Exposed amongst the Hidden:

An Empirical Analysis of the Effects of the Sphinx Packet Unwrap Operation on the Anonymity of the Continuous-time Mixnet

Kuan Lon Vu

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2023

Abstract

Overhead incurred by Sphinx packet processing are accounted using constants in current literature. This project focuses on one aspect of the packet processing - Sphinx packet unwrapping. It is demonstrated that the difference in unwrap time is significant for different test machines, even for the ones with similar processing speeds. Due to the limited related work on this topic, we propose two approaches for mixes to take when transmitting packets. Approach 1 attempts counteract the effects of the unwrap time whilst approach 2 does not. To the best of our knowledge, this is a novel distinction in terms of packet transmission. Approach 1 introduces the concept of over-delayed packets, and three additional novel strategies are proposed to mitigate them.

The experiment framework is obtained by integrating the Sphinx packet unwrap function into a mixnet simulator via Rust bindings. The exponentiality of the packet delays under strategy 1 of approach 1 and approach 2 are evaluated across 4 test machines on 9 end-to-end latencies, ranging from 0.15000001 second to 10 seconds. The results show that the actual packet delays deviate from the sender-intended delay when $d_{E2E} \leq 0.25$ second for most test machines. The percentage of over-delayed packets also falls below 1% when the end-to-end latency is greater than 1 second.

In addition, two new packet-distinguishing attacks are introduced. Increasing the end-to-end latency from the minimal is effective in preventing the single-packet case of the distinguishing attack. Decreasing the proportion of over-delayed packet by increasing the end-to-end latency is effective in reducing the occurrences of multiple-packet case of the attack.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kuan Lon Vu*)

Acknowledgements

First of all, I would like to thank my supervisor, Dr. Tariq Elahi, for giving directions for the project, and all the support and discussions throughout the year. I also thank him for the access to the lab machine on which some experiments are conducted.

I would also like to thank my family for their encouragement and unconditional love for me. Thank you, Mum, for picking up the phone no matter the time across the globe. To my sister for the energy, support and joy you bring, and for believing in me regardless of the circumstances. Ruth, thank you for being there always and all that you do. And for my flatmates and friends: Ruth, Jacqueline and Sarina, thank you for being such lovely people.

I thank my mum, my small group and my friends for their continuous prayers for me.

I thank Wei En and Owen for the discussions. For my sister for proofreading the report and Jingyuan for providing feedback.

Last but not least, Jesus Christ, my Lord and my God, from whom all my strength comes from, and to whom I give all honour and praise of this work.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

With the increase in privacy awareness with communication on the Internet, there is a rise in popularity for communication privacy using forms of anonymous communication systems. The sophisticated state-of-the-art encryption schemes conceal the content of the packets. However they do not obfuscate sensitive network-level metadata such as message sending frequency, size of the messages, and packet departure and arrival times. Many research in current literature highlight that metadata alone is sufficient to de-anonymise the participants of a network [3, 5, 13, 24]. The prominent network anonymity schemes in practice is Tor (The Onion Router). It is a low latency anonymous overlay network which is decentralised yet synchronous; the connections between participants are routed in a multi-hop manner [12]. The sender prepares the packet by wrapping the message and the header in multiple layers of public key cryptography, and sends it through a path consists of several nodes where the nodes decrypt their corresponding layer and forward it to the next node [12]. In such process, there is no need for a single trusted server, as the process is distributed among several nodes. Low latency anonymity systems, such as Tor, offer limited security guarantees against a stronger global adversary, who can monitor the flow of packets across the entire network to analyse traffic patterns. These infrastructures also suffer from traffic analysis attacks, where the adversary is able to exploit the network metadata to correlate and de-anonymise users by performing timing and traffic analysis - inferring the sender and the receiver from the network metadata such as the packet size, arrival and departure times.

In order to protect against such powerful adversary, Chaum [6] conceptualised integrating *mix properties* into the network using mix servers - computers that processes and relay the packets between the senders and the recipients. Each mix would delay and reorder the packets to break the linkability between the incoming and outgoing packets of a mix, for an observing adversary. The additional latency introduced by the delays obfuscates the sending and arrival time of a message between the sender and the recipient. The resultant high latency system is known as a mix network, hereinafter mixnets.

There are different mixing techniques used in mixnet designs. A classical approach is threshold based where a mix batches and shuffles the packets, and forwards them only when a certain threshold of packets is reached. This results in unbounded delay thus unpredictable packet arrival time. This project focuses on continuous-time mixing, presented by Danezis [9]. In continuous-time mixing, each packets are delayed individually and independently with the delay at each mix is randomly selected from an exponential distribution with parameter μ. It is shown to provide optimal anonymity and bounded end-to-end latency. Although the concept of mixnet predates Tor, the development on mixnet is relatively limited due to the more wider adapted Tor, which attracted a large deployment and a diverse user base due to its low latency. There is limited research focused on the computational and latency cost of the mixnet. Transiting from research to commercialisation, Nym technologies unveils the Nym network [11], which is a continuous-time mixnet, based on the Loopix anonymity system [21].

Current literature on evaluating the anonymity property of continuous-time mixnets such as the Nym and Loopix mixnets, does not consider the overhead introduced by the cryptographic operations performed at each mix [11, 21]. Through this work, we focus on the anonymity of the packets transmitted in the mixnet, with respect to the impacts of cryptographic operations performed at the mixes. To the best of our knowledge, this is the first piece of work dedicated on such topic which we wish to provide further insights on. As a starting basis, we concentrate specifically on the effects of the unwrap time of the mix running on different hardware specifications. In addition, we wish to give further insights on the potential suitable machines to operate mixes in a real-word scenario.

1.2 Project Objectives

The primary purpose of this project is to present the impacts of cryptographic packet unwrap operation on the exponentially chosen delays.

This project addresses:

- The significance of the cryptographic packet unwrap at the mix on the anonymity of the packets when evaluating on different hardware specifications.
- Whether there is a value for the parameter μ of the exponential distribution of which the delays are selected from, such that the overall delays of packets at a mix is no longer exponentially distributed, i.e. over-delayed packets.
- A suitable value for the parameter μ to mitigate the effects of the cryptographic overhead.

1.3 Contributions

As part of this project, the following has been achieved, in addition to the ones listed in Section 1.2.

- Developed an experiment framework that integrates Sphinx packet unwrap operation into a mixnet simulator.
- Verified that the appropriate parameters to set up the simulator for the experiments.
- Showed that the differences in unwrap time of the mix between various test machines are statistically significant, even for machines which the different in their processing speeds are statistically insignificant.
- Demonstrated that there exists values of parameter μ of the exponential distribution, such that the sampled sender-chosen delay is shorter than the necessary cryptographic packet unwrap operation performed at the mix. In particular, for μ very close to 0, 100% (rounded to 2 decimal places) of the packets have unwrap time greater than the sender-intended delay.
- Discussed the implausibility of completely eliminating over-delayed packets.
- Proposed and analysed two approaches to mitigate the presence of the unwrap time of the mix on the packet.
- Introduced three strategies for mixes to deploy in the case of over-delayed packets, and provided detailed analysis for one of them.
- Investigated the limitations of the approaches and introduced a novel class of distinguishing attacks a global network adversary can perform to correlate inputs and outputs of the targeted mix.

1.4 Report Organisation

This section outlines the structure of the remaining chapters of this report.

Chapter 2 provides the relevant background on continuous-time mixing and an overview on related work in current literature.

Chapter 3 defines the threat model of concern for this project and introduces the packet transmission strategies which form the basis for the experiments. A description of experimental framework, the relevant modifications and the final workflow of the simulator are also provided.

Chapter 4 presents a summary of key notations and details the parameter configurations of the framework used for the experiments.

Chapter 5 presents and interprets the results obtained from the experiments, and a discussion on their implications.

Chapter 6 critically analyses and evaluates the work conducted in this project.

Chapter 7 concludes the work in this project and presents an overview for further work.

Chapter 2

Background and Related work

In this chapter, the background on continuous-time mixnets and the exponential distribution are presented. Then an overview on the design of the Nym network and the Loopix anonymity system, and the aspects that are relevant to this project are described. We also provide a discussion on the relevant work in the area of continuous-time mixnets and topics that warrant further research in current literature.

2.1 Continuous-time Mixnets

Mixnet is a multi-hop decentralised network that is designed to provide anonymity to its participating users, even with the presence of a strong global adversary. It consists of machines called mixes; packets are routed through a series of these mixes before reaching its final destination. The multi-hop design distributes the trust across the nodes in the network, preventing a single point of failure, since no nodes have the full knowledge of both sender and receiver for any packets.

Mixnets protect the end users against end-to-end correlation attacks by obscuring the relationship between the senders and receivers of messages. The traffic sent by end users are modelled by a Poisson process, which is parameterised by $\lambda_{enduser}$ per second. In this case, the Poisson process is stochastic process that models the times at which the packets arrive at the mix.

Two operations are performed at each mix: mixing and cryptographically transforming the packets. Each mix *mixes* - randomly delays and permutates - the packets it received, which destroys the network patterns of the packets. This prevents a powerful adversary from de-anonymising the senders and receivers by performing traffic analysis on the leaked network metadata such as message arrival and departure times. For example, an adversary observing the endpoints of a low-latency network could correlate the sender and the receiver by analysing the timing between packet departure and arrival times. The latency introduced in the hops destroys the timing pattern which can be exploited by the adversary. The continuous-time mixing technique is individually delaying each packet at the mix, for a period of time that is sampled from an exponential distribution. This allows continuous-time mixnets to achieve high entropy in the reordering of a

sequence of packets.

The mixing technique used in continuous-time mixnet achieves high entropy in re-ordering of the sequence of packets is individually delay each packet at the mix. The independent sender-specified delay at each mix is a random variable following an exponential distribution with parameter μ; all sender-specified delays are drawn from the same exponential distribution. The average delay λ of the packets at each mix is represented by $\frac{1}{\mu}$ seconds.

In terms of the strong anonymity of the continuous-time mixnet, the entropy between incoming and outgoing packets, regardless of their arrival time at the mix, is based on the memoryless property of the exponential distribution. This ensures that the probability of a packet being forwarded from a mix is independent of its arrival time. Hence it is equally probable for any of the packets arrived at the mix to depart at the next moment in time.

2.2 Exponential Distribution

This section highlights the importance of the exponential distribution in the anonymity property of the continuous-time mixes.

The delay of a packet at a particular mix is characterised as a random variable X following an exponential distribution with parameter μ. Its probability density function (PDF) is defined by Equation 2.1. λ denotes the average delay, where $\lambda = \frac{1}{\mu}$. The area under the PDF for any λ equals to 1.

$$f(x;\mu) = \mu e^{-\mu x} \tag{2.1}$$

Or equivalently, in terms of λ:

$$f(x;\lambda) = \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} \tag{2.2}$$

Figure 2.1 illustrates the PDF of different means of the exponential distribution. The expected value of X is the mean of the exponential distribution.

The exponential distribution is the only continuous probability distribution with the memoryless property. In the scenario of the latency on a packet at a mix, the memoryless property offers unlinkability - there is an equal probability of the outgoing packet being any of the previously observed incoming packet to the mix. It is not possible for the adversary to correlate the packets based on arrival times.

In relation to the mean latency of the packets, the smaller value of the parameter μ, the longer the period of time which the packets on average wait for at a mix, increasing the end-to-end latency of the packet from the sender to the receiver. Note that with the exponential delays, while the value of the end-to-end latency can be estimated (hence bounded), it is not possible to determine the exact timings of each packet.

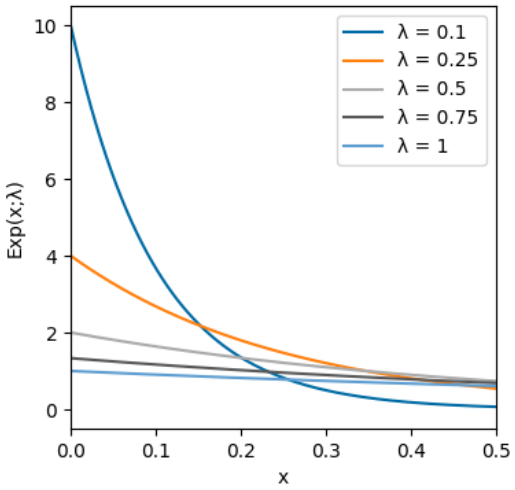


Figure 2.1: Plot of the probability distribution functions of the exponential distribution with λ = 0.1, 0.25, 0.5, 0.75, and 1.

Finally, we present here a proof that a shifted exponential distribution is not memoryless, which is important for the analysis.

Proof. First we define mathematically the notion of memoryless. Suppose X is an exponential distributed random variable such that the range of X is $[0,\infty)$. The probability distribution of X is memoryless if the following holds for any $s,t > 0$:

$$Pr(X > t+s|X > s) = Pr(X > t)$$

Now suppose another random variable Y, obtained by shifting the exponential distribution by a constant value of c, such that $Y = X + c$ and the range of Y is $[0,\infty)$. Then:

$$\begin{aligned} Pr(Y > t+s|Y > s) &= Pr(X+c > t+s|X+c > s) \\ &= Pr(X > t+s-c|X > s-c) \\ &\neq Pr(X > t) \end{aligned}$$

□

2.3 Shannon Entropy

Introduced by Shannon [22], the Shannon entropy (denoted simply as *entropy* in this report) of a random variable is a measure in information theory of uncertainty. In this report, in line with existing work in anonymous systems [4, 20], entropy quantifies the indistinguishablility of the target message of interest amongst the other messages in the network; it denotes the number of packets in logarithmic scale that an adversary can be confused with the target message [4]. Hence the greater the value of entropy, the stronger anonymity in the network. In continuous-time mxining, due to the memoryless property of the exponential distribution, which the per-mix delays of the packets are sampled from, the independent delays of the packets result in high entropy [9, 11, 21].

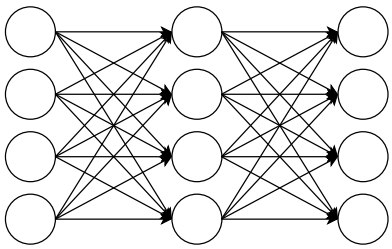


Figure 2.2: The structure of a stratified topological mixnet with 3 layers and 4 mixes per layer, the total number of mixes is 12. The circles denote the mixes.

2.4 The Nym Mixnet and Loopix Anonymity System

The related systems for this project is the Nym network and its preliminary template model, the Loopix anonymity system. This section provides an overview on their designs and highlights the aspects that are relevant to this project, namely the continuous-time mixes and the Sphinx packet format, along with an evaluation on their implementations.

2.4.1 Overview on the Systems

Shannon [22] and Danezis [9] proved that maximum entropy occurs when the probability distribution is an exponential distribution with the mean equal to λ, the average latency of the packets in the mixes. Since then several systems that utilise such property for some protection against timing analysis attacks, have been developed, namely the Loopix system [21] and the Nym network [11].

Nym Technologies has presented the Nym network, a generic infrastructure that consists of many components and different participants to achieve communication privacy. The Loopix anonymity system is a continuous-time mixnet for real-time communications [21]. The core component of the Nym network is the continuous-time mixnet; the design of the Loopix system forms the basis for the Nym mixnet which similarly utilises features such as the Sphinx packet format detailed in Section 2.4.2. The architectural structure of the mixnet is a stratified topology or *restricted route*, illustrated in Figure 2.2 - where the mixes are organised in layers, and each mix in a layer is connected to every mix in the adjacent layers. This reduces the number of connections between mixes as it is not a fully connected topology, or *free route*. The benefits of stratified topology is that there are fewer connections between the mixes and this enables horizontal scaling whilst maintaining anonymity [8]. The capacity of the mixnet can be increased by additional hardware resources to accommodate more traffic without any design modifications on the network. The mixes in the Nym mixnet are controlled by incentivised mix operators.

Besides the mixes for mixing the packets in the mixnet, the Nym network has employed additional features for practical deployment and adaptation such as gateways and validators, as well as bandwidth credentials and service credentials for the proper functionalities of the network for real-world adaptations. These components are beyond the scope of this work hence from this point on, we consider only the mixnet aspect of the Nym network, which is also the Loopix mixnet.

2.4.2 Sphinx Packet Format

The Loopix and Nym mixnets both leverage the Sphinx packet format for privacy protection of the content of the messages between sender and receiver. Sphinx packet format is an efficient cryptographic message format with proven cryptographic security and small amount of overhead [10].

Sphinx packet formation To prepare a Sphinx packet to transmit through a sequence of mixes, the sender derives a set of session keys with each of the mixes in the sequence using a random cyclic group element and the public key of the mix. The header and the payload are encapsulated separately in layers of encryption that is reversed to the order of which the packet is routed through the sequence of mixes. Each mix on the packet's journey would learn inherently at most the preceding mix and succeeding mix, for routing purposes. With multiple hops, no mix has the knowledge of both the sender and the receiver.

Specifically, a Sphinx packet S is routed through mixes $M_1, ..., M_n$, where n is the number of hops to route the packet before reaching the receiver. Note that the numbering on the mixes is sequential for the sole purpose of denoting the order of which the packet is relayed through the mixnet. The outermost layer of encryption is for the first mix M_1; the process repeats for remaining mixes on the route until the final mix M_n, which forwards the packet to the receiver. Figure 2.3 illustrates the structure of a Sphinx packet traversing through a three-hop route. At each layer of encryption, the header contains information for session key derivation, routing information to forward the packet, a delay randomly chosen from the μ-parameterised exponential distribution, and an integrity protection mechanism, namely a message authentication code (MAC) to detect any modifications on the header [10].

Packet processing At each hop, the mix which receives the packet derives the set of shared session keys with the element from the cyclic group and its private key. Once the mix verifies that the header has not been modified, it de-encapsulates ("unwraps") the layer of encryption to extract the routing information to the subsequent destination and the sender-chosen delay for which the packet should wait at the mix before being relayed. The mix pads the packet to identical lengths to conceal the total length of the route and the number of mixes the packet has traversed through. The mix also cryptographically transform the packet before forwarding it such that there is bitwise unlinkability. This means a third party cannot correlate the inputs and outputs of a mix by analysing the their bit pattern, without knowledge of the cryptographic keys used for transforming the packets.

Both packet transformation and unwrapping are cryptographic operations which are computationally expensive and typically subjects to some form of overhead. In this project, we focus on the overhead incurred by the cryptographic operation to unwrap a layer of encryption at a mix, which is denoted as the *unwrap time* of the mix.

2.4.3 System Implementation

The team at Nym Technologies has made available the code of the Nym network infrastructure as well as their implementation of the Sphinx packet format in Rust

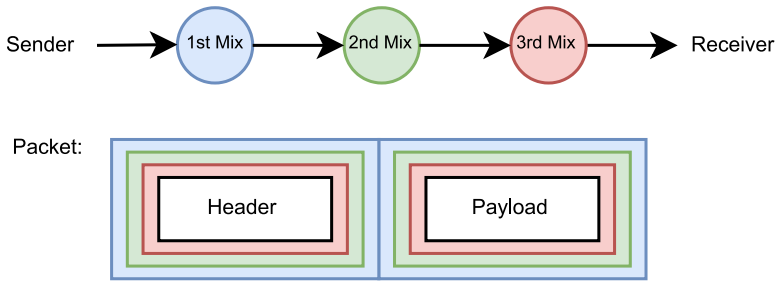


Figure 2.3: The structure of the Sphinx packet, given the packet is routed through three mixes.

under an open-source license [1, 2]. It includes implementations on the components besides the ones for the mixnet such as: Nym blockchain, validator, Nym wallet and its respective components to ensure the proper usage of the network and maintain its functionalities. Since the current codebase of the Nym network offers much additional features beyond the scope of this project, specifically the layered continuous-time mixnet, the discrete-event mixnet simulator is used for the experiments in this work.

2.5 Mixnet Simulation Frameworks

This section discusses **MiXiM**[4] and **Mix Network Simulator**[20] (henceforth, Loopix Simulator), two generic discrete-event simulation frameworks implemented in `Python3` that supports simulations on stratified continuous-time mixnets such as the type of Loopix and Nym. Specifically they are developed using `SciPy` discrete event simulation library. The code for both frameworks are available under open source licences. The frameworks are extendable to enable further research on the various mixnet designs and their configurations.

The Loopix Simulator was developed initially to support anonymity analyses of Loopix under various network configurations and traffic conditions. It is subsequently used for conducting further experiments for the Nym mixnet [20]. The simulator supports many network topology such as stratified and cascade, and offers the flexibility for user-defined configurations. The stratified topology of the mixnet simulator is a simplified version of the Loopix mixnet - it simulates only the network aspect of the mixnet and does not include any cryptographic operations, hence the mixes do not perform any cryptographic operations to unwrap and transform the packets. The provided codebase of the Loopix mixnet is a 3-by-3 layered mixnet where there are three layers and three mixes per layer.

Guirat and Diaz [14] conducted their research on optimising parameters for continuous-time mixnets using MiXiM. It focuses on determining the number of layers and their width, i.e. the number of mixes per layer in consideration of the network deployment configurations and the proportion of compromised mixes with Shannon entropy as the anonymity metric. To the best of our knowledge, it remains unexplored on the topic of selecting a suitable value for the average per-mix delay with respect to the machines hosting the mixes, which reflects the real-world scenarios where mixes are hosted on

machines with different resources that impose variable processing overhead.

Jee *et al.* [15] investigated on optimising various mixnet parameters to reach acceptable performance-anonymity trade-off in real-world mixnet. Their analysis concentrated on parameters such as size of the Sphinx payload, the cover traffic rates, and end user traffic rates.

Limitations It should be noted that whilst the functionality of the network simulation is not affected, both simulators simulate solely the network aspect of the mixnet. With a given configuration for a stratified continuous-time mixnet, the mixnet simulators are designed such that the exponential delays of the packets are randomly sampled at the mixes, contrary to design of MiXiM and true continuous-time mixnets, where the random delay is chosen by the sender. However, note that the packets in MiXiM are not layered-encrypted Sphinx packets, since it is not necessary for the network simulation of the mixnet. Hence, the end users in the MiXiM simulation prepares the packets by encoding the per-mix delays in a list, which is accessed by the mixes on the route by indexing.

As Nym Technologies unveil their general-purpose privacy infrastructure, there are emerging areas that have not been fully explored due to the impact of real-world deployment has on the anonymity guaranteed by the exponential distribution. Most literature on the anonymity property continuous-time mixnet do not specifically consider the significance of the overhead caused by the cryptographic operations when evaluating the anonymity of their proposed system, including Loopix and Nym [11, 21]. From inspecting the source code of the mixnet simulators, the simulation accounts for the general packet processing overhead by manually adding a constant of 0.000386 second to every randomised delay for the Loopix Simulator. The value of 0.000386 is yielded from benchmarking the creation of a new Sphinx packet on a specific test reference machine [2]. In the case of MiXiM, the packet processing is considered holistically with the and altogether accounted conservatively by a constant of 50 ms. The average end-to-end latency d_{E2E} of a packet is calculated based on the sum of three factors: the total of per-mix latencies, network propagation of the links, and the packet processing times.

In particular, this project aims to address the influence of unwrapping time of a mix on the memoryless property of the exponential distribution. We show that while the overhead of unwrapping the Sphinx packet is small, it is not insignificant. This merits further research as in theoretical modelling, the value for the chosen parameter μ can tend to zero such that the average per-mix delay becomes very short. However, in real-world usages, there exists an inevitable packet processing overhead at each mix. Therefore, the packet would wait in the mix for at least the time taken for the mix to process it, regardless of how short the sender-intended delay. This is significant when the sender-selected delay d is less than the unwrap time of the mix u, $d < u$, as the delay is chosen such that the memoryless property of the exponential distribution holds. It calls for further investigation on the impact of such additional delay due to the overhead at a mix, on the anonymity property of the mixnet.

Chapter 3

Methodology and Experiment Framework Structure

In this chapter, the threat model and the packet-distinguishing attacks of interest, are defined in detail. Then, the structure of the mixnet framework for the experiments is outlined.

3.1 Threat Model

Attack Environment and Adversary Capabilities The threat model considered is a global passive adversary (GPA) and the following is assumed for the adversary and the environment. The adversary is able to (1) observe all network communications between the components - end users and mixes - of the mixnet; (2) measure the sending and arrival times of any packets to perform traffic and timing analysis attacks to correlate the sender and receiver of any packet.

The mixnet is assumed to have balanced layers, meaning equal number of mixes in each layer. Moreover, it is assumed that the adversary is not able to influence the layers which the mixes are assigned to. Each mix is a physical machine with certain processing power such that its unwrap time is bounded. The mixnet is assumed to be *honest*; all mixes are honest and the adversary is not able to corrupt them. The end users are also considered to be honest, and they send traffic to the mixnet following a Poisson process that models the rate at which packets are sent into the mixnet. Corrupted mixes can cause packets to be dropped and techniques have been proposed for detecting such malicious mixes [18], therefore we consider them out of the scope of this work. The adversary targets a single honest mix, and it is not able to corrupt this mix. It is also assumed that the adversary has the measures and resources to obtain the unwrap time of each incoming packet to the targeted mix - this assumption will later be relaxed to average unwrap time of the mix near the end of this report. However, discussion on such measures is beyond the scope of this work. Regarding the packets, they are assumed to have identical size, so that the adversary can only correlate the packets using the metadata on timing.

The adversary can cause compromised mixes to hold onto packets for a period of time other than the delay specified in the header. It is able to observe all the incoming and outgoing packets of all mix, including the said targeted mix.

Adversary Goal The goal of the adversary is to gain a non-negligible advantage in linking incoming packets to an outgoing packet of the targeted mix, using the timing information of the packets.

3.2 Packet Transmission Strategies

By including the unwrap time of the mix into consideration, we propose two approaches that mixes can perform for each received packet:

1. Once the mix unwraps the packet and finds the sender-intended delay, the mix reduces the sender-intended delay by the unwrap time so that the overall delay the packet experiences at the mix is the sender-intended delay.
2. The mix unwraps the packet and continues to retain the packet for the duration of delay intended by the sender. The total delay of the packet at the mix is the unwrap time plus the exponential delay.

Approach 1 counteracts the effects of unwrap time whereas approach 2 does not. However, approach 1 introduces the possibility that the designated delay intended by the sender of the packet is less than the mandatory unwrap time, such that once the mix unwraps the packet and discovers the intended delay encoded in the header, the packet has already dwelled in the mix for more than the exponentially chosen delay. These packets are hereafter denoted as *over-delayed* packets. It is important to consider over-delayed packets, as the memoryless property of the exponential per-mix delay contributes to the protection against timing attacks on the packets. In such case, an apparent question arises: *when should these packets be relayed to their next destination?* We propose that there are three potential strategies that can be deployed in order to resolve these over-delayed packets.

1. The mix transmits the over-delayed packet to its next destination immediately; the total delay the packet experiences at the mix is just the unwrap time (the overhead incurred by the cryptographic per-mix packet transformation is left as future work).
2. The sender ensures that the chosen per-mix delay is greater than the unwrap time of the mix; the delay at the mix is the sender-chosen delay since there would not be over-delayed packets.
3. The mix adds some additional delay to the packet on top of the unwrap time and the sender-intended delay, then relay it to the next destination. The total delay of the packet at the mix is unwrap time and sender-intended delay plus some additional delay.

We now discuss briefly about strategies 2 and 3 of approach 1; thorough investigations of the plausibility of these strategies are left as future work. Strategy 2 delegates the responsibility of avoiding over-delay packets to the sender. There could be implications

of de-anonymising packets in the mixnet if the sender is malicious. In addition, since the senders would not include any delays less than the unwrap time, the resultant delay distribution is inherently not an exponential distribution by Equation 2.2. This is because the smallest value a sender can select is at least the unwrap time. Visually, it results in the area of the delay distribution on the left-hand side of the unwrap time to be 0. Such information can be advantageous to the adversary. Strategy 3 passes control to the mix as the mix would essentially be able to delay the packet by arbitrary period of time. Thus a malicious mix could escalate the situation into denial-of-service attacks. Further mechanisms are required to govern the behaviour of the mixes to ensure proper functionality of the mixnet.

In the remaining sections in this report, we focus on strategy 1 of approach 1 and approach 2 in detail as interesting phenomena occurs for them. In particular there is a section dedicated to introducing two types of packet distinguishing attacks, described in this section. These two distinguishing attack scenarios can take place at each mix, in the presence of an adversary who is observing the inputs and outputs of the mix.

3.3 Structure of Experiment Framework

The experiment framework builds on MiXiM to specifically simulate the network aspect of a layered continuous-time mixnet, since MiXiM supports different mixnet topolgies and design options as mentioned in Section 2.5. The mixnet consists of three principal components: end user, mixes, and packets. Their interactions within the mixnet are as such: the end users send packets to each other by relaying the packets through several mixes - the traversal to each mix is a *hop*. The network of mixes are responsible for routing the packets, organised in a stratified topology (more details in Section 2.1).

To address the research questions in section 1.2, we need to measure the unwrap times of the mixes for each packet they receive. As part of the open-sourced code for the Sphinx packet from Nym Technologies, a benchmark module on the creation and unwrapping of a Sphinx packet is also available [2]. Therefore, to construct the experiment framework for this research, the functionalities of MiXiM and the Sphinx packet unwrap benchmark should be combined such that the unwrap time of a packet arriving at a mix can be obtained and recorded extemporaneously.

3.4 Modifications on MiXiM Framework

We obtain the experiment framework for the experiments by adding the Sphinx packet unwrap benchmark to MiXiM. To achieve this, a Rust binding built with the `PyO3` library is used. In this section, the modifications to MiXiM and the resultant workflow of the framework are detailed.

3.4.1 PyO3 Library for Rust Bindings

Since MiXiM is implemented in Python and the Sphinx packet benchmark in Rust, a method is required to combine their functionalities. An approach could be to implement

MiXiM in Rust so that the network aspect of the mixnet is compatible with the Sphinx unwrap benchmark. Considering the overhead in converting the mixnet simulator into Rust, it is common to leverage a language binding framework to bridge two different programming languages. One of such framework is the `PyO3`[1] framework which enables us to create *Rust bindings* - native Python modules for Rust binaries. Therefore, for the purposes of this work, it is possible to take advantage of the ease of use of the Python programming language and the high performance of Rust to incorporate the Sphinx unwrap benchmark into MiXiM via a Rust binding. Another framework that could be used is `rust-cpython`[2], the precursor of `PyO3`. `rust-cpython` has the advantage of offering a greater degree of flexibility offered to the users with some computational overhead than `PyO3`, due to differences in their implementations[3]. For the purpose of this work, we argue that it is sufficient to create Rust bindings using `PyO3` as the types in Rust that are exposed to Python runtime are minimal in the experiments - see Section 3.4.2 for a more in-depth description on the intergration of the Rust binding. Once the Rust binding is created, it can be imported into the code section of the mix in MiXiM such that the the Rust binding can be invoked, in a similar fashion to a function call in Python, and subsequently the compiled Rust code can be executed.

3.4.2 Design of the Rust binding

As the benchmark for the time taken for cryptographic operations on the Sphinx packets are separated from the mixnet simulator, the packets transmitted in MiXiM are not Sphinx packets, with reasons detailed in Section 2.5. Therefore, we simulate the unwrapping of a Sphinx packet by calling the Sphinx packet unwrap benchmark in Rust. At each invocation to the Sphinx packet unwrap benchmark, the time taken to unwrap a single layer of the packet is measured in Rust as follows:

1. When the Rust binding is triggered by the function call in Python, the code in Rust begins to run. Firstly, the time instant is recorded, denoted as `start_time`.
2. The function to unwrap a layer of the Sphinx packet is invoked.
3. Once the unwrap function in the Sphinx packet benchmark returns, the time at that instant is recorded again, and denoted as `end_time`.
4. The final elapsed time for unwrapping is defined as `end_time - start_time`. This elapsed time is measured in microseconds

Note that all these operations are executed in Rust; the Rust binding only returns the lapsed time for a single invocation to unwrap a layer of the Sphinx packet, measured in microseconds. (Note that units are converted where necessary during analysis) This design has several benefits. It introduces a simpler overall design as there is a direct mapping of any integer types in Rust to `int` in Python, which also reduces the computational overhead[4]. Moreover, this maximises the operations performed in Rust

[1] https://pyo3.rs/v0.18.2/

[2] https://github.com/dgrunwald/rust-cpython

[3] https://depth-first.com/articles/2022/03/09/python-extensions-in-pure-rust-with-rust-cpython/

[4] https://www.infoworld.com/article/3687744/how-to-write-python-extensions-in-rust-with-pyo3.html

which is more performant than Python.

3.4.3 Logging System

The logging system in MiXiM is extended to record the delays of each packets at each mix. This includes: (1) the sender-selected delay, (2) the unwrap time of the mix on the packet, and (3) the actual delay of the packet at the mix.

Sender-selected delay This is the intended delay (in seconds) that is encoded in a list with the packet, in the order at which the packet is routed through those mixes. Each mix on a packet's route from the sender to the receiver belongs to separate layers of the mixnet. This is because of the stratified topology detailed in section 2.4.1. Hence the mix accesses the packet's delay at the mix, by the layer at which the mix is located. For example, a packet arrives at its second hop which is on the second layer of the mixnet, so the mix accesses the second element in the list of delays. This design is the part of the original MiXiM implementation.

Unwrap time The unwrap time (in microseconds) of the mix on the packet according to the time returned by the Rust binding which measures the time taken to unwrap a layer of a Sphinx packet. This represents the taken taken to unwrap the packet in the MiXiM simulation as if they were Sphinx packets.

Actual packet delay The actual delay (in seconds) that the packet experiences at the mix, taking in consideration the unwrap time of the mix on the packet. Different values are recorded, depending on the approaches taken towards the additional packet unwrapping overhead, with further details in Section 3.2.

3.4.4 Modifications to assumption in MiXiM literature

Guirat and Diaz view the time for the average network propagation and the packet processing holistically as a constant factor of 50 ms. In this work, we separate the packet processing time in finer granularity as the packet unwrap time and the packet transformation time. We consider the network propagation and the packet transformation time as one single factor of 50 ms, and evaluate the packet unwrap time empirically. We argue that the value of 50 ms is reasonable as packet transformation is a cryptographic operation, so it can be encapsulated in the time duration of 50 ms.

3.5 Simulator Workflow

The design basis of the experiment framework follows the one of MiXiM [4] as the cryptographic benchmark is incorporated into MiXiM. In this section, we outline a summary on the workflow of MiXiM for continous-time mixnet in a stratified topology and any additions to the workflow for the purpose of the experiments.

First, the configurations for a stratified topological continuous-time mixnet is defined in the configuration file. Once the simulation environment is instantiated with the mixes and end-users, according to their description in the configuration file, MiXiM begins the burn-in phrase until the network is stable - where the average number of packets

transmitted and received by the mixes are at the level specified in the configuration file. Since the goal of the simulation is to record the actual delay for each packet at the mix when considering the cryptographic operation to unwrap the Sphinx packet. Therefore for each packet delay, there is a function call to the Sphinx unwrap benchmark.

MiXiM logs packet informations to evaluate the anonymity of the packets in the stable network. The extension to the logging system described in Section 3.4.3 enables the data collection required for this project. The log files produced on the delays at the mixes can then be further analysed.

3.6 Test Machines

This section specifies the specifications of tests machines on which the experiments are conducted as follows:

1. AMD EPYC 7302 16-Core Processor @ 3.00 GHz with 64GB of system memory
2. Apple M2 8-Core Processor @ 3.49GHz with 16GB of system memory
3. AMD EPYC 7302 16-Core Processor @ 3.29 GHz with 512GB of system memory
4. Raspberry Pi 3 Model B @ 1.20 GHz with 1GB of system memory

The machines selected for this work have a range of processing power: test machines 1 and 3 are powerful compute servers whilst test machine 2 is a laptop and test machine 4 is a small-sized microcomputer. The same simulations with identical configurations and packages are run on all the test machines.

Test machine aliases For readability for the remaining report, test machine 1 is simply referred to as *lab machine*, test machine 2 as *Macbook*, test machine 3 as *DICE (SC)*[5], and test machine 4 as *Raspberry Pi*. This section serves as a reference point for their specifications.

[5]DICE is a computing environment available through the Edinburgh University School of Informatics, https://computing.help.inf.ed.ac.uk/what-is-dice. SC stands for the ”student.compute” server.

Chapter 4

Experimental Setup

In this chapter, firstly a summary of key notations is provided. Then the configuration of the mixnet parameters of the simulator is outlined and justified, since this setup is used for all experiments.

4.1 Summary of Key Notations

This section details the key notations used extensively in the remaining of this report and their descriptions.

- μ: The parameter for the exponential distribution at which the delays are randomly sampled; it is represented by $\mu = \frac{1}{\lambda}$.
- λ: The average delay (per second) of the packet at a mix; it is represented by $\lambda = \frac{1}{\mu}$.
- $\lambda_{enduser}$: the rate of packets (per second) at which the traffic from end users enters the mixnet.
- d_{E2E}: the end-to-end latency (in seconds) of the packets from the senders to the receiver.
- u: the unwrap time (in seconds) of a mix on a packet.
- L: the number of layers in the mixnet in a stratified topology.
- W: the width of the layers; the number of mixes in a layer.

4.2 Configuration of the Experiment Framework

The details of the mixnet simulation parameters are defined in the configuration file, which determines the deployment and adversarial scenario for the experiments. We consider the scenario where all the mixes are honest. Therefore, none of the mixes are controlled by the adversary. In terms of the number of layers, Guirat and Diaz [14] demonstrated that for a network GPA that does not corrupted any mixes in the network,

the average entropy, which indicates anonymity, peaks optimally when $L = 2$ for all end-to-end latencies. It confirms that adding a two-layered mixnet achieves the best possible mixing when the mixnet itself is honest [14]. Hence we consider that the value for the number of layers is $L = 2$. For the width of the layers, the mixnet is assumed to be balanced, according to Section 3.1, indicating that all the layers have equal width. Guirat and Diaz proposed that for adversary that controls a constant proportion of mixes or no mixes at all, the optimal width of the layers is the minimum number of mixes required to efficiently route traffic in the network. They showed that for the scenario where $L = 3$ and $d_{E2E} = 1$, the average entropy peaked at $W = 10$, which is the minimum width evaluated [14]. Such findings are reproduced and we verify that for $L = 2$, a similar pattern of decreasing average entropy is displayed as the width increases. We expand on these results in Section 4.3. For the number of end-users (clients) in the simulation, we reason that a set of 100 end-sers is suitable as it is greater than the number of mixes in the network.

Moreover, designs, such as Nym, Loopix and others, leverage cover traffic to maintain the traffic volume for anonymity within the mixnet at all times even when fewer end users are transmitting packets [11, 21, 25]. We consider that the level of input traffic is sustained to a consistent level throughout the duration of the simulation. Therefore, it is not necessary to include additional cover traffic in this work, and we leave the inclusion of cover traffic as future work.

For the user traffic in the simulation, it is modelled as a Poisson process with parameter $\lambda_{enduser}$. One of the services Nym network performs well in is blockchain networks and it is utilised in cryptocurrencies. Two popular cryptocurrencies are Bitcoin and Ethereum. Bitcoin can process 7 transactions per second as maximum throughput [7] whilst it is around 15 transactions per second for Ethereum. Hence in this work, we focus on low user traffic and set the value of $\lambda_{enduser}$ to be 15 packets per second.

The mean of the exponential distribution denotes the average per-mix delay of the packet. The end-to-end latency d_{E2E} is directly proportional to the mean of the exponential distribution from which the per-mix delays are sampled. Therefore d_{E2E} is varied across the experiments to record the effects of per-mix delay in combination with the unwrap time u. Note that in this report, we represent the mean of the exponential distribution with the Greek letter λ, whereas the Greek letter μ is used in the work of Guirat and Diaz in [14]. In line with [14], the average end-to-end latency is directly proportional to the average per-mix latency, i.e. $d_{E2E} \propto \lambda$, and is represented by Equation 4.1.

For a L-layered mixnet with network propagation and packet transform time as γ, the average end-to-end latency d_{E2E} of a packet traverses a L-hop route with $L+1$ links is defined as:

$$d_{E2E} = \lambda L + \gamma(L+1) \tag{4.1}$$

For the experiments, we consider values of d_{E2E} in seconds of 0.15000001, 0.16, 0.2, 0.25, 1, 5 and 10. As $\gamma = 50$ms, the range of possible values for d_{E2E} is when $d_{E2E} > 15$ for a 2-layered mixnet. For example, $d_{E2E} = 0.15000001$ second results in $\lambda = 5 \times 10^{-9}$ second, and $d_{E2E} = 10$ seconds results in $\lambda = 4.925$ seconds. Therefore, the d_{E2E} values selected vary from ones which gives extremely low per-mix latency to latency suitable

for communications.

4.3 Verification of Chosen Parameters

Before integrating the Sphinx packet unwrap benchmark to the mixnet simulation via the Rust binding, the selected mixnet parameters for the simulator is verified to be in line with previous work. Guirat and Diaz show that for $d_{E2E} = 1$, $L = 3$, and $\lambda_{enduser} = 5000$, the average entropy maximised when $W = 10$. Therefore we propose two scenarios with low user traffic: (1) when $d_{E2E} = 1$, $L = 3$, $\lambda_{enduser} = 15$, (2) when $d_{E2E} = 1$, $L = 2$, $\lambda_{enduser} = 15$ for various layer widths and verify that the findings of Guirat and Diaz are applicable.

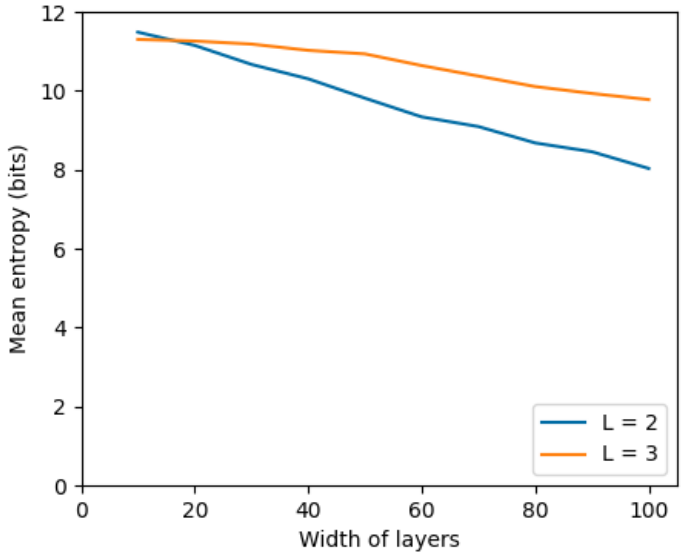


Figure 4.1: The mean entropies when $d_{E2E} = 1$, $\lambda_{enduser} = 15$, and the cases when $L = 2$ (blue) and $L = 3$ (orange), for various widths of the mixnet layers.

Figure 4.1 presents the results: both scenarios demonstrate that the optimal width is $W = 10$, and the mean entropy is greater when $L = 2$; in both cases, there is a gradual reduction of mean entropy as the width increases. This pattern is in line with the investigation of Guirat and Diaz, which they conclude that it is also due to the thinner traffic per mix. Note that the mean entropies for low user traffic ($\lambda_{enduser} = 15$) decrease by around 2 bits compare to the same configuration at higher user traffic rate ($\lambda_{enduser} = 5000$), which confirms that anonymity relies on the presence of large volumes of traffic. For the scenario of $d_{E2E} = 1$, $L = 3$, and $\lambda_{enduser} = 15$, the optimal value of width for maximum entropy is when $W = 10$. This is an analogous finding when $d_{E2E} = 1$, $L = 2$, and $\lambda_{enduser} = 15$. Therefore we verify the anonymity of the chosen mixnet configurations; we also show that lower user traffic displays a reduced yet comparable level of anonymity to high user traffic regarding the average entropy.

Chapter 5

Results and Discussions

In this chapter, we conduct the experiments to address the research questions addressed in Section 1.2.

5.1 Average Unwrap Times

As a preliminary experiment, the average unwrap time of the mix is evaluated on the test machines detailed in section 3.6. The purpose of this experiment is to analyse empirically the possible range of unwrap times of the mixes induced by different hardware specifications. The unwrap time of the mix is defined as the execution time for an invocation of the Sphinx packet unwrap function in Rust from Python runtime.

5.1.1 Experiment Overview

The experiment consists of two parts. The first part acts as an experimental baseline: each test machine records the time duration of invocation to an empty function in Rust from Python runtime. For the second part, the unwrap time of a Sphinx packet on each test machine is measured.

Since the underlying probability distribution of the unwrap time of a mix running on a particular machine is unknown, the unwrap time is modelled as a random variable with unknown distribution. The central limit theorem (CLT) establishes that with sufficiently large sample size, the sampling distribution of the sample mean approximates a normal distribution regardless the actual distribution of the variable, indicating that the average unwrap time of the mix approximately follows a normal distribution given a large sample. Therefore, in terms of the number of iterations and repetitions of the experiment, we consider 10,000 invocations of the empty function and the unwrap time function, and the process is repeated 20 times as they have been found to give meaningful results that follow the CLT [23]. Moreover, the greater the sample size, the greater the precision permitted.

Test Machine	Empty Rust Function (mean ± std. dev.)	Unwrap Sphinx Packet (mean ± std. dev.)
AMD EPYC 7302 16-Core Processor @ 3.00 GHz (64GB RAM)	0.5414 μs ± 5.6 ns	5523.6380 μs ± 11.2527 μs
Apple M2 8-Core Processor @ 3.49 GHz (16GB RAM)	0.6352 μs ± 105.5 ns	7576.2365 μs ± 45.1623 μs
AMD EPYC 7302 16-Core Processor @ 3.29 GHz (512 RAM)	0.5477 μs ± 30.6 ns	5805.7422 μs ± 51.4828 μs
Raspberry Pi 3 Model B @ 1.20 GHz (1GB RAM)	4.4483 μs ± 58.1 ns	59176.1129 μs ± 179.4019 μs

Table 5.1: Elapsed time of calling an empty function in Rust (as control) and Sphinx unwrap packet on different machines

5.1.2 Implementation

The experiment module is built with the `timeit`[1] library. The execution time of the calls to the empty function and the unwrap function is measured by the `timeit()` function in the library. By the design of the `timeit()` function, the execution times of all the invocation to the testing function are aggregated over the iterations. The mean unwrap time on the test machine can then be calculated. The 95% confidence interval of the mean is also computed using the `stats.norm.interval()` function of the normal distribution from the `scipy`[2] library, as the distribution of the average unwrap time approximates the normal distribution with large samples.

5.1.3 Results

From this experiment, a general view of the unwrap time on each of the test machines is illustrated. A summary of the results is shown in Table 5.1. For each part of the experiment, we have listed the specifications of the test machines and the mean times on the corresponding test machines and their standard deviations.

As a preliminary overview on the results, the Raspberry Pi dominates all aspects of the experiment, recording the highest mean elapsed time for the empty Rust function and the Sphinx packet unwrap function. From Table 5, to complete an invocation to the empty function, it takes on average 4.4483 microseconds on the Raspberry Pi, yielding a 7-fold increase from the Macbook and a near 8-fold increase from the lab machine and DICE (SC). In terms of the unwrap time, the average unwrap time on the Raspberry Pi is 59176.1129 microseconds. That is a near 8-fold increase from Macbook, and near 10-fold increase from DICE (SC) and the lab machine. Notice that both aspects of the experiment, the Raspberry Pi and the Macbook record the greatest variations.

The average unwrap times and times of the empty function are now examined with

[1] A built-in Python library for measuring execution time of small number of lines of code, https://docs.python.org/3/library/timeit.html

[2] A Python library for scientific computing, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html

	Empty function				Unwrap function			
Test case	**Variance ratio**	**t-test type**	**t-statistic**	**p-value**	**Variance ratio**	**t-test type**	**t-statistic**	**p-value**
1. Macbook and lab machine	60.1	Welch's	0.83	0.42	605.7	Welch's	274.60	8.92×10^{-46}
2. Macbook and Raspberry Pi	87.3	Welch's	-286.02	1.08×10^{-36}	36.0	Welch's	-1236.81	1.94×10^{-50}
3. Macbook and DICE (SC)	101.6	Welch's	0.04	0.97	1.7	Student's	142.79	1.71×10^{-53}
4. Lab machine and Raspberry Pi	1.5	Student's	-221.95	9.15×10^{-61}	204.3	Welch's	-1300.34	6.55×10^{-49}
5. Lab machine and DICE (SC)	1.7	Student's	-0.47	0.64	9.4	Welch's	-47.79	1.65×10^{-24}
6. Raspberry Pi and DICE (SC)	1.2	Student's	195.60	1.11×10^{-58}	21.6	Welch's	1263.85	3.44×10^{-52}

Table 5.2: A summary of the *t*-tests performed on pairs of the test machines on the average empty function time and the average unwrap time with significance level of 0.05 and degrees of freedom of 19.

independent samples *t*-tests. Note that *t*-tests are valid when the data is continuous and follows the normal distribution. For this case, the average times satisfy both requirements, the latter condition is by the application of the CLT. Each sample is collected on a different test machine so the collected average times are independent. Hence independent samples *t*-tests are suitable. Then, the variances of the data from the 20 repetitions are tested so that the relevant *t*-test is applied according to the common practical guidance of using Welch's *t*-test when the ratio between two samples are greater than four[3].

5.1.3.1 *t*-tests

Empty function

As stated in Table 5.2, for cases (1) to (3), Welch's *t*-test is applied as their ratios of variances are greater than four, whereas Student's *t*-test is conducted for cases (4) to (6) since their ratios are less than four. The *null hypothesis* for each test is defined as the true difference between the average time taken for the empty function across pairs of test machines is zero. The results for the *t*-tests are summarised in Table 5.2

We find that, whilst the average time for the lab machine is marginally smaller than the one for DICE (SC), it is insignificant to cause the difference between the average time taken for empty function on the lab machine and DICE (SC) to be zero, as there is insufficient evidence to reject the null hypothesis ($t[19] = -0.47, P = 0.64$). A similar conclusion of difference in average times is statistically insignificant is drawn for cases (1) and (3) where the *t*-test result are $t[19] = 0.83, P = 0.42$ and $t[19] = 0.04, P = 0.97$ respectively. It is worth noting that the difference in average times for the empty function between Raspberry Pi and all other test machines are significant for *t*-tests with degrees of freedom of 19 and the significance level of 0.05.

Sphinx packet unwrap function

As with the previous part of the experiment, the ratio of the variances are computed in order to categorise the applicable *t*-tests. We find that only case (3) of Macbook and DICE is suitable for Student's *t*-test. Once again, the *Null hypothesis* is the true difference between the average unwrap time of the mixes running on pairs of test machines is zero; the degrees of freedom are 19 and the significance level is 0.05. The summary of the *t*-test results is in Table 5.2. It indicates that all the *p*-values of the

[3]https://online.stat.psu.edu/stat415/lesson/3/3.2

t-tests are less than the significance level of 0.05; there is sufficient evidence to reject the null hypothesis. Therefore there is a statistically significant difference between the unwrap times of any pair of test machines. In conclusion, we propose that it is important to consider the hardware specifications of the potential machines which are operating mixes and their various unwrap times when analysing the infrastructure.

5.1.4 Interpretations and Discussion

The average time of calling an empty function is the longest for the Raspberry pi, whereas the other machines have consistent and shorter average elapsed time for calling the empty function, indicating that the Raspberry Pi is less performant than the rest. Similar findings can also be seen for the average Sphinx packet unwrap times. These results are expected, as referring to their hardware specifications in Section 3.6, the Raspberry Pi has the least amount of processing speed and system memory which limits its performance. The performance of Raspberry Pi is also in line with the results in the experiment of the empty function. The Raspberry Pi is significantly slower than the other test machine statistically. The performance of the Macbook is comparable to the computing servers. Moreover, the difference in the average time between the Raspberry Pi and all other test machines in both baseline test and unwrap function is significantly large at 95% confidence. Whilst the lab machine, Macbook, and DICE (SC) produced average times of the same magnitude for the unwrap time, the differences in average time are statistically significant. Specifically we conclude the following interpretation: the true average unwrap time of a mix is significantly different on all test machines evaluated. Thus, we establish the significance of the choice of the hardware specification to operate a mix on the unwrap time that is imposed on the packets arriving at the mix. To conclude, machines with similar magnitiude of processing speed can nevertheless produce significantly different average unwrap times. Hence cautious is required when the evaluation of the mixnet considers the unwrap time.

In addition, the standard deviation is included as a measurement of variation for both aspects of the experiment. As shown in Table 5.1, the lab machine has the least degree of variation whilst the Macbook has the greatest spread in the data. We suggest that the variance is introduced by the system load of the machine at the time of test.

5.2 Actual Packet Delay Exponentiality Test

In this section, the actual delays of the packets during the simulation is examined to test whether they fit the exponential distribution. Recall that the actual delay of the packet is represented as the overall delay the packet experiences at the mix, taking into account the extra unwrap time. Having established the unwrap time of the mix is significantly different on different test machines, the purpose of this test is to determine whether the probability distributions of the overall delays of the packets retain the memoryless property of the exponential distribution on different test machines.

5.2.1 Experiment Overview

This experiment examines the actual delay of the packets for both approaches 1 and 2 referred to in sSection 3.2. In approach 1, the mix attempts to counteract the overhead incurred by unwrapping the packet whilst in approach 2, the mix proceeds to delay the packet by the sender-intended delay after unwrapping it. This results in a difference in the actual delay - the overall delay that a packet undergoes according to the approach taken. Therefore the experiment consists of two parts; the simulation is run for both approaches and separate log files are produced for analysis.

Defining more formally, M_l is a mix on the l-th layer of a L-layered mixnet, where $1 \leq l \leq L$. Let u_{il} be the unwrap time in seconds of the mix M_l on any packet i that arrives at the mix during the simulation, and let d_{il} be the delay in seconds of the packet i at the mix M_l. For the scope of this work, we consider that the mix M_l forwards packet i to its next destination immediately with negligible processing time. Hence we define the total time the packet i dwells in the mix M_l - the *actual delay* of the packet as U_{il}.

For approach 1, the mix deducts the unwrap time u_{il} from d_{il} such that the actual delay of the packet i, U_{il}, is generally represented as $U_{il} = d_{il} - u_{il}$. This notation is extended in Section 5.2.2 for the different consequences that arise with this strategy of the mix according to Section 3.2. For approach 2, the time at which the mix M_l begins to consider the delay d_{il} is after the unwrap time u_{il} has elapsed. The actual delay is represented as $U_{il} = u_{il} + d_{il}$ for approach 2.

The experiment utilises the experiment workflow detailed in Section 3.5. On each of the test machines listed in Section 3.6, the identical mixnet configuration is applied. The extended logging system for recording delay information in the mixnet simulation is detailed in Section 3.4.3. It collects the sender-selected delay, the unwrap time of the mix for the packet received, the actual delay for every packet transmitted in the duration of the simulation. We repeat the experiment with different end-to-end latencies in seconds for every test machine, i.e. $d_{E2E} = 0.15000001, 0.16, 0.2, 0.25, 0.5, 1, 2.5, 5$ and 10, resulting in 36 runs in total.

5.2.2 Results

In this section, the results of experiment on the test machines are presented in two parts, first on approach 1 and then on approach 2 of relating to the effect of unwrap time of the mix with the test machines.

5.2.2.1 Approach 1

Approach 1 as detailed in Section 3.2, is when the mixes counterbalance the additional delay incurred by their packet unwrap time by reducing the sender-intended delay. The strategies are related to the over-delayed packets - the situation where the per-mix delay encoded from the sender is shorter than the unwrap time of the mix, causing the packet to dwell longer than intended at the mix. Sending strategies are also proposed in Section 3.2 to address when a mix should relay these over-delayed packets. For the scope of this work, strategies 1 is examined; the mix transmits the over-delayed packet

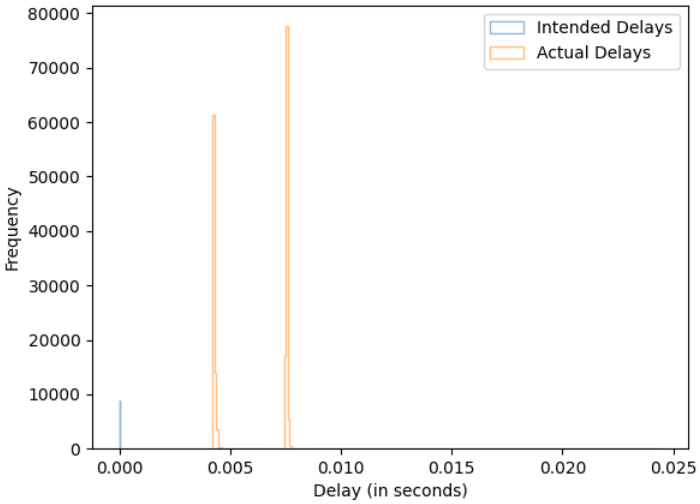


Figure 5.1: Distribution of the actual delays (orange) against the distribution of the sender-intended delays (blue) for $d_{E2E} = 0.15000001$ under strategy 1 of approach 1, on the Macbook.

immediately.

We start by considering strategy 1 and produce histogram plots of the distributions of the actual delays of the packet and their sender-intended delays. Histrograms are suitable as the number of intended delays and actual delays observed are the same so the plots are comparable. The number of bins in the histogram can affect the visual structure of the produced plots. Since the experiment is conducted for 9 different values of end-to-end latency, please refer to Appendix A for all the figures of the delay distributions of the evaluated values of end-to-end latency. Upon inspection, the distributions of the actual packet delays for $d_{E2E} = 0.15000001, 0.16, 0.2$ and 0.25 are visibly not exponentially distributed.

Lab machine For $d_{E2E} = 0.15000001$, the intended delays are centered around 0 - this is expected as the average per-mix delay λ is 5×10^{-9} second, calculated by Equation 4.1. We use the `fitter`[4] library to fit 50000 data points of the actual delays selected at random; a subset is necessary due to the large volume of data and the slow fitting process. 110 distributions are fitted on the data and We select Johnson's S_U-distribution as the best fit by the sum of squared errors. For $d_{E2E} = 0.16, 0.2$ and 0.25, the distributions have a sharp peak at their respective expected value of the unwrap time of the mix when running on the lab machine of 0.0055 second. We denote that the end-to-end latency to be *small* under such scenario, i.e. the end-to-end latency is small when $d_{E2E} \leq 0.25$ for the lab machine. The actual delay distribution begins to the resemble the intended delay as the end-to-end latency increases gradually to 10 seconds.

Macbook For $d_{E2E} = 0.15000001$, the distribution of actual delays is bimodal with

[4]https://fitter.readthedocs.io/en/latest/

			Lilliefors test results	
Test machine	d_{E2E}	Over-delayed packets (%)	Test statistic	p-value
Lab machine	0.15000001	100.00	0.623	<0.001
	0.16	66.81	0.416	<0.001
	0.2	19.65	0.125	<0.001
	0.25	10.36	0.083	<0.001
	0.5	3.09	0.028	<0.001
	1	1.29	0.013	<0.001
	2.5	0.46	0.005	<0.001
	5	0.22	0.002	0.043
	10	0.12	0.002	0.399
Macbook	0.15000001	100.00	0.498	<0.001
	0.16	57.88	0.361	<0.001
	0.2	15.96	0.070	<0.001
	0.25	8.42	0.111	<0.001
	0.5	2.46	0.024	<0.001
	1	1.02	0.010	<0.001
	2.5	0.36	0.004	<0.001
	5	0.18	0.004	<0.001
	10	0.08	0.002	0.222
DICE (SC)	0.15000001	100.00	0.600	<0.001
	0.16	69.72	0.417	<0.001
	0.2	21.37	0.109	<0.001
	0.25	11.22	0.080	<0.001
	0.5	3.40	0.031	<0.001
	1	1.41	0.014	<0.001
	2.5	0.50	0.006	<0.001
	5	0.24	0.003	0.006
	10	0.13	0.006	0.038
Raspberry Pi	0.15000001	100.00	0.629	<0.001
	0.16	100.00	0.630	<0.001
	0.2	90.64	0.566	<0.001
	0.25	69.38	0.436	<0.001
	0.5	28.93	0.145	<0.001
	1	13.20	0.101	<0.001
	2.5	4.73	0.047	<0.001
	5	2.42	0.024	<0.001
	10	1.34	0.013	<0.001

Table 5.3: Summary of the percentage of over-delayed packets and the Lilliefors test results of the actual delay and intended delay distributions at significance level of 5% for various end-to-end latencies on the test machines under strategy 1 of approach 1.

one peak centred around 0.0043 and the other centred at 0.0076 (Figure 5.1). From inspecting the collected 180072 unwrap times, we find that the unwrap times are split into two groups. At the beginning of the simulation, the unwrap times are consistently around the mean unwrap time of 0.0076 second for the first 56% of prackets. Then the unwrap times fall and maintain at a lower value with mean unwrap time of 0.0043 second. Such step-like decrease in unwrap time result in a bimodal distribution which matches Figure 5.1, as the intended delays are much smaller than the unwrap time. The mean intended delay is 5.0×10^{-9} second - a negligible proportion of either of the mean unwrap times. The larger proportion of greater unwrap times explains the higher peak centred at 0.0076. Similar to the lab machine, We consider the end-to-end latency to be small when $d_{E2E} \leq 0.25$ for the Macbook. The peak widens as the end-to-end latency increases and the actual delay distribution eventually is similar to the intended delay distribution.

DICE (SC) The peaks in the actual distributions of $d_{E2E} = 0.15000001$ and 0.16 are less well-defined than in the cases of the lab machine and the Macbook. For all evaluated end-to-end latencies, the highest peak centred near the unwrap time of the mix at 0.0061 indicates greater number of packets with actual delay similar to the unwrap time (see Appendix A.3). Consistent with the previous test machines, the end-to-end latency is small when $d_E2E \leq 0.25$ as there is a noticeable peak centred at the mean unwrap time in the actual delay distribution for $d_{E2E} = 0.15000001, 0.16, 0.2$ and 0.25.

Raspberry Pi For the values of d_{E2E} evaluated, the distributions of actual delays are significantly different compare to the distribution of the intended delays for $d_{E2E} =$ 0.15000001 to 5.0 by performing the Kolmogorov-Smirnov two-sample test at 5% significance level (p-value = 0.00). The corresponding figures in Appendix A reflect such findings as the area of overlap between the distributions reduces as the end-to end latency shortens. There is insufficient evidence to reject the null hypothesis of that the actual delays and intended delays are drawn from the same continuous distribution (p-value = 0.25). Therefore we conclude that the end-to-end latency is small when $d_{E2E} \leq 5$.

Lilliefors Test As the actual delay distribution approximates the distribution of the intended delays when the end-to-end latency increases, we perform the Lilliefors test for an exponential distribution [19] to determine whether the distribution of the actual delays approximates an exponential distribution. The Lilliefors test is a goodness-of-fit test that is a variant of the Kolmogorov–Smirnov test. Such test is valid when the mean and the standard deviation of the population distribution is unspecified, hence it is suitable as the underlying distributions of the actual delays and their true parameters are unknown.

The *null hypothesis* of the Lilliefors test is that the actual packet delays are follow an exponential distribution. The test assumes that the actual delays are consistent with an exponential distribution and estimates the mean of the exponential distribution using the observed actual delays. Table 5.3 presents the test results. The Lilliefors test statistic measures the maximum distance between the empirical distribution of the actual delays and the exponential distribution. Therefore a low value indicates that the empirical distribution is good-fitted to an exponential distribution. As the Lilliefors

test statistic decreases with the increasing end-to-end latency, the difference between the actual delay distribution and the exponential distribution are statistically significant with all evaluated values of d_{E2E} for DICE (SC) and Raspberry Pi at 95% level. For the lab machine and the Macbook, we fail to reject the null hypothesis for $d_{E2E} = 10$, so we conclude that the actual delay distribution bears resemblance to an exponential distribution with 95% confidence level.

5.2.2.2 Approach 2

Approach 2 is, comparably, the simpler scenario. As described in Section 3.2, approach 2 that is the unwrap time of the mix is not considered when the mix proceeds to delay the packet by the sender-intended delay encoded in the Sphinx packet header. Figures B.2 in the Appendix shows that the distributions of the actual delays of the packets is translated more to the right as the end-to-end latency d_{E2E} decreases; the distributions become similar in appearance to the exponential distribution as the latency increases. It is noticeable by the increasing overlap between the distribution with the rising latency such that the distribution of the actual delays for $d_{E2E} = 10$ resembles the exponential distribution of the intended delays. Moreover, similar to the finding of approach 1, the shapes of distributions of the actual delays for $d_{E2E} = 0.15000001, 0.16$ and 0.2 are clearly inconsistent with the exponential distribution by inspection.

To examine whether the distributions of actual delays follows an exponential distribution, we perform the Lilliefors test for the exponential distribution [19] at significance level of 5%. Similar to the analysis performed for approach 1, the *null hypothesis* of the Lilliefors test is that the observed (actual) delays of the packets follow an exponential distribution. The results of the goodness-of-fit test are presented in Table 5.4.

In Table 5.4, the difference in actual and intended delays for all end-to-end latencies except the end-to-end latency of 10 seconds, are highly statistically significant at 95% level of confidence. The rise of the test statistic of the Lilliefors test as the end-to-end latency decreases from 10 seconds, indicates a widening distance between the distributions of the actual delays and the exponential distribution. This pattern affirms the one observed in figures in Appendix B that the distribution deviates from the exponential distribution of the intended delays. The test statistic for the end-to-end latency of 10 seconds is 0.002 (at p-value of 0.222). The p-value of 0.222 is greater than the significance level of 0.05, thus there is insufficient evidence to reject the null hypothesis and we conclude that the actual delays are consistent with the exponential distribution at 95% significance.

5.2.3 Interpretations and Discussion

For approach 1, the sharp peaks in the actual delay distributions are observed when the end-to-end latency is small, such as in Figure 5.2. This is because the unwrap time of the mix dominates the overall delay of the packet at the mix. It results in over-delayed packets and are sent to their next destination immediately. The area at the right-hand side of the peak in the actual distribution is when the sender-intended delays of the packets are greater than the unwrap time of the mix, hence these packets are unaffected by the overhead of the packet unwrap. When the intended and actual delay

		Lilliefors test results	
Test machine	d_{E2E}	Test statistic	p-value
Lab machine	0.15000001	0.628	<0.001
	0.16	0.405	<0.001
	0.2	0.165	<0.001
	0.25	0.095	<0.001
	0.5	0.030	<0.001
	1	0.013	<0.001
	2.5	0.006	<0.001
	5	0.003	0.017
	10	0.003	0.038
Macbook	0.15000001	0.622	<0.001
	0.16	0.372	<0.001
	0.2	0.138	<0.001
	0.25	0.077	<0.001
	0.5	0.024	<0.001
	1	0.010	<0.001
	2.5	0.004	<0.001
	5	0.002	0.046
	10	0.002	0.233
DICE (SC)	0.15000001	0.600	<0.001
	0.16	0.396	<0.001
	0.2	0.165	<0.001
	0.25	0.097	<0.001
	0.5	0.032	<0.001
	1	0.014	<0.001
	2.5	0.005	<0.001
	5	0.002	0.045
	10	0.003	<0.001
Raspberry Pi	0.15000001	0.629	<0.001
	0.16	0.601	<0.001
	0.2	0.502	<0.001
	0.25	0.416	<0.001
	0.5	0.221	<0.001
	1	0.114	<0.001
	2.5	0.048	<0.001
	5	0.025	<0.001
	10	0.012	0.004

Table 5.4: Summary of the Lilliefors test results of the actual delay and intended delay distributions at significance level of 5% for various end-to-end latencies on the test machines under approach 2.

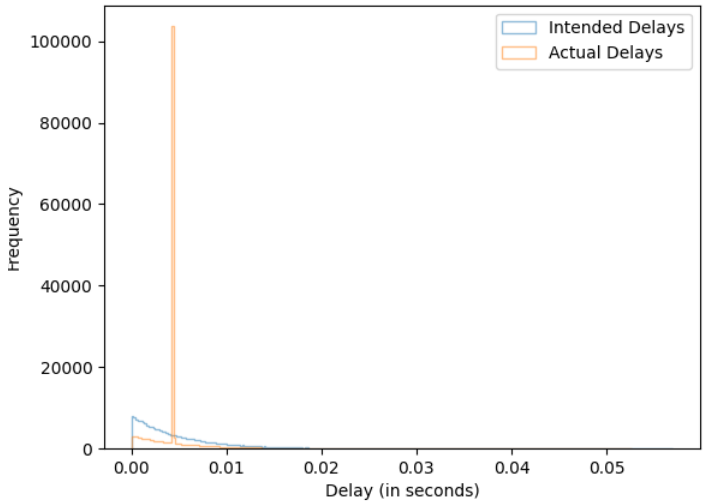


Figure 5.2: Distribution of the actual delays (orange) against the distribution of the sender-intended delays (blue) for $d_{E2E} = 0.16$ under strategy 1 of approach 1, on the Macbook.

distributions resemble each other, it indicates that the unwrap time is not the principle period of time for the actual delay of majority of the packets. Since there is no notion of over-delayed packets for approach 2 as the actual delay of a packet is simply the sum of the unwrap time and the encoded per-mix latency, there is no further implications. However, we can further interpret this: it indicates that the majority of the packets sent are not over-delayed when the end-to-end latency is *not* small; the unwrap time of the mix is less than the intended delay hence the mix is able to counteract its unwrap time by reducing the sender-intended delay appropriately.

The multimodal distributions for some of the actual distributions suggest that the system load on the test machine varied over the duration of the simulation. The unwrap times of the mixes are longer than the intended delay of the packets such that the unwrap time dominates the structure of the actual delay distribution. For example in the case of $d_{E2E} = 0.15000001$ on the Macbook, the distribution of the actual delays reflects the change in unwrap time, described under the results for Macbook in Section 5.2.2.1, in a bimodal distribution.

For approach 1, the actual delays approximately follow an exponential distribution when $d_{E2E} = 10$ for the lab machine and the Macbook at significance level of 5%, hence those actual delays preserve the memoryless property of the exponential distribution. However, for approach 2, the actual delays are only close to be distributed exponentially when $d_{E2E} = 10$ on the Macbook - for all the evaluated d_{E2E} values on other test machines, there is enough evidence to conclude that the actual delay distributions are significantly different to an exponential distribution. In conclusion, greater values of end-to-end latency should be utilised when the mixnet employs approach 2 to mitigate the effects of the unwrap overheads of the mixes, such that the actual delays approximates an exponential distribution. This is because the average per-mix latency λ of the packet

increases with the end-to-end latency. When λ increases, the probability of selecting a delay that is less than the unwrap time of the mix reduces. This is illustrated in Figure 2.1 where the PDF flattens when λ increases as the total area underneath the PDF is 1. Hence the probability of encountering over-delayed packets decreases.

Finally we address the possibility of the distribution of the actual delays ever achieve *true* exponential distribution. we argue that in real-world deployment, it is not possible to achieve an actual delay distribution that is a true exponential distribution which has the memoryless property since the unwrap time of the mix incurs a positive, non-negligible on every packet arriving at the mix. The unwrap time shifts the exponentially distributed delays, causing the resultant distribution be non-memoryless, by Equation 2.2. However, it is possible to reduce the effects of the unwrap time: (1) increase the per-mix delay and (2) decrease the unwrap time. Increasing the per-mix latency comes at the expense of increased end-to-end latency. It is also worth noting that it is not possible to completely eliminate the possibility of over-delayed packets even when the per-mix latency λ is large due to the exponential distribution being right-skewed. The area underneath the graph on the left-hand side of the mean is greater than the right-hand side, referring to Figure 2.1 for examples. Hence increasing λ can at most reduce the proportion of over-delayed packets encountered in a mixnet. Moreover, there is a limit on the human tolerance on the latency, for some real-time messaging application, it may not be acceptable for end-to-end latency to be sufficiently greater than 10 seconds. Reducing the unwrap time can be achieved by faster machines which is a motivation for future work.

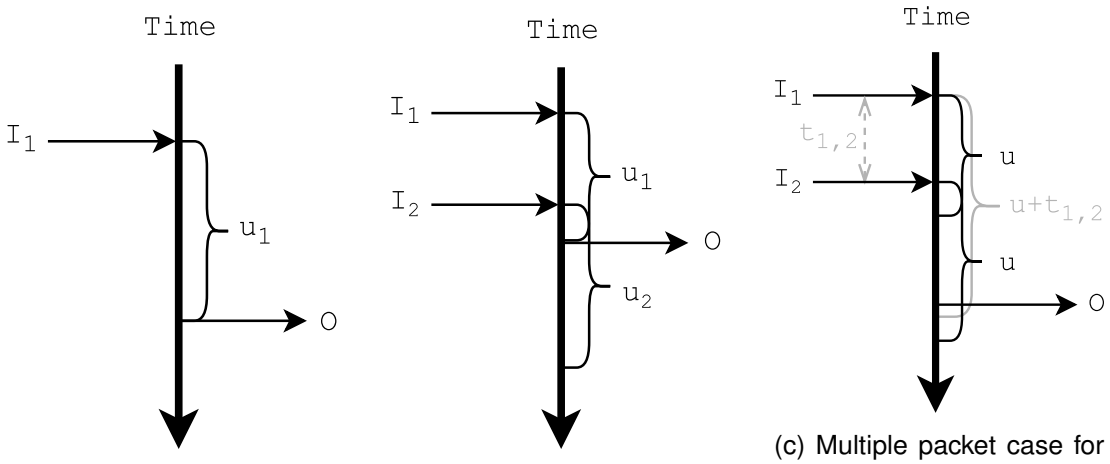
In the next section, we demonstrate the limitations of these approaches.

5.2.4 Packet Distinguishing Attacks

In this section, we address the implications of approaches 1 and 2 on the anonymity of the packets. As briefly mentioned in Section 3.2, there are two possible packet distinguishing attacks that a GPA can perform on the packets when observing arrival and departure times of the incoming and outgoing traffic of a particular mix.

The first attack involves a single packet arriving at a mix with an empty queue, i.e. no messages are waiting at the mix. This is similar to the attack described in [16] where the adversary is only able to correlate an outgoing packet if there are no other packets waiting or arriving at the mix during the delay time. We demonstrate that packets under strategy 1 of approach 1 and approach 2 are vulnerable to this attack.

Figure 5.3a depicts a timeline where there is an incoming Sphinx packet at time I_1, which the mixnode performs the cryptographic operation to unwrap the packet to reveal the delay selected by the sender - *unwrap time*, denoted as u_1. Outgoing packet is denoted as the packet at time O. In this scenario, the adversary is able to correlate packets at times I_1 and at O with non-negligible advantage, even though the outgoing packet had been cryptographically transform to provide bitwise unlinkability between packets at times I_1 and O - a property of Sphinx packet format. This is because the adversary can infer that the intended delay is shorter than the unwrap time u_1 elapses the actual delay; the packet has already been over-delayed at the mix, and is relayed to



(a) Single packet case (b) Multiple packet case (c) Multiple packet case for approach 2

Figure 5.3: Scenarios where the adversary can correlate the outgoing packet with an incoming packet it observed earlier.

its next destination immediately to avoid further delay, resulting in a first-in first-out, queue-like behaviour. Therefore there is no reordering of packets. The same can be applied to packets under approach 2, even though the delay of the packet is longer due to the mix is not counteracting its unwrap time, since the culprit for the attack is the empty queue of the mix.

Since both the packet sending rate and the packet delay time are modelled by two different Poisson processes (which are realised using exponential distributions), the probability of the queue being empty at time I_1 is calculated by $e^{-\lambda/\mu}$ by the method provided in [16]. Thus, for the experimental setup of the continuous-time mixnet detailed in Section 4.2, the parameter μ of the exponential distribution of the per-mix delay by the following formula, given L layers and d_{E2E}:

$$\mu = \frac{L}{d_{E2E} - \gamma(L+1)}$$

As presented Kesdogan *et al.* [16], the probability of the queue being empty when a packet arrives is calculated by $e^{-\lambda_{enduser}/\mu}$. Hence for $d_{E2E} = 0.15000001$, the probability of the packet at time I_1 entering an empty queue is:

$$\mu = \frac{2}{0.15000001 - 0.05 \times 3} = 200000000 \Rightarrow e^{-5000/200000000} \approx 1.0$$

Therefore the adversary has unassailable advantage in launching such attack when the per-mix latency is close to 0. Similarly, we calculate for $d_{E2E} = 0.16$ and the probability is $e^{-5000/200} \approx 1.4 \times 10^{-11}$. An 1.1-fold increase in end-to-end latency reduces the probability of empty queue by a multiple of 7.2×10^{10}. A further increase the end-to-end latency to 0.25 second and the probability is close to 0. Thus we conclude that any value of d_{E2E} slightly greater than the minimal end-to-end value is effective in preventing this first type of attack.

We now extend such attack to a queue with more than 1 packet waiting - over-delayed packets under strategy 1 of approach 1 are vulnerable to this type of attack. It is assumed that only two message are within the queue of the targeted mix, we argue that this is reasonable as the unwrap time of the mix is small. Figure 5.3b illustrates the scenario where there are two incoming packets at times I_1 and I_2 into the mix and an outgoing packet at time O, where $I_2 - I_1 < u_1$ and $O - I_2 < u_2$, u_1 and u_2 denotes the unwrap time for packets at times I_1 and I_2 respectively. We assume that the packet at time I_1 is an over-delayed packet. The adversary can identify the outgoing packet at time O with a specific previously-observed incoming packet, I_1 in this case, with probability greater than 50% (the adversary has probability of 50% of identifying successfully by guessing randomly). This is because with the knowledge of the unwrap times u_1 and u_2, the adversary has the advantage of inferring that the unwrap time of packet at time I_2 lapses the outgoing packet at time O - at time O, the mix would not have completed the unwrapping operation for packet at time I_2, hence the outgoing packet could not be the packet at I_2.

This does not have the memoryless property of the exponential distribution, recalling that the memory property indicates that the probability of packet at time O being packet at time I_1 is equal to the probability of being packet at time I_2, the adversary gains no additional information from the difference in arrival times of I_1 and I_2.

The probability of the occurrence of the second type of attack depends on the number of over-delayed packets. Table 5.3 includes the percentage of over-delayed packets for the evaluated values of d_{E2E}. The possibility of an adversary being able to successfully correlate the traffic falls as the end-to-end latency increases. For end-to-end latency greater than 2.5 second, there is less than 1% of over-delayed packets for all test machines except the Raspberry Pi. Since the Raspberry Pi is at least 10 times more over-delayed packets with the same mixnet configurations. Thus, we do not advise operating mixes on Raspberry Pi's, based on the results presented here. For the remaining test machines, we suggest that a mixnet with end-to-end latency greater than 1 second is acceptable as it was found that a webpage takes 10.3 seconds on average to be fully loaded on a desktop[5]. As mentioned in Section 5.2.3, there will always be over-delayed packets in the network due to the unwrap time. However, increasing d_{E2E} can effectively reduce the adversary's attack surface.

In addition, we discuss briefly that the packets under approach 2 can also be susceptible to the second type of attack. For this section, the adversary has knowledge of the average unwrap time of the mix, relaxing the assumption that the adversary knows every unwrap time of mix for each arriving packet. Such relaxation is suitable as it presents a strong yet more realistic adversary., whilst simplifies the analysis of the attack. The adversary can obtain the average unwrap time of the target machine offline. By the law of large numbers, which states that the sample mean of independently and identically distributed data converges to the theoretical mean of the underlying distribution as the sample size increases. This statement holds even when the underlying distribution is unknown such as the distribution of the unwrap times. Hence we also note that the attack analysis for approach 1 holds even when the adversary only gains knowledge on

[5]https://backlinko.com/page-speed-stats

the average unwrap time.

Suppose the average unwrap time of the mix is u. Let there be 2 packets arriving at the mix at times I_1 and I_2, for $I_1 < I_2$ and $I_2 - I_1 < u$. Let $t_{1,2}$ be the difference in time between the arrival times of the two packets. The sender-intended delays of the packets are unknown to the adversary. If a packet is emitted from the mix at time O for $u < O < u + t_{1,2}$. Then the adversary has non-negligible advantage in correlating the packet at time O with the packet at time I_1. Figure 5.3c illustrates the attack.

Chapter 6

Evaluation

This section provides an overall evaluation on the work in this project. The first section is on the overall system where the experiments are conducted, and the second is on the result and discussion section. We also discuss some technical difficulties encountered during the project and the solutions to resolve them.

6.1 Experimental Methodology, Framework and Setup

Mixnet The mixnet itself is assumed to be honest, with honest mixes and end users which are strong assumptions that are unlikely to reflect any real-world mixnets realistically. However, We consider these strong assumptions apt for the purpose of this project - to focus exclusively on the effects the additional unwrap time alone can cause to detriment the level of anonymity protection offered by the memoryless property of the exponential distribution. The layers of the mixnet are balanced as it has been demonstrated that it is preferred over mixnet with variable layer widths since it produces worst-cases that are less advantageous to the adversary in small networks. This is relevant as the mixnet in this work - 2-layered mixnet with layer width of 10 - is relatively small. These chosen parameters are demonstrated to be optimal in Section 4.3.

Choice of mixnet simulator Current literature presents two discrete-event mixnet simulators, the Loopix Simulator and MiXiM [4, 20]. Both simulators are highly customisable and very suited for the requirements for this project. To decide on a simulator to use for this project, we experimented on both of them and we found that the Loopix Simulator has many additional features such as defining the size of the Sphinx packets for setting up a complex mixnet design that is beyond the goals of this project.

We decided to build on MiXiM to obtain the experimental framework as it was easier to verify our selected mixnet parameters against the work of Guirat and Diaz when running on the same simulator. As a side observation, the average end-to-end latency is adjustable with a specific field in the configuration file for MiXiM whereas the Loopix Simulator opted for adjustable per-mix latency in their configuration file. Note that the end-to-end latency and the per-mix latency are interrelated. The Loopix Simulator can be used for confirming the work conducted in this project. Since both simulators

are implemented with Python, the Rust binding can be easily ported from MiXiM to the Loopix Simulator to perform the experiments. We leave the evaluations of the experiments on the Loopix Simulator as future work.

As a side remark, there is an inconsistency in the Greek symbol used for referring to the mean or the expected value of per-mix latency and the parameter that characterises the exponential distribution from which all per-mix delays are sampled. In the literature of MiXiM, μ represents the former - the mean of the exponential distribution, i.e. the average per-mix latency, whereas μ represents the latter - the parameter of the exponential distribution. We confirmed this by running trial executions on both simulators. As a clarification, we use μ to represent the parameter of the exponential distribution and λ to denote its mean - the average per-mix latency. For further details, please refer to the summary of notations in Section 4.1.

Structure of Rust binding The original Sphinx packet unwrap benchmark is built with `criterion`[1], a feature-rich, sophisticated Rust benchmarking framework that is popular for measuring code performances in Rust. Since the unwrap time is included in the time of which the packet waits in the mix during the simulation, the initial idea was to obtain the unwrap time of the packet extemporaneously by invoking the Sphinx unwrap benchmark. This was well-thought approach as the design of `criterion` minimises the measurement overhead. However, despite a long period of substantial research and experimenting, we were neither able to find a way to invoke the Sphinx packet unwrap benchmark via the Rust binding, nor effectively return the unwrap time benchmarked in Rust to the Python runtime. This period of trial and error inherently slowed down the progress of the project. The challenges faced in this part of the project was further amplified by our limited experience with the Rust programming language. Hence substantial amount time was spent to gain sufficient knowledge in the Rust program structure, in order for the appropriate modifications to be made for this project.

As a result of the restrictions in the framework, we finally resorted to using a simple start-stop timer in Rust to measure the elapsed time to acquire the unwrap time. Thus, small fluctuations in the unwrap time were observed, hence also in part, motivated the experiment on the average unwrap time.

Overhead of Rust binding From inspecting the source code of the provided Sphinx benchmark, we find that part of the Sphinx packet unwrap function technically includes creating a copy of the packet for setup. It is a small portion of overhead compare to the computationally expensive cryptographic packet unwrapping [2]. Therefore such overhead is amortised over the duration of the simulation. Moreover, it is important to consider whether the overhead introduced by the Rust binding impacts the delay measurements. We confirmed this by calling an empty Rust function and found that the additional overhead due to the PyO3 library is of magnitude 10^{-5} of the unwrap time, regardless of the test machine. Thus this overhead is considered to be negligible.

[1] https://docs.rs/criterion/latest/criterion/

6.2 Results and Discussions

Choice of evaluated d_{E2E} values For the experiments are involved varying the value of d_{E2E}, the following values were chosen: 0.15000001, 0.16, 0.2, 0.25, 0.5, 1.0, 2.5, 5, and 10. Although the intervals between the values are not system, they are not arbitrarily chosen. The lowest values of 0.1500001 is selected as we wanted a small value that can highlight the gap between between transition from the theoretical models to physical implementations. This is a common problem in research. Whilst theoretical models can evaluate the processing time of any program, any empirical evaluation encounters noises introduced by the environment such as the operating system context switching of the physical test machine. In our case, an end user could theoretically set the value of d_{E2E} so small such that the average per-mix latency is or close to zero, to achieve a low overall latency. By testing 0.15000001, we can evaluate the effect of such small lambda of a continuous-time mixnet in a real-world scenario. We showed that this results in 100% over-delayed packets. Moreover, whilst the decision on the d_{E2E} values to evaluate does not affect the findings in this project, the values selected for end-to-end latency greater than 1 is relatively sparse compared to the values less than 1.

Histogram Plots The delay distributions were visualised using histograms. There was a key question to address: *how many bins should be used to generate the histograms?* There is a trade-off between the bin numbers. Small bin numbers result in some essential characteristics of the distributions to be overly rounded. Large bin numbers factor in the fluctuations caused by sampling. To best reproduce the structure of the which can be affected by suboptimal bin widths, several bin numbers were evaluated and a bin number of 250 selected which best balance between the representation of the delay distributions and the rounding of the data. We further supported our investigations by using various statistical analysis.

However, we recognise that it is difficult to gauge the optimal number of bins by the common method of trial and error. Knuth [17] propose an algorithm based on Bayesian probability theory which attempts to obtain the optimal bin number with limited or no information on the actual shape of the underlying distribution. It also does not assume the data to be normally distributed. Such method is implemented in the data visualisation module of the `astropy`[2] library to compute automatically the optimal number of histogram bins based on the data. Whilst we found that Knuth's rule computed the bin number efficiently, the algorithm advises fewer bins than the number we found; this is in line with the evaluation presented in [17]. This results in more rounding to the delay distributions such that some essential characteristics of the delay distributions have been overly rounded. Given longer project timeline, this approach could be explored further as it showed promising results.

Packet distinguishing attacks Two variants of the second type of distinguishing attacks are illustrated for strategy 1 of approach 1 and approach 2 in Figure 5.3b and Figure 5.3c respectively. We were able to compute the probability of an adversary launching such attack for strategy 1 of approach 1 by computing the proportion of over-delayed packets.

[2]Whilst the library itself is designed for astronomy purposes, the data visualisation module, specifically the function for plotting histograms is applicable for the usage in this report. Link: https://docs.astropy.org/en/stable/visualization/index.html

However, since approach 2 results in no over-delayed packets by definition, the current experimental setup does not provide means to compute the likelihood of occurrence for the attack. Specifically, there would be necessary modifications to the simulator and the logging system such as recording precisely the packet arrival and departure times, so that anaylsis can be completed. The entire data collection process would have to be repeated. Therefore, in consideration of the project timeline, we leave the detailed computation of the distinguishing attack on approach 2 as future work.

Chapter 7

Conclusions

7.1 Summary of Results

It is evident that the unwrap time of the mix has significant effect on the overall time the packet dwells in the mix. On the various test machines, the unwrap time of the mix running on one of the test machines is significantly different to the one of another test machine with a different hardware specification. The Loopix Simulator uses a constant to represent the Sphinx packet processing time on every mix whilst MiXiM accounts for the processing time and the network propagation together as a constant value that occurs per hop. The findings show that such design is no longer suitable, and encourage future work to take packet processing time, especially unwrapping time, in a sense that it is variable. Moreover, we advise extra attention is required when analysing a mixnet that consists of machines with various hardware specifications. This is because even machines are similar in processing speeds can still produce significantly different unwrap times.

We demonstrated that for small end-to-end latency, there is a sharp peak in the histogram depicting the actual delay distribution. This indicates that the unwrap time is forming an overwhelming proportion of the actual delay which a packet experiences. For most of the evaluated test machines, the end-to-end latency is small when $d_{E2E} \leq 0.25$. The exception is the Raspberry Pi which is significantly slower than the other machine; its end-to-end latency is small when $d_{E2E} \leq 5$.

Over-delayed packet occurs when the per-mix latency is shorter than the sender-intended delay. For μ value close to 0 (0.00000001 in our experiments), this studying finds that such small value results in 100% of packets being over-delayed for all the machines. In fact, for all the end-to-end latencies evaluated (hence μ values evaluated), there is a non-zero percentage of over-delayed packets. However, the percentage drops below 1 when the end-to-end latency is 10 seconds for all test machines except for the Raspberry Pi. In conclusion, a suitable value for the parameter μ of the exponential distribution depends on the hardware specification of the machine used for operating the mix. We advise suitable adjustments must be made accordingly for more accurate analyses.

To the best of our knowledge, the packet-distinguishing attacks presented in this report

are novel. The adversary can perform timing analysis attack by exploiting the knowledge of the unwrap time, and gains advantageous information to correlate incoming and outgoing traffic of a mix. In addition, we have showed that any value slightly greater than the minimal end-to-end latency permitted by the network latency is effective in preventing the first type of packet-distinguishing attack. we presented that the memoryless property breaks down for the over-delayed packets which occurs in strategy 1 of approach 1. This is because the exponentially distributed delays are shifted by the unavoidable unwrap time, and we also provided the relevant proof.

To conclude, it is to the best of our knowledge that, this project is the first piece of research focusing exclusively on the implications of the cryptographic Sphinx packet unwrapping operation on the anonymity of the mixnet. Sphinx packets are a fundamental component of real-world mixnets such as Nym [11]. We show the anonymity impacts of the machine-dependent unwrap time of the packet and their mitigations. The limitations and vulnerabilities of the mitigations have on the packet anonymity are discussed. With the empirical analysis on the Sphinx packet unwrap presented in this report, we hope to have provided further insights and motivations for future investigations in this crucial yet less-explored topic on the continuous-time mixnet.

7.2 Future Work

Throughout the report, we have suggested several potential areas for further work, which are summarised in this section.

Having established the significance of unwrap time on the actual delay of the packet, it is natural to continue the investigation on the other aspect of Sphinx packet processing - packet transformation. When a Sphinx packet arrives at a mix, the packet is cryptographically transformed by the mix before being relayed to its next destination. Such cryptographic operation is also computational expensive. In this report, the packet transformation overhead is accounted by assuming a constant factor. Looking the findings on unwrap times, this assumption is likely to be false. This motivates for further analysis to be performed in order to formalise its implications on the anonymity of the packets.

In terms of the architecture of the experiment framework used, we have used MiXiM as the underpinning simulator for this work. Alternatively, modifications can be made for the Loopix Simulator to be used instead.

For the packet distinguishing attack, the analysis of the likelihood of attack occurrence for approach 2 can be formalised. For strategies 2 and 3 of approach 1, future work can investigate the plausibility and the relevant mechanisms to ensure the correct functioning of the mixnet, as well as limiting the impacts in case of malicious senders or mixes.

Finally, the work presented in this paper is the starting basis on analysing the anonymity of the packets in the presence of cryptographic overhead. We have started by using a simplified mixnet design. To advance the investigation, we can add the remaining anonymity mechanisms such as cover traffic and link traffic which reflects more realistically the holistic mixnet deployed in Nym and Loopix. In future works, we can analyse the anonymity of such system when taking into account the unwrap times of the mixes.

Bibliography

[1] nymtech/nym, October 2022. original-date: 2020-01-07T11:42:53Z.

[2] nymtech/sphinx, September 2022. original-date: 2019-11-01T10:24:40Z.

[3] Manan AlMusallam and Jalal AlMuhatdi. De-correlating user profiles: Exploring anonymity tools. In *Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems*, pages 220–222, 2014.

[4] Iness Ben Guirat, Devashish Gosain, and Claudia Diaz. Mixim: Mixnet design decisions and empirical evaluation. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, WPES '21, page 33–37, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Alex Biryukov and Sergei Tikhomirov. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 172–184, 2019.

[6] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[7] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains: (a position paper). In *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20*, pages 106–125. Springer, 2016.

[8] George Danezis. Mix-networks with restricted routes. In *Privacy Enhancing Technologies: Third International Workshop, PET 2003, Dresden, Germany, March 26-28, 2003. Revised Papers 3*, pages 1–17. Springer, 2003.

[9] George Danezis. The traffic analysis of continuous-time mixes. In *Proceedings of the 4th international conference on Privacy Enhancing Technologies*, PET'04, pages 35–50, Berlin, Heidelberg, May 2004. Springer-Verlag.

[10] 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.

[11] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym Network. pages 1–38, February 2021.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[13] Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger. The devil is in the metadata—new privacy challenges in decentralised online social networks. In *2012 IEEE international conference on pervasive computing and communications workshops*, pages 333–339. IEEE, 2012.

[14] Iness Ben Guirat and Claudia Diaz. Mixnet optimization methods. *Proceedings on Privacy Enhancing Technologies*, 1:22, 2022.

[15] 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.

[16] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *Information Hiding*, 1998.

[17] Kevin H Knuth. Optimal data-based binning for histograms. *arXiv preprint physics/0605197*, 2006.

[18] Hemi Leibowitz, Ania M Piotrowska, George Danezis, and Amir Herzberg. No right to remain silent: isolating malicious mixes. In *PROCEEDINGS OF THE 28TH USENIX SECURITY SYMPOSIUM*, volume 28, pages 1841–1858. USENIX Association, 2019.

[19] Hubert W Lilliefors. On the kolmogorov-smirnov test for the exponential distribution with mean unknown. *Journal of the American Statistical Association*, 64(325):387–389, 1969.

[20] 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.

[21] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *Proceedings of the 26th USENIX Security Symposium*, pages 1199–1216. USENIX Association, August 2017. 26th USENIX Security Symposium : USENIX 2017 ; Conference date: 16-08-2017 Through 18-08-2017.

[22] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[23] Zachary R Smith and Craig S Wells. Central limit theorem and sample size. In *annual meeting of the Northeastern Educational Research Association, Kerhonkson, New York*, 2006.

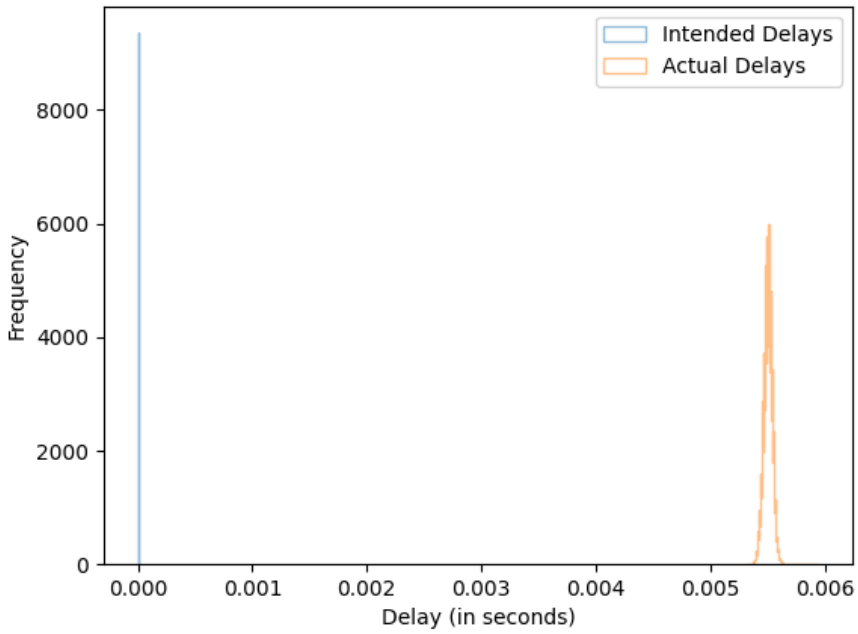
[24] Qingfeng Tan, Xuebin Wang, Wei Shi, Jian Tang, and Zhihong Tian. An anonymity vulnerability in tor. *IEEE/ACM Transactions on Networking*, 30(6):2574–2587, 2022.

[25] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, 2015.

Appendix A

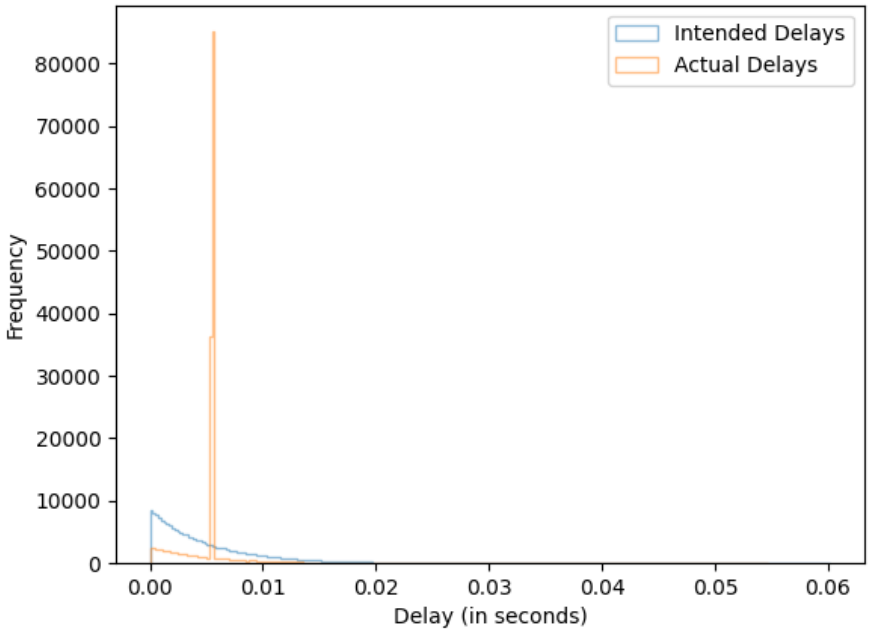
Intended and Actual Delay Distributions of Strategy 1 of Approach 1

This section provides all the distributions of actual and intended delays on the test machines listed in section 3.6, when considering strategy 1 of approach 1 of section 3.2. Figures A.1, A.2, A.3, and A.4 cover strategy 1 of approach 1 for the lab machine, Macbook, DICE (SC) and Raspberry Pi respectively.

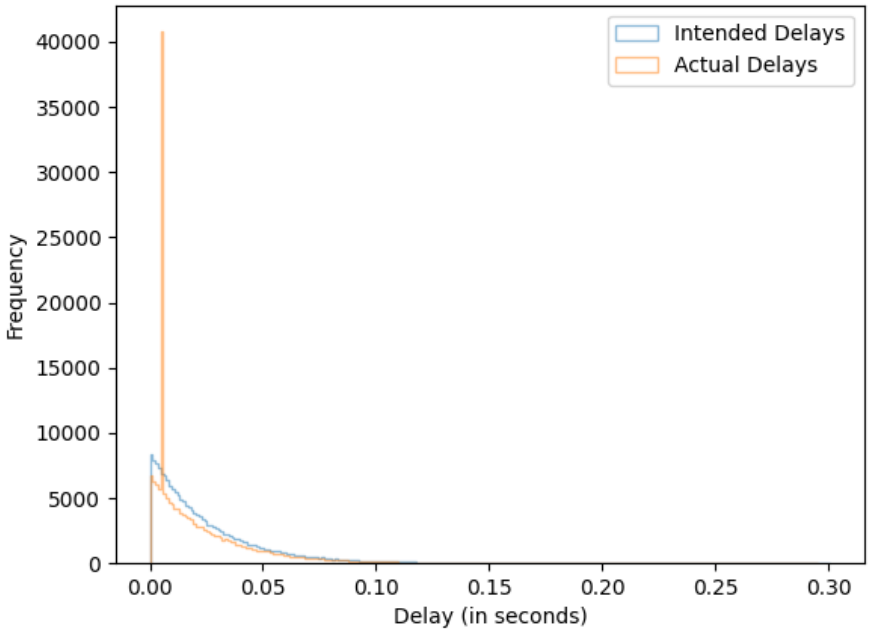


(a) $d_{E2E} = 0.15000001s$

Figure A.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 1 in section 3.2.

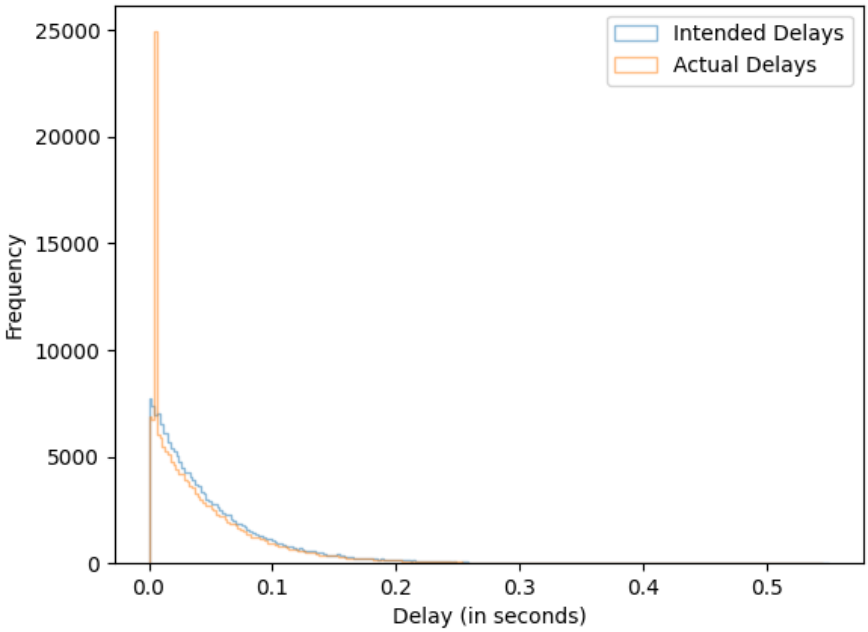


(b) $d_{E2E} = 0.16s$

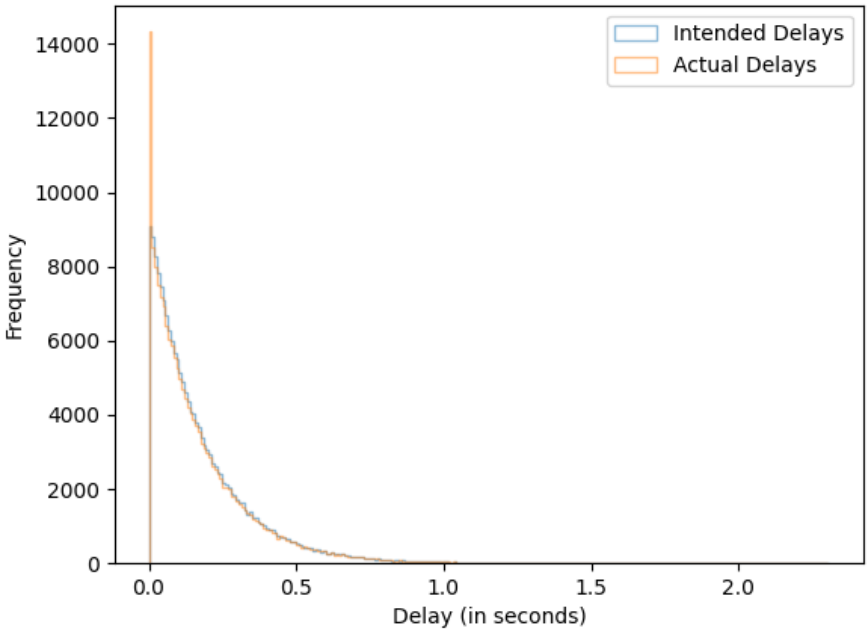


(c) $d_{E2E} = 0.2s$

Figure A.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 1 in section 3.2.

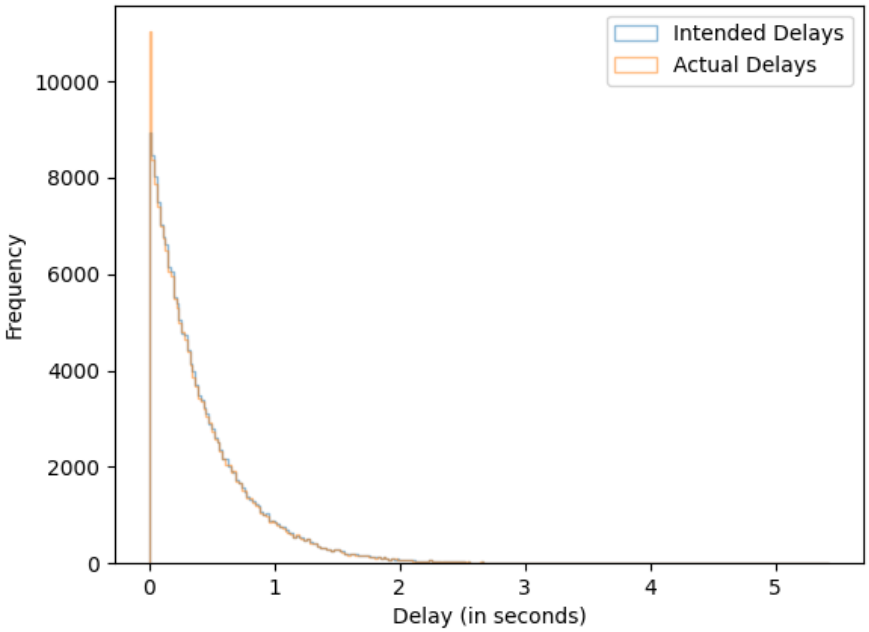


(d) $d_{E2E} = 0.25s$

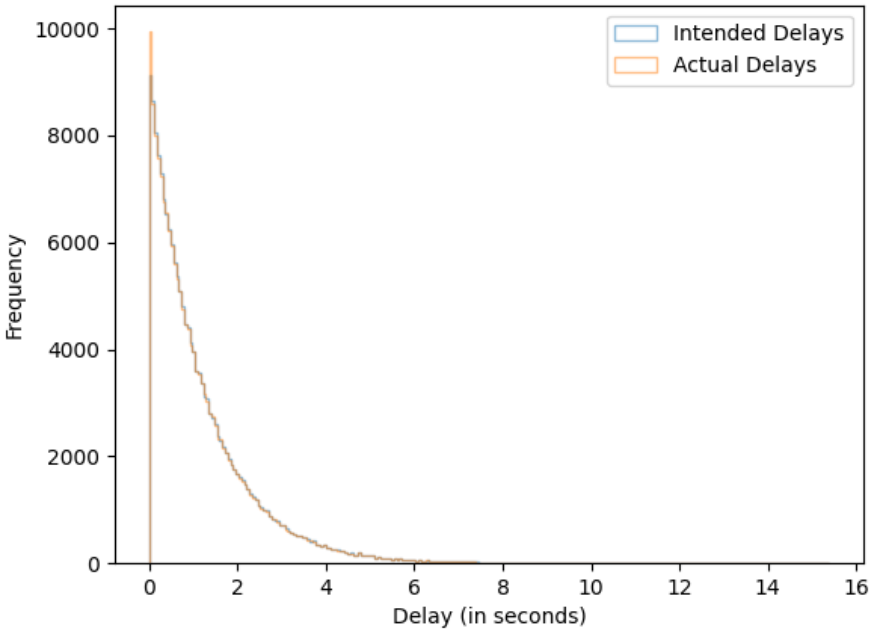


(e) $d_{E2E} = 0.5s$

Figure A.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 1 in section 3.2.

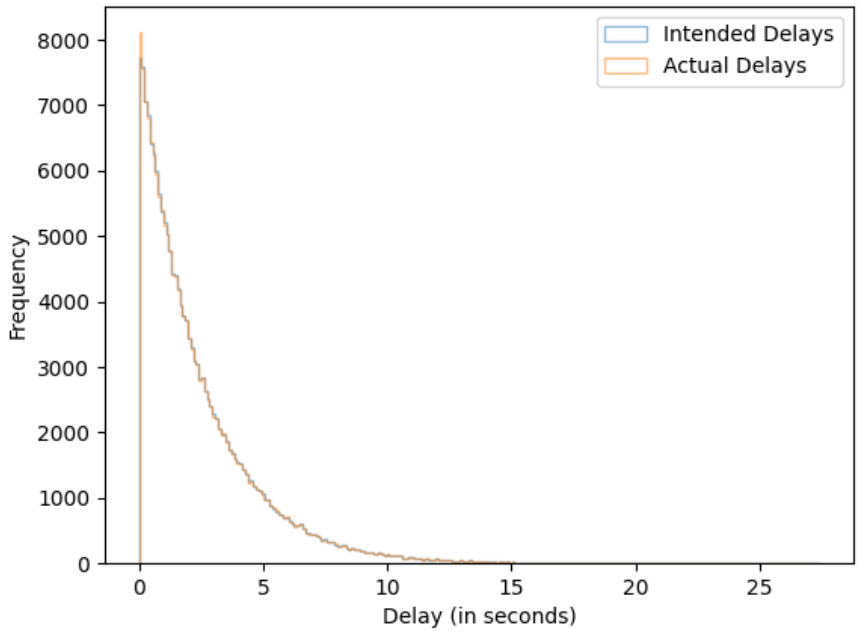


(f) $d_{E2E} = 1.0s$

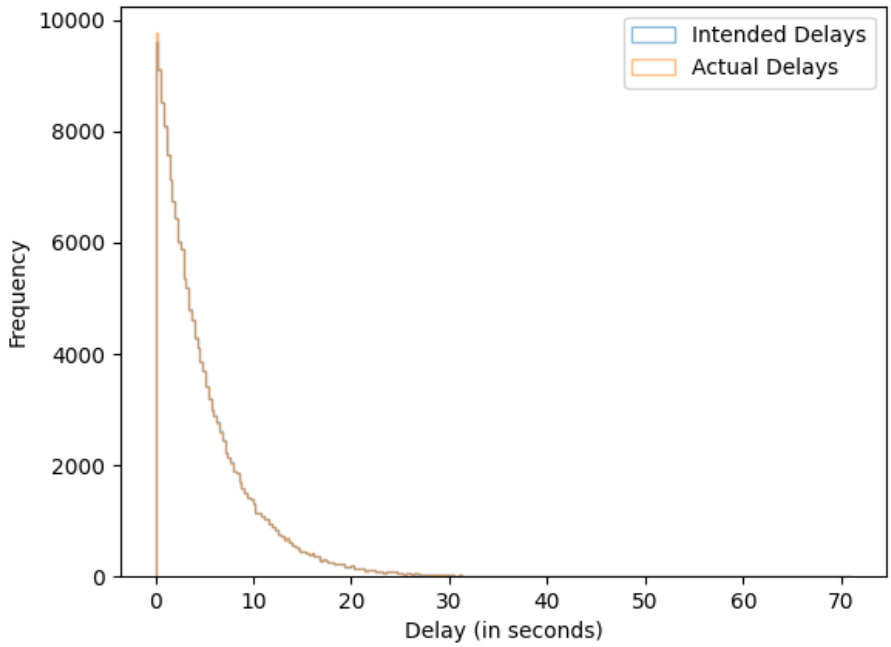


(g) $d_{E2E} = 2.5s$

Figure A.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 1 in section 3.2.

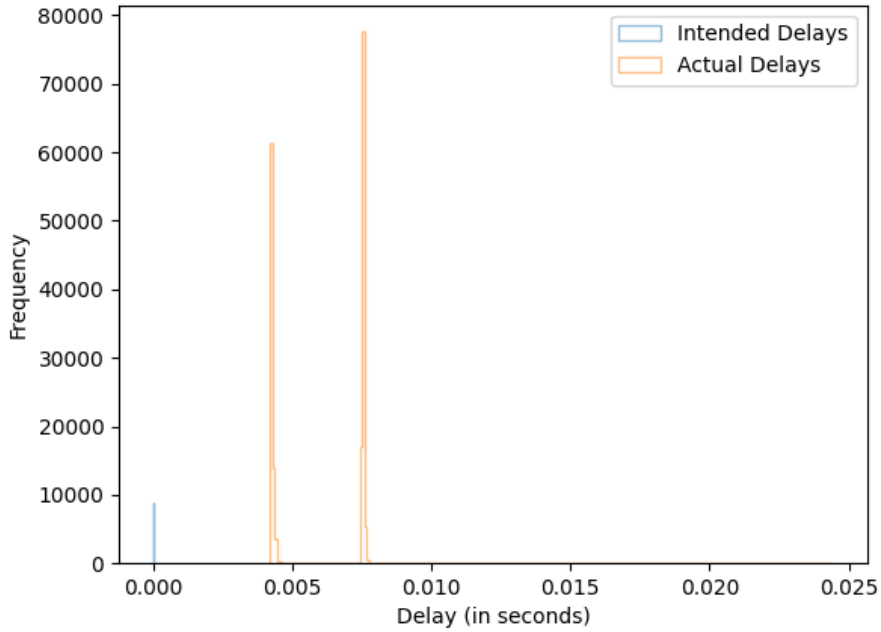


(h) $d_{E2E} = 5.0s$

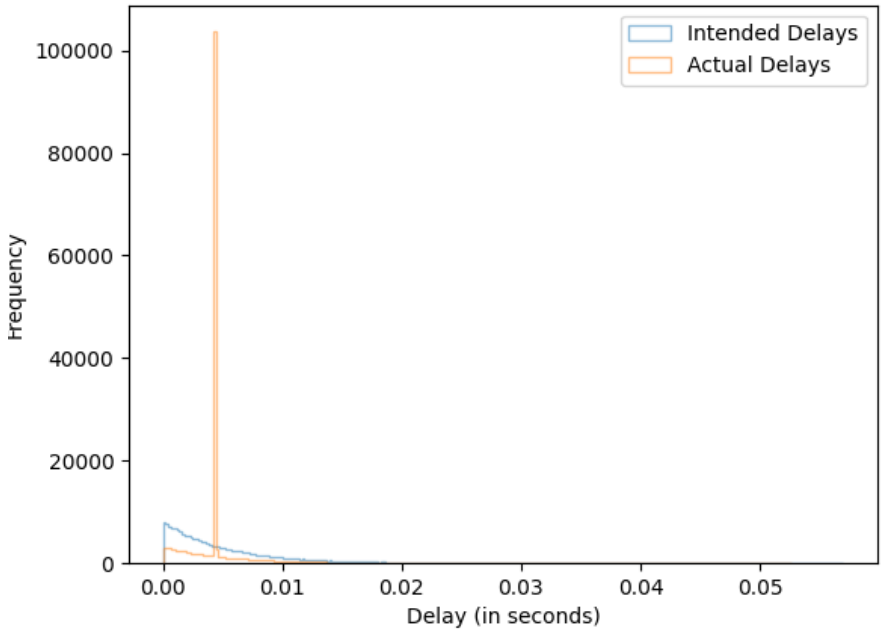


(i) $d_{E2E} = 10.0s$

Figure A.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 1 in section 3.2.

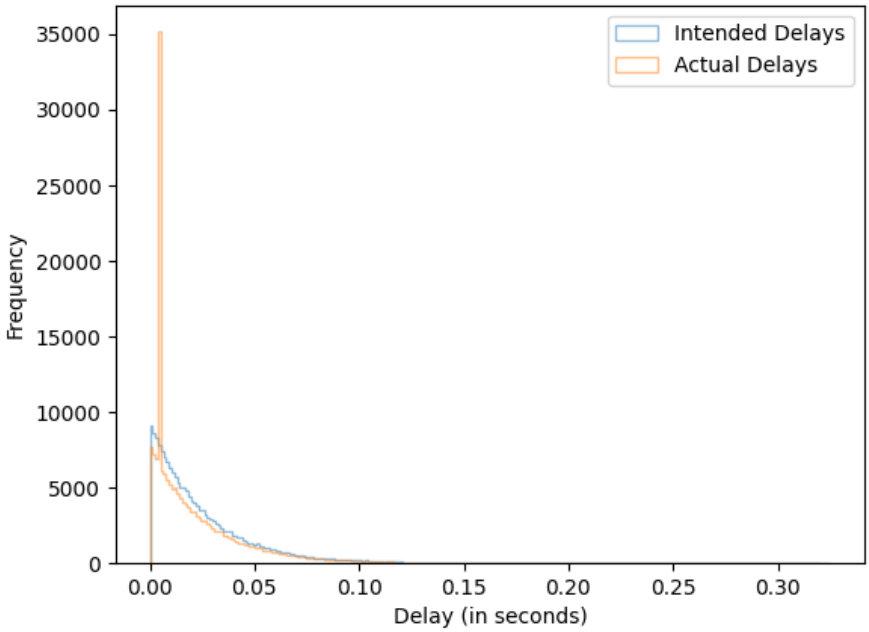


(a) $d_{E2E} = 0.15000001s$

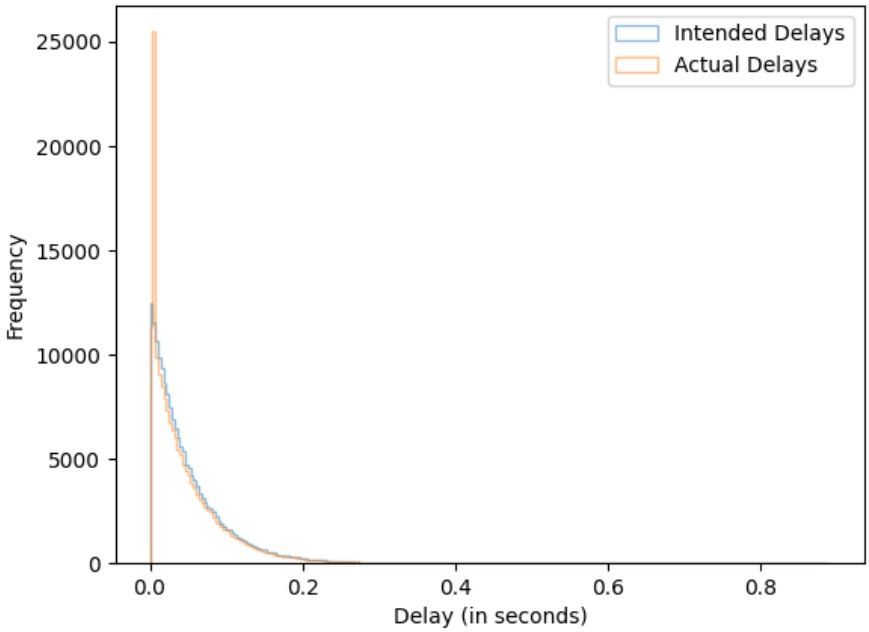


(b) $d_{E2E} = 0.16s$

Figure A.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 1 in section 3.2.

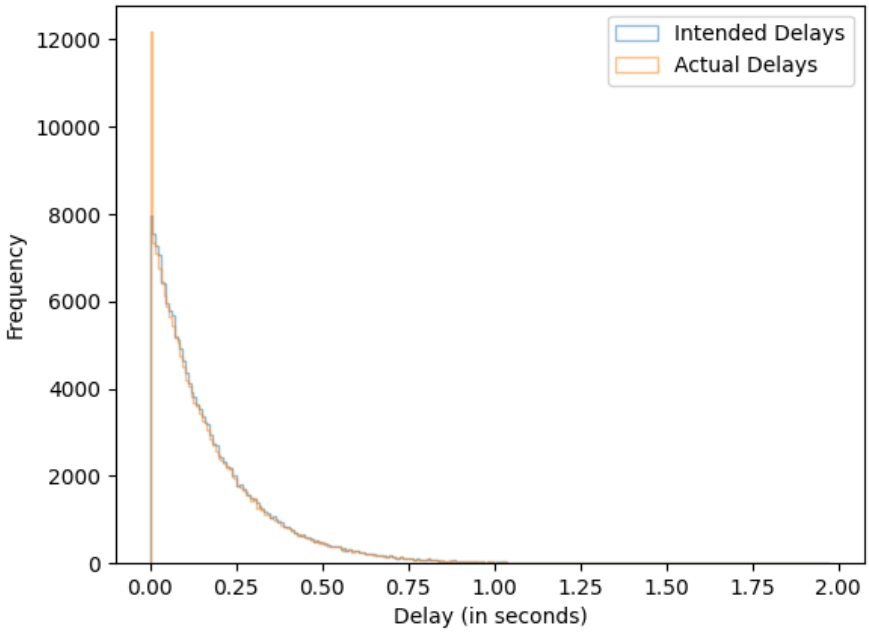


(c) $d_{E2E} = 0.2s$

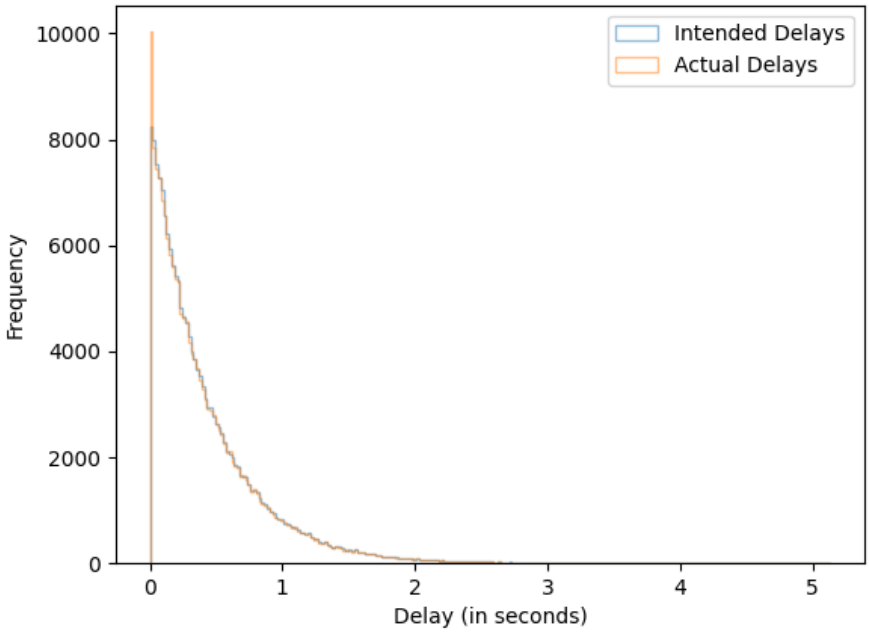


(d) $d_{E2E} = 0.25s$

Figure A.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 1 in section 3.2.

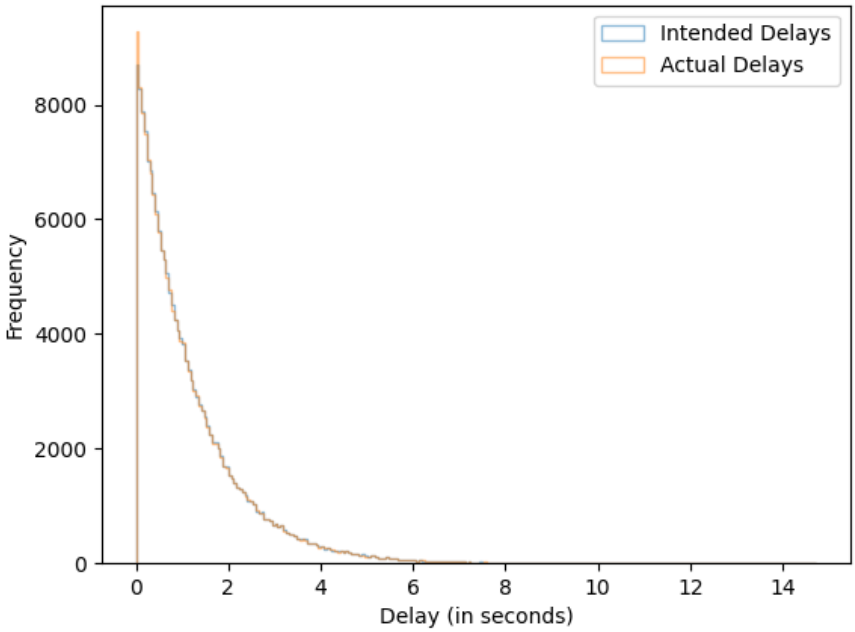


(e) $d_{E2E} = 0.5s$

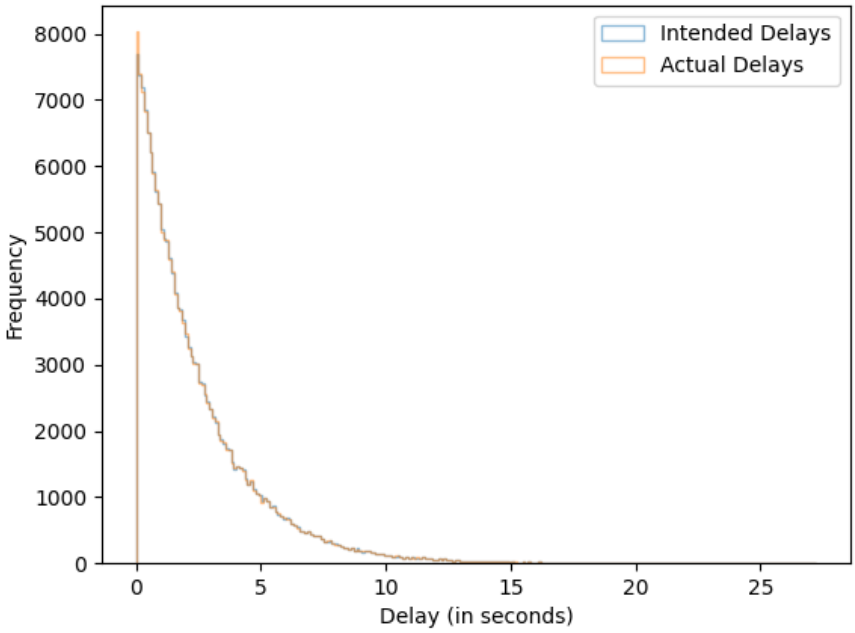


(f) $d_{E2E} = 1.0s$

Figure A.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 1 in section 3.2.

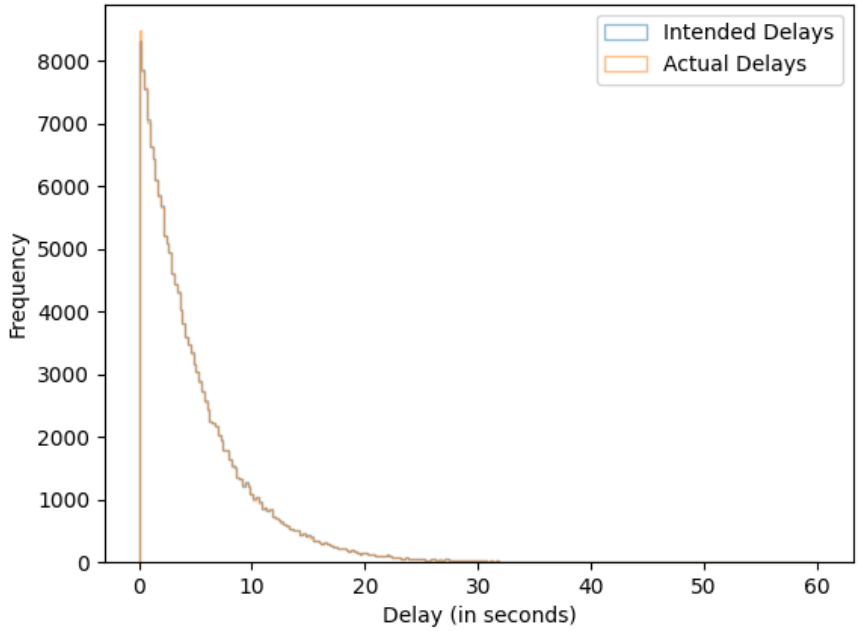


(g) $d_{E2E} = 2.5s$



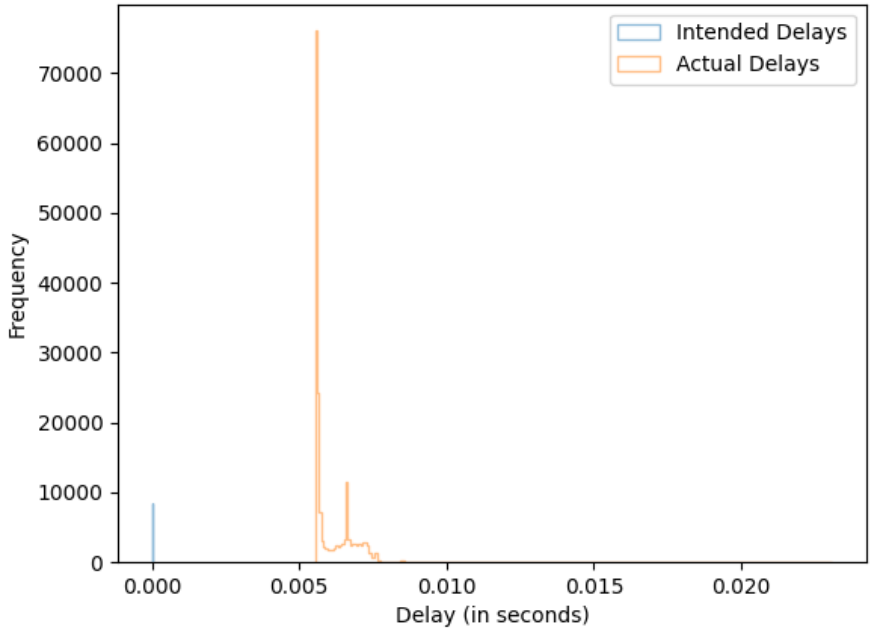
(h) $d_{E2E} = 5.0s$

Figure A.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 1 in section 3.2.



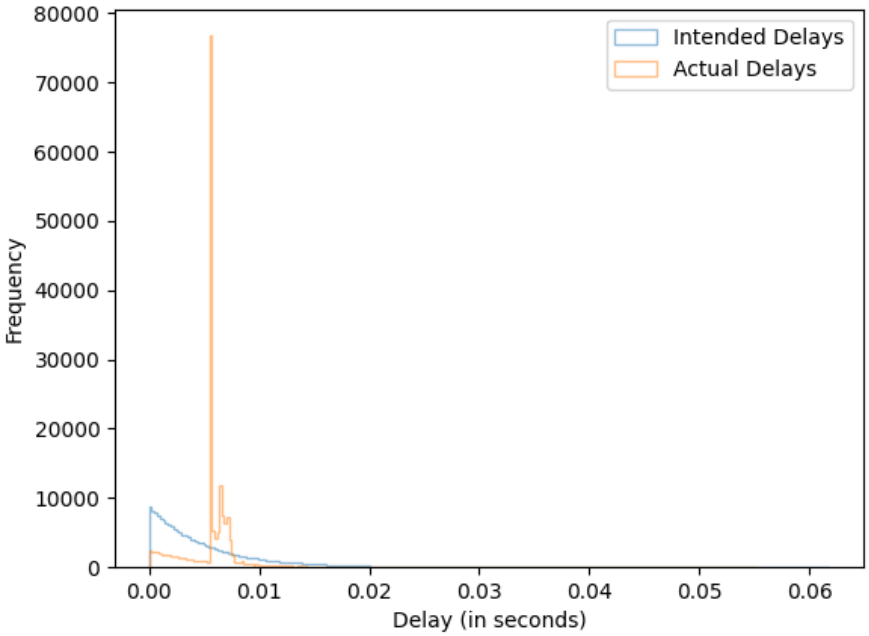
(i) $d_{E2E} = 10.0s$

Figure A.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 1 in section 3.2.

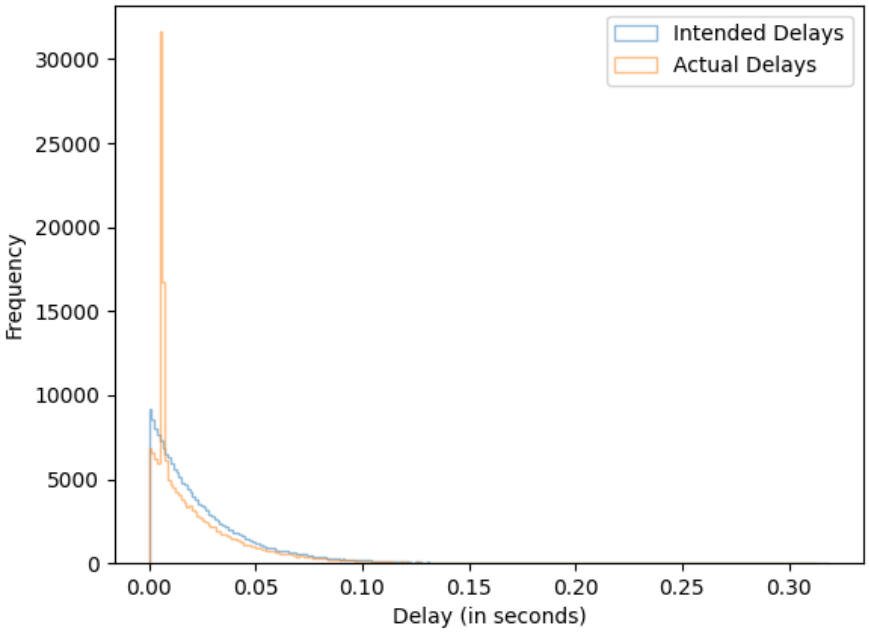


(a) $d_{E2E} = 0.15000001s$

Figure A.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 1 in section 3.2.

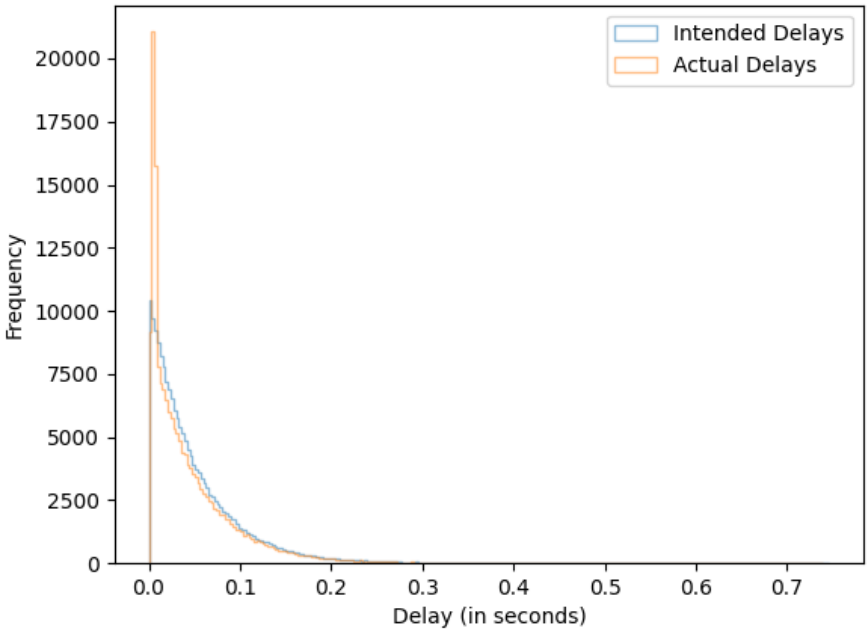


(b) $d_{E2E} = 0.16s$

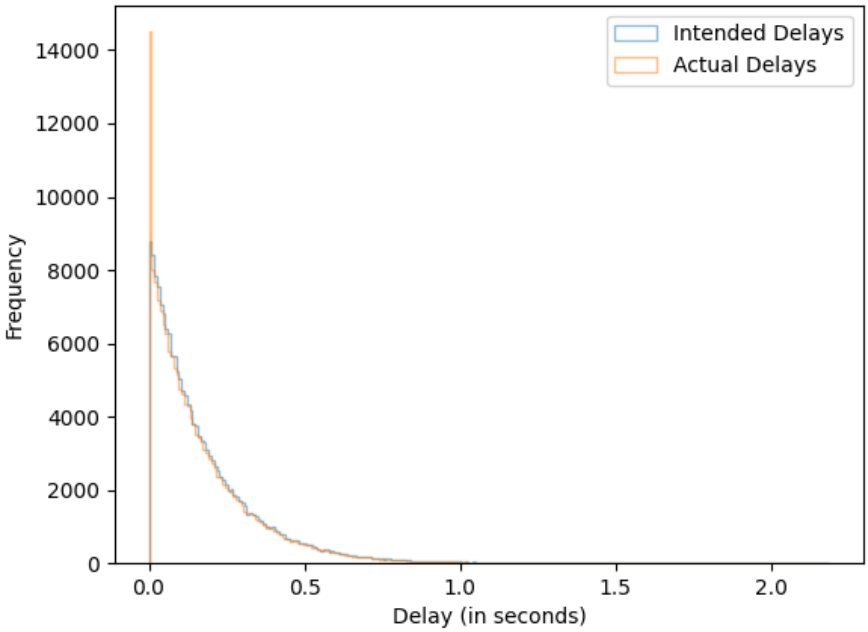


(c) $d_{E2E} = 0.2s$

Figure A.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 1 in section 3.2.

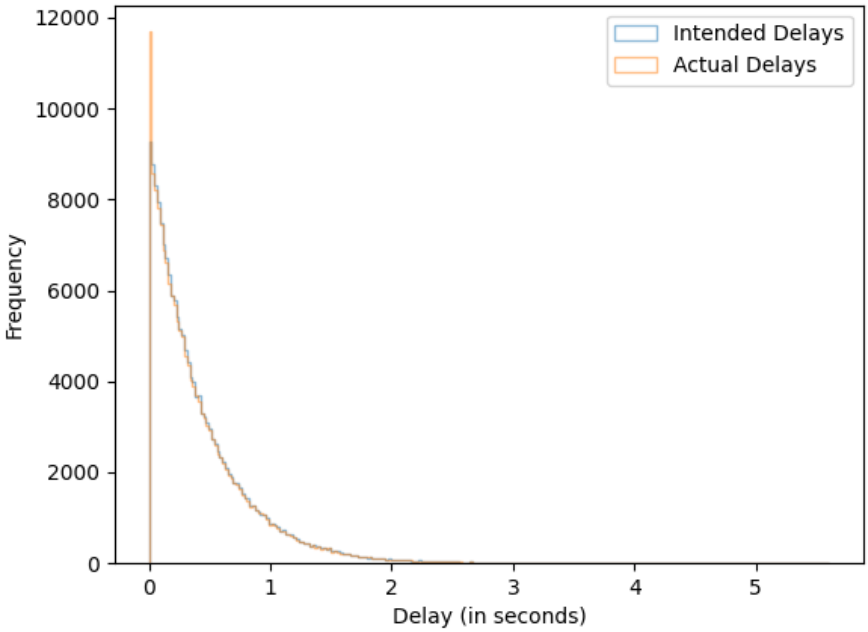


(d) $d_{E2E} = 0.25s$

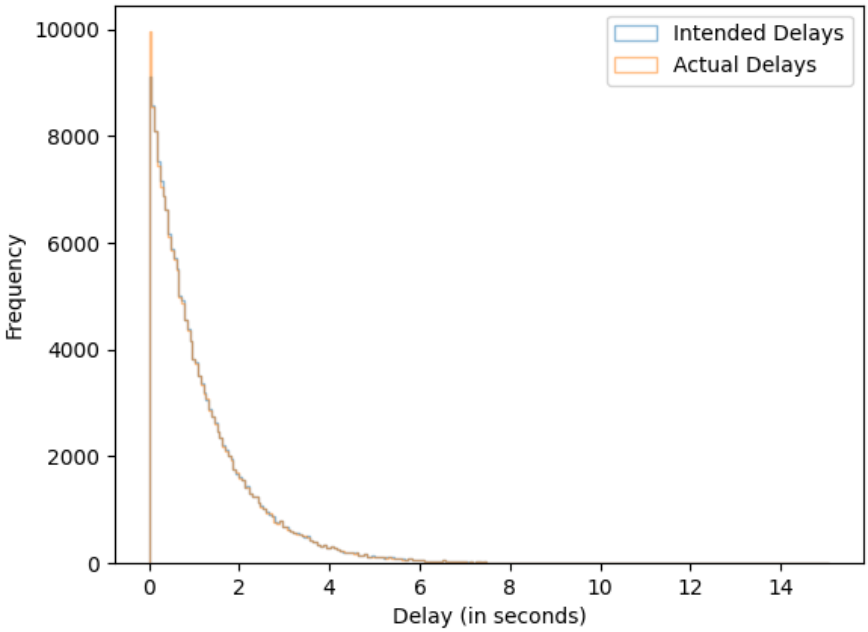


(e) $d_{E2E} = 0.5s$

Figure A.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 1 in section 3.2.

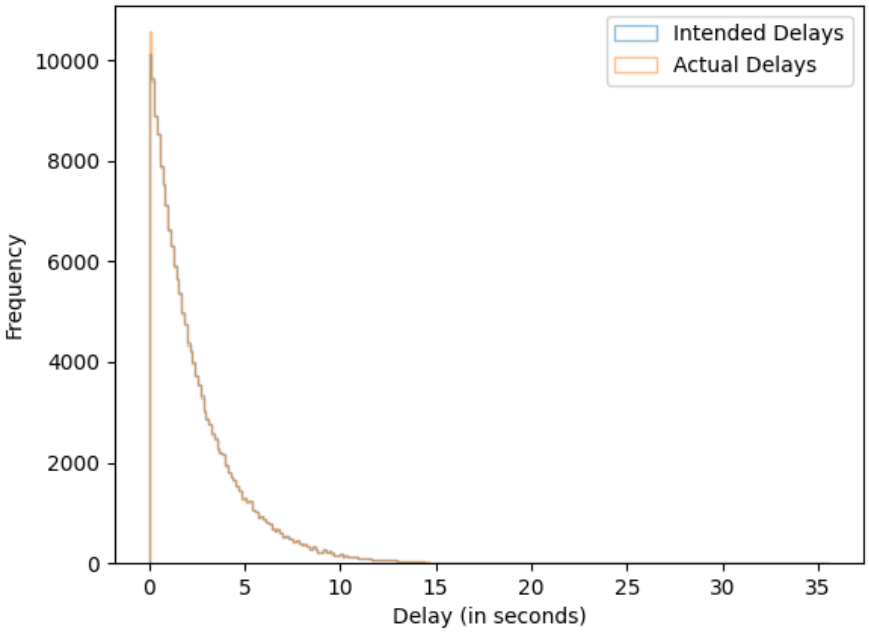


(f) $d_{E2E} = 1.0s$

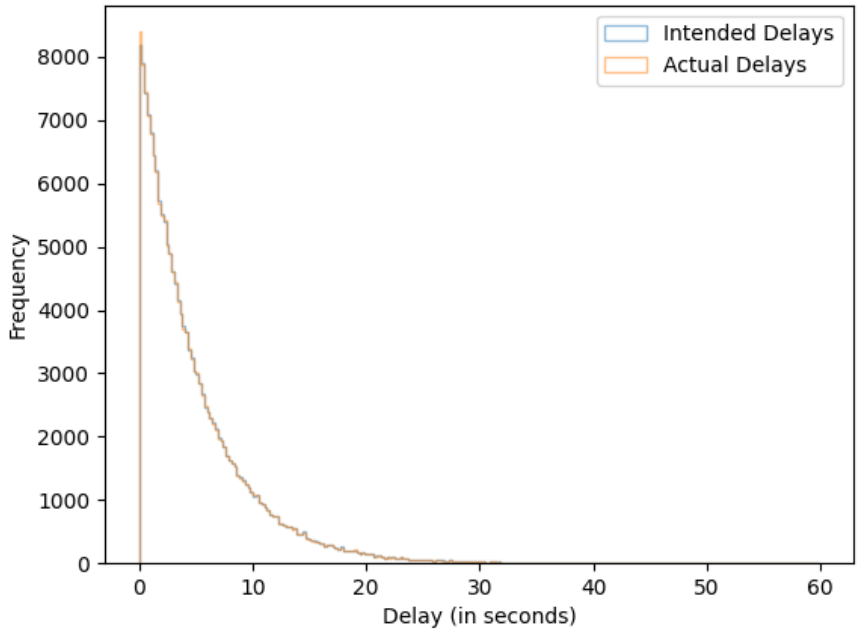


(g) $d_{E2E} = 2.5s$

Figure A.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 1 in section 3.2.

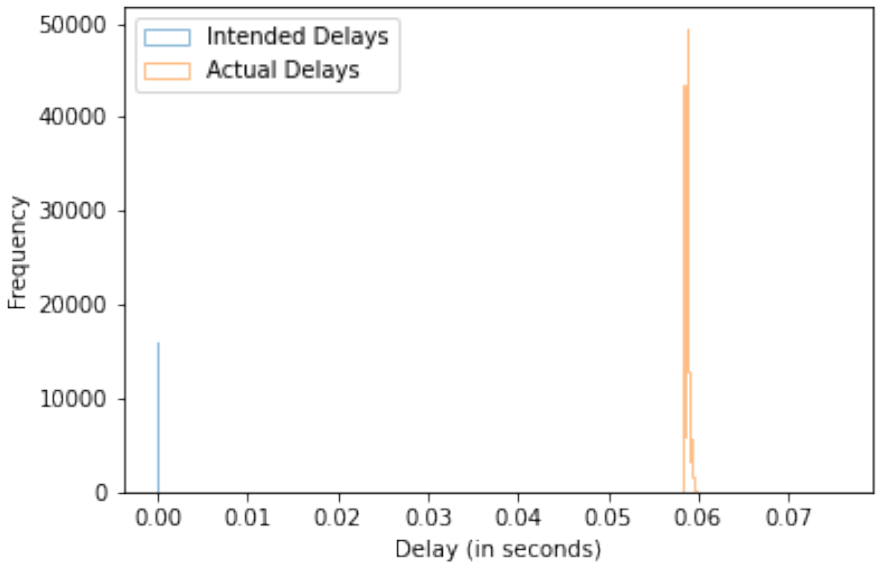


(h) $d_{E2E} = 5.0s$

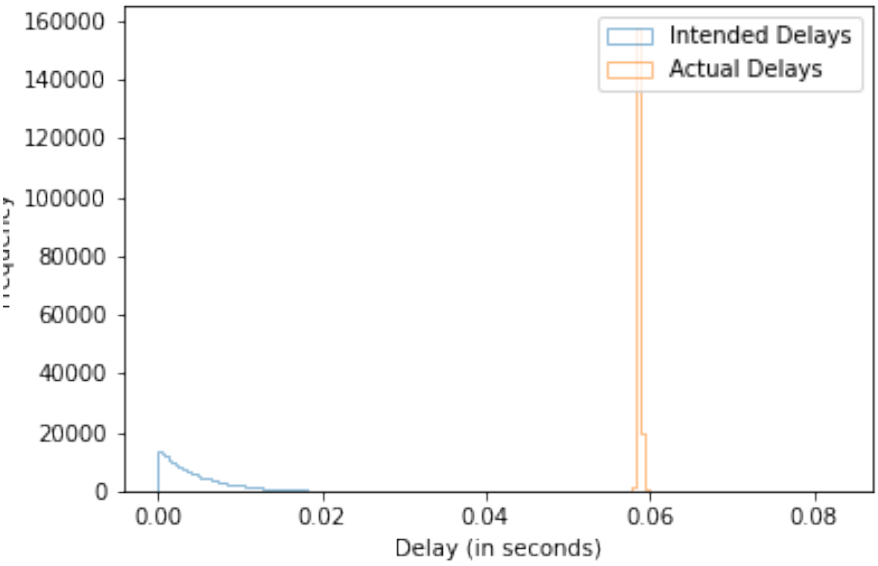


(i) $d_{E2E} = 10.0s$

Figure A.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 1 in section 3.2.

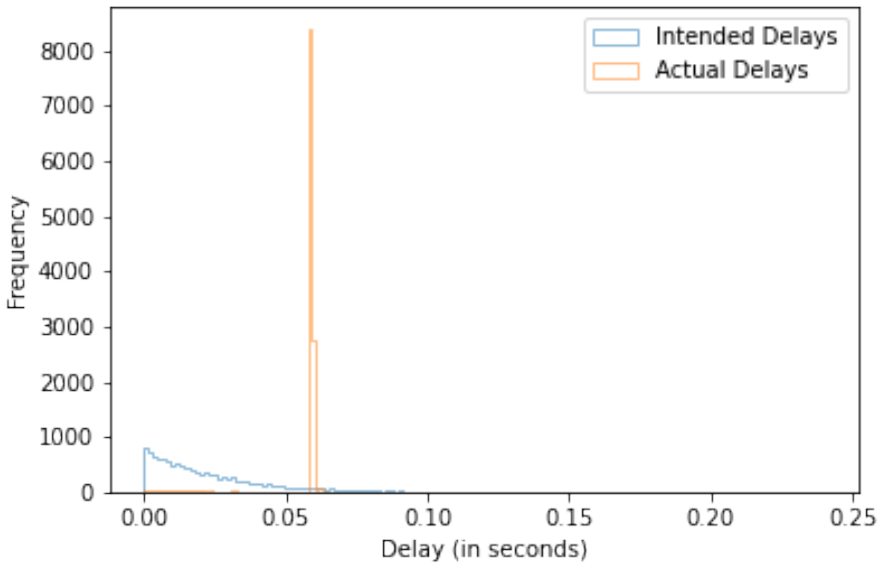


(a) $d_{E2E} = 0.15000001s$

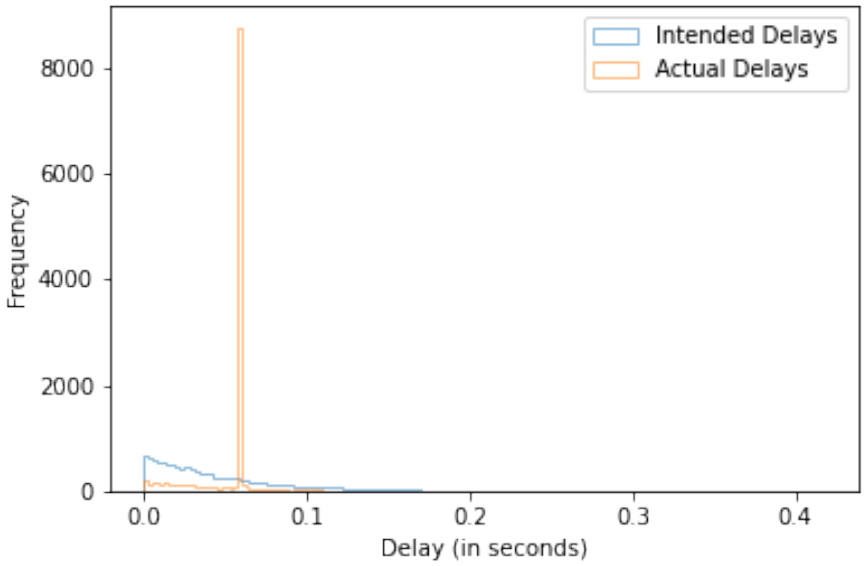


(b) $d_{E2E} = 0.16s$

Figure A.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 1 in section 3.2.

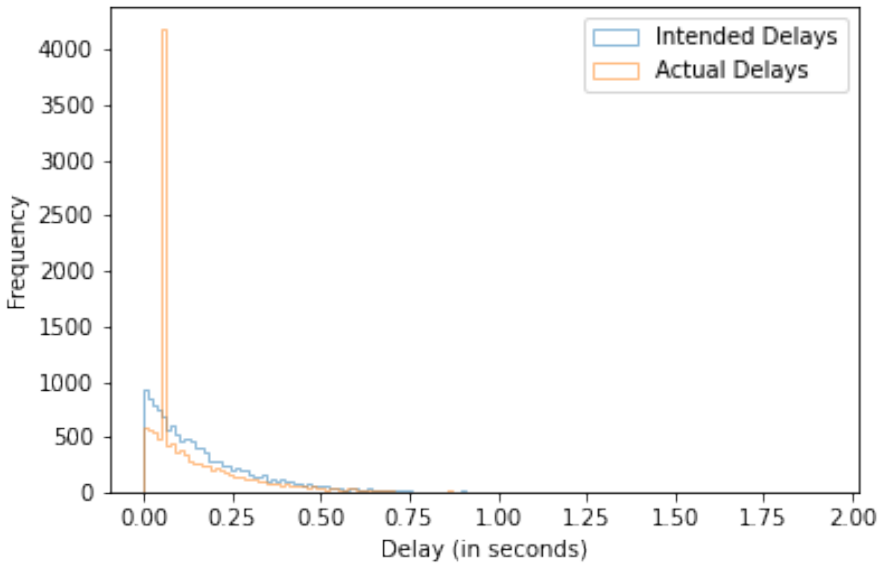


(c) $d_{E2E} = 0.2s$

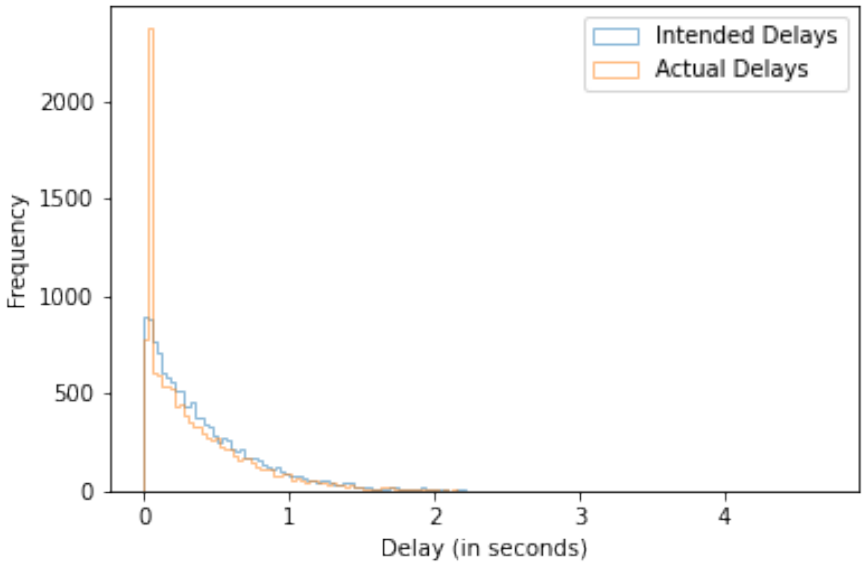


(d) $d_{E2E} = 0.25s$

Figure A.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 1 in section 3.2.

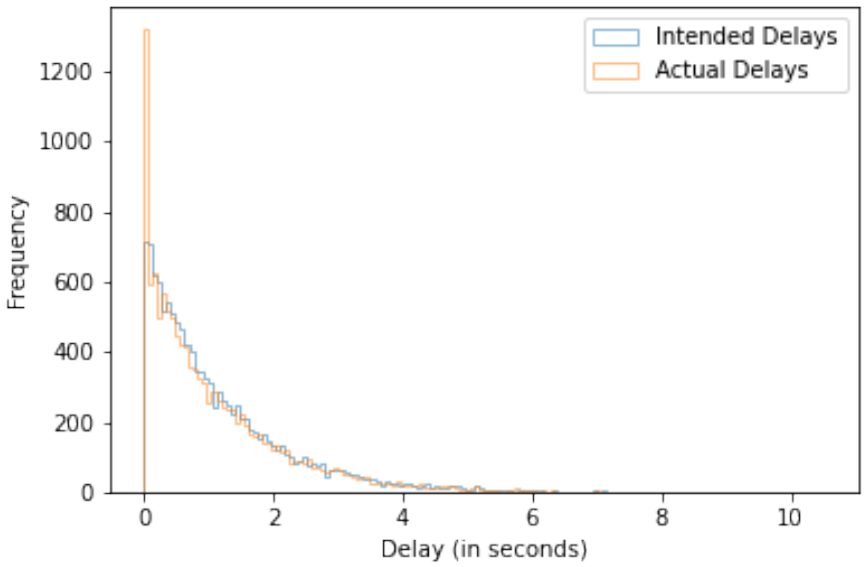


(e) $d_{E2E} = 0.5s$

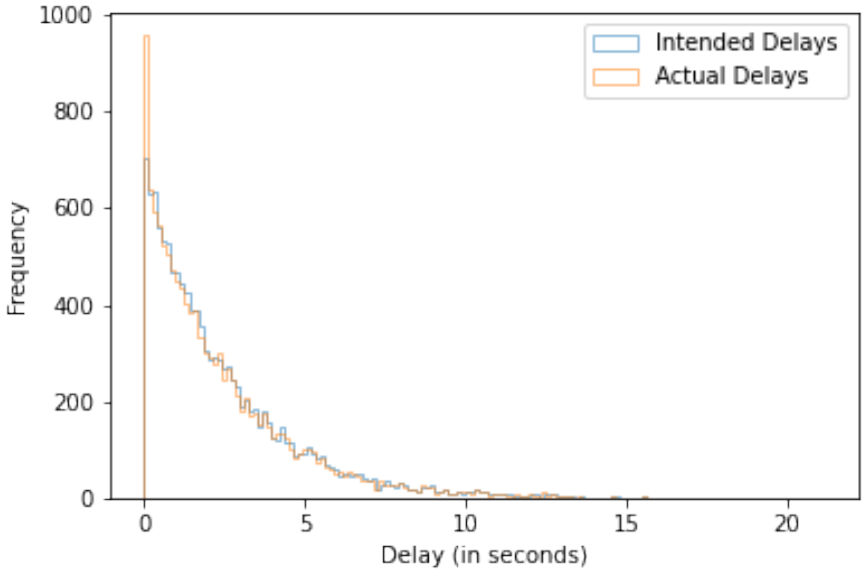


(f) $d_{E2E} = 1.0s$

Figure A.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 1 in section 3.2.

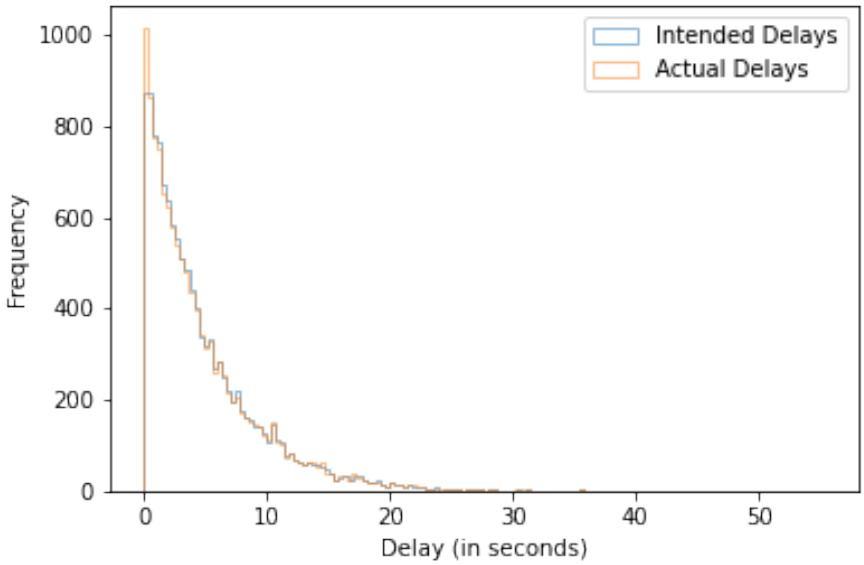


(g) $d_{E2E} = 2.5s$



(h) $d_{E2E} = 5.0s$

Figure A.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 1 in section 3.2.



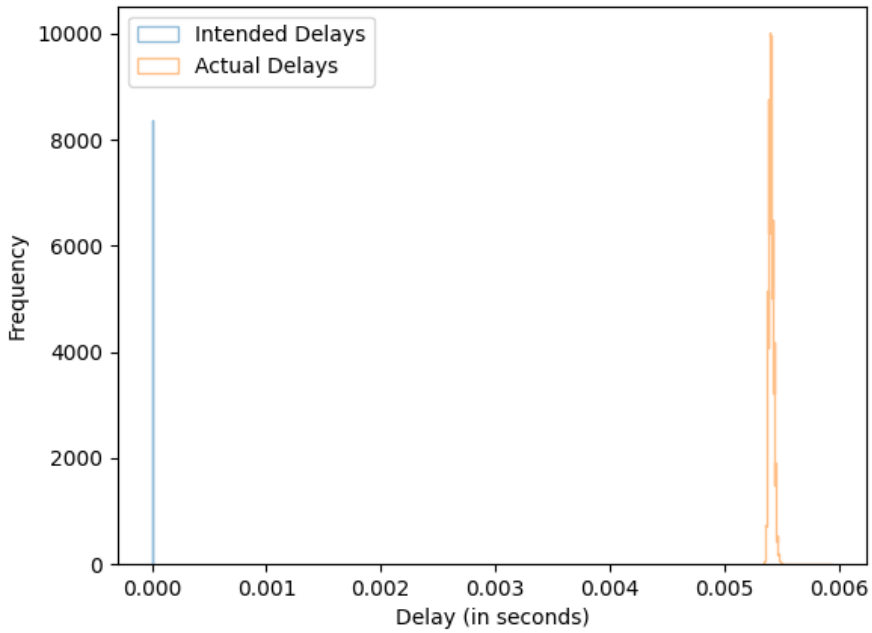
(i) $d_{E2E} = 10.0s$

Figure A.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 1 in section 3.2.

Appendix B

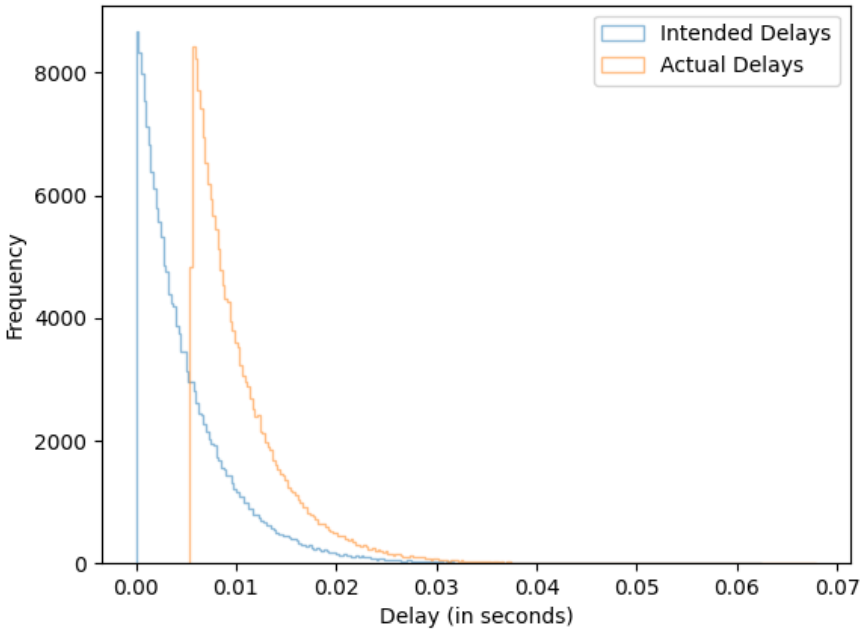
Intended and Actual Delay Distributions of Approach 2

This section presents the figures of delay distributions related to approach 2. Figures B.1, B.2, B.3, B.4 for the lab machine, Macbook, DICE (SC) and Raspberry Pi respectively.

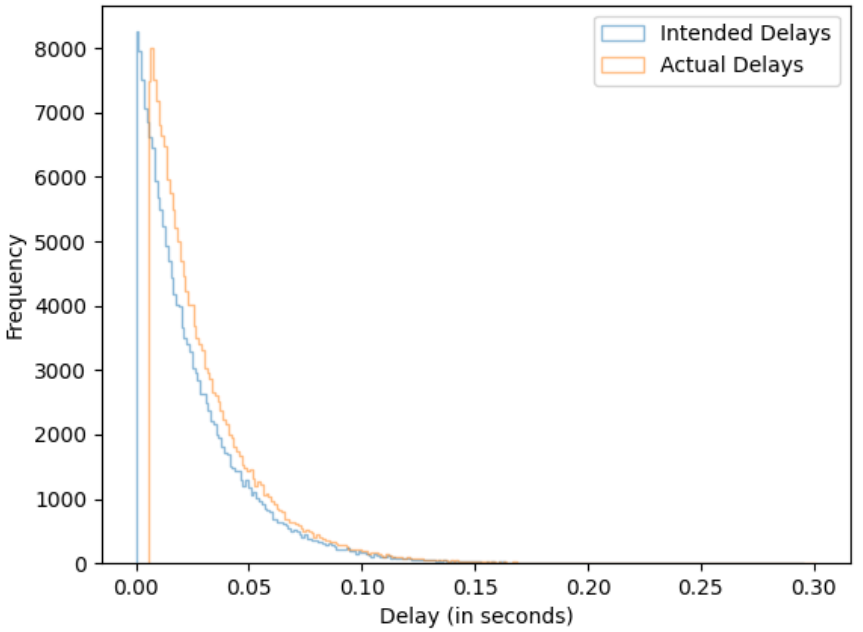


(a) $d_{E2E} = 0.15000001s$

Figure B.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 2 in section 3.2.

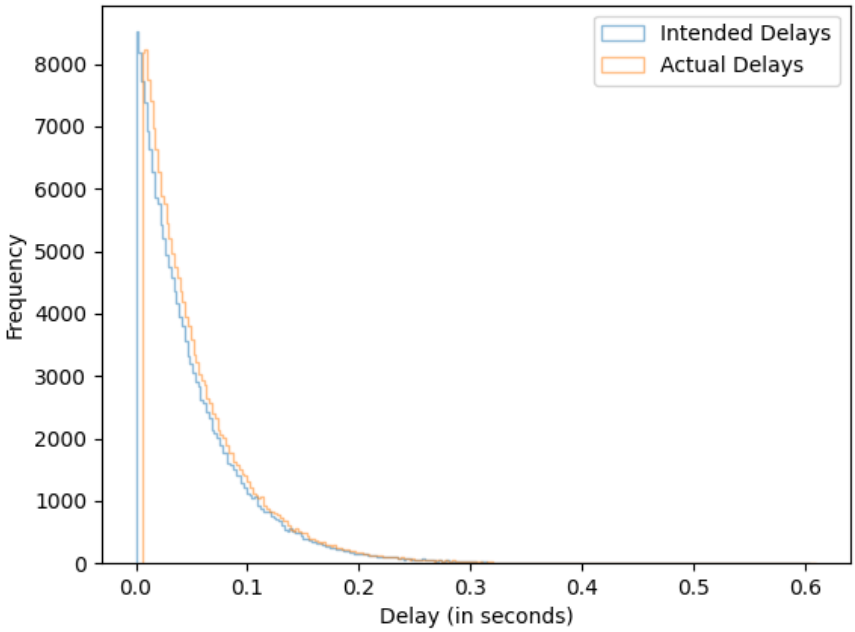


(b) $d_{E2E} = 0.16s$

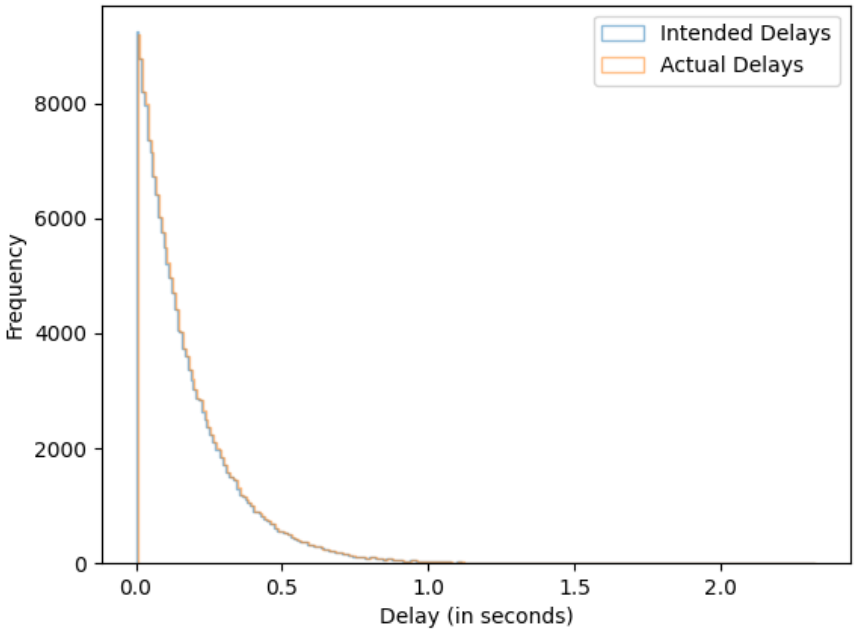


(c) $d_{E2E} = 0.2s$

Figure B.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 2 in section 3.2.

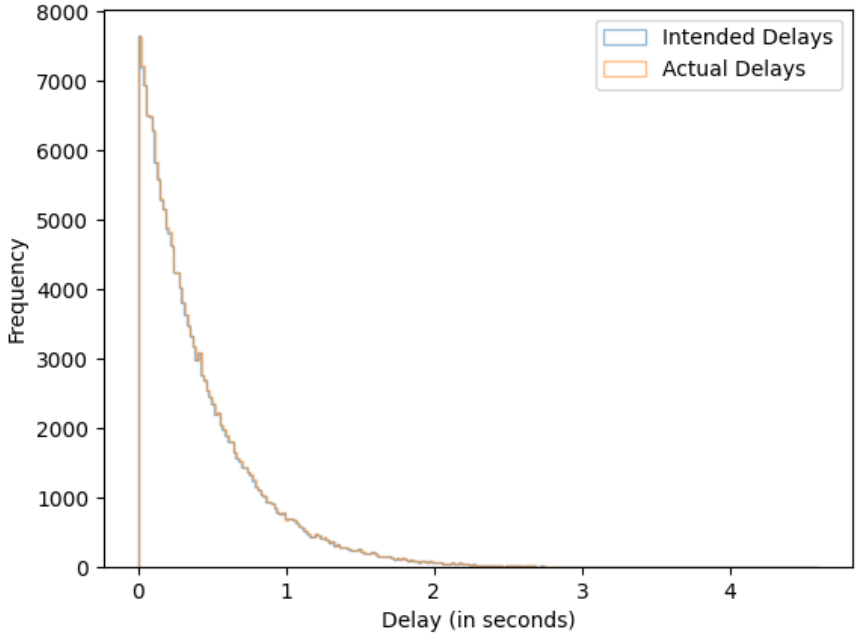


(d) $d_{E2E} = 0.25s$

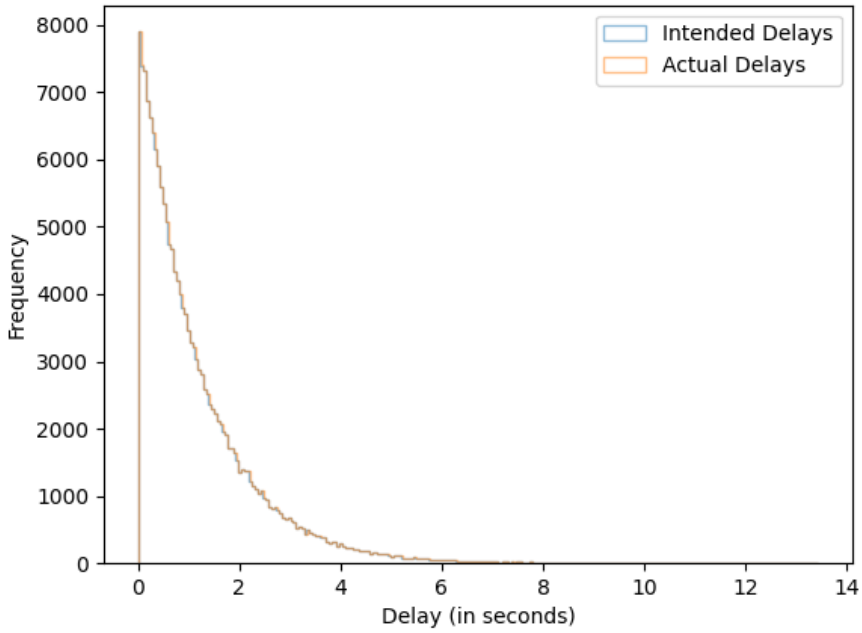


(e) $d_{E2E} = 0.5s$

Figure B.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 2 in section 3.2.

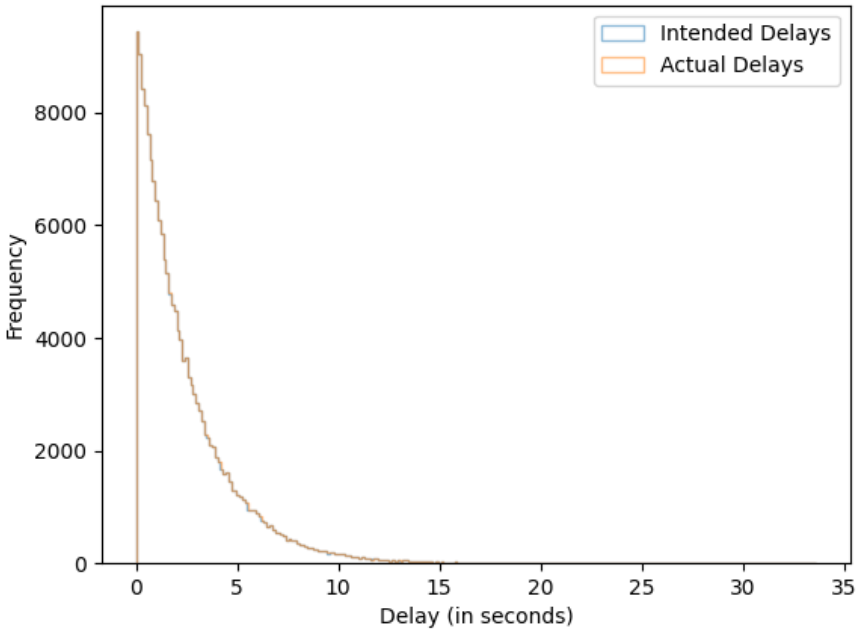


(f) $d_{E2E} = 1.0s$

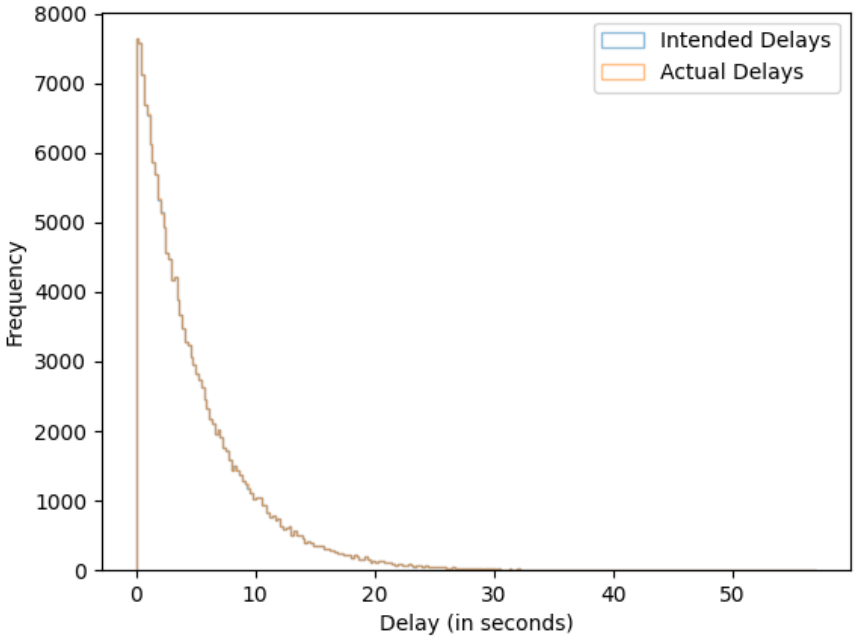


(g) $d_{E2E} = 2.5s$

Figure B.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 2 in section 3.2.

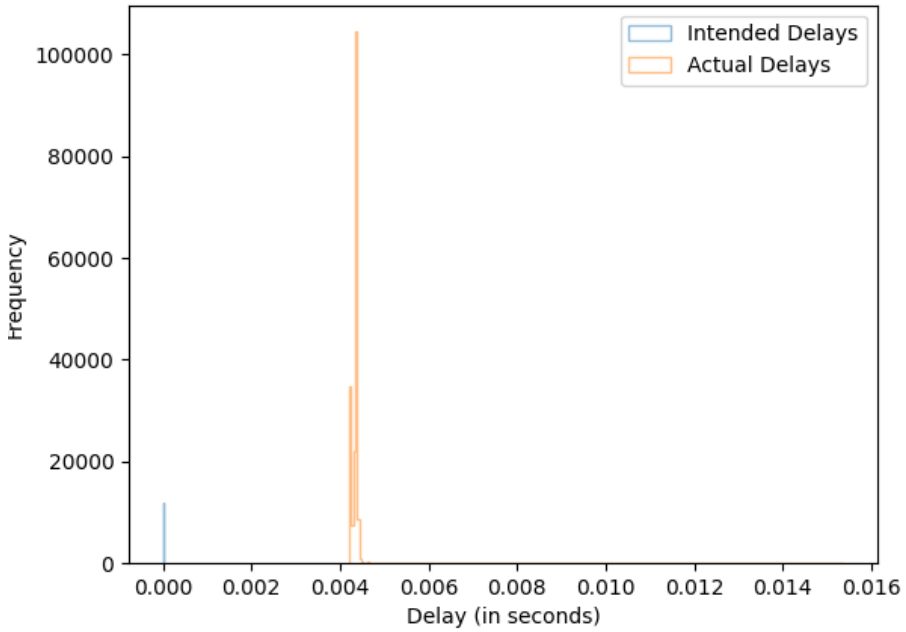


(h) $d_{E2E} = 5.0s$

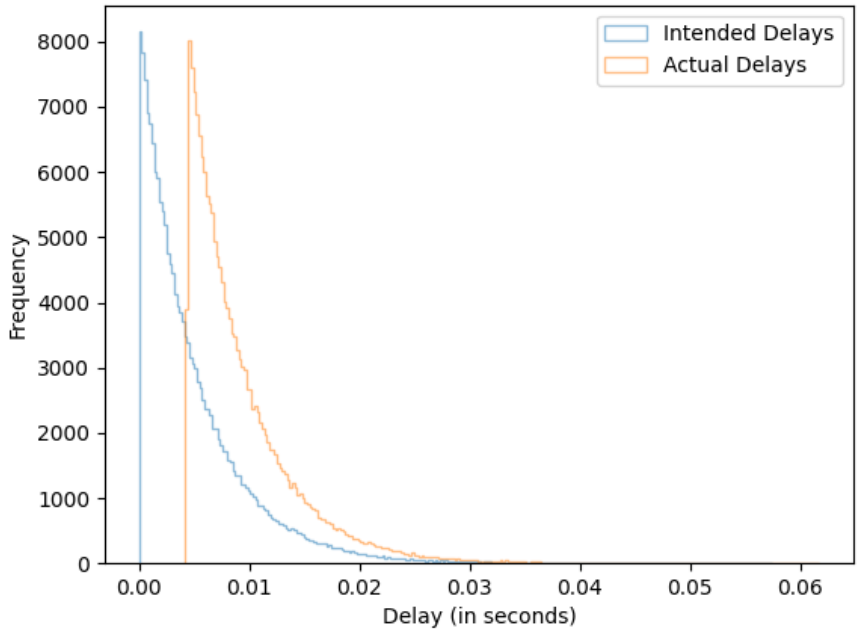


(i) $d_{E2E} = 10.0s$

Figure B.1: Distributions of the intended and actual delays of the packets in the simulation on the lab machine under approach 2 in section 3.2.

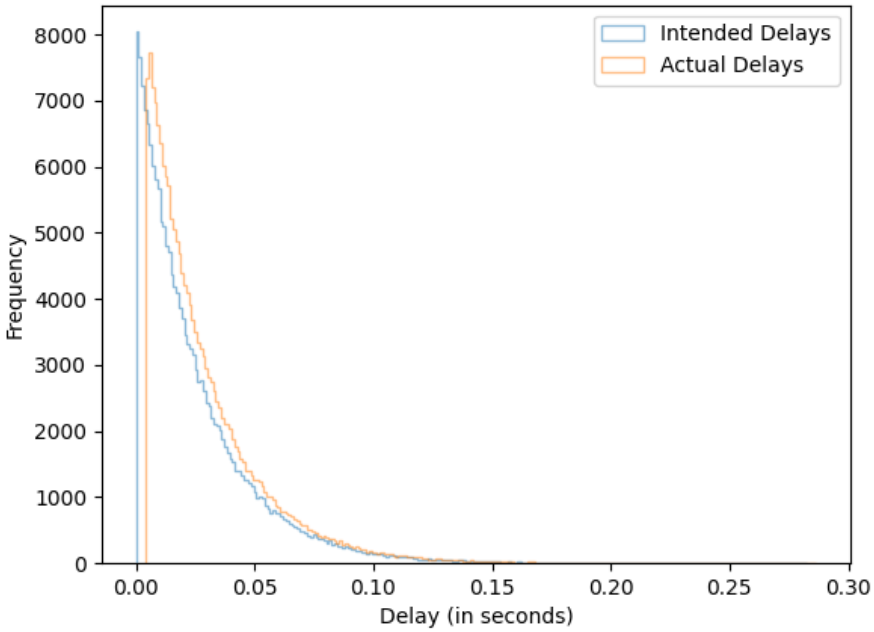


(a) $d_{E2E} = 0.15000001s$

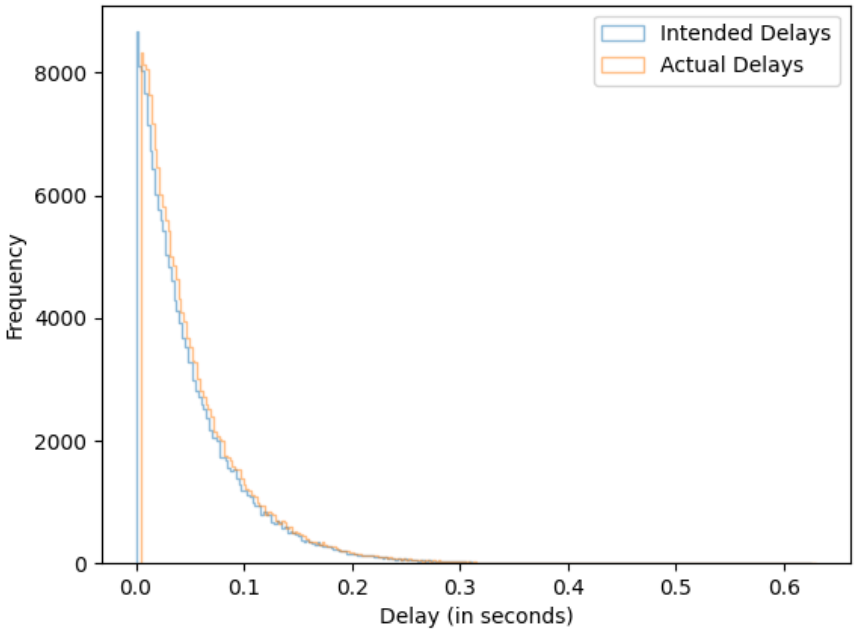


(b) $d_{E2E} = 0.16s$

Figure B.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 2 in section 3.2.

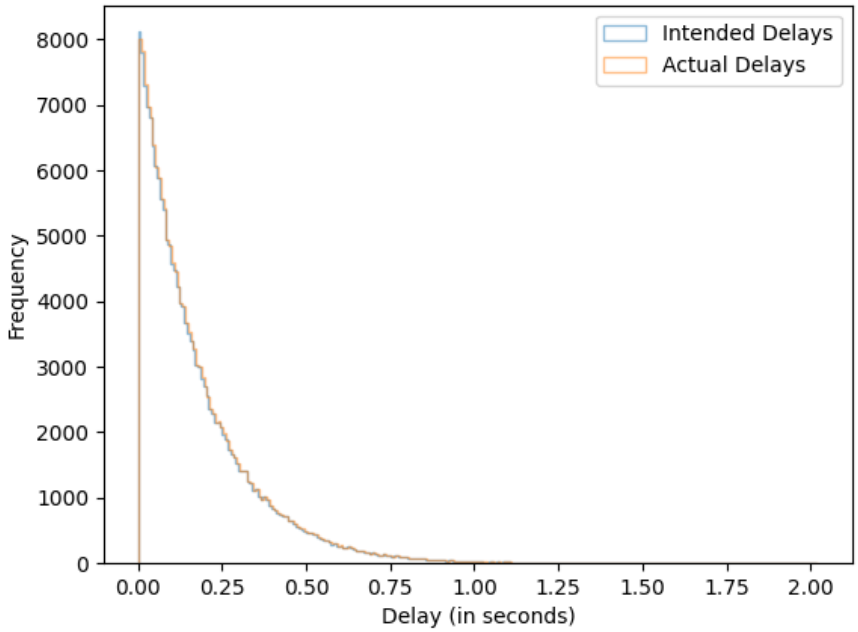


(c) $d_{E2E} = 0.2s$

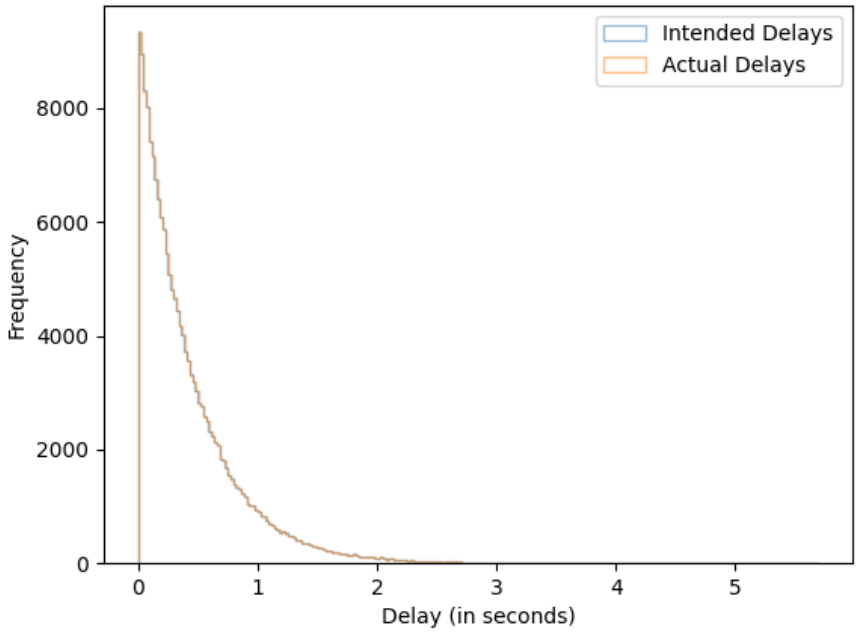


(d) $d_{E2E} = 0.25s$

Figure B.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 2 in section 3.2.

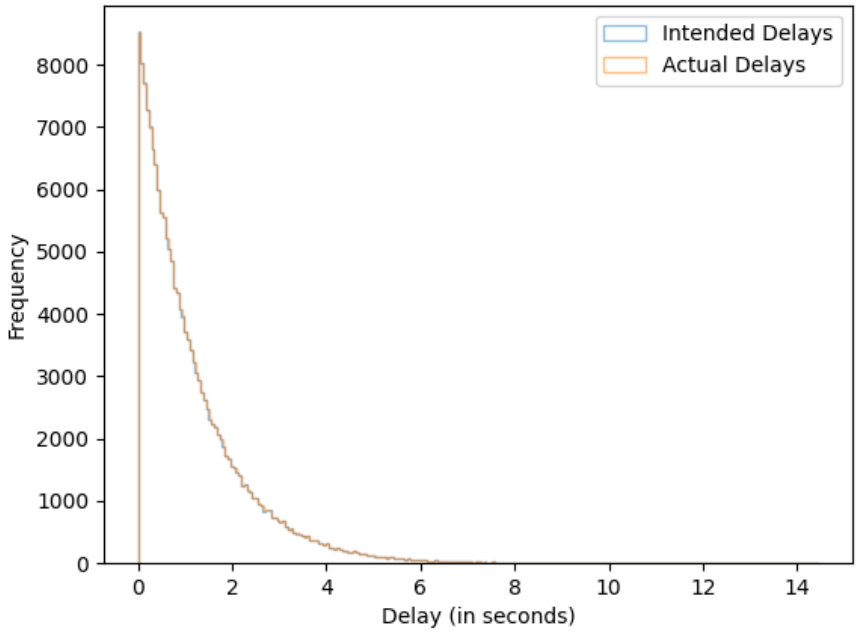


(e) $d_{E2E} = 0.5s$

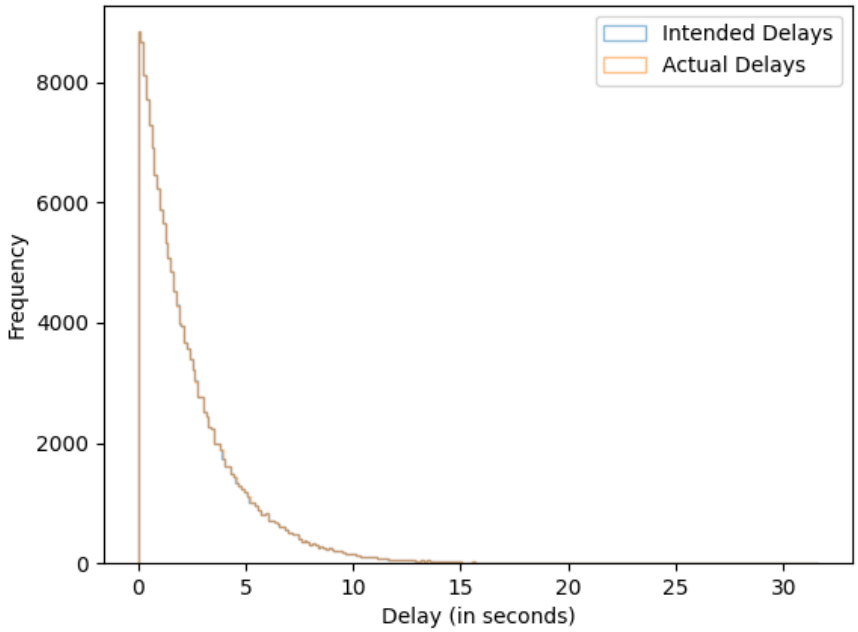


(f) $d_{E2E} = 1.0s$

Figure B.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 2 in section 3.2.

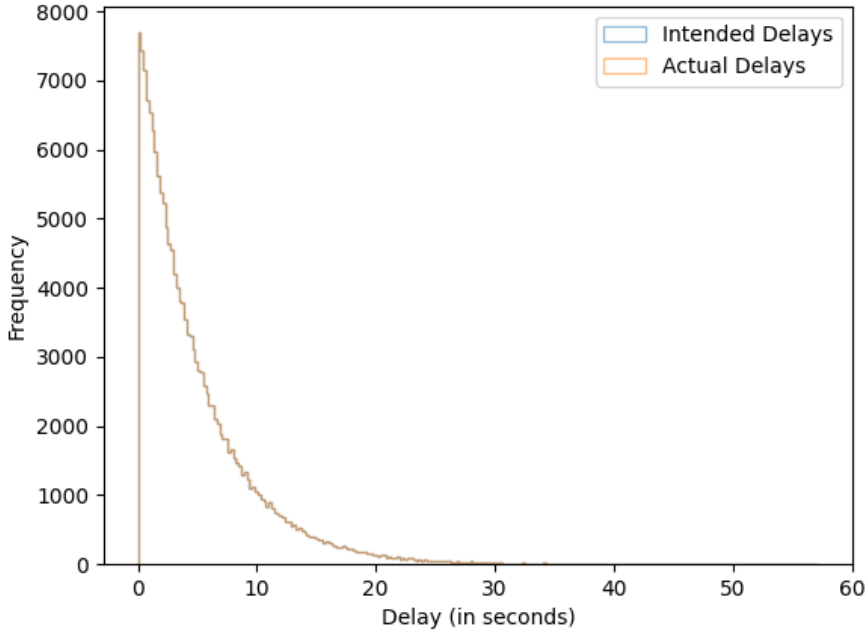


(g) $d_{E2E} = 2.5s$



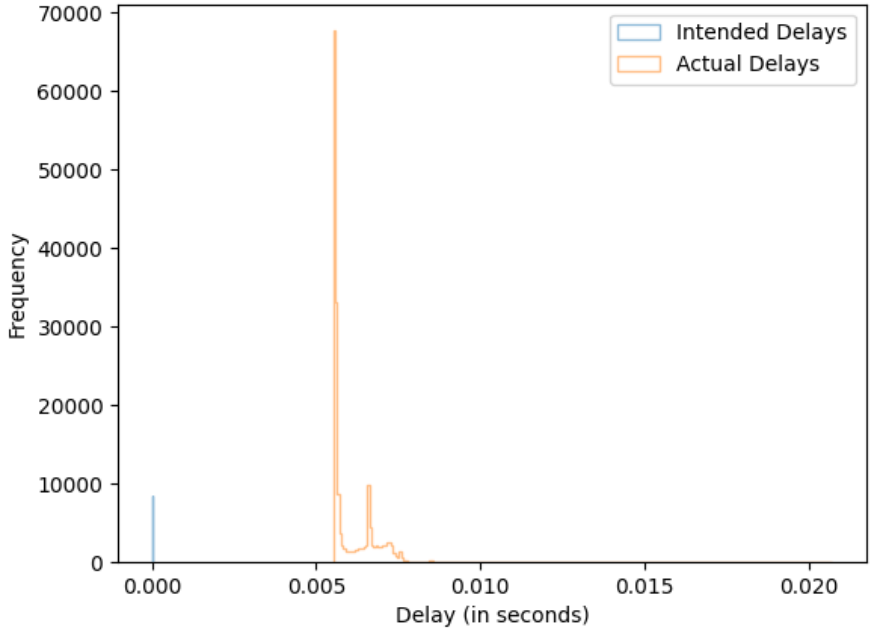
(h) $d_{E2E} = 5.0s$

Figure B.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 2 in section 3.2.



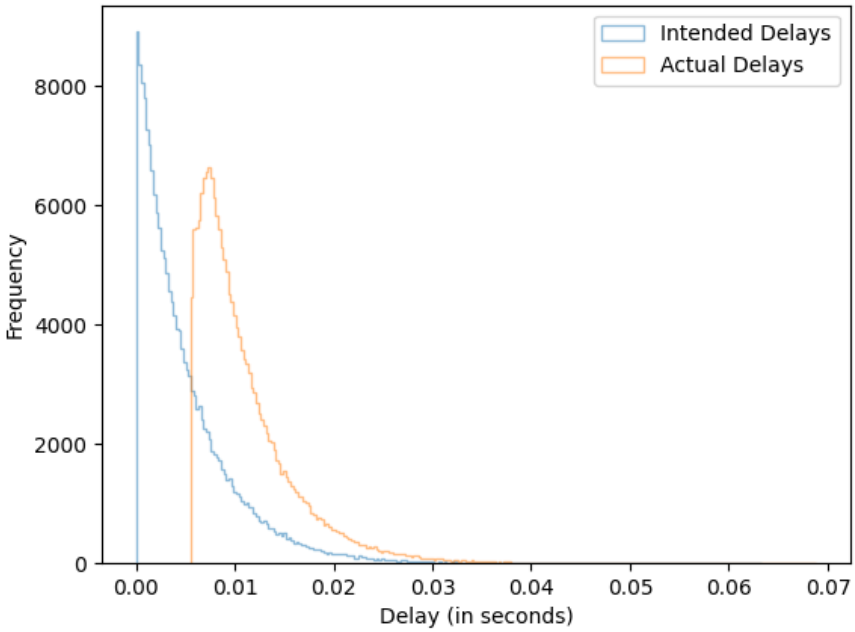
(i) $d_{E2E} = 10.0s$

Figure B.2: Distributions of the intended and actual delays of the packets in the simulation on the Macbook under approach 2 in section 3.2.

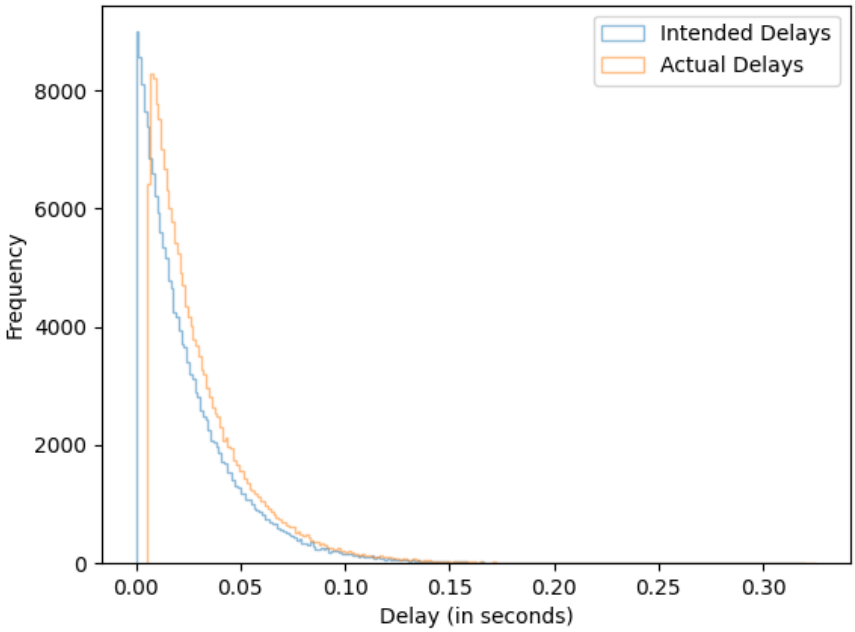


(a) $d_{E2E} = 0.15000001s$

Figure B.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 2 in section 3.2.

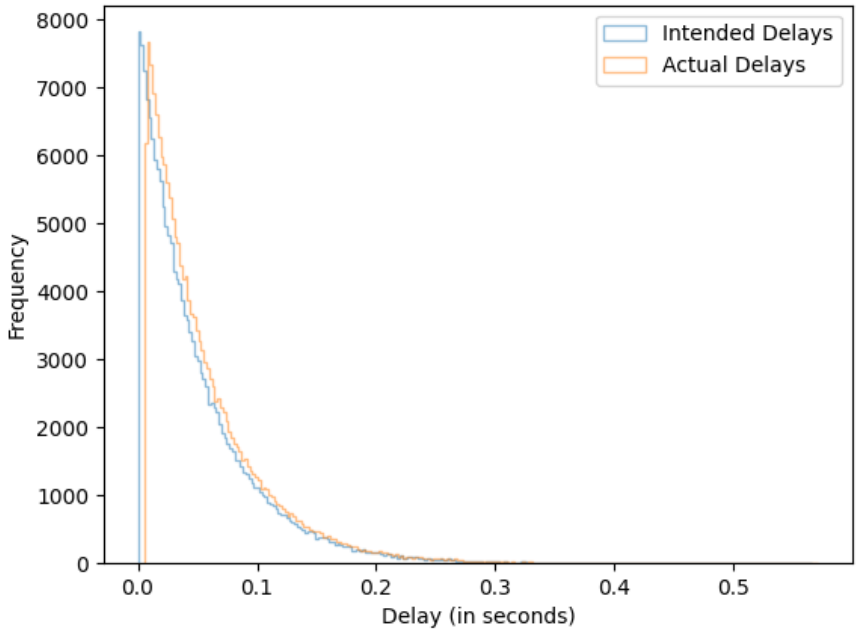


(b) $d_{E2E} = 0.16s$

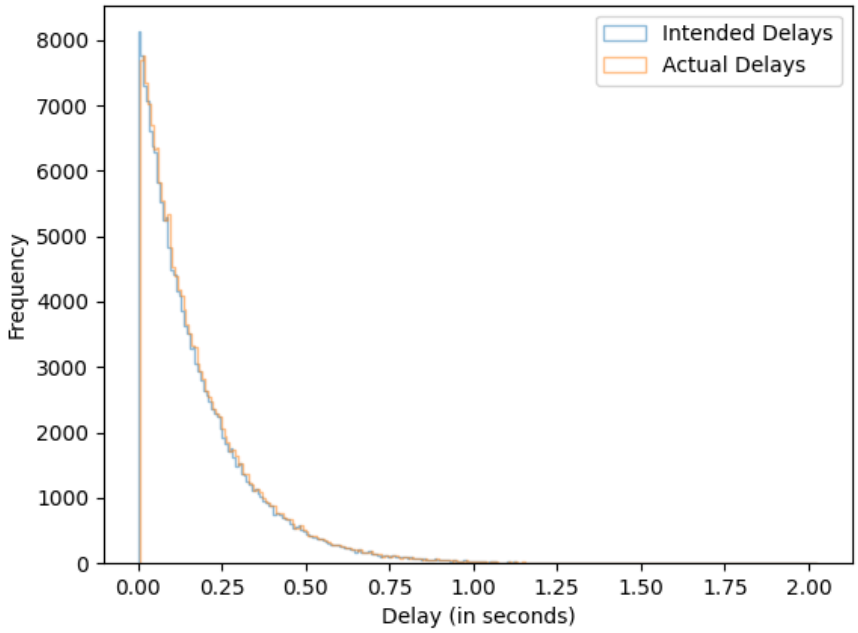


(c) $d_{E2E} = 0.2s$

Figure B.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 2 in section 3.2.

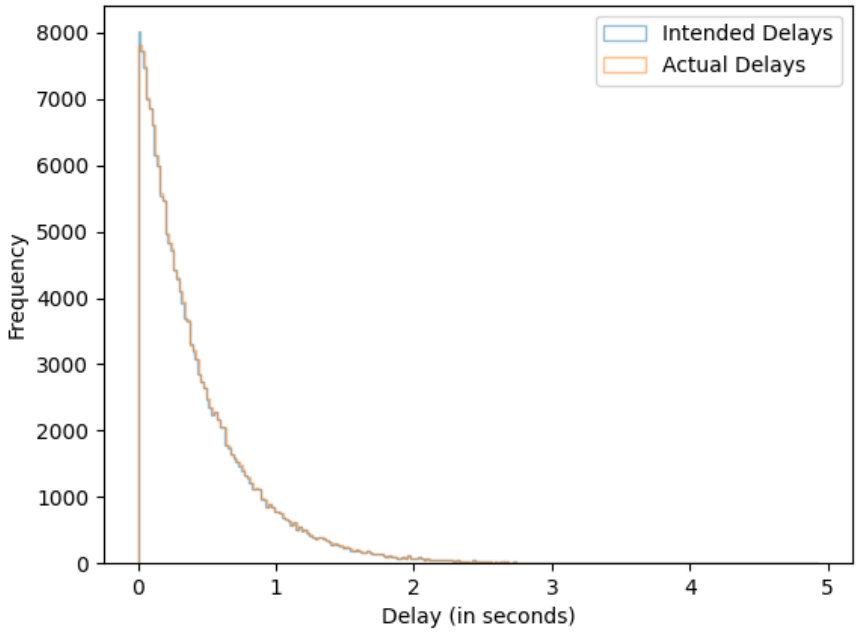


(d) $d_{E2E} = 0.25s$

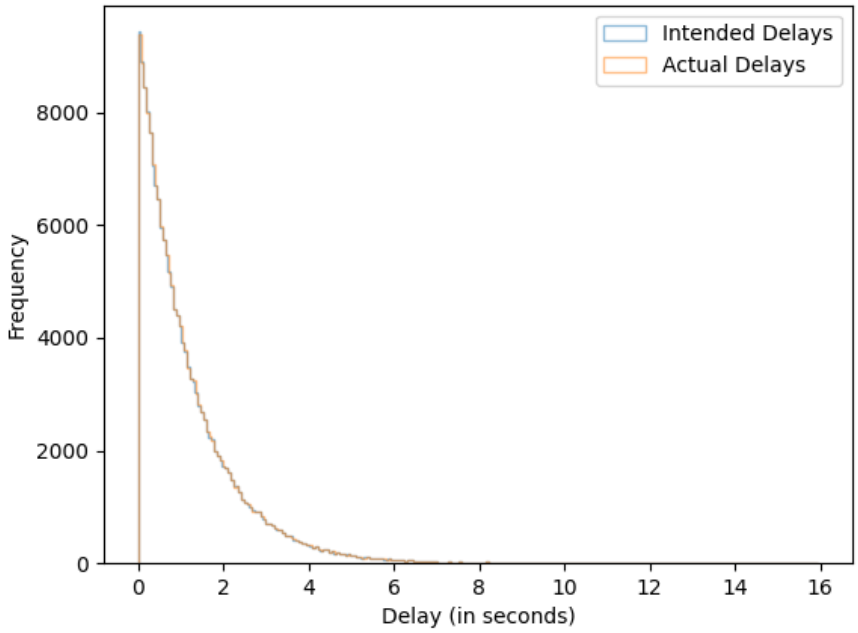


(e) $d_{E2E} = 0.5s$

Figure B.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 2 in section 3.2.

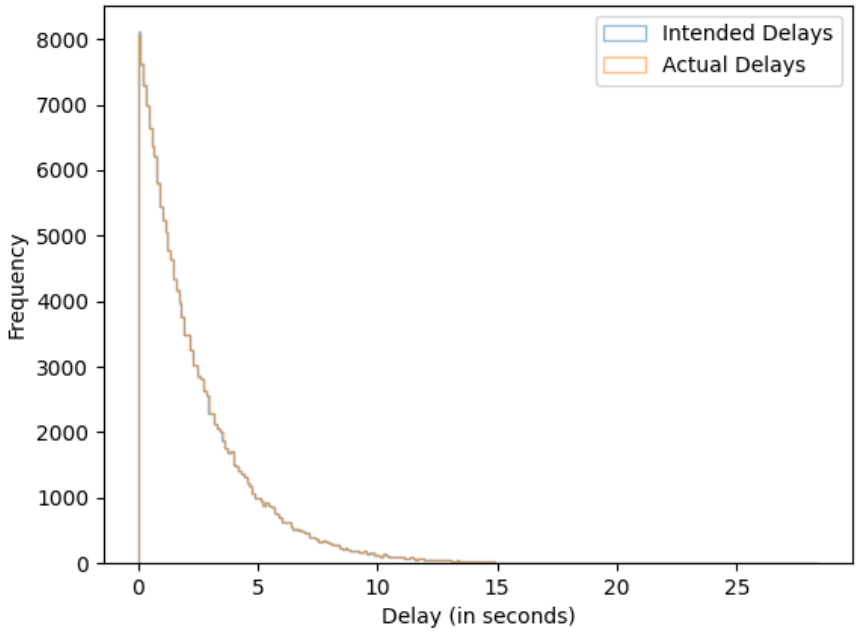


(f) $d_{E2E} = 1.0s$

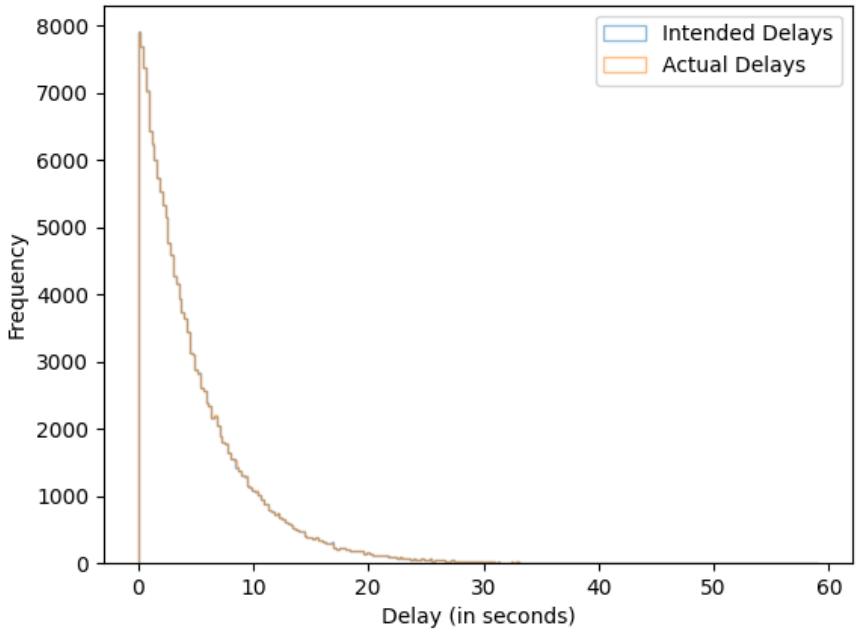


(g) $d_{E2E} = 2.5s$

Figure B.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 2 in section 3.2.

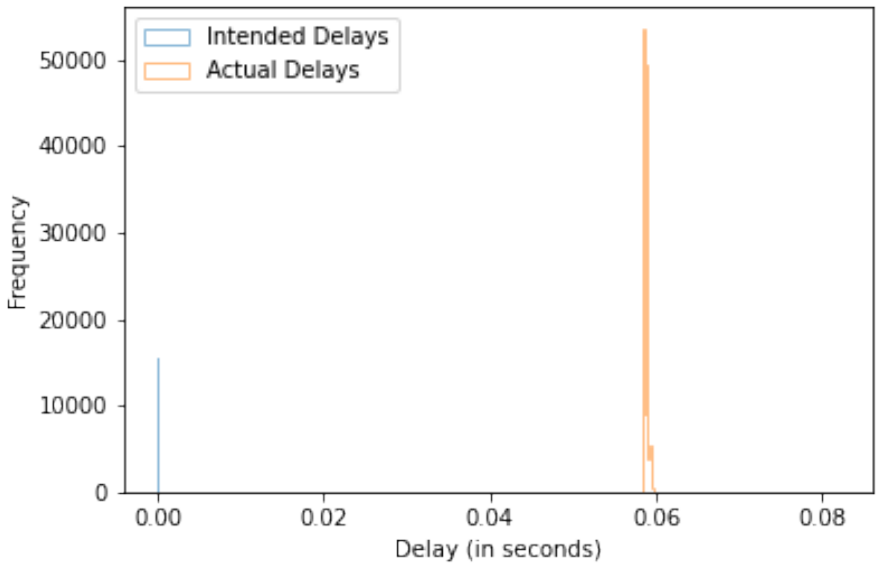


(h) $d_{E2E} = 5.0s$

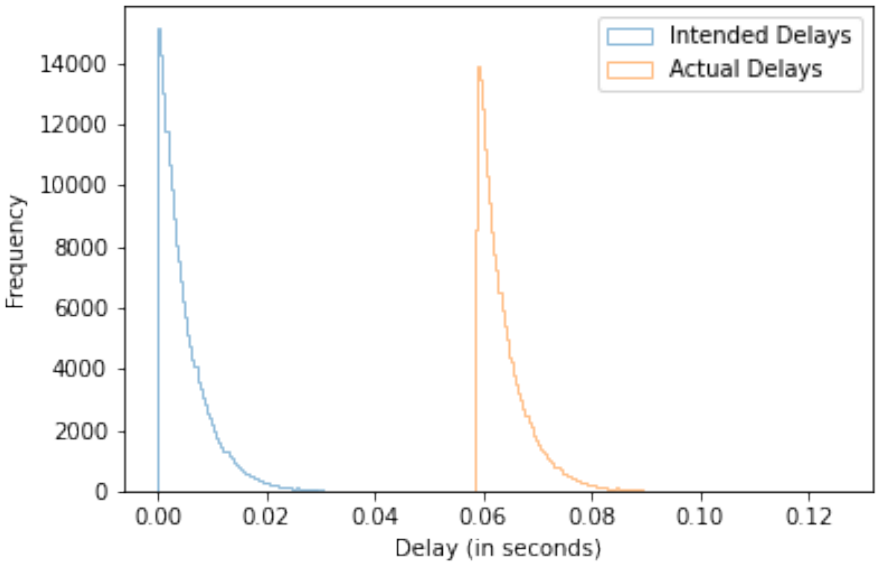


(i) $d_{E2E} = 10.0s$

Figure B.3: Distributions of the intended and actual delays of the packets in the simulation on DICE (SC) under approach 2 in section 3.2.

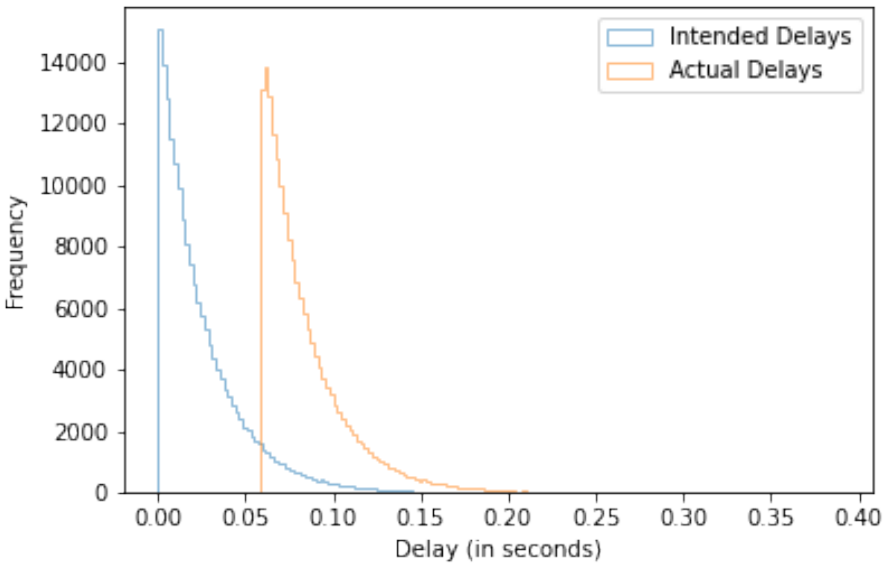


(a) $d_{E2E} = 0.15000001s$

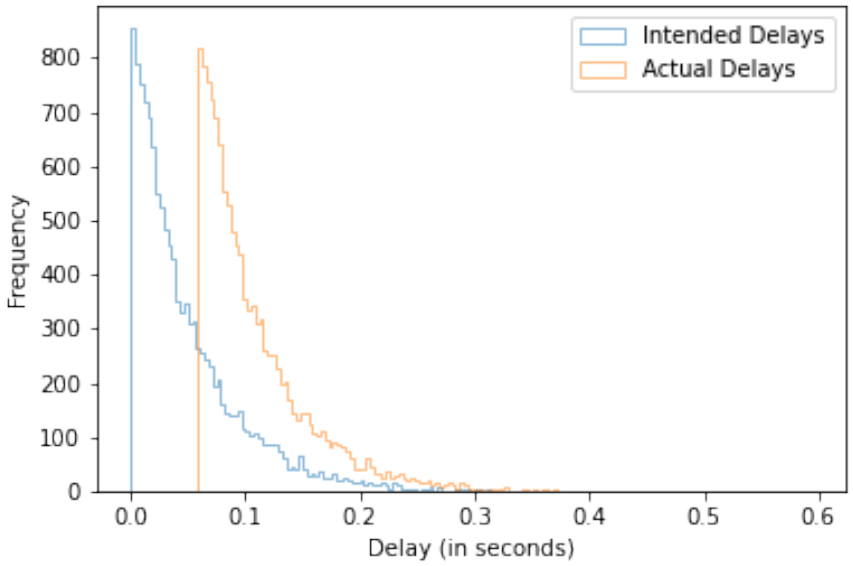


(b) $d_{E2E} = 0.16s$

Figure B.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 2 in section 3.2.

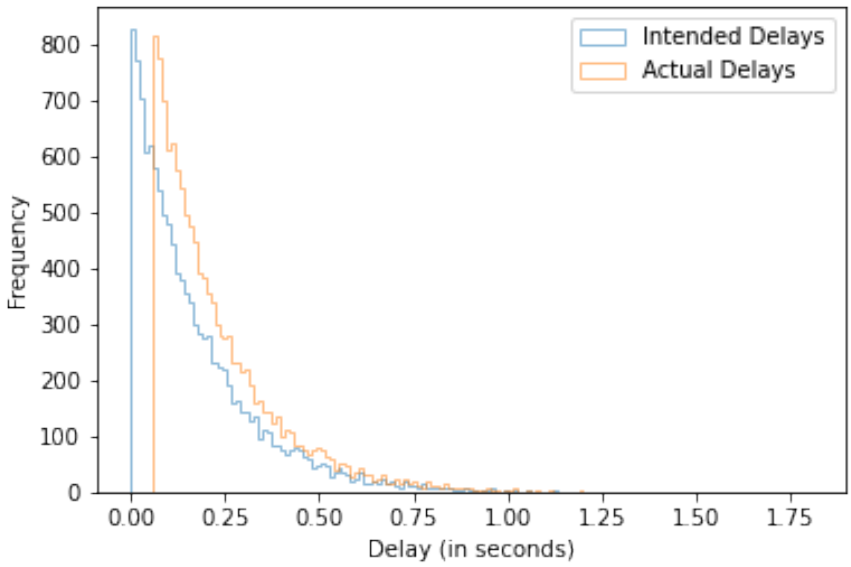


(c) $d_{E2E} = 0.2s$

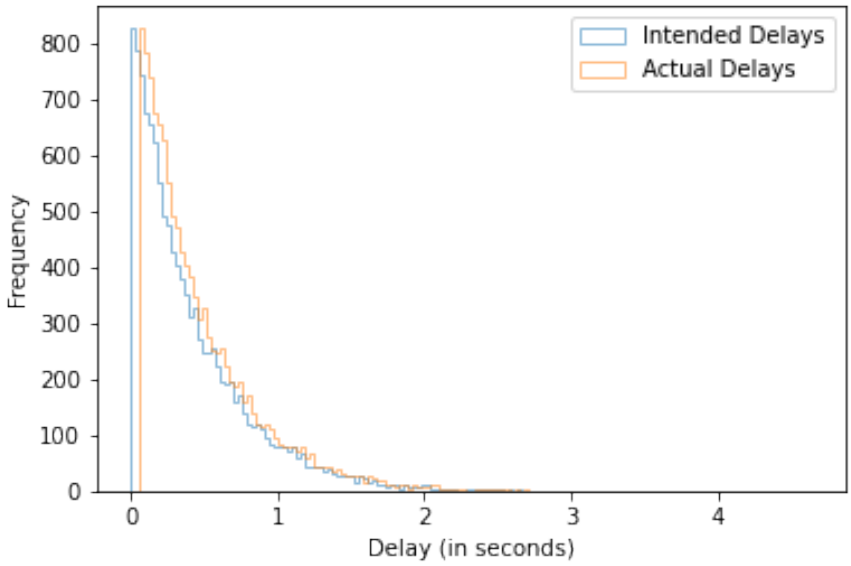


(d) $d_{E2E} = 0.25s$

Figure B.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 2 in section 3.2.

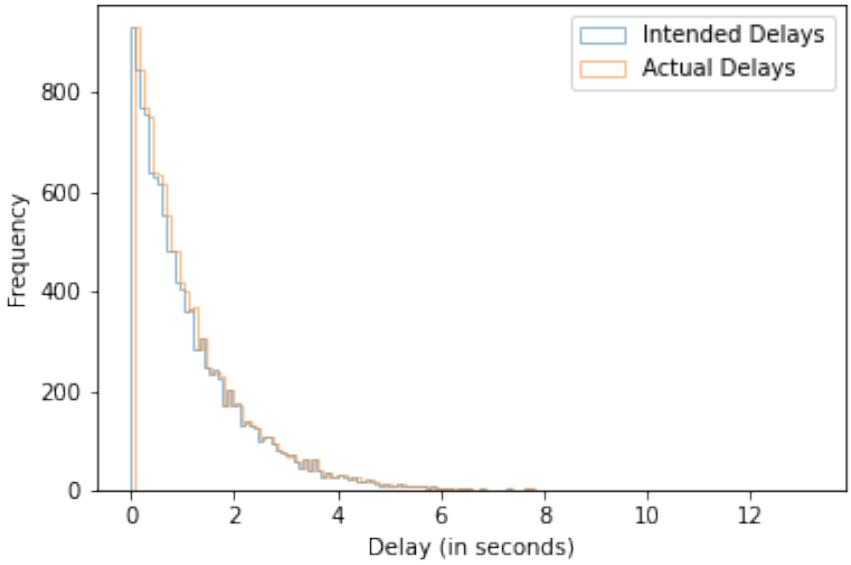


(e) $d_{E2E} = 0.5s$

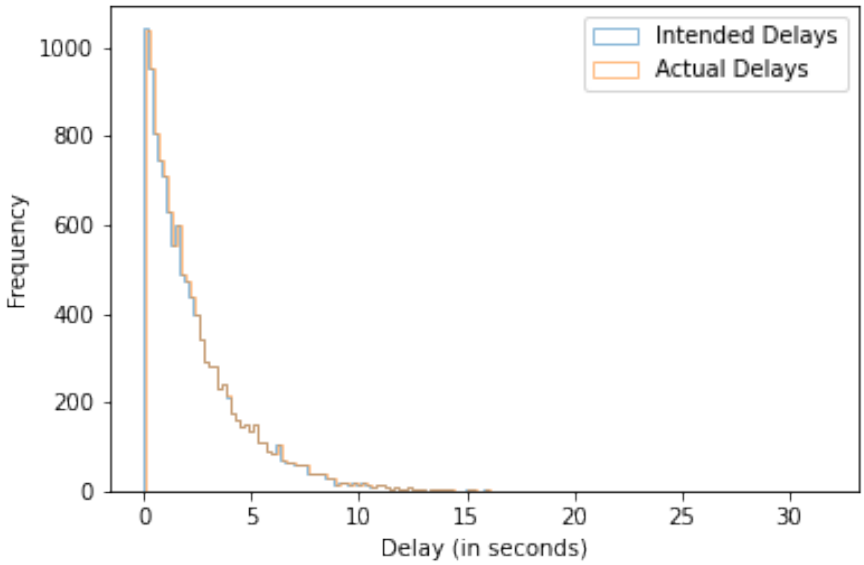


(f) $d_{E2E} = 1.0s$

Figure B.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 2 in section 3.2.

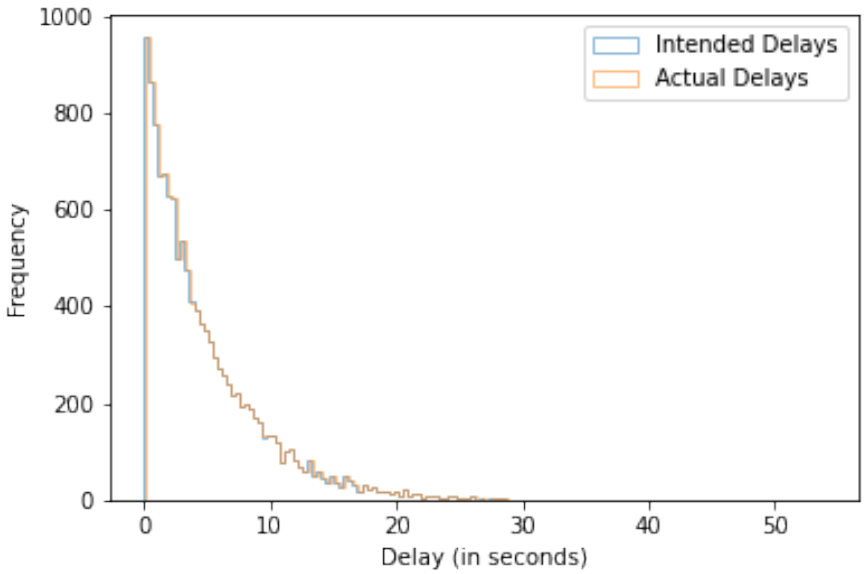


(g) $d_{E2E} = 2.5s$



(h) $d_{E2E} = 5.0s$

Figure B.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 2 in section 3.2.



(i) $d_{E2E} = 10.0s$

Figure B.4: Distributions of the intended and actual delays of the packets in the simulation on the Raspberry Pi under approach 2 in section 3.2.