rewards are based on that metric.

Nym tokens also serve as a medium of exchange to access the mixnet. Along with the credentials, they provide a flexible way to get Nym services. For example, internet service providers could include an amount of NYM tokens with some of their subscriptions, or service developers may allow some users to connect through the mixnet by giving them tokens under conditions.

Nym Blockchain : All the transactions with NYM tokens are written in the Nym blockchain by the validators. This provides a decentralized and unique logbook of the network's operations such as the credential issuance, the operations with Nym tokens (including node's rewards), the aforementioned commitment of forwarded messages, etc. The blockchain technology is based on CosmWasm[2], a smart contracts platform built on top of the Cosmos SDK[3], a framework for building blockchain applications.

[1] Currently the credentials are only enabled in the testnet, a separate, smaller network than the mainnet.

[2] CosmWasm: https://cosmwasm.com/

[3] Cosmos SDK: https://github.com/cosmos/cosmos-sdk

Currently, the measurement packets are still a feature to develop and the mixnet can be used freely without tokens.

3.1.4 Staking mechanism

The validators determine the active set of mixnodes based on the nodes' stake, the higher the stake, the higher the probability for a mixnode to be selected during the next epoch. Every mixnode can invest NYM tokens to increase its reputation and also its chance to be selected[4], which leads in turn to a token reward at the end of every epoch the node was selected.

This mechanism serves as a Sybil prevention, spawning a considerable amount of malicious nodes would require a massive amount of tokens. Stakeholders of NYM tokens, besides running mixnodes, can also delegate part of their possession to existing ones, increasing the stake and reward to whom they delegate to. The stakeholders then receive in return part of the reward from the delegated node. The detailed mechanisms of the tokens scheme can be found in the NYM token economic paper [10].

3.2 Nym Network Architecture

In this section, we provide a comprehensive breakdown of the architecture of the Nym network. We begin by outlining each component that comprises the network, followed by an explanation of their interactions with one another.

3.2.1 Mixnodes

Mixnodes serve as intermediary relays responsible for forwarding users' traffic within the Nym network. Their key operations involve the cryptographic processing of received packets, such as key derivation and decoding a single layer of Sphinx encryption. Once these operations are completed, the mix node gains knowledge of the mixing delay, which determines the duration for which the packet should be retained before being forwarded to the next hop.

For each epoch of the network, the validators determine from the set of mixnodes a subset of active ones. Mixnodes with a higher stake have a higher probability of being selected. The selected nodes are then spread across three layers, based on a verifiable random function. The validators attribute a routing score to each node based on how well they perform at mixing. After each epoch, the nodes earn rewards proportionally to their routing score[5], creating an incentive for nodes to mix packets well. The path

[4]The mixnodes' inclusion probability: https://validator.nymtech.net/api/v1/status/mixnodes/inclusion_probability

[5]Mixnodes' rewards: https://validator.nymtech.net/api/v1/mixnodes/rewarded

taken by each packet is chosen by the client upon its creation. It is always made of one node of layer one, then one node of layer two, and finally one node of layer three.

Mixnodes also run the VerLoc [28] algorithm, which measures the link latencies in the network. During startup, and then every 12 hours, each mixnode will measure its latency to every other mixnode (including non-active ones) running compatible software. The results are publicly exposed at http://*node_ip*:8000/verloc. Currently, details about mixnodes are provided in the network explorer and in an API endpoint.[6][7][8]

3.2.2 Mixnet topology

The Nym network is based on a stratified topology comprising three layers i.e. the mixnodes are each assigned a layer, and two consecutive ones are fully connected. The client assigns each packet, upon its creation, a random path going through the three layers.

Every hour, the mixnet enters a new epoch, which serves as a clock to make the network evolve over time. At the beginning of each epoch, the active set of nodes and their layers are shuffled, rewards are distributed, routing scores are computed, and temporary keys are refreshed to preserve forward secrecy. The current epoch information is accessible through the API[9]. The duration of epochs may change in the future, as stated in the whitepaper [9], to adjust the tradeoff between network flexibility and overhead.

In addition to the active set choosing nodes based on their routing score, mixnodes and gateways with reliability lower than 50% in the last 24 hours are blacklisted. These nodes are excluded from participating in the active set and receive no reward at the end of each epoch.

3.2.3 Gateways

Clients are not directly connected to the mixnodes, rather they establish a bond with a gateway serving as a proxy to access the Nym network.

The gateways are responsible for

- Checking a client credential, to ensure that this client has the right to use the Nym network[10].

[6] Network Explorer: https://explorer.nymtech.net/network-components/mixnodes
[7] List of all mixnodes: https://validator.nymtech.net/api/v1/mixnodes
[8] Active mixnodes: https://validator.nymtech.net/api/v1/mixnodes/active
[9] Current epoch: https://validator.nymtech.net/api/v1/epoch/current
[10] This feature is still under development. Currently, by default, no check is performed and credentials are disabled.

- Forwarding the traffic of authorized sending clients to the mixnet. The gateways do not perform any cryptographic operation on packets, they are simply relaying the traffic into the Nym mixnet.

- Receiving the traffic coming out of the mixnet and forwarding it to the receiving client.

- The gateways could also perform useful tasks, such as keeping packets while clients are offline. Users can later retrieve the stored packet, as in a "packet mailbox".

The link between clients and gateways is made by spending credentials, this ensures the client remains anonymous. Upon bounding, the gateway checks the credential and marks it as spent in the Nym blockchain. If the operation succeeds, the gateway and the client agree on a temporary pseudonym for the client. The address of a client is then composed of the client's gateway address and the client's pseudonym. When encoding a packet, the sender's client attaches the recipient's address at the end of the mix route. To process an incoming message, the gateways extract the pseudonym from the address and forward the message to that client.

In addition, gateways could also serve as a censorship resistance feature, enabling user access through disguised connections. This is not possible with mixnodes that must be publicly listed in order for the network to properly work. Currently, the list of public gateways can be found in the web interface[11] and as an API endpoint [12].

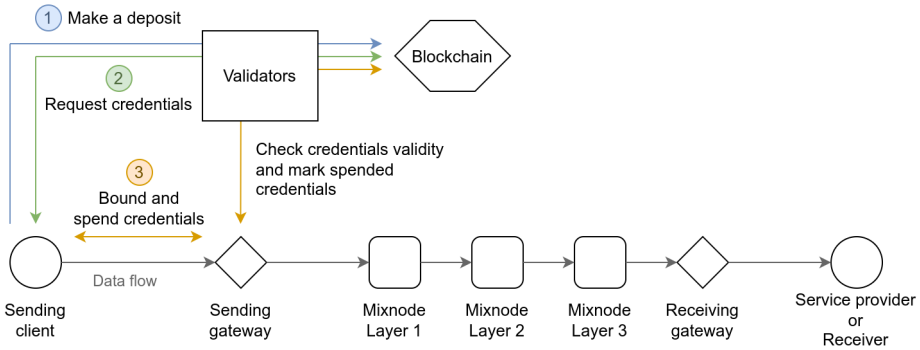

Figure 3.1: Overview of the steps to access the mixnet.

[11] Gateway web interface: https://explorer.nymtech.net/network-components/gateways

[12] Gateway API: https://validator.nymtech.net/api/v1/gateways/

3.2.4 Clients

The clients are the network endpoints. They offer a convenient way for any application to connect to the Nym network.

The clients are responsible for

- Choosing a gateway. This can be configured by the user, otherwise, a default random gateway is picked.
- Receiving the traffic from an application and converting it into a flow of Sphinx packets.
- Retrieving the current network topology for node selection, and selecting a route of three mixnodes per packet.
- Selecting one delay per packet per hop, sampled from an exponential distribution.
- Ensuring the reliability of the protocol by detecting and retransmitting packets if they exceed a given timeout.
- Sending cover traffic to provide unobservability for the user.
- Sending back acknowledgements when receiving a packet.

Each of the clients uses the same underlying module, the *client core*, which provides all the common functionalities of the Nym client. Nym is currently providing three different types of clients.

- WebSocket (Native) client. The native client serves as a proxy for applications supporting WebSocket proxies. The application redirects the traffic through the client, which then forwards the traffic through the mixnet.
- SOCKS5 client. Similarly to the WebSocket client, the SOCKS5 client redirects a SOCK5 application flow through the mixnet.
- WebAssembly client. This client is intended to be used with web applications, providing connectivity to the Nym network directly in the browser. It works in the same way as the two aforementioned clients.

Before running a client, it should first be initialized. During this step, the client picks a gateway to use at random and derives its private and public keys. It is also possible to configure the client to use a specific gateway, a specific API endpoint, or choose the parameters of the WebSocket protocol. Once initialized, the client should be started. It then actively listens for messages to send through the mixnet.
Upon reception of a message, the client proceeds as follows:

- The client splits the message into Sphinx packets. For each packet, a route of three random mixnodes is chosen. For each mixnode, a mix delay is sampled from an exponential distribution of mean $\mathbb{E}[X] =$ 50 ms. The client then creates each packet by running all the required cryptographic operations.

- Once all the packets are created, the client puts each packet into the *sending queue*.

- An independent Poisson process runs in parallel to the one splitting messages into packets. This process will pop one packet from the sending queue at an exponentially distributed interval of mean $\mathbb{E}[X] =$ 20 ms. If the sending queue is empty at that time, a loop cover message is generated instead. This guarantees that the sending rate of the client is constant and that a sending event is independent of others, providing unobservability. An adversary spying on the first link of a path learns if the user is running a Nym client, but does not learn if the user is actively communicating or not.

- For each packet sent, a retransmission time is computed following the formula $Timeout_{retransmission} = MixDelays \cdot AckWaitMult + AckWaitAdd$ where $Timeout_{retransmission}$ is the time after which the packet is retransmitted, $MixDelays$ is the sum of all the mix delays of the mixnodes along the packet's path and along its acknowledgement path, $AckWaitMult$ is a multiplicative amount of default value 1.5 and $AckWaitAdd$ is an additive amount of default value 1500 ms.

- Upon reception of a packet, the client will store it until all the packets of the corresponding message have arrived. The client then reconstructs the message and forwards it to the application.

3.2.5 Validators

Validators are responsible for maintaining the blockchain in which the network-wide information is stored (mixnet topology, mixnodes public keys, token transactions, rewards, etc). They provide a decentralization piece of technology. Validators are also issuing credentials granting access to the mixnet. When users want to use the mixnet, they first need to make a token deposit which is inscribed in the blockchain. They can later request the validators for credentials. If the user has deposited enough tokens, the validators issue a credential testifying that the user possesses that amount of tokens. The user can then spend that credential while bounding to a gateway. If the credential is valid, the tokens are inscribed in the blockchain as spent and the gateway grants access to the mixnet. Those steps are summarized in figure 3.1.

This whole process ensures that users can access the mixnet without revealing their identity. The zk-nyms credentials guarantee that users can prove they possess valid credentials, without revealing any information beyond that simple fact.

3.2.6 Service provider

Service providers within the Nym network are external applications that offer services to users. An example of a service provider can be a network requester, an RPC point, or an instant messaging server. These providers run Nym client software, furnishing endpoints, so users can connect and use the services. The network explorer is maintaining a list of endpoint addresses[13]. Nym is currently providing NymConnect[14], a client connecting to SOCKS5 service providers currently supporting Telegram, Keybase, Blockstream, and Electrum connections.

To simplify integration, Nym credentials are designed to be utilized by service providers as an alternative to traditional subscription systems that typically involve compromising user anonymity. By doing so, service providers can benefit from the privacy properties of the zk-nyms.

[13]Service providers: https://explorer.nymtech.net/network-components/service-providers

[14]NymConnect: https://nymtech.net/download-nymconnect/

Chapter 4

Empirical Analysis of Latency in the Nym network

4.1 Methodology

In this chapter, we provide a comprehensive empirical analysis of the latency experienced within the Nym network. We begin by introducing the methodology employed for conducting the measurements. Subsequently, we present the collected data and analyse the impact of various processes on network latency. This analysis enables us to gain valuable insights into potential methods for enhancing network performance and identifying any existing bottlenecks.

4.1.1 Measurement scripts

We implemented two Python 3.10.6 scripts to instrument the Nym client and collect measurements from the network. The first one, referred to as Starter script, is responsible for setting up and executing the Nym client (v1.1.10). It simulates a user employing the client to send a given amount of data through the network. The second one, referred to as the Controller script, coordinates a fixed number of Starters to conduct measurements. This configuration enables the simultaneous execution of multiple Nym client instances, allowing them to collectively perform measurements and gather data points.

Controller and Starters

Every Starter script is provided with parameters from the Controller, which describe the configuration of the Nym client to be launched. The Controller ensures each client shares the same log files and that they run on a different port. It also orchestrates the Starters so that the work is distributed among them and launches a new Starter in case one of them fails.

Each Starter script proceeds to initialize its Nym client based on these parameters. During this initialization process, the Nym client establishes a connection with the designated gateway, generates cryptographic keys for secure communication, and configures its settings accordingly. Once the initialization is completed, the Starter script starts its Nym client. This action creates a new Nym client process waiting for incoming messages.

The Starters and the Controller scripts are communicating through message queues. Once a Starter has finished initiating and starting a client, it will issue a READY message. The Starter then waits for a command from the Controller. The commands are START, STOP, and WAIT. The Starter state machine is shown in figure 4.1.

Upon receiving a START command, the Starter opens a WebSocket connection to its Nym client and sends messages one by one, waiting for a previous message to come back before sending the next one. This avoids any queuing delay of multiple messages waiting on the Nym client side. We defined a timeout for every message so that the Starter does not hang if a message is lost. Once the number of requested messages is reached, the Starter sends a new READY.

Upon receiving a WAIT, the starter simply ignores the message and issues another READY. We needed this feature because sometimes the Controller consumes a READY, but decides not to start a new measure. For example, this happens at the end of a measurement, when the Controller is polling the Starters but doesn't have any new message to send.

Upon receiving a STOP, the Starter kills the Nym client process, sends a FINISHED message and terminates.

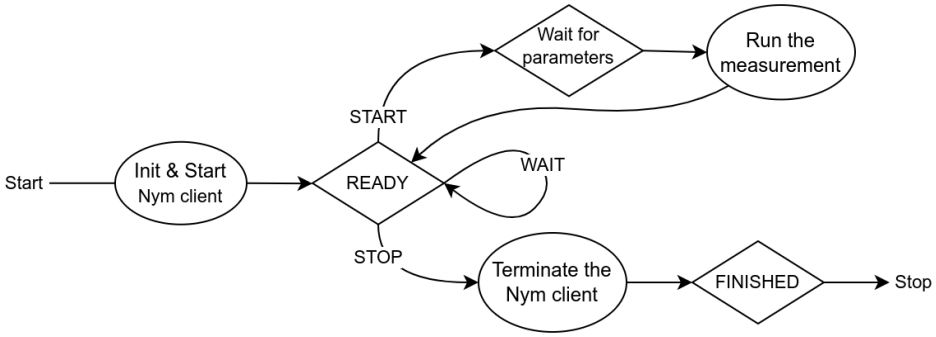

Figure 4.1: Starter state machine. Diamonds are states, ovals are actions.

Additionally, the Starter script has the capability to accept separate sending and receiving gateways as parameters. In such cases, the Starter creates two distinct Nym clients and associates each one with its respective gateway. During the measurement process, the Starter utilizes the sending client to transmit messages while simultaneously employing the receiving client to receive those messages.

Configuration parameters

The controller script serves as the central location for defining global parameters for the measurements. It orchestrates the parallel execution of multiple Starters and provides each of them with the following arguments:

Logging directory Each time the Starter script is launched, a new log folder is selected with a name based on the current time. This log folder serves as the destination for both the Starter and the Nym client to store their respective data.

Logging file The script allows specifying a filename where the data is written, giving the option to instruct a specific file for the clients to log their data.

Client ID For every spawned client, a unique client ID is assigned, and it is incremented for each subsequent client created.

Message size the size of the message to send through the Nym mixnet.

Sphinx packet size the Sphinx packet size, which can be 8, 16, or 32. If another value is provided, the client will automatically fallback to the default Sphinx packet size. This option permits to change the packet size between each message without restarting the Nym client.

Number of messages The number of messages for the client to send.

Gateway identifier The gateway identity key, as listed in the Nym Network Explorer[1].

Client port The port on which the Nym client is started. The Controller keeps a minimal number of ports used, starting from a base port.

To measure latencies without introducing significant modifications to the Nym client, we pass the necessary parameters wrapped inside the payload data. The Nym client then reads these parameters when processing the message and adjusts its configuration accordingly, enabling the measurement of latencies while maintaining compatibility and minimizing alterations to the client's existing structure.

[1]List of gateways: ttps://explorer.nymtech.net/network-components/gateways

4.1.2 Data collection

With every execution of the Controller script, a new dataset is generated and stored within a customizable directory. This directory is organized into a new folder, named based on the current time, to ensure uniqueness. To facilitate convenient storage and processing of the experiment results, a dedicated dataset hierarchy has been established. It is designed to maintain a consistent structure for all experiments, making them easily manageable and conducive to further analysis. The Controller divides the work into multiple measurements, each of them counting multiple messages later split into multiple packets, as shown in figure 4.2. A given packet within a dataset is then identified by the tuple $(measurement_{id}, message_{id}, packet_{id})$.

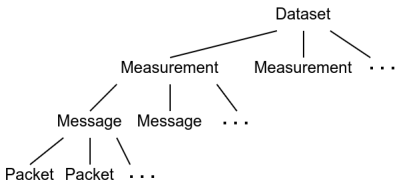

Figure 4.2: Dataset hierarchy

A dataset is made of the following files :

- *infos.json* stores global information about the dataset itself. Currently, it contains only the gateway ID but can be expanded in the future.
- *messages.csv* is a file created by the Python script that logs all the messages-related data such as the sending and receiving time, Sphinx packet size, and message length.
- *packets.csv* is a file created by the Nym client that logs packets before they are pushed into the sending queue.
- *packet_sent.csv* and *packets_received.csv* are two files created by the Nym client respectively when a packet is selected to be sent by the Poisson process and when a packet reaches the receiving queue.
- *retransmissions.csv* is a file created by the Nym client that logs all the packets retransmitted.

In Table 4.1, we list the fields logged during messages and packet lifetime.

4.2 Latencies in the Nym network

4.2.1 Setup and Nym client modifications

All measurements were made using a Nym native client (using WebSockets) in version 1.1.10. All the experiments were run on a Linode [2] machine in

[2]Linode: https://www.linode.com/

Name [Unit]	Description
CLIENT_ID	ID of the client sending the packet
MEASUREMENT_ID	ID of the measurement the packet belongs to
MESSAGE_ID	ID of the message the packet belongs to
FRAGMENT_ID	Unique ID per packet assigned internally by the Nym client, used to join files together
PACKET_ID	ID of the packet
LENGTH [bytes]	Length of the message
WEBSOCKET_SEND_TIME [ns]	When the message was sent by the python script
WEBSOCKET_RECEIVED_TIME [ns]	When the message was received by the python script
SPHINX_SIZE	Sphinx packet size used for the message, with values in [0,8,16,32]
EXPECTED_RTT_DELAY [ns]	Sum of the packet delays and its ack delays along the full round-trip
PROCESS_START_TIME [ns]	When the Nym client receives the message from the python script
PROCESS_END_TIME [ns]	When the Nym client has split the message and created the packets
HOP_X	IP address of the mixnode number X along the path, for X in [1,2,3], in base58
GATEWAY	IP address of the gateway used to send the packet, in base58
HOP_X_MIX_DELAY [ns]	Packet mix delay at the mixnode number X along the path, for X in [1,2,3]
PACKET_SENT_TIME [ns]	When the packet was picked by the Poisson process to be sent through the network
PACKET_RECEIVED_TIME [ns]	When the receiving Nym client received the packet
POISSON_DELAY [ns]	Delay between the sending event of the current packet and the next one

Table 4.1: Logged fields in datasets

the United Kingdom running Ubuntu 22.04.2 LTS with a processor AMD EPYC 7713 at 2GHz, 4 cores and 8 GB of memory. The measured upload bandwidth of our machine was 1500 Mbit/s and the download was 2400 Mbit/s. The number of parallel Nym clients running was kept below 15, so that it does not overload the CPUs at any time. However, we had to keep that number below 10 when processing Sphinx packet larger than 2048 bytes because these are requiring more processing power. We also reduced the number of clients while using gateways with limited bandwidth.

All the experiments in this work were made on the Nym mainnet, using the default mixnode selection if not stated otherwise i.e. only active nodes[3] running a compatible version of the Nym mixnode software were selected. The client is compiled in release mode, which is of significant importance. Debug mode, on the other hand, results in a considerably slower and inconsistent creation of Sphinx packets, rendering the results unusable. Therefore, utilizing the release mode ensures optimal performance and reliable outcomes for analysis, and reflects a real-world deployment of the Nym client.

Unless otherwise stated, the Nym clients ran the default configuration options, including the default Poisson and mix delays. The default sending interval of the Poisson process is 20 ms on average and the default mix delay for all the hops is 50 ms on average, so 150 ms for a three hops path. We chose

[3]The list of active mixnodes used by the Nym client to build paths can be found on https://explorer.nymtech.net/network-components/mixnodes/active and https://validator.nymtech.net/api/v1/mixnodes

a gateway in Germany among the ones available at that time and tested that it had a stable connection with an RTT between 50 and 80 milliseconds. We measured the RTT by sending Sphinx packets with an empty path to the gateway, making it send the packets back without forwarding them through the mixnet. This gateway stayed the same throughout the whole experiment presented in this chapter.

We made the following modifications to the client, trying to keep them as small as possible to preserve the original client's behaviour.

- The following parameters were added to the client command line
  - *log_folder*: Base path where the log files are written, enabled the python script instrumenting the client to change folder at each run
  - *client_log_id*: Numerical ID of the client, to differentiate packets send and received by clients
  - *packet_size*: The size of extended Sphinx packets. Can be a value in [8, 16, 32] to use a specific extended Sphinx size. Any other value will result in the default packet size.
- The packets and messages are logged at the following times during their processing.
  - When the message is initially sent by the Python script
  - When a message is received by the Nym client, just before the client starts splitting the message and creating the Sphinx packets
  - When a message has been split and all the corresponding packets are added to the sending queue
  - When a packet in the sending queue is chosen by the Poisson process, just before the packet is sent through the network
  - When a packet arrived in the receiving queue of the receiving Nym client
  - When a packet times out due to a missing acknowledgement and the packet is retransmitted
  - When a message is finally received back by the Python script
- To maintain a constant sending rate and avoid introducing additional delays, the client's throttling mechanism for the sending rate has been disabled. This ensures a consistent rate of data transmission without artificial interruptions. It was ensured that the measurements did not overload the gateways, guaranteeing their smooth operation during the experimentation process.

4.2.2 Message timing analysis

In this section, our measurements were made using the same sending and receiving gateway located in Germany. For each packet size and message size, we sent 250 messages further split into packets depending on the message size. A random path was selected, according to the Nym client default behaviour, for each packet. This experiment was repeated multiple times with the same results.

Measurements across epochs

Since the Nym mixnet topology is rotated every hour (*epoch*), the measurement in this section spans across 11 epochs, without significant changes observed between them. During all our experiments, most of the tests were conducted across multiple epochs, and no substantial differences were observed. The median round trip time for packets remained relatively consistent across epochs, with the maximum difference measured as less than 20 ms. The round trip time is primarily influenced by rare outlier paths that encounter substantial delays. Upon analysing these paths, it was discovered that they contained nodes that exhibited poor behaviour, either by dropping traffic or becoming overwhelmed by its volume. Fortunately, with the implementation of the updated routing score, these problematic nodes were mostly eliminated from the active node set in subsequent epochs. More details are given in section 4.2.3.

Notation

Throughout the remainder of this document, we distinguish between two terms: a *message* and a *(Sphinx) packet*. A message refers to a specific amount of data sent from the Python script to the Nym client, this client in turn sends the message through the network. On the other hand, a (Sphinx) packet is a network packet utilized by the Nym client to transmit messages across the network. If a message exceeds the payload capacity of a single packet, it is divided into multiple packets, each of which is transmitted separately through the network.

We establish the following definitions regarding messages in the Nym network:

- The *End-to-end latency* is the total amount of time elapsed from the moment the Python sends a message until it received it back. This corresponds to the observed latency from the end-users' point of view.

- The *WebSocket delay* is the total amount of time for the Python script to send the message to the Nym client, and for the Nym client to transfer it back to the Python script. We added these two values as

they are delays induced by the WebSocket communication, which we do not analyse further in this work.

- The *Processing delay* is the amount of time for the Nym client to split the message into Sphinx packets. This operation includes the cryptographic operations required to create the Sphinx packets.

- The *Transmission delay* is the amount of time for a message to be put into the Nym network i.e. for the Poisson process to pick all the message's fragments and send them.

- The *Propagation delay* is the amount of time for a message to travel from one end to the other end of the network, including packet mixing delays at each hop sampled from an exponential distribution.

The delays described above are represented in figure 4.3. The WebSocket and processing delay are induced by the processing power of the sending machine, whereas the propagation delay is the result of the network speed and capacity. The transmission delay, as we will show, is capped by the Poisson process.

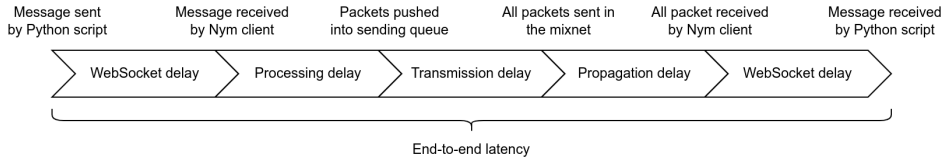

Figure 4.3: Message delays

End-to-end latency

The end-to-end latency encompasses the cumulative delays mentioned earlier. Figure 4.4 illustrates the progression of the end-to-end latency given varying message sizes. Additionally, we investigate how the size of the Sphinx packet format impacts this latency. We observe that latency starts to grow as soon as messages require multiple packets. For example, the blue line (Sphinx packets of 2048 bytes) starts to increase when messages are bigger than 2 kB. We can theorize four reasons for the increase :

1. More packets need to be created, increasing the processing delay

2. Packets need to be picked by the Poisson process before being sent. On average one packet is picked every 20 ms, so more packets increase the transmission delay.

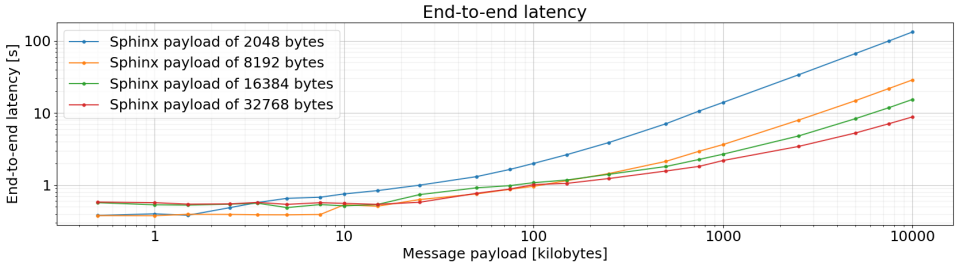

Figure 4.4: End-to-end latency by Sphinx packet size and payload size.

3. To reconstruct a message, all the packets need to be received. If one packet experiences a higher delay in the network, the whole message is delayed. So the more packets, the higher the risk that one of them delays the whole message, increasing the propagation delay.

4. Bigger messages are a bigger amount of data to transmit between the Python script and the Nym client, inducing a higher WebSocket delay.

In the following sections, we analyse each of the delays listed above to understand their impact on the end-to-end latency.

Processing delay

The processing delay is the time the Nym client takes to split the message into packets and perform all the cryptographic operations required by the Sphinx [12] format. The evolution of the processing delay in regard to the message size and Sphinx packet size is shown in figure 4.5. We notice that

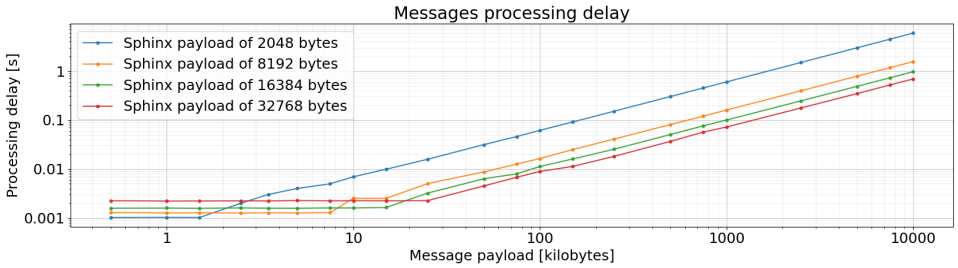

Figure 4.5: Processing delay by message size and Sphinx packet size

the processing delay grows linearly with the number of packets. Because of the efficient implementation, the processing times are negligible compared to the full end-to-end latency. For small packet sizes, the time is less than 1 percent, and for the largest message size and packet size, the processing time takes less than 10 percent of the end-to-end latency.

Transmission delay

The transmission delay is the time during which packets are waiting in the sending queue before being picked by the Poisson process. This process picks a new message from the sending queue every 20 ms on average (sampled from an exponential distribution). Figure 4.6 shows the result from our experiments.

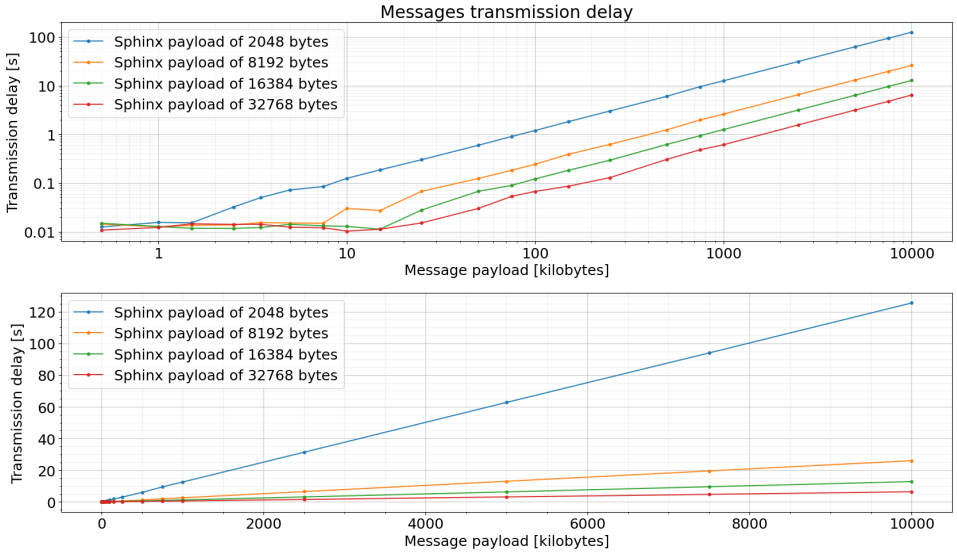

Figure 4.6: Transmission delay by message size and Sphinx packet size. Top graph with a logarithmic scale, bottom graph with a linear scale.

The transmission delay grows linearly with the number of packets. When the message leads to only one packet, the transmission delay is less than 20 ms on average, because the Poisson process will pick this packet at its next event, for which the timer has already started. When the message is split into two packets, the transmission delay is slightly higher than 20 ms. This is due to the first packet being picked in the same way as stated above, and the next packet being picked on average 20 ms later.

With p the packet size, o the payload overhead caused by the acknowledgement and the Diffie-Hellman key included in the payload, and r the sending rate of the Poisson process, we can compute the theoretical transmission delay for a message of size m as follows: $transmission_{delay} = \lceil \frac{m}{p-o} \rceil \cdot r$. In practice, the overhead is of fixed size $o = 437$ bytes, and the Poisson process events follow an exponential distribution of parameter $\lambda = \frac{1}{20}$ so $r = \frac{1}{\lambda} = 20ms = 0.02$ seconds. We observe that the experimental measures in figure 4.6 closely match the theoretical values.

Overall, the transmission delay takes a larger proportion of the end-to-end latency the bigger the message. Figure 4.10 shows that for standard

Sphinx packets, the transmission time becomes dominant for messages larger than 75 kB. In these cases, the bottleneck is the Poisson process that, by design, fixes the sending rate of the client.

The Poisson process: This process is managing the packet sending rate so that, by its memoryless property, packet sending times are not linked to each other. We measured the distribution of the Poisson process and found, as expected, that it follows an exponential distribution of parameter $\lambda = 1/20$ (4.7). 20 milliseconds being the default average sending rate of the Poisson process used by the Nym client.

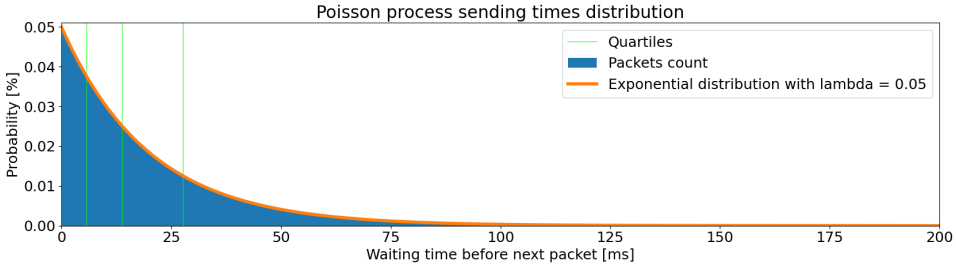

Figure 4.7: Distribution of the Poisson process sending times. Its distribution follows an exponential distribution of parameter $\lambda = 1/20$.

Propagation delay

The propagation delay is the time for the message to go through the network, once the sending client has sent its last packet. We can observe that when the

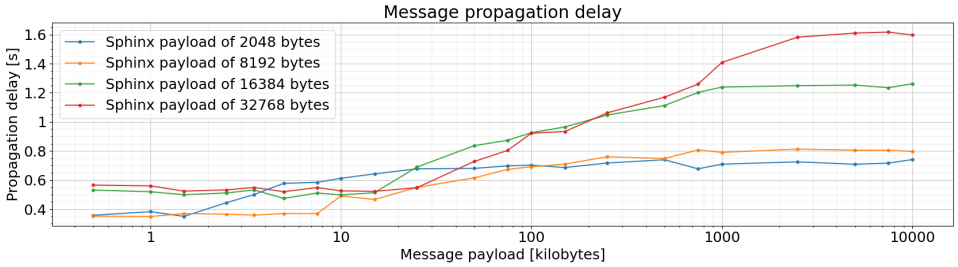

Figure 4.8: Propagation delay by message size and Sphinx packet size

message leads to only one packet i.e. when the packet size is bigger or equal to the message size, the propagation delay stays stable. When a message is divided into multiple packets, the propagation delay tends to increase. This is because, in order to reconstruct the original message, all packets must reach the receiving client, resulting in the propagation time being defined by the arrival of the last packet. If a particular packet encounters significant

delays within the network, it can delay the entire message, and this risk intensifies with the number of packets.

However, it is important to note that this effect does not lead to a linear increase in the propagation delay. Each line depicted in Figure 4.8 eventually reaches a plateau as the number of packets becomes substantial. For a packet to significantly delay the message, it must be among the last packets received. As transmission times gradually lengthen for larger messages, it becomes increasingly unlikely for the packets sent earlier to be the ones causing significant delays. Consequently, only the last few packets are likely to experience notable delays that might impact the overall propagation delay.

Figure 4.10 shows that for messages smaller than 75 kB, the propagation delay is the dominating cost in terms of latency. We study this delay in more detail in section 4.2.4.

WebSocket delay

The WebSocket delay is the time taken by the WebSocket protocol to send the message from the Python script to the Nym client before sending it through the mixnet, and upon reception by the receiving Nym client to transfer it back to the Python script.

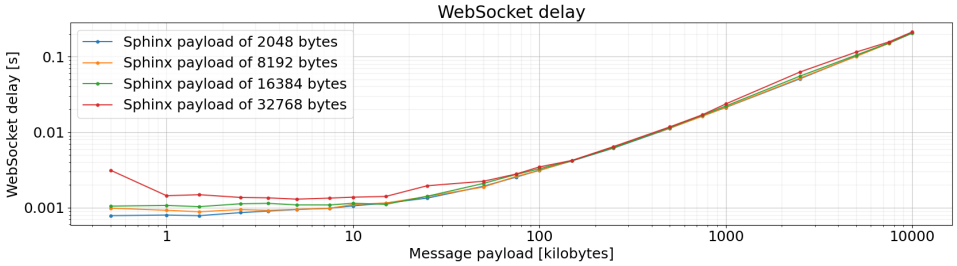

Figure 4.9: WebSocket delay by message size and Sphinx packet size

We computed the WebSocket delay to ensure that it stays negligible in regard to the end-to-end latency measurement. As seen in figure 4.9 it grows with the message size. Theoretically, the WebSocket delay should not be impacted by the packet size used, as the delay occurs when transferring the message between the Python script and the Nym client, before and after the message was split into Sphinx packets. However, in figure 4.9, the coloured lines are not perfectly stacked. This is due to our timing measurement, logging the timestamp when the Nym client receives packets. There are a few operations between the reception of the last packet and the message transfer to the Nym client that are also included in this graph such as the message reconstitution, which explains why the lines are not combined. Nevertheless, as users may choose any client (WebSocket, SOCKS5, WebAssembly) to connect to the mixnet, the main goal of quantifying the WebSocket delay is

to ensure it does not interfere with the other latencies in the network. We confirm that it stays mainly negligible, taking less than 5 % of the end-to-end latency in any case, as presented in the next section.

Overall view and goodput

The latencies presented in this chapter can be summarized in figure 4.10. This figure shows the separation of latencies for standard Sphinx packets (2048 bytes). Similar figures for other Sphinx sizes can be found in appendix A.

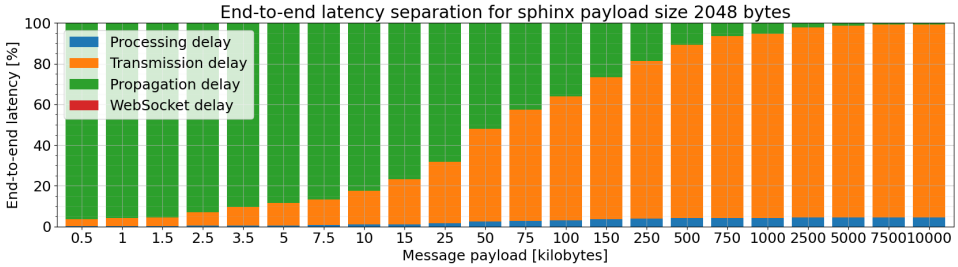

Figure 4.10: Message E2E latency split into its delays for Sphinx packets of size 2048 bytes

Finally, we can divide the message size by the end-to-end latency to get the network's goodput. Figure 4.11 shows the goodput for various message and packet sizes. We note that for small messages (up to 10 kB), Sphinx packets of size 8192 bytes are preferable. For bigger messages, Sphinx packets of size 32768 bytes achieve the best goodput. This is an interesting result, as using 2048 and 16846 bytes Sphinx packets does not seem to bring any advantage, besides consuming less bandwidth. This observation was made in all our experiments, also using different gateways at different times.

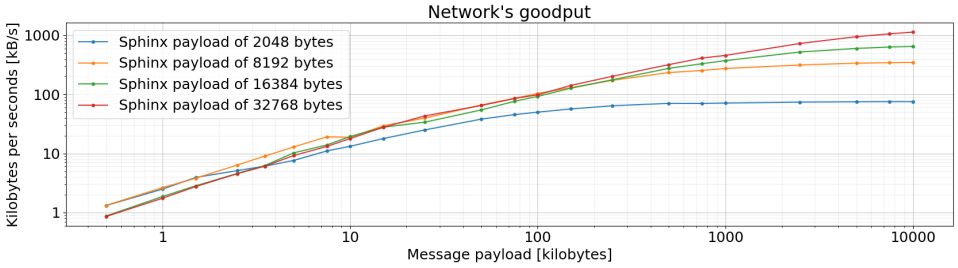

Figure 4.11: Network goodput by packet and message sizes

4.2.3 Retransmissions

When a packet sent by the client times out, the client retransmits that packet. To do so, the client retrieves the packet's data stored in memory and builds a new packet with it (the path, mix delays, and other parameters are computed again as it would be done for a fresh new packet). This ensures the retransmissions are identical to any other packet travelling in the network. In particular, an adversary looking at the connection is unable to differentiate retransmissions from other packets. The client puts the retransmitted packets in a dedicated sending lane, offering the possibility to handle them differently. Currently, the client prioritizes the lanes with few messages (the *small* lanes), and then the lanes with the higher number of messages sent already (the *old* lanes). Since retransmissions happen sporadically, their lane is generally small and is prioritized.

Retransmissions across the network may occur in different scenarios.

- When a packet sent is lost during its propagation. This leads to packet loss.
- When an acknowledgement is lost during its propagation back to the sending client. The initial packet still reached its destination, but the sending client will retransmit because it will not receive the confirmation.
- When a packet takes more time to reach its destination than the timeout, the client retransmits the packet anyway.

To differentiate between these three options, we isolate all the messages logged as retransmitted and proceeded as follows :

1. We logged for each packet the sum of all the mix delays for the packet and its acknowledgement.
2. We compute the timeout of each packet, following the formula implemented by the Nym client. The formula is $\texttt{timeout} = \texttt{totalMixDelay} \cdot \texttt{ackWaitMultiplier} + \texttt{ackWaitAddition}$ where, by default, $\texttt{ackWaitMultiplier} = 1.5$, $\texttt{ackWaitAddition} = 1500$ ms and `totalMixDelay` is the sum of all the mix delays for the packet and its acknowledgement.
3. If the packet is received before its retransmission timeout triggers, we are sure that the retransmission occurs because of a lost or delayed acknowledgement. We consider these cases as *Lost acknowledgement*.
4. If it is not a lost acknowledgement, and the number of times the packet was sent is higher than the number of times the packet was received, then, at least once, the packet got lost. We consider these cases as *Lost packet*.

5. Finally, if it is not a lost acknowledgement and the number of packets sent is equal to the number of packets received, then the packet's initial transmission experienced a delay that made the packet timeout but all the packet's transmissions finally reached their destination. We consider these cases as *Delayed packet*.

The distributions of the aforementioned retransmissions can be found in figure 4.12. We did not take into account exceptional cases such as delayed

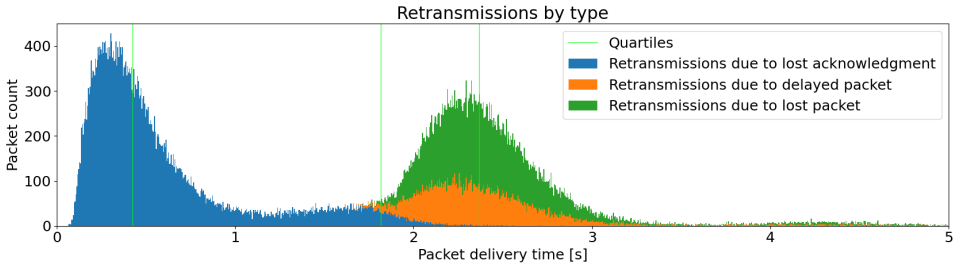

Figure 4.12: Types of retransmissions by delivery time

packets, for which the retransmission experiences a lost acknowledgement, triggering a second retransmission. Such cases are extremely rare and do not influence our analysis.

It is interesting to note that the *lost acknowledgement* packets (blue curve) have a bimodal distribution. We believe this is a consequence of an underlying retransmission mechanism with a shorter timeout than the one for Sphinx packets. Its presence and proportion vary with experiments, so we deduced it is linked to the packets' path and the subjacent network layer congestion. We can also notice a second, minuscule, distribution in green between 4 and 5 seconds. These are the messages lost twice in a row that required two retransmissions.

In all our experiments, we found a consistent retransmission rate between 1% to 3%. Among all the packets, between 0.4% to 0.7% were lost packet retransmissions. We concluded that retransmissions do not significantly influence network delays. Bigger packets lead to a higher rate of retransmissions, as shown in table 4.2. The numbers in this table average over 3 experiments spreading across more than 8 epochs each, and have a variance smaller than 0.5. However, as discussed next, this heavily depends on the number of disruptive nodes.

During all our measurements, retransmissions originated primarily from a small set of nodes having a high drop rate. For each mixnodes, we measured the number of retransmissions along the paths containing these mixnodes. For each layer, around 3 nodes are showing a retransmission rate of over 10%, which lets us conclude that those nodes are dropping a significant amount of packets. Over the 240 active nodes, this represents about

Sphinx packet size [bytes]	Retransmissions [%]
2048	1.75 %
8192	1.79 %
16384	2.97 %
32768	3.72 %

Table 4.2: Packet retransmission rate per packet size

9 nodes at each epoch behaving "badly". However, as the number of disruptive nodes is small, the dropping rate varies heavily between epochs. Moreover, as stated in section 3.2.2, nodes failing to process traffic see their probability of being picked in the active set decreased. This makes the set of unsound nodes change at each epoch and challenging to study in more detail.

4.2.4 Packet timing analysis

Previously, we looked at the latencies regarding the messages in the network. In this section, we focus on the packet's latency and its delays.

Definitions

We define the following timings regarding packets in the Nym network. They are illustrated in figure 4.13.

- The *Packet delivery time* is the time between a packet being sent by the sending client and its arrival at the receiving client.
- The *Mix delays* are the amounts of time a packet is delayed at each mixnode along its path. Note that these delays are drawn upon the packet creation in the sending Nym client.
- The *Processing delay* is the amount of time spent in each mixnode to compute the cryptographic operations required by the Sphinx packet format. Note that the gateways do not perform such operations and are only forwarding the packet.
- The *packet Propagation delay* is the time for a packet to go from one hop to the next one in the path.
- The *Gateway delay* is the time for a packet to go from the sending client to its gateway, added to the time for a packet to go from the receiving gateway to its destination client.

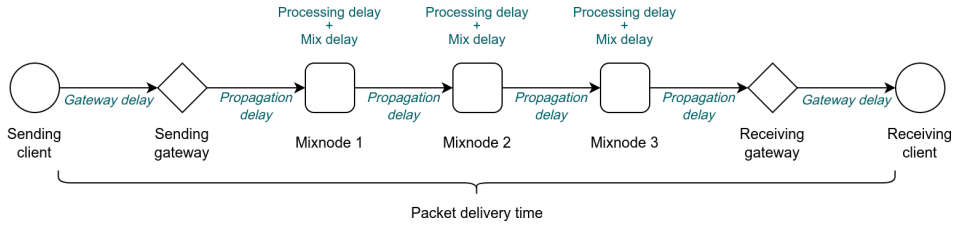

Figure 4.13: Packet timings

Packet delivery time

The packet delivery time is the total amount of time a packet takes to travel from the sending client to the receiving client. Figure 4.14 confirms the intuition that packet delivery time does not depend on the message size, as the Nym client does not differentiate packets based on their parent message. Packets of size 16 kB and 32 kB take more time on average to be delivered, this is a behaviour observed on multiple measurements made on different gateways. We think it is the effect of an underlying packet-splitting mechanism that handles distinctively 2/8 kB packets and 16/32 kB packets.

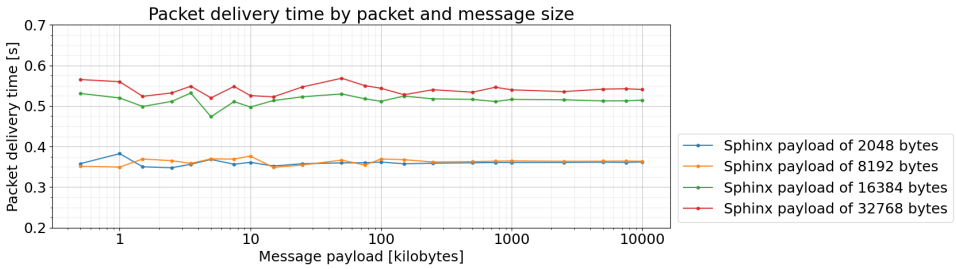

Figure 4.14: Median packet delivery time

Mix delays

We measured the average mix delays at each hop of the network. The default mix delay is set to 50 ms, so we expected the mix delays to follow an exponential distribution of parameter $\lambda = 1/50$. Figure 4.15 shows that it is exactly the experimental result. For a path comprising n nodes, the average mixing delay a packet will experience is $n \cdot 50$ ms, so 150 ms for a three-hop path.

Processing delay

The processing delay is the amount of time a mixnode requires to decipher the packet, extract the routing information, and perform the cryptographic

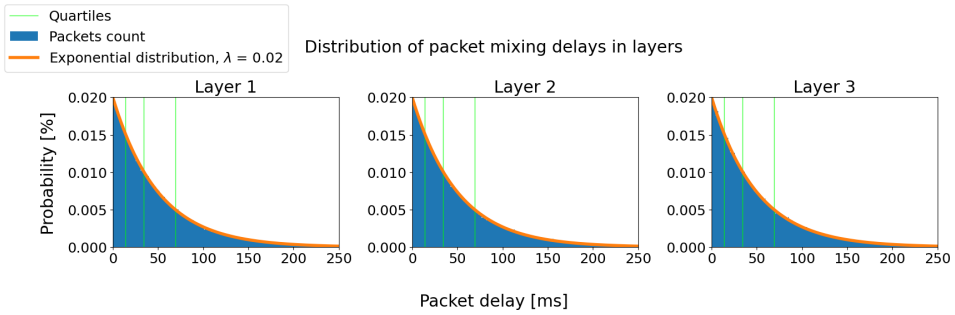

Figure 4.15: Distribution of the mix delays at each hop. The distributions follow an exponential law of parameter $\lambda = 1/50$.

operations so that the packet is bitwise unlinkable with its input version.

Using the Sphinx packet format, these operations, called *unwrapping*, are measured to be under 1/6 millisecond[4]. This makes the processing delays of the mixnodes along the paths negligible in regard to other delays.

Propagation delay

The packet propagation delay is the time the packet takes to travel through the wire from a mixnode to the next one. Note that this does not imply a direct connection between mixnodes, the Nym network is an overlay network working on the standard internet infrastructure. When travelling, a packet can (and most of the time will) go through multiple traditional IP routers, inducing more delay than a direct link.

The propagation delay is mainly correlated to the distance a packet needs to travel. We observe this phenomenon in figure 4.16. To compute the distances between nodes, we used the IP geolocation service https://ip-api.com/ to retrieve the assumed latitude and longitudes of nodes based on their IPs.

Figure 4.16 shows the correlation between the packet delivery time and the total distance the packet travelled (including the distance between the gateway and the first mixnode, and the distance between the last mixnode and the gateway). As the data was extremely noisy, we grouped the points by buckets of 50 km and took the median of each bucket. The information is travelling in the mixnet at about 23% of the speed of light.

Gateway delay

The gateway delay depends on both the sending gateway and the receiving one. The user can manually choose which sending gateway to use but doesn't have any influence on which gateway the receiver is bound to. Users

[4]Documentation of Sphinx implemented in Rust: https://github.com/nymtech/sphinx

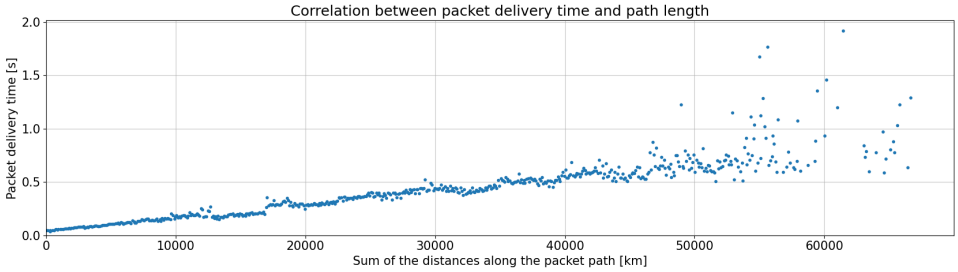

Figure 4.16: Correlation between the packet delivery time and its path length.

may choose their gateway based on multiple parameters such as location, bandwidth, popularity, or security rating. To reduce the gateway delay, the client should bind itself to a proximate gateway, in order to reduce the packets' travelling time.

However, as the gateway is mostly a user choice motivated by various parameters, we chose not to study it in depth. The nature of latency encountered by this choice is the same as the packet propagation delay mentioned in the previous section.

In the final implementation of the mixnet, gateways will additionally ensure that clients have the right to use the network by checking their credentials. This will induce additional overhead as gateways will also need to measure the amount of data users are transmitting in order to request additional credentials when needed.

4.2.5 Packet outliers

Following our analysis in the previous chapters, we were also interested in understanding the latencies of outliers and their origin. Outliers can be an issue during message transmission because, as reconstructing a message requires all the packets to reach their destination, having one outlier at the end of the transmission may delay the whole message by several seconds.

As we saw that the processing delay was mostly negligible, we wanted to understand if outliers had an over-average mixing delay or propagation delay. To do so, we considered as outliers the top 1% of packets with the highest delivery time, we then separately plotted their mixing and propagation delay. The results are in figure 4.17. We observed that for outliers, the mixing delays are similar to the rest of the packets. However, outliers are experiencing higher delivery times because of their propagation time. This is mainly due to the path selection algorithm. As the path is selected at random among all the mixnodes, it may happen that a path is especially long or congested. For example, a path where each link is crossing an ocean experiences a considerable propagation delay.

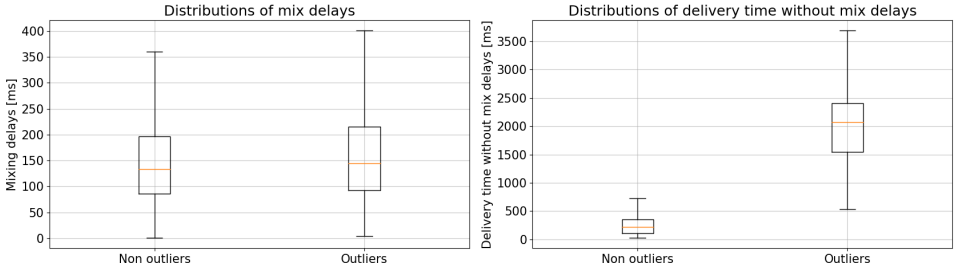

Figure 4.17: Comparison of delays between outliers and non-outliers

4.2.6 Summary

In this chapter, we proposed a methodology to analyse the network latencies by first describing the scripts we used, their interactions, and the data architecture. We then split the message end-to-end latency into four delays that we measured and discussed. The transmission and the propagation delays were the two biggest shares of the overall latency. The former is a consequence of the Poisson process maintaining the sending rate to a fixed value, the latter because of the time packets take to travel through the mixnet. We then further analysed the packets' latencies and found that the largest delays were the mixing times, of 50 ms on average per hop, and the packet propagation time, which strongly correlates with the distance travelled. Lastly, we discovered that outliers are the result of the path selection algorithm, selecting mixnodes at random, which sometimes generates excessively lengthy paths.
These observations motivated the next chapter, dedicated to improving the path selection algorithm and reducing the packets' propagation time.

Chapter 5

Reducing communication latencies

The empirical measurements presented in the previous chapter offer valuable insights into how various processes, involved in message sending, affect the latency overhead within the Nym network. When a message is transmitted through the network, it encounters delays at each hop along its path. These delays can arise from factors such as network congestion, routing inefficiencies, or physical distance. Since a single Sphinx packet must navigate through multiple hops, the cumulative impact of these delays can result in high communication latency.

Furthermore, the inherent nature of the Nym network architecture, which involves anonymization and mixing of traffic, adds an additional layer of complexity. The intricate routing mechanisms employed by the network introduce extra computational overhead and contribute to the overall latency experienced during message transmission.

Through our observations, we have identified that the distance between the selected nodes plays a pivotal role in determining the overall latency experienced, which we explore further in this chapter.

5.1 Impact of nodes geo-location

In Figure 5.1, we present examples of how impactful the inter-node distances, in which mixnodes locations play a major role, are on the packet latency. We also noticed that filtering nodes by hardware specifications do not lead to significant differences, because the computational time of cryptographic processing of the Sphinx packets is orders of magnitude lower than mixing or transmission delays. However, hardware plays an important role in the reliability of the network.

The results from figures 4.17 and 5.1 pointed out that a significant part of the message propagation delay was devoted to packets travelling through

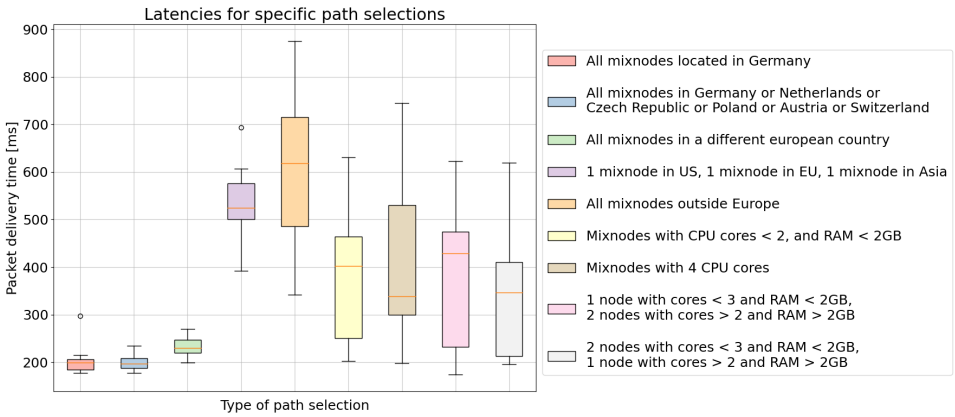

Figure 5.1: Packet latencies with specific path selection algorithms. Sending and receiving gateways were in Germany

the internet wires, and, moreover, that a random path selection induces high travelling distance that can be optimized. This leads to our path selection algorithm proposal, which strives to reduce the packets' travelling distance without compromising too much on privacy.

5.2 Path selection algorithm

The propagation delay, as seen in section 4.2.4, induces a high latency in the network. The random path selection occasionally causes itineraries to go back and forth around the globe.

Many path selection algorithms for anonymous communications have been proposed in the past, especially to optimize the path selection for Tor. Most of them revolve around two core ideas. The first is to restrict the hop selection by discarding hops that are too far apart or cause a too high latency [22][2][36][23]. The second is to compute multiple paths, and discard the slower ones by following a metric such as the latency between hops [3][23]. Some attempts were also made for mixnets such as [25] which has a system of messaging to extract the shortest path in ad-hoc networks. In [16] a reputation system is used to improve the performance of mail delivery networks using mixing servers, as designed by Chaum [7].

To reduce both the average latency of the network and avoid these outliers paths, we propose a path selection algorithm for mixnets that tries to minimize the distance between nodes while keeping some amount of privacy. In order to reduce the propagation delay, the algorithm needs to select nodes that are closer to each other, in particular, we want to minimize the number of significantly large jumps such as ocean crossing. Paths in the Nym network are made of 5 components: the sending gateway, the receiving gateway,

and one mixnode per layer.

As the receiving gateway is the choice of the receiver who binds to it, the sender does not have any influence on it, hence the algorithm must operate with a fixed known gateway. On the contrary, the client may select any sending gateway. Currently, users have the option to specify a gateway in the configuration file, otherwise, the client chooses a gateway at random. This choice has an impact on the network latency, as choosing a gateway far from the user induces a delay in all the packets sent and received by the client. However, as users are not expected to switch gateway frequently (this requires cryptographic keys derivation and a temporary pseudonym choice), this parameter should remain in control of the user. Moreover, gateways are intended to serve as a censorship countermeasure [9], so people may want to choose some gateways that are far from them on purpose. Finally, gateways are the only actors aware of the sender's true IP address, which reveals information. Users may want to choose gateways they trust, even if these are not the ones providing minimal latency. For all those reasons, we chose not to consider gateways for the path selection. Our algorithm assumes one fixed sending gateway transferring packets to one fixed receiving gateway.

Currently, for each packet, the mixnodes are selected at random among the active mixnodes of the corresponding layer. While this keeps privacy at its best, as stated, it is far from being optimal in terms of latency. To improve the propagation delay of the Nym network, we propose algorithm 1. An example run of the algorithm is presented in figure 5.2.

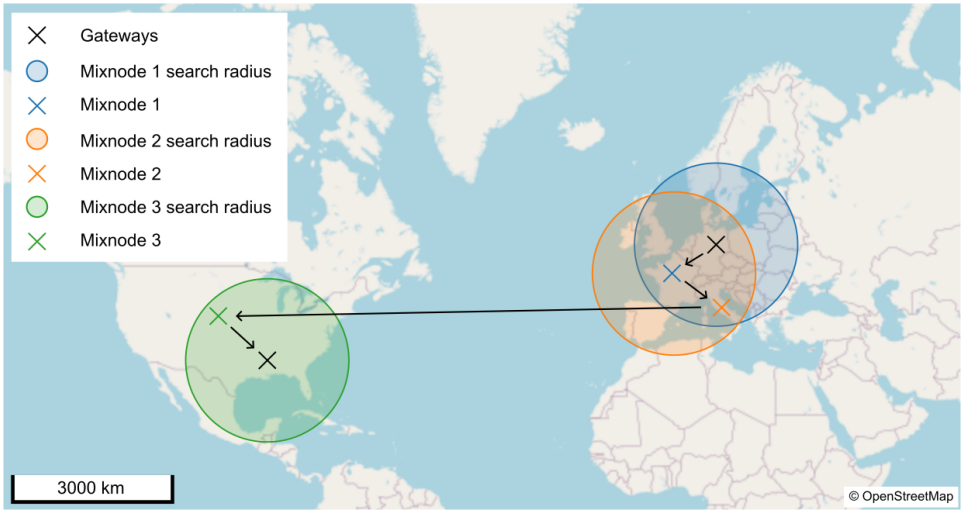

Figure 5.2: Example run of the algorithm width $d = 3000$ km, with a sending gateway in Germany and a receiving gateway in the US. The first mixnode is chosen within a radius d of the sending gateway, the second mixnode is within a radius d of the first mixnode, and finally, the third mixnode is within a radius d of the receiving gateway.

Algorithm 1 Optimized path selection

Require:

$\boldsymbol{d}$, the maximal search distance for mixnodes
$\boldsymbol{n_{min}}$, the minimal number of mixnodes candidates
$\boldsymbol{M_i = (m_1, m_2, ..., m_k)}$, the list of actives mixnodes in layer $\boldsymbol{i}$
$\boldsymbol{G_s}$, the sending gateway
$\boldsymbol{G_r}$, the receiving gateway
$\boldsymbol{dist(x, y)}$, returns the distance between nodes $\boldsymbol{x}$ and $\boldsymbol{y}$
$\boldsymbol{pickOneIn(S)}$, pick one element at random in the set $\boldsymbol{S}$

procedure computePath
    $mixnode_1 \leftarrow getCloseMixnodes(G_s, 1, d)$
    $mixnode_2 \leftarrow getCloseMixnodes(mixnode_1, 2, d)$
    $mixnode_3 \leftarrow getCloseMixnodes(G_r, 3, d)$
    return $(mixnode_1, mixnode_2, mixnode_3)$
end procedure

procedure getCloseMixnodes($origin, layer, currentDistance$)
    $result \leftarrow \varnothing$
    for all m in M_{layer} **do**
        if $dist(origin, m) \leq currentDistance$ **then**
            $result \leftarrow result \cup m$
        end if
    end for
    // If not enough mixnodes nearby, try doubling the distance
    if $|result| < n_{min}$ **then**
        return $getCloseMixnodes(origin, layer, 2 \cdot currentDistance)$
    end if
    return $pickOneIn(result)$
end procedure

In the next sections, we justify the algorithm's design while simultaneously exploring its variations and tradeoffs.

5.2.1 The unrestricted distance

Our algorithm's core idea is to restrict at each hop the set of nodes within a range d where the propagation latency is acceptable. For a reasonable distance d, it implies no costly jumps (1) between the sending gateway and the first node, (2) between the first node and the second node, and (3) between the third node and the receiving gateway. Any ocean crossing or major jump takes place between the second and the third node, restricting such jumps to at most one per path. This major jump is labelled as the "*unrestricted distance*" in figure 5.3

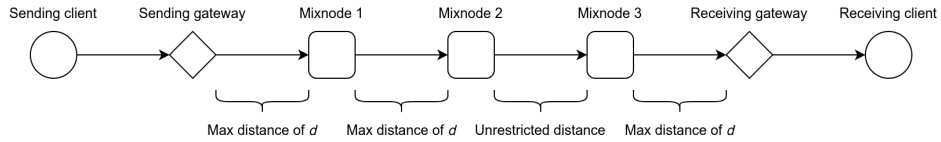

Figure 5.3: Distance restriction within our path selection algorithm

Considering the position of the unrestricted distance in the path, we observe the following :

- Placing it between the sending gateway and the first node allows a passive adversary to know where the traffic from a given user is heading within a maximum distance $3d$, just by looking at the first mixnode's location.
- Placing it between the first and second nodes results, for an adversary controlling the first node, in knowing both (1) the sender and (2) where the packets are heading within a distance $2d$.
- Placing it between the second and the third node results, for an adversary controlling the third node, in knowing both (1) the receiver and (2) where the packets are coming from within a distance $2d$.
- Placing it between the third mixnode and the receiving gateway allows a passive adversary to know where the traffic to a given user is coming from within a distance $3d$, just by looking at the third mixnode's location.

The third option in this list appears to be the least compromising in terms of privacy.

To see if the location of the unrestricted distance impacts the latency, we tested all the possibilities. They are presented in graphs 5.4 and 5.5. The

labels "*Jump in X – Y*" means the unrestricted distance is placed between nodes X and Y, with G referring to the gateway. We experimented with both having a receiving gateway far from the sending one (path from Germany to the United States in figure 5.4) and near the sending one (path from Germany to Germany 5.5). We added as well three special path selection strategies to each plot.

- The default algorithm, selecting nodes at random, denoted by "Random choice".
- An algorithm that always selects the closest node to the previous, denoted by "Closest node".
- An algorithm that does not select any mixnode, but rather makes the sending gateway directly send its message to the receiving gateway. This is equivalent to not using the mixnet at all. It is denoted by "No mixnodes".

The last two algorithms are here for comparison, it is clear that they are mediocre from a privacy perspective due to their determinism.

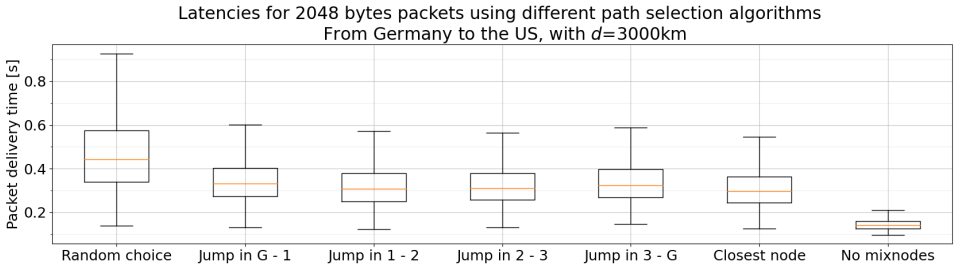

Figure 5.4: Packet delivery time with various path selection strategies, from Germany to the US

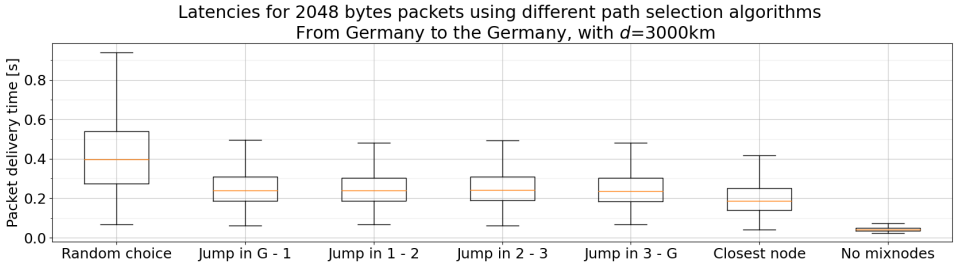

Figure 5.5: Packet delivery time with various path selection strategies, from Germany to Germany

We make the following observations :

- Even with a wide distance of 3000 km, our algorithm reduces on average the latency by about 30%. It is even more efficient when the sender and receiver are located close to each other, where the speed-up reaches 40%.

- Our algorithm reduces also the outliers, on both graphs the distributions are narrower.

- The nodes between which the *unrestricted distance* is located have almost no influence regarding the algorithm efficiency.

- Selecting a large value for d (3000 km) does not considerably penalize the latency compared to the naive "Closest node" selection. We further discuss this topic in the next section.

- The mixnodes are still substantially penalizing the network's speed, compared to the "No mixnodes" strategy. This is primarily due to the mixing delay adding 150ms latency on average.

In regard to the privacy argument mentioned above and the empirical measurements we conducted, we conclude to put the *unrestricted distance* between the second and the third mixnode along the path.

5.2.2 Parametrizing the algorithm

How to find a good value for the parameter d ? Understandably, reducing its value leads to shorter paths, increasing the network's speed but shrinking the anonymity set. This value can be seen as a "fader" between an algorithm selecting mixnodes at random and an algorithm striving for the shortest path, as seen in figures 5.4 and 5.5. Our objective is to find a good tradeoff between reducing latency and conserving privacy.

To quantify the impact of this parameter and learn how much picking a smaller value of d reduces the latency, we selected multiple sending and receiving gateways across the globe and built paths using our algorithm with an increasing distance. Figure 5.6 present the result for sending and receiving gateways in Germany, other similar graphs for all the tested paths can be found in appendix B.

By looking at the graphs, we can empirically observe that most increases in latency occur after 3000 km. Even if this varies considerably between gateways, a value of d between 2500 km and 3000 km seems to be the highest value we can consider before paying high latency costs.

To further explore some parameters, we also plotted figure 5.7, with the following algorithms that impact privacy more aggressively:

- *Random choice*: Default path selection of the Nym client, no filtering

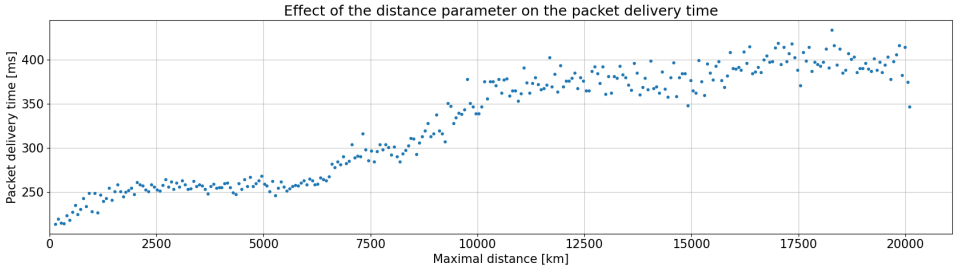

Figure 5.6: Packet delivery time with increasing node selection radius. Sending and receiving gateways both in Germany.

- *d=3000 km*: Our proposed algorithm, with a maximum distance of $d = 3000$ km
- *d=250 km*: More aggressive filtering with $d = 250$ km
- *d=250 km, 2 Nodes*: Reducing the number of mixnodes along the path to two mixnodes, with $d = 250$ km
- *d=250 km, 1 Node*: Reducing the number of mixnodes along the path to one mixnode, with $d = 250$ km. Besides the mixing of packets happening at the mixnode, this is similar to a VPN setup.
- *Closest Node*: Reducing the number of mixnodes along the path to one mixnode and taking the closest mixnode.
- *No mixnodes*: Direct connexion from the sending gateway to the receiving gateway, no mixing.

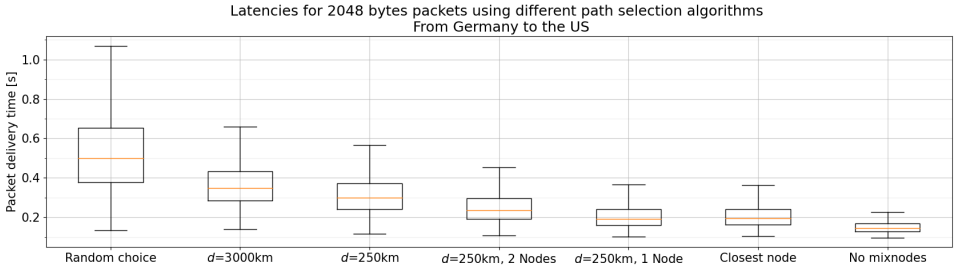

Figure 5.7: Other path selection algorithms strongly impacting privacy

We observe that dividing the value of d by 12 only improves the latency by about 50 ms. However, reducing the number of nodes along the path significantly lowers it, which is mainly the result of sparing mixing delay. For each mixnode we remove from the path, we save about 50 ms (the average mixing delay). Note that having one mixnode with $d = 250$ km does not cost more than always selecting the closest node.

5.2.3 Privacy implications and discussion

Lowering latency does not come without a privacy tradeoff. In this section, we discuss the implications our algorithm has on privacy and justify implementation choices.

First, our threat model considers a global adversary, capable of spying on any link in the network. This implies gateways are not providing any privacy, as the adversary can execute traffic correlation attacks and link back the outcoming packets to their originating user. The adversary is also considered active i.e. capable of controlling or spawning some malicious nodes.

Passive adversary: By limiting the mixnodes selection distance, we are reducing the number of nodes from which packets are coming. We must ensure that each mixnode receives traffic from enough different users to keep the anonymity to an acceptable level. With a fully random path selection, the set of users using any mixnode is the total number of users using the network. It corresponds to the maximal anonymity set that can be achieved. With our algorithm,

- For mixnodes in layer one, this set shrinks to the users bound to gateways within a distance d of the mixnode. This number is hardly quantifiable, as a gateway can be bound to any number of users, having more gateways within a mixnode range tells nothing about the number of users using that mixnode.
- For mixnodes in layer two, the same principle applies, but the set of users using those mixnodes are the users using gateways within a range of $2d$.
- For mixnodes in layer three, symmetrically to mixnodes in layer one, the set of users are all the users sending traffic to gateways within a range d.

Nevertheless, to give an intuition on how this affects privacy, we can consider the scenario where users choose their gateways uniformly at random. However, we should keep in mind that users can behave differently and that the network is not homogeneous, as discussed in section 5.2.3.
With N the total number of users and G the total number of gateways, every gateway handles $\frac{N}{G}$ users. It means that for k gateways from which a packet can originate, the anonymity set is $S = \frac{kN}{G}$.

Following the definition of anonymity degree proposed in [15], we can compute the anonymity degree of a given packet observed in the mixnet implementing our algorithm. Intuitively, an anonymity degree of 1 (the maximal value) means that a given packet has the same probability to come from any user, while an anonymity degree of 0 indicates that the sender

of a packet can be determined unambiguously. In our case, the anonymity degree is $\frac{log_2(S)}{log_2(N)} = 1 - \frac{log_2(G) - log_2(k)}{log_2(N)}$, where $\frac{log_2(G) - log_2(k)}{log_2(N)}$ can be interpreted as the anonymity cost.

For a packet potentially coming from all the gateways (which is the case when using a fully random path selection), $k = G$ and the anonymity cost is null (all the users have the same probability to have sent a given packet). On the other hand, if only one gateway is possible, then $k = 1$ and the cost reaches its maximal value of $\frac{log_2(G)}{log_2(N)}$ (only the users connected to that one gateway could have sent a given message). Figure 5.8 shows the evolution of the anonymity degree with an increasing ratio of potential gateways. We notice that even when building paths where a limited subset of gateways could have sent packets, the anonymity degree is satisfactory, for example, considering only 15% of gateways leads to an anonymity degree of 0.8.

Finally, we note that when considering numerous users (more than 5000), the number of users does not affect significantly the curve shown in figure 5.8.

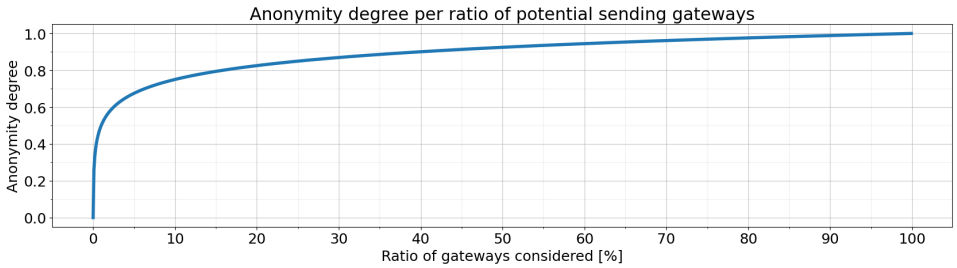

Figure 5.8: Anonymity degree per ratio of mixnodes considered for path selection, assuming 10'000 users.

Active adversary: In addition to passive observations, if the adversary is in control of some nodes in the network, the following concerns arise.

- If the adversary controls the first node in the path, by doing traffic analysis on the gateway it is able to learn which user is sending the packets. Since the first node is under control, it also learns which is the mixnode in layer two for each packet. However, this reveals nothing about the destination of packets.

- If the adversary controls the second node in the path,

  - The adversary learns the nodes in layers one and three for each packet.
  - Knowing which mixnode is in layer one reveals the sending gateway location up to d.

  - Knowing which mixnode is in layer three reveals that the receiving gateway is within a range d of this mixnode.
  - However, such an adversary does not learn anything more about the sender or receiver's identity.

- If the adversary controls the third node, by traffic analysis on the receiving gateway it is able to learn which user is receiving the packets. Since the third node is under control, it also learns which is the mixnode in layer two for each packet. This reveals the location of the sending gateway up to $2d$.

To summarize, an adversary controlling the first node (and so knowing the sender) does not learn anything about the receiver, and an adversary controlling the third node (and so knowing the receiver) infers the sender with a precision of $2d$. If the adversary controls any two mixnodes, by the same reasoning, we conclude it is at best able to infer the location of the unknown network end by d. For large values of d, like the one we proposed in section 5.2.2, we assume the privacy to be acceptable.

Full path disclosure probability: Even if an adversary controlling two mixnode is not a major threat, we want to avoid at all costs an attack where all three mixnodes on a path are malicious. This would reveal the whole communication and make the mixnet useless. We assume that for big values of d, this would be extremely hard to achieve. The number of nodes within the distance d should be large enough so that the adversary, even by spawning new nodes, has a negligible probability that packets travel through 3 adversarial nodes.

For an adversary controlling n mixnodes among N, it will lead to, on average, $n_{layer} = \frac{n}{3}$ corrupted nodes per layer and $N_{layer} = \frac{N}{3}$ mixnodes per layer. The probability to choose a fully compromised path is then $(\frac{n_{layer}}{N_{layer}})^3 = (\frac{n}{N})^3$ for each packet sent through the network. This is assuming sufficiently large n and N so that mixnodes are evenly distributed among layers, and that the adversary achieves to possess n corrupted mixnodes in the active set.

This result tells us that the probability to encounter a fully compromised path depends on the ratio of compromised nodes. The more mixnodes are operating, the more compromised mixnodes an adversary will need. Using our algorithm, by restricting the number of considered mixnodes, if the adversary has evenly distributed corrupted nodes over the globe, the probability of having a fully corrupted path does not change (the ratio of malicious nodes stays the same). However, it facilitates an attack where an adversary targets a specific region. If the path selection algorithm considers only $\frac{N}{k}$ mixnodes, an adversary still controlling n nodes among the ones considered will have a probability of $(\frac{k \cdot n}{N})^3$ to compromise the full path. In that case,

the risk has been multiplied by k^3. This tells us that if the path selection algorithm considers only $\frac{1}{4}$ of the nodes, and the adversary still controls the same number of nodes, it multiplies the probability by $4^3 = 64$ compared to a fully random node selection.

As mentioned in the introduction, current adversaries are global and can collude. To evade their surveillance, the network should be distributed in such a way that entities working for the network share different interests, and so maximize the number of different jurisdictions, internet providers, countries, etc that the mixnodes uses. This is corroborated by the previous analysis concluding that adversaries must be prevented from controlling a significant part of the active node set, meaning that this set should be as diverse as possible.

To prevent such scenarios, Nym has put safeguards in place.

Staking mechanism The stacking described in 3.1.3 serves as a reputation mechanism. To spawn multiple mixnodes, an adversary will be required to invest a considerable amount of tokens and maintain its nodes' reputation through staking and delegation mechanism. Moreover, as token holders can delegate tokens to mixnodes they trust, Sybil attacks should be improbable.

Mixnodes families Entities owning multiple mixnodes can declare them as belonging to the same *mixnode family*. During the active set selection, the validators avoid selecting too many mixnodes belonging to the same family in the active set. This mechanism ensures that no entity will get a significant probability of having packets routed uniquely through their mixnodes. However, for now, the family announcement is purely based on a voluntary metric.

Geographical nodes distribution

By looking at the current locations of mixnodes, we noticed that the distribution is heavily biased, with more than half of them located in Germany, Finland, and the United States. Table 5.1 shows the first 10 countries hosting the most mixnodes. This has significant privacy and latency implications.

Regarding latency, paths have a higher probability to go through countries with more mixnodes, which may reduce the latency of communications near those areas and increase the latency of other paths. For example, based on data from table 5.1, communications between two nodes in Europe have a high chance to stay in Europe because of nodes located in Germany and Finland, reducing their travelling distance on average. Similarly, communications between two clients far from Europe still have a high probability to include European nodes, causing a bias toward lengthy paths, and increasing latency.

Country	Number of mixnodes	Share of all mixnodes
Germany	209	25.4 %
Finland	122	14.8 %
USA	118	14.3 %
France	40	4.9 %
Netherlands	29	3.5 %
Switzerland	28	3.4 %
United Kingdom	26	3.2 %
Russia	25	3.0 %
Singapore	16	1.9 %
China	14	1.7 %

Table 5.1: Top 10 countries with the most mixnodes on mainnet, June 2023, taken from https://explorer.nymtech.net/nodemap, accessed 08.06.2023

Regarding privacy, if a given distance d is enforced, some gateways or mixnodes far from the others may be unusable because it would be impossible for clients to construct paths including them. Furthermore, an adversary can perform Sybil attacks in areas lacking mixnodes, and easily drain a significant amount of traffic through corrupted mixnodes. To avoid such pitfalls, we included in our algorithm a minimal number of mixnodes n_{min}. If there are fewer than n_{min} mixnodes at distance d, then we double the value d for that specific mixnode selection. This both guarantees that a path will always be found, as long as the topology is valid, and that an adversary can not target isolated gateways.

One could be tempted to further improve the mixnodes selection by considering only n_{min} mixnodes among which one is chosen randomly, ignoring the maximum distance d. However, in this case, an adversary can spawn nodes close to the target client and bias the node selection algorithm. By keeping a distance d in which all the nodes are even candidates, adversaries nodes are competing with every other node no matter their precise location toward the client.

Figures 5.9 and 5.10 show the impact of different values of n_{min} on the latency. We observe that when the mixnode density is high, like in Germany, in figure 5.9, a large n_{min} can be chosen while keeping an reasonable latency. When the density is sufficient, like paths from Germany to the US in figure 5.10, choosing $10 \leq n_{min} \leq 50$ has a small impact on latency. Smaller values don't have an effect at all.

Figure 5.11 presents a suboptimal case, where both the sending gateway in India and the receiving gateway in Canada have very few mixnodes to choose from. When writing these lines, there were 12 mixnodes operating in India and 10 in Canada, from which only a few were part of the active set. In this figure, when the number of mixnodes is not enforced

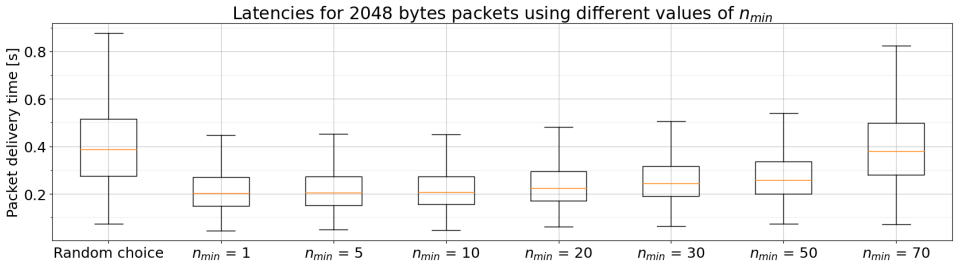

Figure 5.9: Packet delivery time with various values of n_{min}, with d = 250 km. Sending and receiving gateways in Germany.

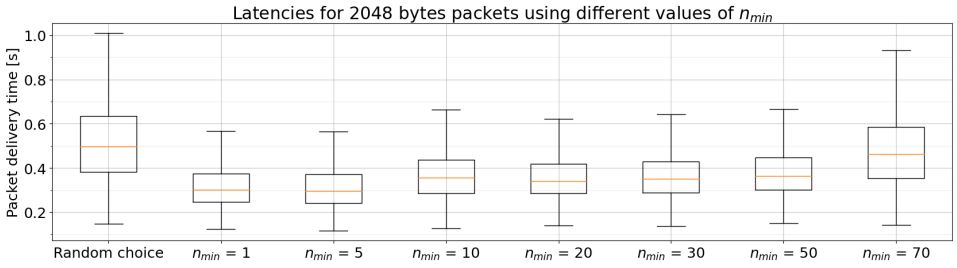

Figure 5.10: Packet delivery time with various values of n_{min}, with d = 250 km. Sending gateway in Germany and receiving gateways in the US.

($n_{min} = 1$), our algorithm operates as described previously. The latency is low, but the chosen paths lack mixnode diversity. When increasing n_{min}, the latency immediately goes up, as the algorithm drastically increases the search radius to fulfil the number of mixnodes requirement. Interestingly, for $20 \leq n_{min} \leq 50$ the latency diminishes. To keep n_{min} high enough, the algorithm considers mixnode in Europe, and, as currently the majority of mixnodes are located there, most paths are then going through Europe. It happens, by chance, to be a good midpoint between India and Canada and benefit from a high-speed network, which unintuitively diminishes the latency. We can verify this assumption by looking at figure 5.12 which shows that paths from Brazil to Canada do not present this behaviour.

Node selection based on latency

One other solution we explored to better select paths in the network was to base our algorithm not on physical distances, but on latencies between nodes. This seems more intuitive as this is the primary metric we are trying to minimize. We implemented the same algorithm as algorithm 1 but instead of using a distance d, we filtered the mixnode based on a maximal latency t. For latency between mixnodes, we used the publicly available VerLoc [28] data that each mixnode exposes, and for latency between gateways and

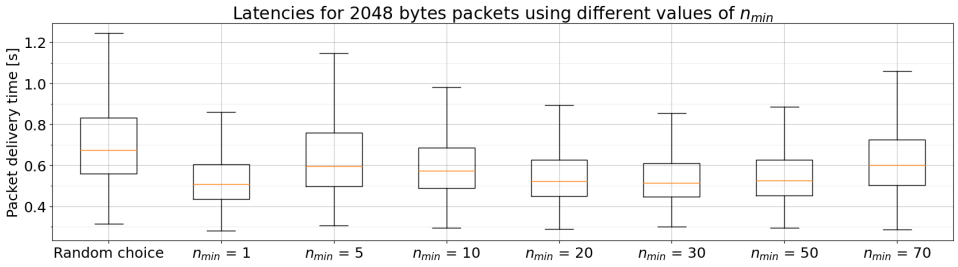

Figure 5.11: Packet delivery time with various values of n_{min}, with d = 250 km. Sending gateway in India and receiving gateways in Canada.

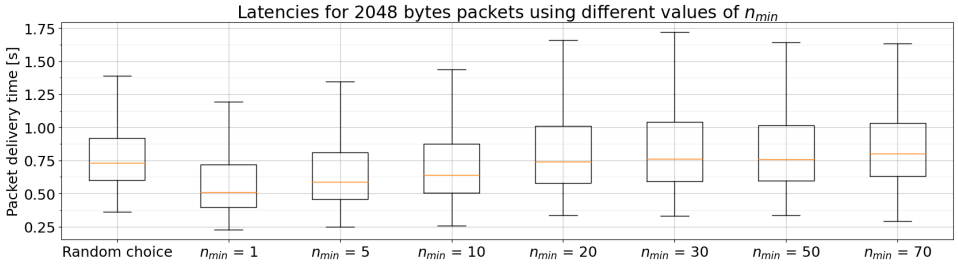

Figure 5.12: Packet delivery time with various values of n_{min}, with d = 250 km. Sending gateway in Brazil and receiving gateway in Canada.

mixnodes we used our own measurement script. By comparing figures 5.6 and 5.13, we observe that the graphs look similar, but the latter doesn't show a plateau when the latency is kept smaller than 200 ms. The same behaviour is noticed by comparing other pairs of graphs. Selecting paths based on constraining latency or distance leads to the same delivery time improvement, in both figures 5.6 and 5.13 the best delivery time achievable is about 220 ms.

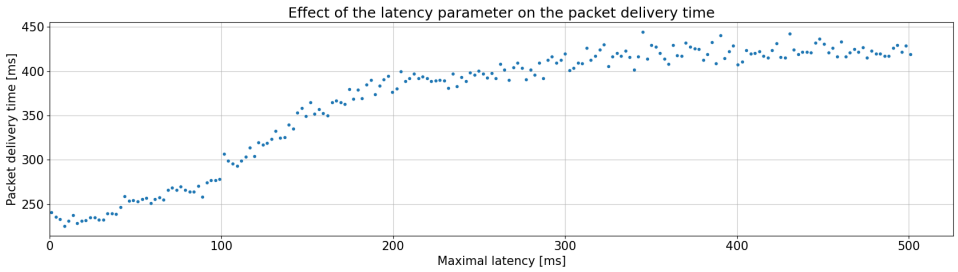

Figure 5.13: Packet delivery time with increasing node selection latency. Sending and receiving gateways both in Germany.

One key drawback of using latencies instead of locations, which justified the latter for our algorithm, is that latencies are computed on a single link

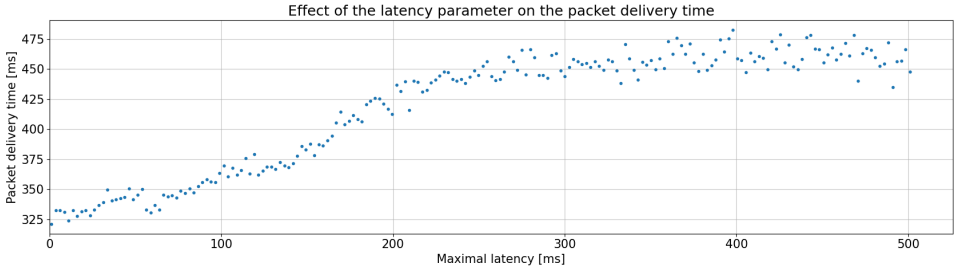

Figure 5.14: Packet delivery time with increasing node selection radius. Sending gateway in Germany, receiving gateway in the United States.

by a single machine. An adversary who controls the network infrastructure and some nodes can add latencies between each pair of nodes not belonging to them, biasing the path selection. This would largely increase the risk for paths to contain three compromised mixnodes. On the other hand, the locations of nodes can be computed by triangulation from multiple different links and run by multiple independent measurers. VerLoc [28] is such a system, it uses a decentralized measurement method and shows that the true location can be determined even with 20% of adversarial nodes. Knowing that both metrics lead to the same latency improvement, we chose to use distance constraints for our algorithm.

5.2.4 Summary

In this chapter, we showed that the default path selection algorithm implemented in the Nym client can create overlong paths by selecting mixnodes at random. We proposed our own algorithm, measured its efficiency, and discussed its tradeoffs. Our solution aims at reducing the path length by fixing a maximal distance d at which each next hop is selected, except for what we name the *unrestricted distance* allowing at most one jump larger than the maximal distance. We showed that such a path selection spares between 30 and 40% of the packet delivery time.

We then conducted an analysis of the privacy tradeoff this algorithm induces. Passive adversaries nor active ones do not gain a substantial advantage. Senders and receivers can't be linked significantly more easily than before. With a large value for d we concluded that the privacy risk was minimal. We then discussed the probability that a packet travels to a fully compromised path and argued that our algorithm does not importantly increase the risk. Finally, we analysed the geographical distribution of nodes and noticed that it was heavily biased. To ensure that no path is vulnerable to Sybil attacks exploiting a lack of mixnode in some regions, we added a minimal number of mixnodes, n_{min}, among which the algorithm must be able to choose, otherwise it increases d.

Chapter 6

Future work

In the previous chapter, we conducted an in-depth analysis of the Nym network latencies and proposed an algorithm to reduce the propagation delay. In this chapter, we discuss further work regarding latency reduction and pitfalls that may arise.

6.1 Further work involving propagation delay

6.1.1 Service provider locations

During our previous discussion on the design of our path algorithm, we emphasized that the choice of the receiving gateway was determined by the receivers themselves. As a result, our algorithm assumed a fixed receiving gateway for message transmission. However, even with our algorithm in place, it is evident that the distance travelled by packets remains one of the primary factors contributing to message propagation delays within the Nym network.

To mitigate this delay further, it is crucial to incentivize service providers to establish multiple endpoints across the globe. By having a distributed network infrastructure with endpoints located in various geographical regions, users can connect to the closest available endpoint. This approach significantly reduces the travelling distance of packets between the sender and receiver.

When users connect to the nearest endpoint, the overall latency and message propagation delay can be significantly reduced. By minimizing the physical distance that packets need to traverse, the network can achieve faster and more efficient communication. Service providers should be encouraged to expand their global presence and establish endpoints strategically, considering factors such as user density and geographical coverage.

Incentivizing service providers to host multiple endpoints across the globe not only improves the user experience in terms of reduced latency but also enhances the overall robustness and reliability of the Nym network.

By optimizing the network's infrastructure to minimize packet travelling distances, we can enhance the efficiency and performance of message transmission within the network.

6.1.2 Gateway selection

In the most recent implementations of the Nym client, if the user does not manually select a gateway, one is chosen randomly. However, this random selection can potentially result in increased latency for all packets if the selected gateway happens to be located far from the client.

It is important to note that gateways in the Nym network do not provide any privacy protection since a global adversary can correlate upstream and downstream traffic. Given this limitation, selecting the closest or fastest gateway does not compromise the security of the network. In fact, opting for the nearest or fastest gateway can help mitigate latency issues by minimizing the physical distance packets need to travel between the client and the gateway.

By considering proximity or network performance metrics when automatically selecting gateways, the Nym client can enhance overall communication efficiency and reduce latency. This approach does not compromise the network's security since the privacy vulnerabilities lie in the correlation of traffic rather than the selection of specific gateways.

Ultimately, choosing the closest or fastest gateway when the user does not explicitly specify one can contribute to an improved user experience by minimizing latency and optimizing message transmission within the Nym network.

Chapter 7

Conclusion

In this work, we provided an overview of the different components comprising the Nym mix network and explained how they collectively create a privacy-preserving overlay network for the Internet. We then conducted an in-depth empirical analysis of the latencies induced by this technology, and we learned that the end-to-end latency originates primarily from transmission and propagation delays.

The transmission delay is capped by the Poisson process, holding a constant sending rate in order to maintain unobservability, a key aspect to avoid correlation attacks over time. Adapting this delay while using the network would reveal its usage and break unobservability.

The propagation delay is the time taken by packets to travel through the network. As we measured, two processes are engendering this delay. The first one is the mixing process at the core of the mixnet, ensuring at each mixnode that incoming and outcoming packets can't be correlated. Reducing this delay would directly impact the anonymity set of packets through the network. The second one is the time for packets to channelize between mixnodes which scales following the distance travelled.

We then proposed an algorithm that aims at reducing this travelling distance while keeping the privacy primitives at the heart of mixnets. It is supported by our observation that paths are mostly penalized by large leaps such as ocean crossing. By avoiding these bounces, we showed that it is possible to reduce the packet latency by more than 30% for most paths while preserving privacy.

Although there is still progress to be made for mixnets to achieve widespread adoption, speed undoubtedly plays a crucial role in providing a technologically acceptable solution. However, it is equally important to raise awareness and educate users about the significance of privacy and why it is essential. By highlighting the importance of privacy and its role in preserving individual freedoms, we can foster a deeper understanding and appreciation for privacy-enhancing technologies like mixnets.

Bibliography

[1] Mullvad VPN AB. Mullvad. https://mullvad.net/.

[2] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. Lastor: A low-latency as-aware tor client. In *2012 IEEE Symposium on Security and Privacy*, pages 476–490, 2012.

[3] Robert Annessi and Martin Schmiedecker. Navigator: Finding faster paths to anonymity. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 214–226, 2016.

[4] Lamiaa Basyoni, Noora Fetais, Aiman Erbad, Amr Mohamed, and Mohsen Guizani. Traffic analysis attacks on tor: A survey. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 183–188, 2020.

[5] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 605–616, New York, NY, USA, 2012. Association for Computing Machinery.

[6] Jake S. Cannell, Justin Sheek, Jay Freeman, Greg Hazel, Jennifer Rodriguez-Mueller, Eric Hou, Brian J. Fox, and Dr. Steven Waterhouse. *Orchid: A Decentralized Network Routing Market*, 2019.

[7] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, feb 1981.

[8] Fallon Chen and Joseph Pasquale. Toward improving path selection in tor. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6, 2010.

[9] Harry Halpin Claudia Diaz and Aggelos Kiayias. The nym network. https://nymtech.net/nym-whitepaper.pdf, 2021.

[10] Harry Halpin Claudia Diaz and Aggelos Kiayias. Reward sharing for mixnets. https://nymtech.net/nym-cryptoecon-paper.pdf, 2021.

[11] George Danezis. Mix-networks with restricted routes. In Roger Dingledine, editor, *Privacy Enhancing Technologies*, pages 1–17, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[12] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *2009 30th IEEE Symposium on Security and Privacy*, pages 269–282, 2009.

[13] P. Dhungel, M. Steiner, I. Rimac, V. Hilt, and K. W. Ross. Waiting for anonymity: Understanding delays in the tor overlay. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–4, 2010.

[14] Claudia Diaz, Steven J. Murdoch, and Carmela Troncoso. Impact of network topology on anonymity and overhead in low-latency anonymity networks. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies*, pages 184–201, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[15] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In Roger Dingledine and Paul Syverson, editors, *Privacy Enhancing Technologies*, pages 54–68, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[16] Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. A reputation system to increase mix-net reliability. In Ira S. Moskowitz, editor, *Information Hiding*, pages 126–141, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[17] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, August 2004. USENIX Association.

[18] B. Evers, J. Hols, E. Kula, J. Schouten, M. den Toom, R.M. van der Laan, and J.A. Pouwelse. Thirteen years of tor attacks. https://github.com/Attacks-on-Tor/Attacks-on-Tor, 2016.

[19] Saman Feghhi and Douglas J. Leith. A web traffic analysis attack using only timing information. *IEEE Transactions on Information Forensics and Security*, 11(8):1747–1759, 2016.

[20] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In Ross Anderson, editor, *Information Hiding*, pages 137–150, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[21] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, page 31–42, New York, NY, USA, 2009. Association for Computing Machinery.

[22] Kyle Hogan, Sacha Servan-Schreiber, Zachary Newman, Ben Weintraub, Cristina Nita-Rotaru, and Srinivas Devadas. Shortor: Improving tor network latency via multi-hop overlay routing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1933–1952, 2022.

[23] Mohsen Imani, Mehrdad Amirabadi, and Matthew Wright. Modified relay selection and circuit selection for faster tor. *IET Communications*, 13(17):2723–2734, 2019.

[24] Mathieu Jee, Ania M. Piotrowska, Harry Halpin, and Ninoslav Marina. Optimizing anonymity and performance in a mix network. In Esma Aïmeur, Maryline Laurent, Reda Yaich, Benoît Dupont, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 53–62, Cham, 2022. Springer International Publishing.

[25] Shu Jiang and N.H. Vaidya. A mix route algorithm for mix-net in wireless mobile ad hoc networks. In *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE Cat. No.04EX975)*, pages 406–415, 2004.

[26] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 337–348, New York, NY, USA, 2013. Association for Computing Machinery.

[27] jrandom (Pseudonym). Invisible internet project (i2p) project overview. https://geti2p.net/_static/pdf/i2p_philosophy.pdf, August 2003.

[28] Katharina Siobhan Kohls and Claudia Díaz. Verloc: Verifiable localization in decentralized systems. In *USENIX Security Symposium*, 2021.

[29] S.J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195, 2005.

[30] Vasile Perta, Marco Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei. A glance through the vpn looking glass: Ipv6 leakage and dns hijacking in commercial vpn clients. In *Proceedings on Privacy Enhancing Technologies*, volume 1, 04 2015.

[31] Ania M. Piotrowska. Studying the anonymity trilemma with a discrete-event mix network simulator. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, WPES '21, page 39–44, New York, NY, USA, 2021. Association for Computing Machinery.

[32] Sentinel. Sentinel : A blockchain framework for building decentralized apn applications. https://sentinel.co/, 2020.

[33] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *CoRR*, abs/1802.07344, 2018.

[34] Matteo Varvello, Iñigo Querejeta Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Goncalo Pestana, and Benjamin Livshits. Vpn-zero: A privacy-preserving decentralized virtual private network. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–6, 2021.

[35] Chris Wacek, Henry Tan, Kevin S. Bauer, and Michael E. Sherr. An empirical evaluation of relay selection in tor. In *Network and Distributed System Security Symposium*, 2013.

[36] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware path selection for tor. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security*, pages 98–113, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Appendix A

Latencies analysis

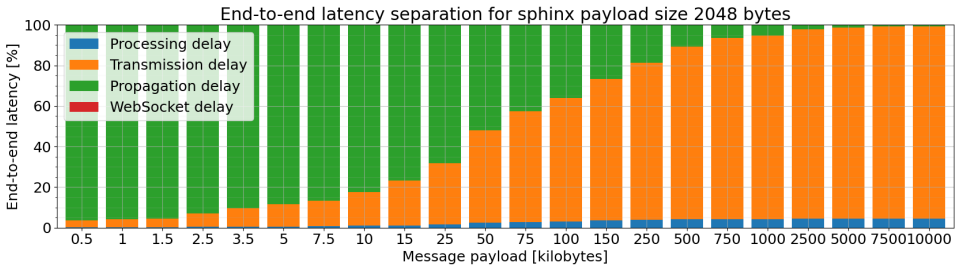

Figure A.1: Message E2E latency separation for packets of size 2048 bytes

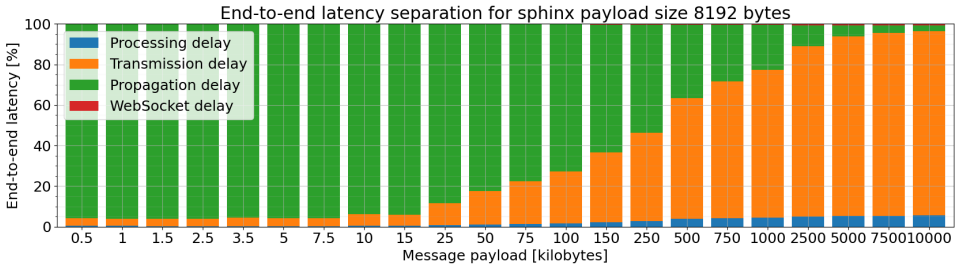

Figure A.2: Message E2E latency separation for packets of size 8192 bytes

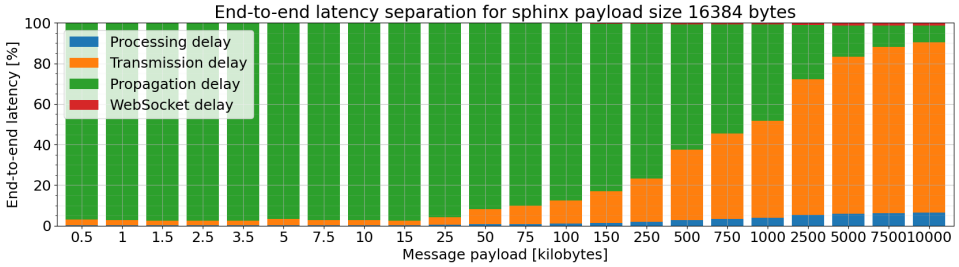

Figure A.3: Message E2E latency separation for packets of size 16384 bytes

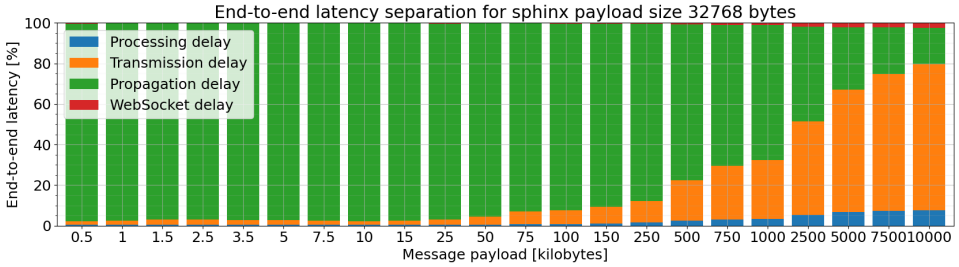

Figure A.4: Message E2E latency separation for packets of size 32768 bytes

Appendix B

Packet delivery time by node selection radius

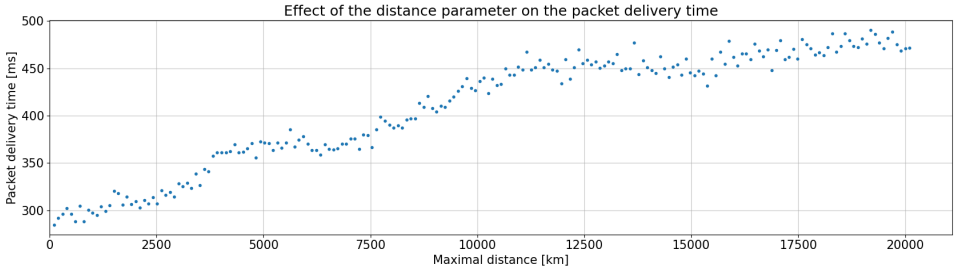

Figure B.1: Packet delivery time with increasing node selection radius. From Germany to the US

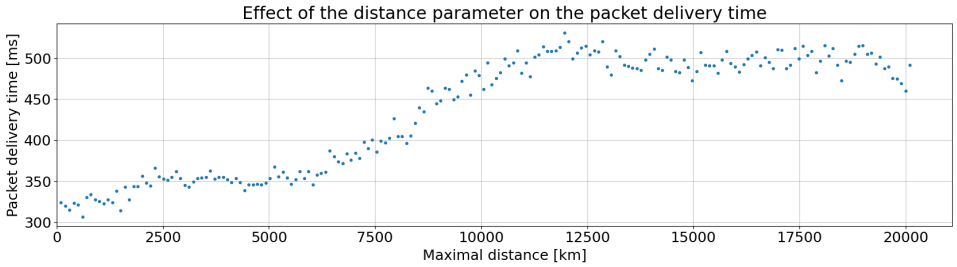

Figure B.2: Packet delivery time with increasing node selection radius. From Germany to Russia

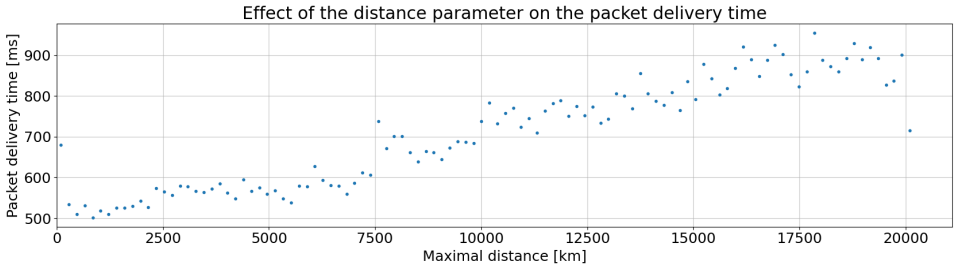

Figure B.3: Packet delivery time with increasing node selection radius. From Germany to Brazil

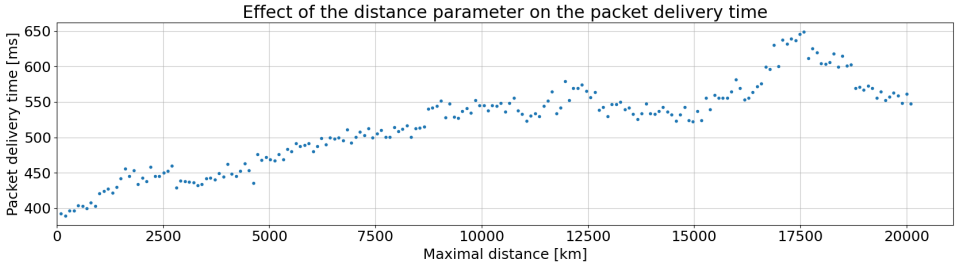

Figure B.4: Packet delivery time with increasing node selection radius. From Germany to Singapore

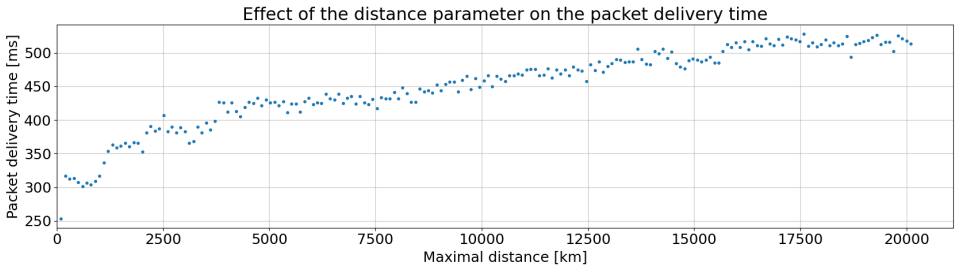

Figure B.5: Packet delivery time with increasing node selection radius. From the US to the US

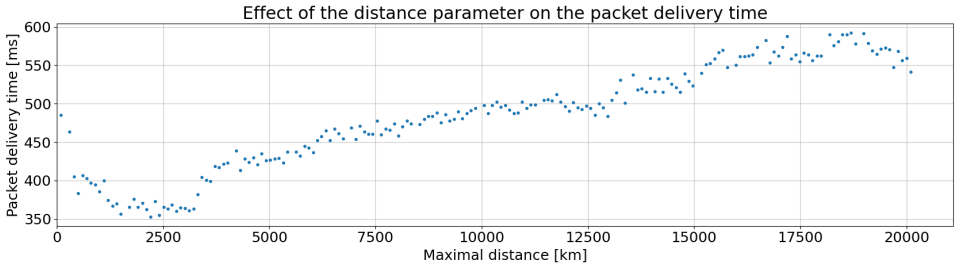

Figure B.6: Packet delivery time with increasing node selection radius. From the US to Canada. The high delivery time observed for smaller distances is a consequence of a lack of mixnodes in Canada. This forced the algorithm to increase its search distance to fetch at least one mixnode.

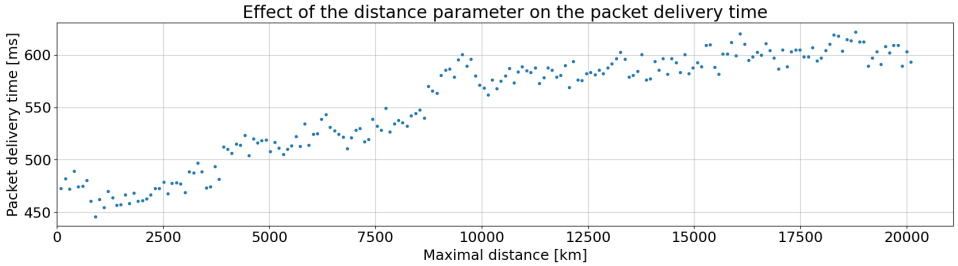

Figure B.7: Packet delivery time with increasing node selection radius. From Singapore to the US

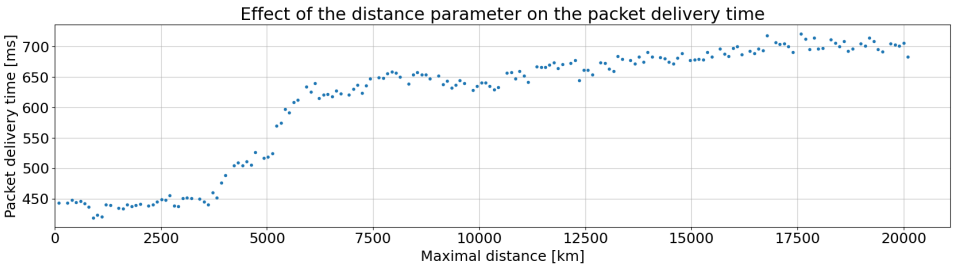

Figure B.8: Packet delivery time with increasing node selection radius. From Singapore to India