



University of
Zurich^{UZH}

Peer-Based Mixing for Decentralized Federated Learning

Sandrin Raphael Hunkeler, Linn Anna Spitz
Zurich, Switzerland
Student ID: 18-253-815, 17-923-665

Supervisor: Dr. Jan von der Assen and Chao Feng
Date of Submission: September 19, 2025

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, September 16, 2025



Sandrin Hunkeler

Zürich, September 16, 2025



Linn Spitz

Abstract

Deutsch

Diese Arbeit stellt ein praktikables DFL-Framework (Decentralized Federated Learning) für unverlinkbares, dezentralisiertes föderiertes Lernen vor, das den Modellaustausch in dynamischen und instabilen Netzwerken mithilfe von Mixnets ermöglicht. Das Framework verwendet ein integriertes, peer-basiertes Mixnet für den fragmentbasierten Modellaustausch, das unabhängig von der Identität des Senders funktioniert. Eine kontinuierliche Überwachung der Peers ermöglicht eine dynamische Neukonfiguration der Lern- und Mixabläufe, um schnell auf hinzukommende oder aussteigende DFL-Teilnehmer zu reagieren. Eine robuste, auf Sphinx-Paketen basierende Onion-Routing-Netzwerkschicht bietet anonyme Erhaltensbestätigungen, verlässliche Wiederholungen von Nachrichten und Erkennung von Duplikaten, um eine hohe Stabilität unter nicht-optimalen Netzwerkbedingungen zu gewährleisten. Das integrierte, peer-basierte Mixnet bietet vollständig konfigurierbares Mixen von Verzögerung, Durchmischung, Routing und die Generierung von inhaltslosem Nachrichtenaustausch zur Verbesserung der Unlinkbarkeit. Ein vollständig integrierte Infrastruktur zur Überwachung ermöglicht Echtzeiteinblicke in Lernqualität, Kommunikationsverhalten, Ressourcenverbrauch und Rundensynchronisation, zugänglich über eine intuitiv gestaltete Benutzeroberfläche.

Experimente und Analysen zeigten Kommunikations- und Leistungsabstriche sind invers proportional. Kurze, stochastisch gleichmässige Sendeintervalle halten das Netzwerk reaktionsfähig, erhöhen jedoch die Last auf die DFL Teilnehmer. Eine theoretische Entropieanalyse bewertete die Parametrisierung und zeigte, dass eine grosse Outbox die Entropie verbessert, ohne Latenzen signifikant zu erhöhen. CPU-Zeit wurde als begrenzende Ressource identifiziert, da kontinuierliches Mischen im Hintergrund zunehmend mit dem Lernen in allen Phasen des DFL-Ablaufs konkurriert. Die Dauer der individuellen Runden steigt ungefähr linear mit der Anzahl der Teilnehmenden und Zwischenhops, während der Arbeitsspeicherverbrauch moderat bleibt.

Der anonyme, fragmentbasierte Aggregationsalgorithmus führte zu stabilen globalen Modellen, die das durchschnittliche lokale Modell in der Genauigkeit übertreffen und unter optimalen Bedingungen mit FedAvg gleichgestellt sind, während sie robust gegenüber spontanen Änderungen in der Topologie und gegenüber partiell übertragenen Modellen bleiben.

Einschränkungen des neu implementierten Frameworks umfassen die Anforderungen an die Rechenleistung der Teilnehmer, die Latenz sowie das Fehlen von Methoden zur Erkennung und Bekämpfung von böartigem Verhalten.

English

This thesis presents a practical DFL framework for unlinkable decentralized federated learning that achieves model exchange in dynamic and unstable networks using mixnets. The DFL framework deploys an integrated peer-based mixnet for fragment-based model exchange, independent of sender identity. Continuous peer health monitoring allows reconfiguring learning and mixing workflows dynamically to support spontaneously joining or exiting nodes during training. A robust onion routed networking layer built on Sphinx packets provides anonymous acknowledgments, reliable resending, and duplicate handling for high resilience in non-optimal networking conditions. The integrated peer-based mixnet provides fully configurable mixing for delaying, shuffling, routing, and generating dummy-traffic for improved unlinkability. A full integrated observability stack offers real-time insights into quality of learning, communication behavior, resource consumption, and round synchronization, accessible through an intuitive graphical user interface.

Experiments and analysis illustrated communication and performance trade-offs. Short, stochastically uniform sending intervals keep the network responsive but increase peer load. Theoretical entropy analysis evaluated parametrization and showed that a large outbox size can increase entropy without significantly worsening the latency of the system. CPU-time was identified as the limiting resource as continuous mixing in the background increasingly competes with learning across all stages of the DFL workflow. Round duration increases approximately linearly with the number of nodes and intermediate hops while memory remains moderate.

The anonymous fragment-based aggregation algorithm resulted in stable global models that outperform average local models and match FedAvg in optimal conditions, while remaining robust to spontaneous changes in topology and partial transmissions.

Limitations of the newly proposed framework include computational overhead, latency, and the absence of methods for detection and mitigation against malicious behavior.

Acknowledgments

We would like to express our sincere gratitude to our supervisor, Dr. Jan von der Assen. His consistent support, valuable feedback, and generous time investment have been crucial throughout this thesis.

Additionally, We are grateful to Prof. Dr. Burkhard Stiller for the opportunity to complete our Master's Project at the Communication Systems Group (CSG) of the University of Zurich. This experience has been personally enriching and has enhanced our scientific skills. Thanks to everyone who has been part of this journey.

Contents

Declaration of Independence	i
Abstract	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	2
2 Background	5
2.1 Federated Learning	5
2.1.1 Privacy and Security	6
2.1.2 Nebula	6
2.2 Mixnets	6
2.2.1 Fundamental Logic	6
2.2.2 Attacks on Mixnets	7
3 Related Work	9
3.1 Cryptography	9
3.1.1 Threshold Based Decryption	9
3.1.2 Homomorphic Encryption	10
3.1.3 Ring Keys	10
3.2 Networking	10

3.2.1	Tor-Protocol	10
3.2.2	Blockchain	11
3.3	Model Obfuscation	11
3.4	Identity	12
3.5	Mixnets	12
3.6	Summary	15
3.6.1	Discussion	15
4	Architecture	19
4.1	Design Decisions	19
4.2	Unlinkability Trough Mixnet	19
4.2.1	Mixing Logic	21
4.3	Components	22
4.3.1	User Interface	23
4.3.2	Manager	23
4.3.3	DFL / Mixing Node	23
4.4	Workflows	23
4.4.1	DFL Workflow	24
4.4.2	MixNet Workflow	24
5	Implementation	27
5.1	Components	27
5.1.1	Manager	27
5.1.2	User Interface	27
5.1.3	Node	29
5.1.4	Learning	29
5.1.5	Communication	31
5.2	Workflows	31
5.2.1	Manager	32
5.2.2	Learning	32
5.2.3	Communication	32

<i>CONTENTS</i>	ix
6 Evaluation	37
6.1 Dynamic Topology	37
6.1.1 Implications on Workflows	37
6.1.2 Observed Metrics Topology Change	38
6.2 Communication and Mixing	42
6.2.1 Network Metrics	42
6.2.2 Entropy	45
6.3 Computational Resources	48
6.3.1 CPU-Time	48
6.3.2 Memory	50
6.3.3 Time	51
6.4 Learning and Accuracy	52
6.5 Mitigation of DFL Threats	54
6.6 Comparison to Nebula	55
7 Summary and Conclusions	57
7.1 Limitations	58
7.2 Future Work	59
Bibliography	61
Abbreviations	67
List of Figures	67
List of Tables	70
.1 Source Code and Reproducibility	71

Chapter 1

Introduction

Conventional machine learning has several key limitations introduced by its centralized orchestration, including the risk of data leakage and a centralized point of failure [1]. Decentralized Federated Learning (DFL) addresses these issues by decentralizing the training process, ensuring data privacy by keeping the data localized within the distributed nodes [2]. Instead of sharing raw training data, DFL enables nodes to collaboratively train models by periodically exchanging locally trained model parameters [3].

Despite only sharing the model parameters, DFL still exposes participants to the risks of being linked to each other. To mitigate this risk, this thesis introduces a framework of peer-based mixing in Decentralized Federated Learning, leveraging the concepts of *Mixnets*, first proposed in 1981 by Chaum [4]. The following chapter motivates the system and provides an overview of the structure of this work.

1.1 Motivation

DFL enables distributed nodes to jointly train machine learning (ML) models without sharing raw data. However, the mutual trust required during model exchange introduces vulnerabilities that can compromise participants' privacy or network unlinkability. ML models retain traces of private data they were trained on, making it possible to extract sensitive information through unintended memorization [5]. Additionally, revealing the models' network origin during the exchange exposes its owner and undermines the global privacy of peers [6].

The literature review conducted within this project highlights that existing research has primarily focused on enhancing privacy and unlinkability in centralized federated learning (CFL), which utilizes a centralized aggregator. Related work often extends CFL with trusted entities to shuffle or obfuscate models before aggregation. While these methods improve privacy and network unlinkability, they reduce the decentralized and distributed nature of FL, partially defying its core purpose. Moreover, trusted entities introduce another single point of failure, creating vulnerabilities to attacks, exposing identities and network links.

This project identifies a critical research gap in applying peer-based mixing for anonymous model fragments in DFL. Addressing this gap motivates the proposed framework that enhances privacy and unlinkability while preserving the decentralized architecture of DFL.

1.2 Description of Work

This thesis introduces and implements a new framework for peer-based mixing in DFL. Unlike existing approaches, this framework adopts a fully decentralized and distributed architecture for mixing and learning. Leveraging the architectural properties of DFL, each peer has two roles as both a mixing node and a DFL node, operating independently without centralized orchestration.

Each node splits its model into multiple anonymous fragments, whereas each fragment is tagged with its index position in the model. Aggregation is performed using a fragment-based *Federated Averaging* (FedAvg), which is robust to only partially transmitted models, caused by spontaneous changes in topology or exited nodes. The framework is designed to adapt dynamically to changes in topology, ensuring workflows are resilient against participants newly joining or unexpectedly exiting.

All fragments are transmitted through a peer-based mixnet, hosted directly by the DFL nodes themselves. Each fragment is encrypted independently into an onion packet, which is routed, shuffled, delayed, and acknowledged anonymously as part of the mixing process. To obscure sensitive time periods when model fragments are exchanged, each node maintains a stochastic output rate by sampling delays between consecutive transmissions from a normal distribution to obscure timing patterns. Additionally, in the absence of meaningful messages, the framework generates dummy traffic to ensure consistent network activity and further enhance unlinkability.

Both the mixnet and the DFL workflows are fully configurable and can be observed by the frameworks' self-hosted monitoring infrastructure. The framework consists of three main components: the frontend, which displays various real-time metrics of the ongoing scenario; the manager, responsible for deploying and managing scenarios; and the nodes, operating as independent and autonomous DFL participants.

The framework is evaluated across multiple dimensions, including architectural properties, performance, and resource utilization. Furthermore, the impact of the mixnet on both the learning and communication workflows is assessed. Computational resource utilization, including time, CPU, and memory usage is evaluated to determine the framework's trade-off between using resources for learning and increasing privacy. At last, the theoretical properties of the mixnet, as a source of unlinkability and enhanced privacy, are examined from a theoretical perspective.

The next chapter covers key background concepts. After this, a literature review examines DFL and mixnets, which leads to the research gap addressed by this work. Next, important design decisions are addressed as the architecture is explained, with implementation details in the subsequent chapter. This is followed by the experiments and corresponding

evaluation. Finally, the research and its findings are summarized and possibilities for further work are highlighted.

Chapter 2

Background

The following chapter summarizes the theoretical foundation of this work. The initial section introduces federated learning (FL). The second section introduces *Nebula*, which served as the architectural reference for the foundational design of the proposed framework. The third subsection covers the relevant aspects of mixnets and their common threat models.

2.1 Federated Learning

Conventional machine learning has several key drawbacks introduced by its centralized orchestration, such as data leakage and centralized point of failure [1]. FL addresses these drawbacks by decentralizing data collection to maintain data privacy [2]. With this novel approach, distributed nodes train a ML model locally while periodically sharing the model parameters instead of their private training data [3]. When exchanged, the shared model parameters are aggregated until convergence is reached [7].

By transmitting the model instead of the vast amounts of locally collected data, FL not only increases the data privacy but reduces the network overhead [1]. While data privacy is increased, the distribution of the model reduces confidentiality, which enables more advanced attacks by gaining insights on the internals of the model itself [7].

Federated learning can be categorized by the topology of how the locally trained model parameters are exchanged and aggregated [3]. In centralized federated learning (CFL), the training FL nodes send their local model to a central node for aggregation [7]. CFL still faces similar disadvantages similar to traditional ML such as single point of failure and limited scalability [8]. Semi-distributed federated learning (SDFL) reassigns the role of the aggregator to one or multiple nodes at each round.

While CFL is the most common and established topology, distributed federated learning (DFL) builds upon the advantages of FL [3]. DFL omits the central aggregator node by

assigning each node both roles, aggregator and trainer [3]. DFL has the natural advantages of decentralized systems, such as high fault tolerance, scalability, and mitigating the risks of a single point of failure [3]. However, it also introduces common disadvantages of centralized systems such as increased communication overhead [3]. Furthermore, techniques for aggregation which result in high training optimization are more challenging than for centralized topologies [3], [9]. Computational resources for training, aggregation, and communication are key challenges of networks consisting of consumer-grade DFL nodes [3], [9].

2.1.1 Privacy and Security

Privacy in FL can be classified by local and global privacy, whereas global privacy focuses on communication while local privacy concerns the training and aggregation process [6]. DFL has a higher local privacy compared to CFL by omitting the centralized trusted entity [6]. However, both approaches involve sharing parameters between entities which allows extracting sensitive information through unintended memorization [5]. In addition, compromised nodes can gossip tampered models through the federation against which aggregating nodes deploy various defense mechanisms, such as anomaly detection [10].

2.1.2 Nebula

Nebula is a flexible platform for deploying FL scenarios with extensive options to configure [11]. Its front end guides users during the scenario configuration for setting up network topology, model architecture, algorithms, and datasets. Furthermore, it provides real-time monitoring of the communication and the learning progress. The platform mainly supports training image classification on datasets like MNIST and CIFAR-10, training models such as LeNet5, CNN, and MobileNet. Nebula's communication module is based on asynchronous messaging and incorporates gossiping mechanisms, allowing messages propagate through nodes.

2.2 Mixnets

Mixnets were first proposed in 1981 by Chaum [4] as a solution to the *traffic analysis problem*. The problem defines the cryptographic challenge to hide the sender's and receivers' identity, rather than the message's content itself. The problem is also known as the *unlinkability problem*.

2.2.1 Fundamental Logic

Mixnets consist of a network of intermediary nodes, known as *mixes*, which obfuscate the sender and receiver of a message. A message that is sent through a mixnet, traverses the

mixes, which hide the path of the message. In order to do this, the mixes perform two fundamental steps [12]:

- Perform a cryptographic operation on the message, for which various techniques can be used.
- Collect a batch of messages overtime, permute the messages, and pass them on within one randomized batch.

In addition to the advantage of unlinkability of the original sender and receiver, mixnets require no central governing authority [4]. Their use of decentralized techniques make them naturally suited for increasing global privacy in DFL.

In addition to the fundamental principles, mixes deploy a variety of techniques to prevent passive or active entities from undermining the anonymity. Senders add junk bits to the original message to send only packets of equal size to prevent length-based inference [13]. Furthermore, to counter time-analysis mixes may cache incoming packets and send them after a randomized delay or in shuffled batches [13]. Various strategies of *flushing* the messages exist, such as using a static threshold, randomized batch sizes or using additional dummy messages [14].

Latency is a major consideration in mixnets. The batching process requires aggregating messages over time, which by nature results in delays. Additionally, cryptographic operations can be computationally expensive and can have negative effects on communication overhead, as they often increase message size [12]. Thus, mixnets are impractical for use in real-time communication such as most web surfing or live communication via audio or video. Furthermore, disrupting a single node can result in a loss of the sent information [15].

2.2.2 Attacks on Mixnets

Attacks on mixnets can be classified into passive and active attacks. Active attackers manipulate computation and message, whereas passive attacker are limited to observing the network [14].

Passive Attacks

Passive attacks on mixnets rely on network analysis techniques. Eavesdroppers know the topology of the network and the algorithmic properties of randomized routing, batching and delaying, but do not know the actual realization of randomized processes [13].

To protect themselves from passive attacks, mixnets maximize the entropy of their networking properties to lower the probability of an eavesdropper identifying a source-destination pair [13]. High delay and traffic increase the anonymity but are constrained by the performance requirements [16].

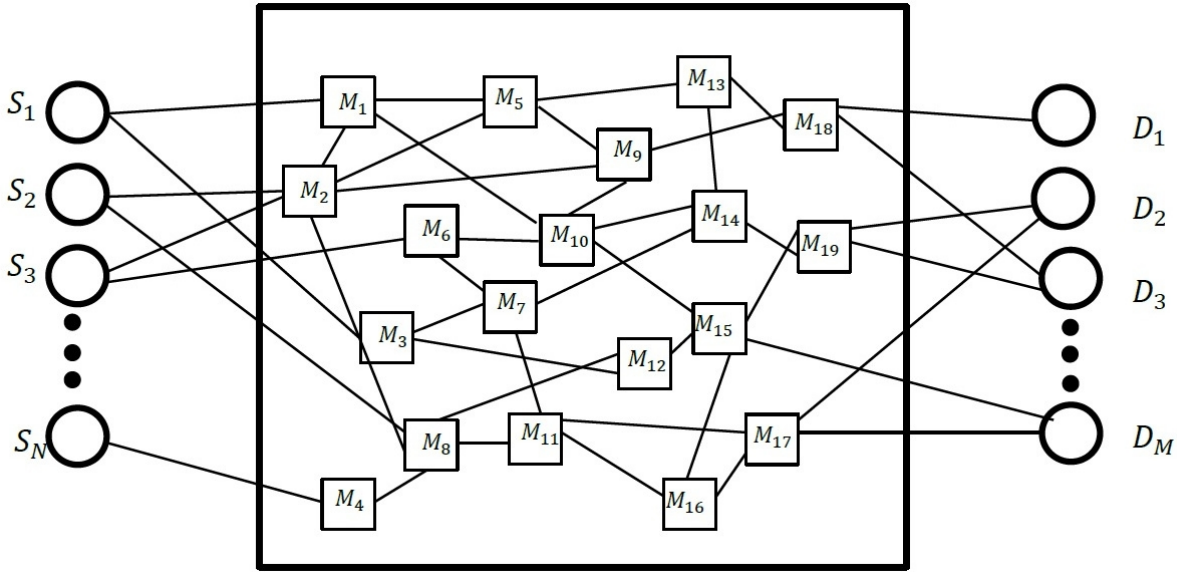


Figure 2.1: System Model of Mixnets
[13]

Passive brute force attacks record every routed message to identify sender-receiver pairs or reduce the whole set to possible combinations [14]. Dummy messages and high variability in selecting the next hop are countering the attack by increasing the computational cost for the attack [14].

Active Attacks

In an $N-1$ attack, an adversary sends $t - 1$ messages to a mix node with a batch limit of t . These messages are identifiable by the adversary to isolate and track the remaining external message. Mix-nodes remix the messages, sending them as original messages to the network itself, makes it harder for the attacker to identify the message [14].

Timing attacks aim at uncovering routing patterns by analyzing transmission times of packets. Randomized artificial delays are a counter measure but cannot prevent an attack since the variability in round trip time (RTT) cannot be hidden completely [14].

Contextualized attacks are among the most dangerous, as they make use of situational awareness. Having insights on user behavior, such as synchronous communication involving two end-nodes being similar active, enables identifying communicating users by various techniques such as packet counting [14].

Denial of service attacks (DOS) on mix nodes affects the overall routing behavior of the network and increases the attack surface for data analytics [14].

Send and seek attacks are performed by malicious senders which try to uncover the recipient's identity by sending messages in identifiable messaging patterns which can be tracked through the network [14].

Chapter 3

Related Work

The following chapter provides an overview of approaches to achieve unlinkability and privacy. The goal is to prevent an adversary from linking updates to specific worker nodes and from retrieving private information from the models themselves. Each of the following techniques was used in some combination by the analyzed frameworks.

3.1 Cryptography

Cryptographic approaches for ensuring unlinkability mostly rely on differential privacy. Three main approaches are used frequently, Shamir’s secret sharing, Paillier homomorphic encryption, and ring keys.

Shamir’s secret sharing introduces a threshold based decryption, which allows key holders to jointly decrypt an encrypted message. Paillier homomorphic encryption allows mathematical operations on encrypted models while maintaining privacy. Ring keys are used by a group of senders to sign their models as a group, which hides the original sender by hiding among peers.

3.1.1 Threshold Based Decryption

In [17], a trusted authentication server creates threshold-based key pairs for worker nodes. Each worker node can be deanonymized by a minimal number of worker nodes. A reputation based scheme determines trustworthy worker nodes and assigns them the role of aggregators and a random aggregator as a tracer. Ring keys are used to sign model updates. If a ring is suspected of having a malicious participant, clients collaborate with the tracer to deanonymize the malicious node among them. If the tracer receives the necessary number of fragments for identifying the malicious node, it informs the authentication server which reveals the malicious node’s identity.

[18] uses Shamir’s secret sharing, a threshold-based cryptographic scheme for publishing global model updates to ensure they can only be decrypted by collaboration of the semi-trusted FL participants. Worker nodes get a pseudonym assigned by a trusted authority for every round to improve unlinkability.

3.1.2 Homomorphic Encryption

[18] proposes that the worker nodes encrypt their model using Paillier homomorphic encryption, which allows aggregation without decryption. This protects the worker nodes’ privacy from honest-but-curious aggregators.

3.1.3 Ring Keys

[19] uses ring keys to send their model update to a group leader which aggregates the models and sends the group update to the central aggregator. The architecture makes use of one-time addresses (OTD) and one-time private keys (OPK) to publish the rewards for the participation. The rewards are stored on a blockchain and only accessible by the individual worker node. The rewards cannot be linked to a worker node and can only be accessed by the worker node by scanning the blockchain by checking access with their credentials.

[20] makes use of onion-like encryption using multiple layers of symmetric and asymmetric encryption involving homomorphic techniques.

3.2 Networking

Modern network communication addresses challenges such as privacy, security, and unlinkability. Various architectures have been designed to enhance anonymity and protect users from surveillance or correlation attacks. Among these, the Tor project, an open-source onion routing system, is widely known to provide privacy and unlinkability [21]. Other approaches, such as blockchain networks and mixnets, also focus on ensuring privacy and decentralization, each employing distinct mechanisms to break up communication patterns and mitigate adversarial analysis.

3.2.1 Tor-Protocol

[22] introduces *FedTor*, which implements a lightweight version of the Tor protocol, that enables computationally weak IoT devices to execute the protocol’s fundamental cryptographic processes. Each worker node has a dual role as a FedTor node. A newly introduced router selection server stores the reported bandwidth of the nodes, sent by the neighboring nodes. On request, the router selection server provides the worker nodes with the ratings

for them to select a path. Compared to the original Tor protocol, FedTor does not rely on the self-reported bandwidth.

[23] implemented the Tor-based GPor protocol with increased reliability compared to the original. Using ad-hoc on-demand distance vector (AODV) routing for lightweight path finding. The path from the worker node to the central aggregation server is determined by a centralized server upon request. Each path consists of a static number of hops through predefined groups of worker nodes. The groups themselves are formed by the authentication server upon registration of a worker node. Their implementation showed higher availability compared to the Tor implementation.

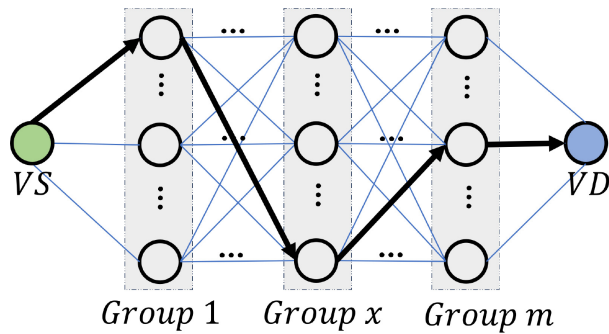


Figure 3.1: Grouped Onion routing [23].

3.2.2 Blockchain

[20] makes use of a modified private blockchain by embedding the aggregation process into the blockchain protocol. The blockchain is utilized by the worker nodes to store metadata about the models and transactions. Combined with an on-chain reputation system, the framework ensures high variability in the model aggregation process.

3.3 Model Obfuscation

Model obfuscation introduces noise to prevent an attacker from tracing model updates to a worker node. There exists a trade-off between obfuscation and model quality.

[24] computes the sign matrix of its model update $(-, +)$ and with a low probability replaces parameters at random. The randomized 1-bit model representation will still converge, even with induced noise, but with increased training time. On the other hand, the 1-bit representation of the gradient reduces computational and network overhead. The sign matrix is used by the central aggregator to reduce or increase the global model parameters.

In [18], workers use blinding factors to obfuscate their homomorphically encrypted model updates. The factor is removed only after the final aggregation and decryption of the latest global model. Similarly, [20] masks parts of each gradient update before applying

homomorphic encryption. [25] showed the impact on the accuracy of the model with increased privacy due to noise added to the gradient before sending it to a centralized shuffler.

3.4 Identity

[26] introduces a trusted authority (TA) that assigns pseudonyms to worker nodes to enforce anonymity while maintaining traceability. Cryptographic randomization in pseudonym and signature generation prevents adversaries from linking updates to specific workers. The aggregating node communicates outliers to the TA which can revoke the admission to the federation.

Similarly, [22], [24], and [27] introduce an authentication server for admission to the network. The TA mitigates *sybille* attacks by preventing the random generation of fake identities.

3.5 Mixnets

The introduction of mixnets in 1981 [4] enabled the anonymization of existing applications such as email [28], [29] and e-voting [12], [30]–[32]. The use of mixnets in anonymous browsing remains limited due to high latencies [33]. However, [34] introduced *NYM*, a consumer virtual private network (VPN), which offers a mixnet-based mode to achieve untraceability. Mixnets have never been applied publicly in FL at this point in time. In contrast to mixnets, the Tor protocol has been used for FL in the past [22], [23].

The Tor browser is a prominent app for anonymous browsing [35]. The underlying protocol bears similarities to mixnet systems, but with some key differences. In both Tor and mixnet systems, traffic is anonymized using intermediary nodes, in order to make it untraceable [21]. In mixnets however, the nodes perform two important steps. Firstly, they perform a cryptographic operation to make packets unrecognizable. Secondly, they permute, batch or randomly delay messages to obfuscate their temporal relations. This second step is absent in the Tor protocol [21], which is why Tor can transmit messages with low latency. However, if an observer can monitor the start and endpoint of a communication path, the anonymity is compromised [36]. Mixnets provide a stronger anonymity guarantee so that even if each intermediary node is observed, traffic stays anonymous [36]. Previous research has explored the usage of Tor in anonymous FL [22], [23]. The stronger anonymity of mixnets with respect to an observable network makes them a subject of considerable interest, especially with no prior public research within this domain.

In all mixnets, intermediary nodes mix messages through delays or batching and forward them. There are a variety of cryptographic strategies that mixes can employ. The cryptographic operations differ in several ways. Firstly, decryption, re-encryption and hybrid methods can be used. Another important distinction lies in the usage of public

asymmetric, or secret symmetric keys. This is further linked to the specific cryptographic algorithms employed [12].

The original Chaumian mixnet [4] is a public-key decryption network. It works as follows. M denotes the message to be sent by sender S to A , the recipient's address. The message is encrypted using the recipient's public key, K_A . $K_A(M)$, A denotes the encrypted message with the receivers' address. The message, however, is routed through mixes. Every mix in the network has a public key K_i , and the sender further encrypts the message using each node's public key in order. Using Chaum's notation in a simplified manner, mixnets can be defined as follows [4].

$$K_n(K_{n-1}(\dots K_2(K_1(M), A)\dots)),$$

M is passed through n mixes, whereas each mix decrypts its corresponding layer of encryption and sends it to the next recipient in the chain.

The Chaum network is the original mixnet design, but it has several drawbacks. Layered public-key encryption is expensive and increases message sizes [12]. To mitigate this, other strategies have been developed. One such approach is to use hybrid-encryption mixnets. It encrypts the main body of the message using a symmetric secret key. This secret key is encrypted using the mix's public key. Together, they form the cryptographic message. The public key operation thus only has to be performed on the secret key and not the whole message, saving on computation and message length [12].

Another method is based on ElGamal encryption, which is an asymmetric algorithm but with the addition of the re-encryption property [37]. It allows mixes to re-encrypt messages using their secret exponent, thus making the package unrecognizable. The holder of the original secret key, i.e. the receiver, can still decrypt the message. As mixes do not decrypt the message, control information, such as subsequent addresses, cannot be included [12]. For that reason, re-encryption networks can only be used for fixed-path mix networks. This was used by *Helios* [38], which was the first publicly available e-voting system. *Helios* performs anonymization through re-encryption and shuffling, which is provided by a single server. Another system that used re-encryption is the *ProvoMN* framework, where ballot shuffling is performed by nodes in a distributed ledger (DL) network. The DL nodes double as mixes of a re-encryption network. The nodes shuffle the votes and pass them in a circular manner [31]. The authors found that it is possible to achieve a decentralized re-encryption network in combination with DL technology, in order to make the voting process transparent and provably correct. The *Selene* system uses a threshold-decryption scheme in combination with ElGamal re-encryption [30], where the secret key is further distributed between a set of nodes, called *tellers*. *Helios*, *ProvoMN* and *Selene* are instances of e-voting systems. The high anonymity requirements imposed by this domain can be extended to the FL use-case. In contrast, verification is less critical in FL. While these examples provide forms of unlinkability and anonymization, none support flexible topologies. In DFL, supporting dynamic and fault-tolerant topologies is a advantageous property.

Another, rarely used strategy is to employ encryption based on symmetric keys, and entirely avoiding public keys. In [39], one binary tree for each mix is created, where the corresponding mix forms the root node. The rest of the mixes form the nodes of

the binary tree. Each sender sends fragments of their secret key to the leaf nodes of each binary tree. Using this system, a sender can share its secret key with the mixes without revealing their identity. Symmetric key operations are more efficient than their asymmetric counterparts. However, low latency comes at the cost of fault tolerance [39], which is a considerable drawback in the area of DFL, where, in many cases, flexibility is essential.

Topology [40]

Table 3.1: Example Mixnets in Comparison

Work	Domain	Cryptographic Operation	Key Type	Topology	Additional Techniques
[4] 1981	General Purpose	Decryption	Asymmetric	Cascade	-
[39] 2024	General Purpose	Secret Key	Symmetric	Unknown	Binary-Tree Key Fragmentation
[38] 2008	E-Voting	ElGamal Re-Encryption	Asymmetric	Centralized	-
[31] 2022	E-Voting	ElGamal Re-Encryption	Asymmetric	Cyclical	Distributed-Ledger
[30] 2016	E-Voting	ElGamal Re-Encryption	Asymmetric	Unknown	Threshold encryption
[40] 2017	Messaging	Sphinx	Combination	Stratified	Cover Traffic, Poisson-Mixing
This work	DFL	Sphinx	Combination	Fully Connected	Cover Traffic, Fixed Average Sending Rate

The mixing techniques employed by mix-nodes can be distinguished into two categories: threshold mixing (or batch mixing) and continuous mixing [41]. The original mixnet design by Chaum [4] is an instance of the former, while systems such as *Loopix* employ random delays to achieve low-latency continuous mixing [40] for messaging applications. Continuous mixing schemes are considered less secure in terms of anonymity [41], [42]. While high delays impose disadvantages in most domains, latency is not as critical in FL as it is in user-facing applications such as messaging and browsing.

The potential advantages of mixnets applied to FL have been suggested several times [10], [43], [44], yet this remains an unexplored topic. While mixnets in DFL are promising due to their strong anonymity guarantee, their integration also present challenges. Many mixnet implementations rely on the full availability of all nodes to provide complete message decryption [4], [12], [39]. Other implementations require a cascade topology. Thus, these systems have little fault tolerance, which is a central design principle of

decentralized architectures like DFL systems. Additionally, mixnets are prone to high latencies, with a trade-off between low latency and strong anonymity guarantees [36], [42].

3.6 Summary

Ensuring privacy and unlinkability in federated learning requires a combination of networking, obfuscation, and cryptographic techniques. Cryptographic methods such as Shamir’s secret sharing, homomorphic encryption, and ring signatures reduce the risk of inference attacks. Network frameworks, such as onion routing and blockchain, introduce decentralized mechanisms that break up networking patterns. Additionally, model obfuscation techniques, such as gradient masking and adding noise, reduce the vulnerability against attacks targeting unintended memorization.

A key challenge is the trade-off between unlinkability, model accuracy, and computational efficiency. Increasing unlinkability often comes at the cost of increased communicational overhead and, as network latencies increase, slower convergence.

3.6.1 Discussion

Various techniques have been proposed to improve unlinkability and privacy in FL. The majority of frameworks relies on introducing trusted centralized entities to obfuscate, encrypt or shuffle the models before aggregation. While these entities increase privacy, they do not eliminate the underlying vulnerability of a centralized point of trust.

Decentralizing the federated learning network breaks up the direct link between the centralized entities and the worker nodes. Using existing frameworks such as mixnets and Tor naturally integrates well into the domain of DFL. Many have pointed to the potential of mixnets in addressing unlinkability concerns in FL. However detailed literature and implementations have not been published. This presents a research gap for DFL frameworks deploying an integrated peer-based mixnet for fragment-based model exchange, specifically for dynamic and unreliable network conditions.

Table 3.2: Comparison of Related Work I

Work	Unlinkability	Implementation	Experiments
[23] 2024	Onion Routing	Network Simulator, Python, TOR	Node Anonymity, Reliability
[17] 2024	Ring Signatures	Ethereum, PyTorch	Accuracy, Computational Overhead
[18] 2024	Secret Sharing, Pseudonyms, Homomorphic Encryption	Theoretical Experiments	Analysis, Computational and Communication Cost, Security, Privacy
[24] 2024	Local Obfuscation, Global Shuffler, Sign-Based Gradient, Byzantine-Robust Aggregation	Pytorch Experiments, Theoretical Analysis	Accuracy vs. Byzantine Poisoning Attacks, Privacy
[26] 2024	Pseudonyms, Signatures, Outlier Detection	Cryptographic Operations Only	Computational Cost, Communication Overhead, Security
[22] 2022	Onion Routing, Reputation-Based Routing	TOR-Based Lightweight Routing	Computational Cost, Communication Overhead, Security
[25] 2024	Locally Obfuscated Gradient, Centralized Gradient Shuffler	Isolated Experiments	Accuracy vs. Privacy Budget, Computational Overhead of Shuffling
[27] 2024	Homomorphic Encryption, Secret Aggregation with Collaborative Model Selection	PyTorch Experiments	Runtime, Communication Overhead, Intrusion Detection
[19] 2021	Ring Signature for Upload, Smart Contract for Rewards	Ethereum Hardfork, Ganache-Truffle, CryptoNote	Gas Costs, Runtime
[20] 2024	Homomorphic Encryption, Blockchain	Modified Blockchain	Private Accuracy Of Training Algorithm

Table 3.3: Comparison of Related Work II

Reference	Domain	Attack Type	Technique	Description
[45]	FL	Byzantine	FLTrust	Maintain server model to compare client updates with
[46]	FL	Byzantine	Krum aggregation	Aggregation algorithm for SGD that is resilient to Byzantine attacks
[47]	DFL	Privacy attacks	GEnc	Gradient encryption algorithm
[48]	DFL	Privacy attacks	Differential privacy	Obfuscate information by introducing noise
[48]	DFL	Poisoning Attacks	SGD	Aggregation across node, where no nodes knows the exact value of others
[49]	FL	Privacy attacks	BatchCrypt, HE	Batch-wise HE encryption to reduce computation and communication costs

Chapter 4

Architecture

This chapter outlines the architecture for the proposed DFL framework using peer-based mixing. The first section highlights the high-level design decisions, followed by a detailed mixing strategy and an overview of the structural components. The final section provides details about the learning and mixing workflows.

4.1 Design Decisions

Unlike related work, the proposed architecture eliminates the need for centralized trusted entities to obfuscate model updates or to enhance unlinkability. Instead, it integrates decentralized techniques from mixnets and DFL into peer-based mixing for DFL. By running DFL and mixnets on the same peers, deployment is simplified, and the mixnet can be fully tailored to the requirements of DFL. Moreover, it ensures that privacy-preserving techniques are not dependent on third-party entities. This implies peers are not only DFL participants but also mixes and senders, combining all roles of DFL and mixnets. Any external centralized FL or mixing entity could introduce vulnerabilities through its access points and would contradict the decentralized nature of DFL.

However, the duality tightly couples two separate concerns and limits scalability. The coordination of the mixnet’s routing and the federation’s model aggregation must be aligned. This results in additional challenges in managing peer failures of dynamic network changes. Despite its trade-off, the underlying work focuses on an integrated approach and its advantages in achieving fully decentralized FL.

4.2 Unlinkability Trough Mixnet

For many typical applications of DFL, such as IoT architectures, *sender online unobservability* represents an additional property worth considering alongside the unlinkability constraint. Guaranteeing this property implies a user’s online status is fully obscured,

meaning an observer cannot infer whether a user is communicating meaningful messages at all.

The proposed fundamental sending strategy follows [40]’s solution, where outgoing messages are placed in an outbox. The outgoing message stream is constant, with a randomly sampled interval between outgoing packages. Cover traffic is generated in the absence of DFL messages. Any model being broadcast over the mixnet is split into fragments of a common size for independent network messages. Mixing nodes shuffle, delay, and forward each fragment independently, whereas the original sender of a message is untraceable. The final receiver of a model fragment collects model fragments, unaware of the fragments’ origins. A fragment’s only meaningful content is the model parameters and their indices encoding their position in the global model. To our knowledge, this form of strong unlinkability in DFL is novel.

All collected model fragments are aggregated by a fragment-based FedAvg algorithm. This increases the learning algorithm’s robustness for partially delivered models in the case of network failures or crashed nodes. The challenge of anonymous model fragments lies in assessing if all model fragments of the DFL scenario were received. To address this, the proposed framework defers the responsibility of assuring the complete transfer of a local model from the receiver to the sender by leveraging anonymous acknowledgments. Any encrypted payload is extended by its own acknowledgment message. On receipt of a message, a receiver can access the acknowledgment included in the message. The receiver sends this acknowledgment through the network, where it is forwarded to the next receiver addressed in the message. The receiver of the acknowledgment, i.e. the original sender, is never revealed to the sender of the anonymous acknowledgment, that is the original receiver of the fragment. The anonymous acknowledgments are implemented using *single use reply blocks* (SURB). This system is illustrated in figure 4.1. If a sender does not receive the SURB in time, it resends the original payload and embeds a new unique SURB. This ensures unlinkable, reliable, and fault-tolerant communication for the proposed peer-based mixing for DFL.

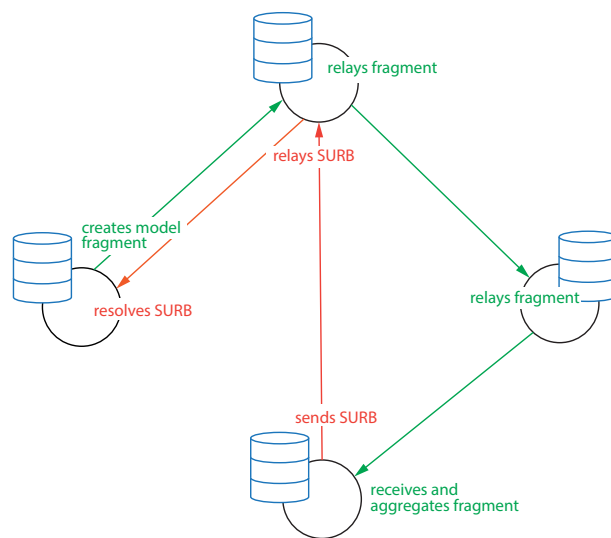


Figure 4.1: High-Level View of Round Trip

A single universal outbox is used for all concurrent polymorphous roles of each peer,

which ensures the secure mixing of any outgoing or relayed message. In general, the mixnet traffic is comprised of three distinct message types: model fragments, SURBS and covers. The outbox ensures persistent mixing techniques for all message types and centralizes the mixing logic. Any message placed in the outbox is delayed and shuffled independent of its type.

4.2.1 Mixing Logic

One challenge that arises from the dual responsibility of the DFL nodes is the highly variable volume of messages produced during model training. While mixnets are suited to steady data streams, the model training results in an alternating pattern of long idle periods followed by a burst of fragments. This makes it difficult to estimate the distribution of fragments in the network at any given point in time. To provide a security guarantee, the system implements a fixed outbox size that provides a lower bound of the system’s entropy, independent from a node’s training phase.

Another vital step besides the mixing of packages lies in the cryptographic step performed by the mixes. When a relay package arrives at a mix-node, the corresponding layer of onion encryption is peeled back, resulting in a cryptographic transformation that guarantees the unlinkability in terms of package appearance. Additionally, it is ensured that all packages are of the same size and that fragments, SURBs and covers are indistinguishable.

To guarantee a minimum entropy, the each node has two compartments where it stores outgoing messages, the queue and the outbox. The queue is a simple *first in first out* (FIFO) structure, where a node’s fragments, incoming relay traffic and SURBs equally are routed to. The outbox’s size is a configurable global parameter in the system, referred to as O from now on. As soon as the outbox is empty, it pops O -many packages from the queue. If the queue contains less than O packages, cover packages are generated. The outbox is then shuffled. The packages are sent out from the outbox at a steady rate that is sampled from a truncated normal distribution, with mean μ and standard deviation σ , which are configurable global parameters.

The outbox size O therefore provides a minimum amount of indistinguishable packages that a relayed message is guaranteed to be shuffled with. Consider a scenario with a global adversary, that can observe all traffic that passes through the network. It observes node n with the aim of linking relayed traffic entering and exiting n . The probability of correctly linking an incoming package m_i to an outgoing message m_o can be bound as such:

$$P(m_o = m_i) < \frac{K-1}{K} * \frac{1}{O}$$

Where the parameter K is the maximum hops that of a package in the mixnet, which is a globally configurable parameter. All path lengths from 1 to K are equally likely. The factor $\frac{K-1}{K}$ accounts for the probability of a package terminating at the observed node n an not being relayed at all.

4.3 Components

The proposed architecture, shown in Figure 4.2, consists of a *User Interface* and a *Manager*, which together allow users to configure and deploy DFL scenarios. The Manager deploys the nodes based on the user’s preferences and acts as a metrics hub receiving, storing, and streaming metrics data. Once deployed, the nodes independently handle learning and mixing in a decentralized manner, completely independent of the Manager.

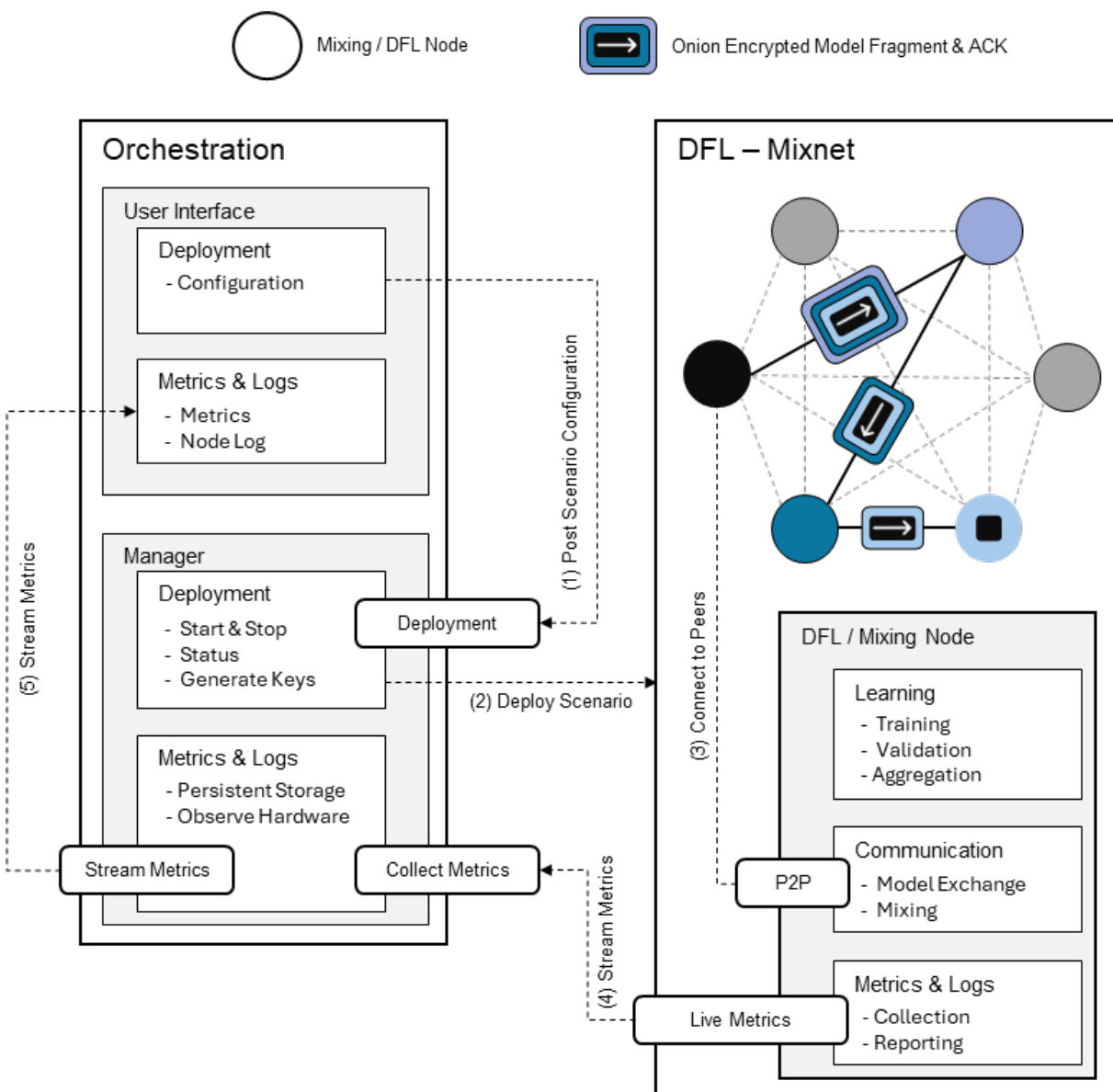


Figure 4.2: Architecture Overview

4.3.1 User Interface

The *UI* serves two primary purposes: scenario deployment and observation. It enables users to start and stop scenarios and observe their progress in real time. A dashboard provides an overview of the nodes' status, health and up-time. Moreover, various metrics can be selected for visualization.

4.3.2 Manager

The Manager handles the deployment of the DFL scenarios and their observation once deployed. It consists of multiple controllers providing services to the nodes and the UI.

The *Deployment Controller* configures and deploys peers, generating the necessary keys for the mixnet. Once the peers are fully deployed, the *Metrics Controller* provides the nodes with an endpoint to report various metrics and system health. Additionally, the Metrics Controller stores the metric data persistently, including computational resources usage of the peers. This allows the UI to receive metrics as a stream, visualizing them in real time.

4.3.3 DFL / Mixing Node

Each node consists of three major modules: learning, communication, and metrics. The communication module connects to peers and manages the connection's health. It includes logic for mixing, resending, and acknowledgment. The learning module handles the dataset, training, and aggregation. It is responsible for handling the fragmentation of a model and for reassembling the fragments during aggregation.

The communication module contains the logic for transport, which handles peer-to-peer connections, and the Sphinx sub-module. The sub-module implements the Sphinx library [50] and is responsible for the encryption, routing, and the anonymous acknowledgments. In conjunction with the acknowledgments, it is also responsible for ensuring a package's reliable arrival. Therefore, it implements a resend mechanism that handles unacknowledged packages. Additionally, the communication module also contains the mixing logic, handling the outbox.

Both the learning and communication module make use of the metrics module to report metrics, which are then periodically reported to the Manager's Metrics Controller.

4.4 Workflows

This section describes the decentralized workflows for DFL and the mixnet, which together enable peer-based mixing for DFL. The DFL workflow focuses on the training, aggregation, and validation, while the mixnet workflow enables the DFL workflow to anonymously send and receive models.

4.4.1 DFL Workflow

During the bootstrap phase, each node initializes a communication server and establishes connections with all other nodes in the scenario. At the same time, nodes load their unique local training dataset along with a shared validation dataset.

Once the bootstrap phase is complete, each node begins training locally on its dataset for a single round. The accuracy of the model is evaluated using the shared validation dataset. After training, the local model is split into fragments, with each fragment containing a subset of the model parameters. These fragments are labeled with their indices to specify their positions within the model, but do not contain any identification of the sender. The communication module then broadcasts these fragments anonymously to all other nodes using the mixnet, ensuring privacy and unlinkability.

Each node awaits all SURBs and the total expected number of fragments. The total number of fragments is the product of nodes and the calculated number of fragments per model. If the early stopping conditions are not met before the configured threshold, the node proceeds. All received model fragments are aggregated to form a new global model using the proposed fragment-based FedAvg algorithm. After aggregation, the updated global model is validated again using the shared validation dataset. The validation results are recorded, and the process is repeated for all following rounds.

4.4.2 MixNet Workflow

The mixnet ensures the model fragments are transmitted and acknowledged anonymously. The *Mixer* is the part of the system unique to mixnets and is responsible for the temporal obfuscation and unlinkability of incoming and outgoing traffic.

Within a node, one can distinguish between three types of traffic: fragments generated by the node itself, relay traffic, and traffic addressed to this node. The traffic that is addressed to this node has to be acknowledged. The fragments, the relay traffic, and the acknowledgments are all treated uniformly by the Mixer.

Onion Routing

Each model fragment is wrapped in multiple layers of encryption using onion encryption. The path is randomly generated for each fragment and receiver. Similarly, the acknowledgment of the message is encrypted using a unique and independent path before being embedded into the message.

Each node receiving a message decrypts the message's outermost encryption layer and accesses the routing information. If the message was not addressed to the processing node itself, the message is padded to have a uniform size and forwarded to the next node in the path.

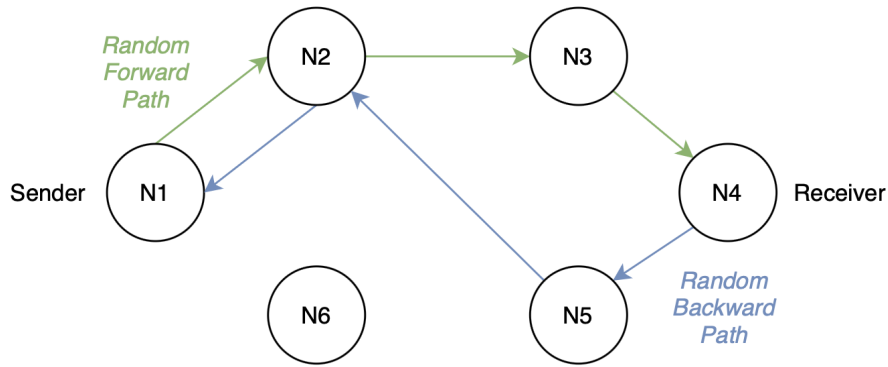


Figure 4.3: Forward- and backward-path of a fragment through the mixnet.

Anonymous Acknowledgments

Once the message was transmitted to its final recipient, the model fragment and the acknowledgment message can be decrypted and accessed. The acknowledgment message does not reveal its original sender and is mixed in the same fashion as the model fragment on the forward path. The fragment part of the message is stored and eventually aggregated by the training module. If the original sender does not receive the acknowledgment within the configured time, the sender resends the original payload. The resent packet is freshly encrypted using a new randomized path. Figure 4.3 shows the corresponding example fragment’s path through the network, including the acknowledgment’s backward path.

The corresponding onion encryption schema of the same example fragment can be conceptualized as follows:

$$K_2(K_3(K_4(\mathbf{frag}, K_5(K_2(K_1(\mathbf{ack})))))),$$

where K_n refers to a given node’s public key. Layers up to the fragment element correspond to the forward path, and subsequent layers correspond to the backward path.

Mixer

All outgoing traffic in the node passes through the Mixer, including relay traffic, model fragments, and acknowledgments. All messages are first placed in the queue of the Mixer. Over time, the packages progress to the Mixer’s outbox of configurable size O . Each time the outbox is filled it shuffled. A single package is sent from the outbox at a randomly sampled interval. If the outbox is empty, the Mixer either pops new messages from the queue or generates cover traffic. Cover traffic is a form of dummy traffic, for which the payload is randomly generated. It is encrypted identically to the fragments, including a random path, though they do not include SURBs and therefore are not acknowledged. The encryption sub-module ensures a uniform package appearance and pads messages to a consistent length.

Chapter 5

Implementation

The following chapter outlines the implementation of the DFL framework leveraging a peer-based mixnet for anonymous model exchange. The first section provides an overview, followed by a walkthrough of the deployment, learning, and mixing processes.

5.1 Components

The *User Interface* based on React.js and the FastAPI-based *Manager* allow users to configure and deploy DFL scenarios. The Manager deploys the containerized nodes using the Docker SDK. Using a metrics-controller, the Manager acts as a metrics hub receiving, storing, and streaming metrics data. Once deployed, the nodes independently handle learning, mixing, and reporting metrics in a decentralized manner. Figure 5.1 shows an overview of the high-level components.

5.1.1 Manager

The Manager is implemented using FastAPI, deploying two controllers: the Metrics Controller and the Node Controller. The Metrics Controller provides a REST endpoint for the nodes to report locally connected data and an SSE endpoint for any client to register for live metrics streams.

The Metrics Controller makes use of Service classes to observe the Docker containers hosting the scenario. Using the Docker Python SDK, the uptime, CPU-time, memory usage, and container health are observed. The metric data is added to the metric cache and streamed to the metrics file as well as to the connected clients.

5.1.2 User Interface

The User Interface is implemented as a webpage using React. The dashboard has three primary components: the nodes' health, timeseries visualization, and scenario control.

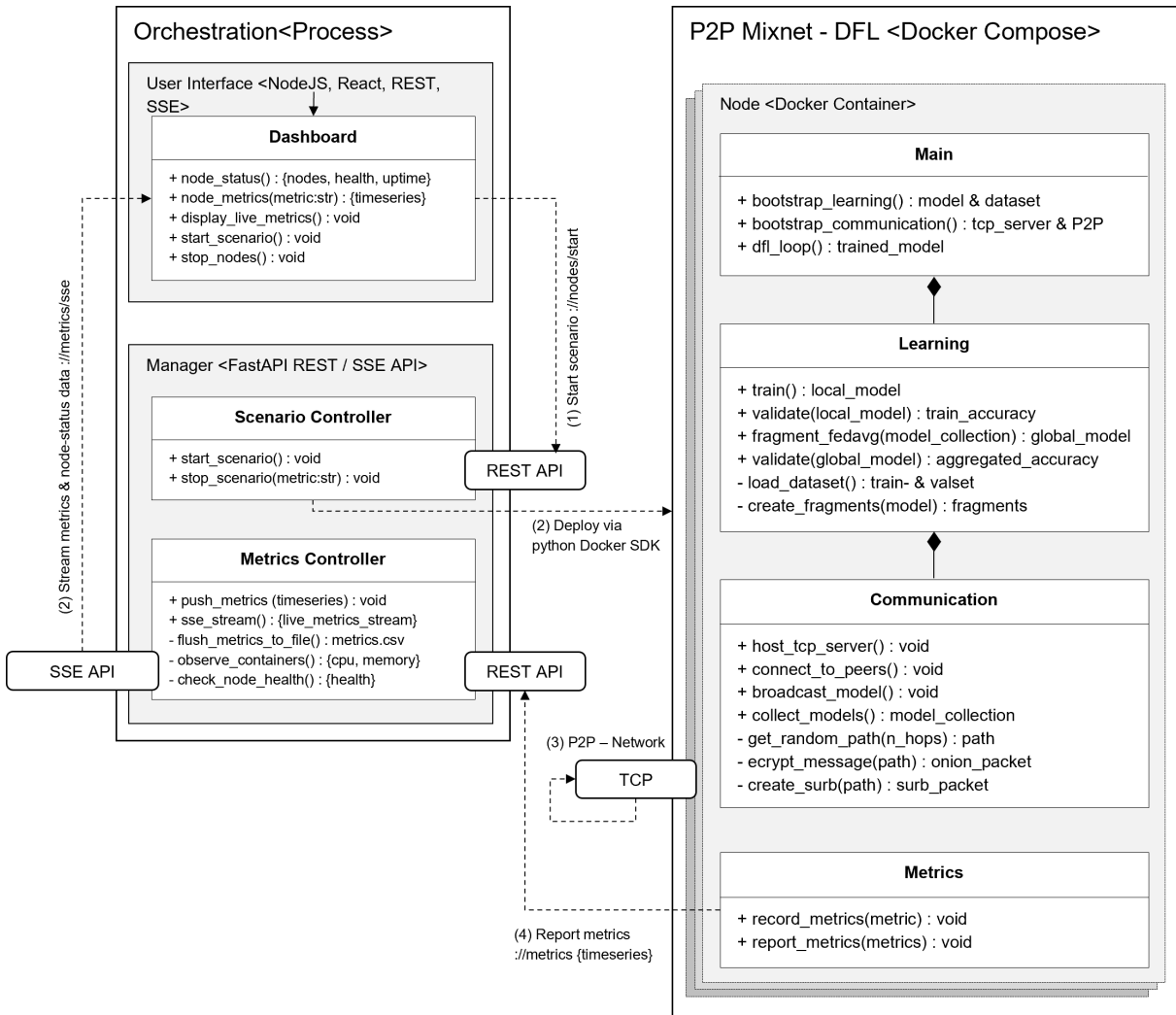


Figure 5.1: Implementation Overview.

Peer Based Mixing Start Stop

Node Status				
Node	Status	Uptime	Round	ETA
node_0	running	4m 49s	3/10	11m
node_1	running	4m 49s	3/10	11m
node_2	running	4m 49s	3/10	11m
node_3	running	4m 48s	3/10	11m
node_4	running	4m 48s	3/10	11m
node_5	running	4m 48s	3/10	11m

Figure 5.2: Node Overview in Frontend

The nodes' health is visualized using a table listing the nodes, their uptime, and current round such as in Figure 5.2. The information displayed is requested periodically from the *Scenario Controller*. The timeseries component consists of a filter panel and plots visualizing the selected timeseries such as shown in Figure 5.3. Upon selection, the graphs are added to the visualization. Each browser client connects to the Manager's Metrics Controller, registering for the *Server Side Events* (SSE) stream. The Metrics Controller periodically streams all metrics received in the last period. Newly connected clients initially receive all previously cached metrics data before starting the live stream.

5.1.3 Node

Each node is deployed by the Manager as an isolated Docker container. Each container gets its configuration assigned by its environmental variables during deployment. The configuration consists of various parameters adjusting the learning, mixing, and communication processes. During the nodes' bootstrapping process, each submodule is instantiated asynchronously using the Python *asyncio* library. This ensures the non-blocking execution of tasks without stopping background tasks such as mixing or acknowledging.

5.1.4 Learning

The Learner module is responsible for two main tasks: Firstly, it performs local training and, secondly, it manages the aggregation of model fragments and thus the creation of the global model. It is implemented with *Pytorch*, making use of its robust data management and learning features. The model handler module is responsible for the training, the fragmentation of models, the reassembling and aggregation of incoming parts, and the evaluation using the validation set that is shared across all nodes. The dataset itself is loaded from Pytorch online resources and split evenly by using the nodes' indices to ensure non-overlapping datasets. The system uses the MNIST [51] dataset implemented in



Figure 5.3: Exemplary Selection of Metrics out of 33 Options in Frontend

torchvision [52] to simulate a plausible DFL scenario. Moreover, the model is instantiated as a Pytorch convolutional neural network. As part of the Learner module, the Message-Handler is responsible for sending out model fragments and caching incoming chunks for aggregation.

5.1.5 Communication

TCP Connection

Peers make use of the Docker networks' DNS to connect to all peers using their hostnames, loaded from the deployment configuration passed by the Manager. Each peer connects to its peers via a directed TCP connection, having both roles as a TCP server for incoming connections and as a client for outgoing connections. Connections are persistent and maintained during the scenario. In addition to message handling, the connections are used to determine the health of a peer. Inactive peers are disconnected and immediately excluded from the DFL and mixing process. If a node reconnects or newly joins the federation, the node is added to the list of active nodes and integrated into the node's learning and mixing workflows. This allows any node to leave or join the federation without disrupting the mixing or learning processes.

Sphinx

The *Sphinx Module* takes its name from the *Sphinx* cryptographic library [50] used for onion encryption and onion routing. The Sphinx Module is responsible for all cryptographic processes such as encryption, decryption, and onioning. Moreover, it serves as a generator for randomized paths, SURBs, and indistinguishable cover-traffic. To ensure cryptographically secure randomness, the system additionally uses Python's *secrets* library [53].

Mixer

The mixer module manages the outbox, which must be passed by any outgoing message. The mixer's logic can be separated from the Sphinx module, as the mixer is exclusively responsible for the temporal unlinkability of incoming and outgoing messages, excluding cryptographic processes. The mixer module handles shuffling and ensures a uniform output to achieve the mathematical properties required for temporal unlinkability.

5.2 Workflows

In this section, the major workflows that are executed during a scenario are described in detail. This includes the deployment, the learning, and the mixing.

5.2.1 Manager

Once a user has initiated the deployment of a scenario in the frontend, a REST request is sent to the Scenario Controller of the Manager. The Manager proceeds to generate the unique cryptographic keys for each node and configures the individual deployment parameters for each node. Using the Docker SDK for Python, each node is deployed as a Docker container having the keys and the parameters embedded. The Manager's Scenario Controller stays idle and provides REST endpoints to query the nodes' health status and to abort a scenario. Meanwhile, the Metrics Controller opens a new metrics file and buffer to store all incoming metrics data persistently while streaming it to any requesting client using SSE in real-time.

5.2.2 Learning

Upon initialization, the Learner component instantiates the Model Handler and the Message Manager. They create the Pytorch model and download the nodes' unique training dataset. After the setup, the Model Handler component starts training on the predefined number of batches. Once the training is finished, the node enters the communication phase, tasking the Message Manager to split up the model and generate valid fragments to be sent to the peers. The fragments are serialized using Pytorch, which flattens the model, and split into JSON payloads. Concurrently, the Message Manager buffers incoming model fragments to be aggregated later on. In the background, nodes are sending out their fragments, mixing intermediate traffic, and collecting incoming fragments.

After the communication phase has terminated, the Model Handler aggregates the fragments into a global model, applying the fragment-based FedAvg. Each aggregated fragments payload is assessed by its hash to filter out duplicates. This is required since fragments cannot be identified otherwise as duplicates, as they do not have any embedded identification. Each fragment only consists of parameters and their indices in the global model. Each parameter of the final global model is an average of the received values for this specific parameter. This method is independent of the sender's identity as well as the number of fragments per parameter subset. Therefore, the number of fragments containing a given subset of parameters can vary in scenarios. Situations like these may arise due to joining or exiting nodes and can be handled without threatening the robust aggregation of the global model. Finally, the global model is evaluated using the global validation set.

5.2.3 Communication

Figure 5.4 shows a simplified sequence diagram visualizing the DFL process, including the mixing sequences. In reality, the system's operations are highly asynchronous and do not follow the strict sequence as illustrated. Several processes are required to run concurrently, such as mixing, sending, and caching.

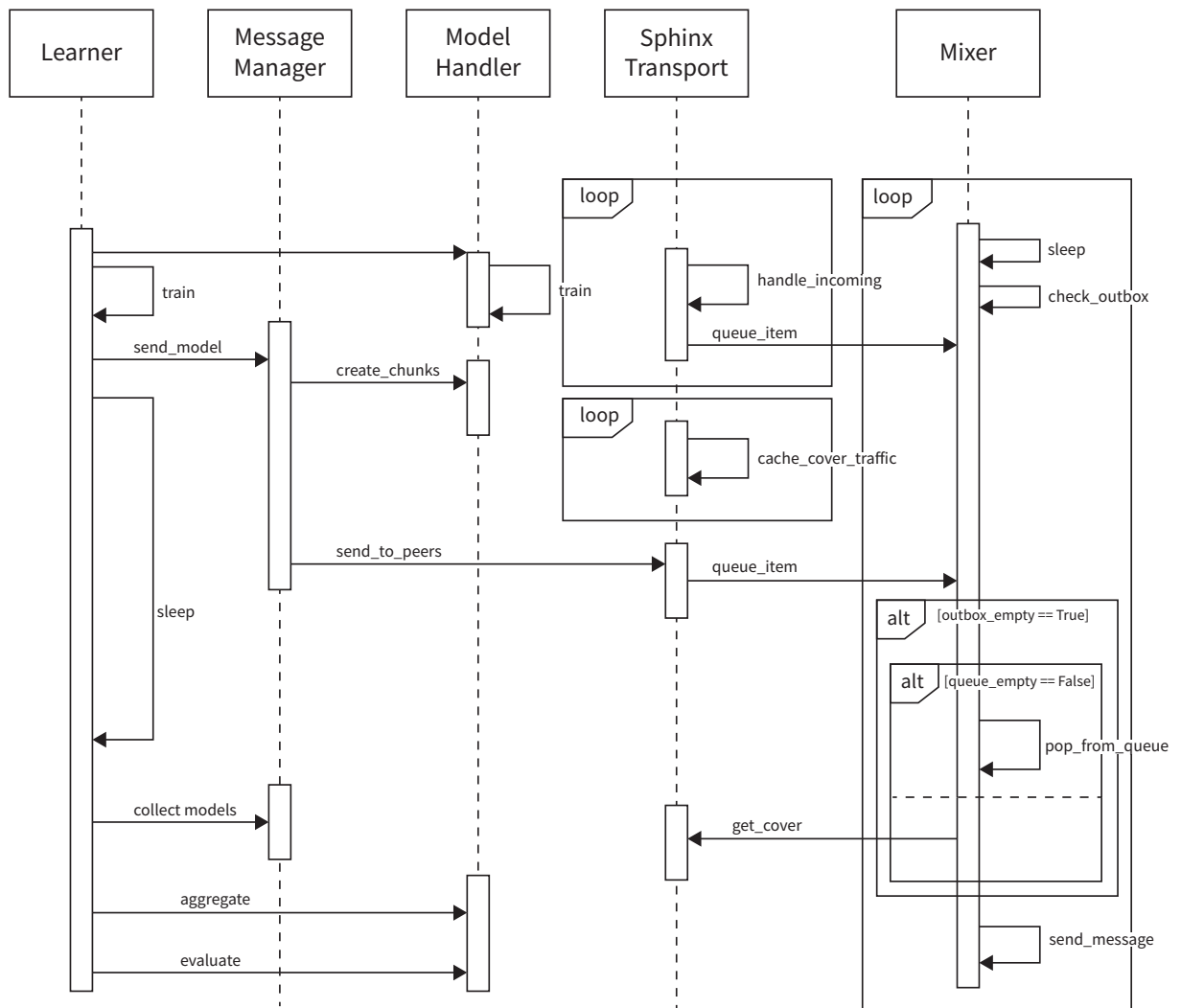


Figure 5.4: Sequence Diagram of Learning and Communication Process

Managing the Outbox

Firstly, the mixer runs a constant asynchronous task that handles the periodic checking of the outbox. Within this task, the mixer samples a random interval from a truncated normal distribution. The task then sleeps for the duration of the sampled interval. Subsequently, the outbox is checked. If the outbox is not empty, the top package on the stack is sent. If the outbox is empty, the mixer pops packets from the queue. If there are not enough queued messages to fill the outbox, the mixer refers to the *Sphinx Transport* module, which generates cover traffic. After the outbox is refilled, it is shuffled. Furthermore, the mixer exposes a public function that allows other components to add messages to the queue.

Handling Incoming Traffic

The Sphinx Transport component runs an asynchronous task that is responsible for handling incoming traffic. In order to handle incoming traffic, the Sphinx Transport first strips the outermost layer of encryption from the package. This allows it to access the traffic flag. There are three types of incoming traffic that the Sphinx Transport can distinguish. First, there is relay traffic, which is to be mixed by this node. Second, destination traffic, which is a fragment addressed to this node. Third, an acknowledgment addressed to this node, originating from a piece of traffic that this node originally sent out. Additionally, the Sphinx Transport component is responsible for encrypting and preparing fragments that are generated by this node.

Original Fragments

When the message handler calls the Sphinx Transport to encrypt a payload it must (1) randomly generate a forward path, (2) generate an anonymous acknowledgment in the form of a SURB, (3) generate a random backward path, (4) wrap the acknowledgment using onion encryption corresponding to the backward path, (5) add the acknowledgment to the original message and wrap the package into the onion encryption layers corresponding to the forward path. Subsequently, the Sphinx Transport adds the new outgoing package to the mixer's queue. Furthermore, the Sphinx Transport saves the SURB data generated by this outgoing fragment in a cache. This allows the system to keep track of fragments that have not yet been acknowledged.

Relay Traffic

The Sphinx Transport component handles relay traffic by removing the outermost layer of encryption, which is generated by the sender using a shared secret derived from the mix's public key. This reveals the routing information necessary to pass the package on to the next node in its path. Additionally, it transforms the appearance of the package, so that an observer cannot recognize the outgoing message. The relaying node does not have

access to any further information within the new package. This package is thus addressed to the next node and added to the mixer's queue.

Destination Traffic

If a package has the destination flag, it was addressed to this node. This means that this node can remove the last encryption layer and obtain the fragment. The fragment is added to a cache. Additionally, this package contains a SURB. The SURB already contains all routing information and corresponding encryption, as this was generated by the original sender. The SURB is thus added to the mixer's queue.

SURBs

If a package is a SURB, this means that a fragment sent from this node that has been acknowledged. The Sphinx Transport can decrypt the SURB and obtains a hash that can be looked up in the node's cache. The corresponding fragment can be marked as acknowledged.

Producing Cover Traffic

The Sphinx Transport is responsible for generating cover traffic. Cover traffic is not acknowledged by the system, therefore no SURB is included with them. In order to ensure a smooth output rate, the cover traffic must be generated efficiently. In order to achieve this, a stack of cover traffic is continuously generated in the background and cached. When the mixer requests a cover package, it can be provided from the cover cache, ensuring that operations such as encryption do not delay the regular output.

Receiving Cover Traffic

Receiving cover traffic is analogous to destination traffic, i.e. receiving a fragment. However, cover traffic is marked by a cover flag. Cover traffic does not require acknowledgment and therefore does not include a SURB. A piece of cover traffic is therefore simply dropped by the recipient node.

Resend Mechanism

To ensure each node eventually receives all broadcast fragments, a cache of payloads and their final destination is maintained. This cache records which of the sent fragments have been acknowledged. Unacknowledged messages that are older than a preconfigured time are resent, whereby they are encrypted using the Sphinx module and added to the outbox, using a new unique path.

Chapter 6

Evaluation

In this chapter, the implemented DFL framework for peer-based mixing is evaluated. The chapter is split into three sections. The first section evaluates the federated learning workflows, showing the resilience of the framework against dynamic changes in the peers. The second section evaluates the communicational properties of the peer-based mixing and model exchange. The third section analyzes the computational resources used depending on the properties of the configured DFL scenario.

Experiment Scenario

Unless specified otherwise, the experiments use a baseline scenario. This scenario consists of a fully connected mesh of 6 nodes ($N = 6$), an outbox size of 10 ($O = 10$) and two maximum hops ($K = 2$), runs over 10 rounds ($R = 10$) and uses an average sending interval of 0.005 seconds ($\mu = 0.005$) with a standard deviation of 0.001 ($\sigma = 0.001$). For a detailed description of the parameters' roles, refer to Chapter 4.

6.1 Dynamic Topology

The implemented framework is designed to be resilient against dynamic changes in the topology. Both the mixnet and the DFL workflows demonstrate robustness when nodes exit or join the network. The following experiments illustrate the network and learning properties during such an event, emphasizing the framework's ability to maintain stability and functionality.

6.1.1 Implications on Workflows

Joining and exiting nodes directly impact the DFL and mixing workflows. When a node exits unexpectedly, it may still be part of encrypted onion messages being routed at the time of its exit. The exited node will not decrypt or route any Sphinx packets, which causes

the loss of all SURBs and fragments being routed over the exited node. Furthermore, a change in the number of active nodes disrupts the early stopping condition of the model collection stage described in Chapter 4 in subsection 4.4.1. The framework mitigates this through the immediate exclusion of the inactive node from each node’s local view of the topology. This ensures that missing fragments or SURBs are not awaited further from the exited node, and that any unacknowledged fragments routed over the exited node are resent.

Similarly, a spontaneously joining node must establish connections with all existing nodes of a topology to participate in the learning and mixing workflows. The framework ensures any spontaneously joining or rejoining node is immediately utilized for encrypting Sphinx packets and exchanging fragments. Furthermore, the early stopping criteria are updated concurrently, ensuring instant adaptation to the new topology during any stage of the DFL workflow.

Overall, the framework has various mechanisms implemented to adapt quickly and recover from any change in topology. The following measurements illustrate the framework’s recovery from nodes joining and exiting during a DFL scenario.

6.1.2 Observed Metrics Topology Change

To illustrate the framework’s resilience against dynamic changes in the topology, the default scenario is extended to include both a joining and an exiting node. *Node 0* is configured to join the topology after any node reports to the Manager that it has reached Round 3. Subsequently, *Node 0* is set up to exit after partially broadcasting its fragments in Round 5, thereby disrupting the flow of fragments between nodes. The other nodes therefore have to react immediately to this change in topology.

Detection of Change in Topology

The framework running on each node continuously observes the health of all TCP connections to its peers. If a node fails to respond or terminates its connection, it is promptly marked as inactive. Similarly, any newly requested TCP connection triggers an immediate reconfiguration of the node to adapt to the new topology.

Figure 6.1 illustrates the self-reported number of active connections held by any node. Around minute one, the nodes bootstrap the topology and report all being connected to the available four peers, which currently excludes *Node 0*.

Just after minute four, all nodes report being connected to five peers, illustrating the swift adaptation to the newly joining *Node 0*. Between minute nine and ten, *Node 1* exits and reports being connected to none of its peers. The other peers detect the exit of *Node 1* and report being connected to four peers until the end of the scenario.

The adaptation to a joining node is quicker than to an exiting node. This is intentional as the framework optimistically assumes that a temporarily unresponsive node may still

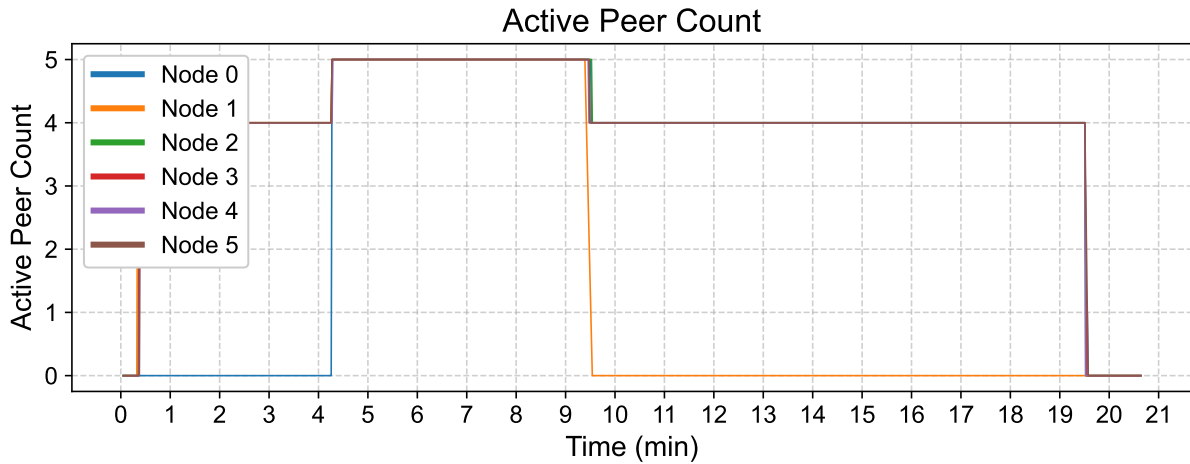


Figure 6.1: Self-Reported Number of Peer Connections per Node

recover within a timeout. This approach avoids unnecessary reconfiguration in cases of short-lived unresponsiveness. After the timeout elapses, the exiting node is excluded, and the topology is reconfigured.

Natural Synchronization

Figure 6.2 shows the self-reported stage each node is currently in. The stages are divided into five distinct stages, representing each node’s primary focus in the learning workflow. Although each node reports its current stage, the framework is implemented using asynchronous programming, ensuring that no task blocks background processes. This design allows nodes to concurrently handle critical operations such as mixing, resending unacknowledged fragments, or adapting to dynamic changes in topology.

The major stages of the learning workflow are defined as follows:

0. Stage: Bootstrap & Finish
1. Stage: Local model training on a distinct training set
2. Stage: Local model evaluation on shared test set
3. Stage: Broadcasting, forwarding, and collecting of fragments
4. Stage: Global model evaluation on shared test set

All nodes naturally synchronize due to the early stopping criteria during Stage 3. Aside from this, there is no other external or communicational synchronization mechanism. The high degree of synchronization is clearly visible, as the stages across appear to change almost simultaneously. However, three incidents show deviation in the synchronization.

During the first incident in minute four, the spontaneous join of *Node 0* does not disrupt the other nodes significantly more than during earlier stages. Moreover, the newly joined *Node 0* achieves similar levels of synchronization starting from the next round.

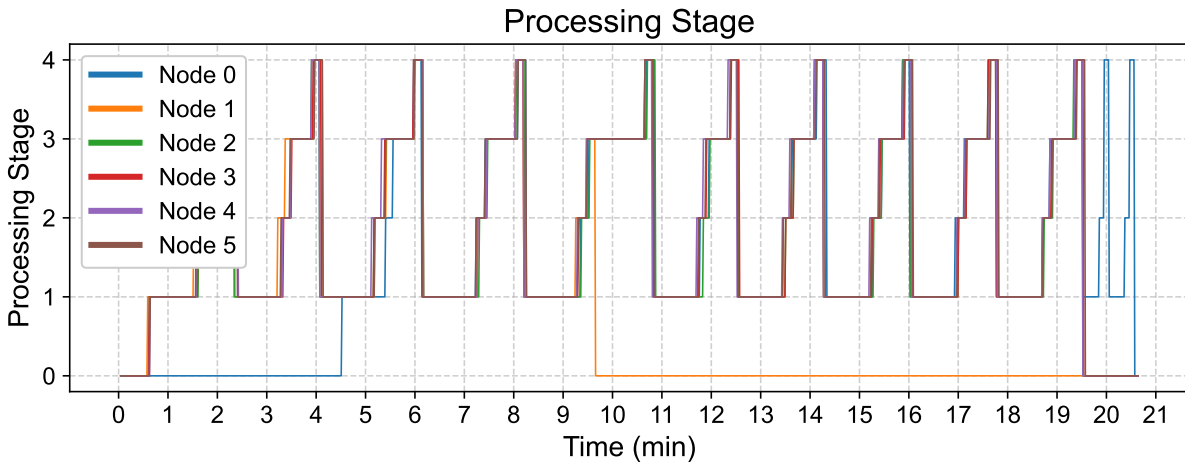


Figure 6.2: Stage Information per Node in Topology

In Round 5, after minute nine, during the second incident, *Node 1* exits during Stage 3. While this event does not visibly disrupt the level of synchronization of the other nodes, one can observe that this does increase the duration of Stage 3. The delay can be explained by the early stopping mechanism awaiting the SURBs of the retransmitted fragments, indicated in Figure 6.4. Although Stage 3 took longer, the remaining nodes of the topology do not desynchronize and proceed without being disrupted.

The third incident shows Round 11 and 12 being processed only by *Node 0* after it joined late to Round 3. *Node 0* continues without interruption and without waiting for its remaining peers to send their own model fragments. Due to the model collection being skipped without any peers to expect model updates from, *Node 0* iterates alone through the stages and finishes the assigned number of rounds successfully.

Overall, this shows that the system implements a strong natural synchronization mechanism, without the use any explicit coordination messages. Furthermore, it shows the synchronization's robustness against unexpected events such as delayed joins into the federations or early shut-downs of nodes.

Retransmissions

Figure 6.3 shows the number of cached fragments still awaiting acknowledgment. Two major events illustrate a change in the topology and the reaction of the participating nodes.

The first event occurs during the third spike, indicating the third round, where *Node 0* joins the topology. Upon joining, *Node 0* immediately begins sending its model fragments, which are acknowledged by its peer nodes without any noticeable delay. This illustrates the quick integration of a newly joined node into the mixing and anonymous acknowledgment workflow.

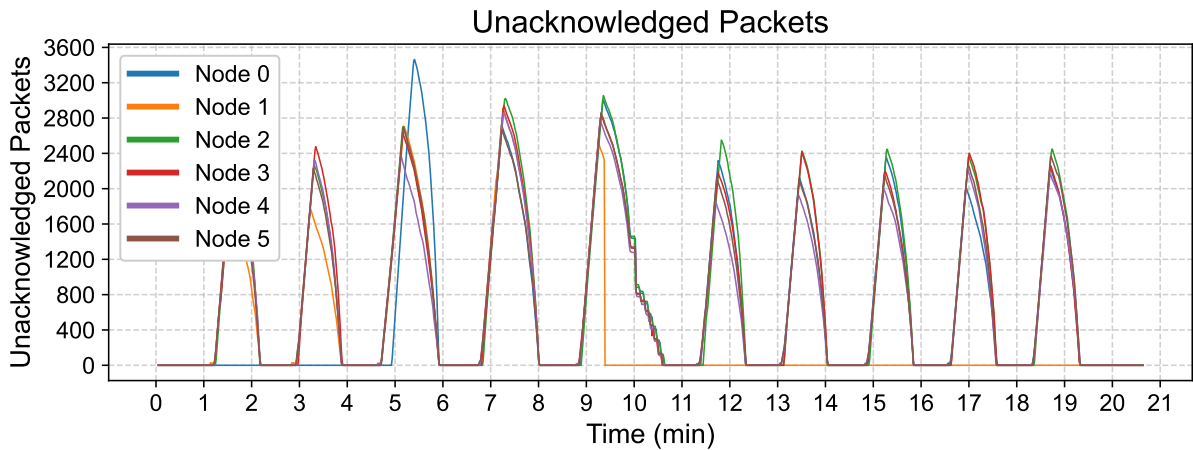


Figure 6.3: Unacknowledged Messages per Node in Topology

The second event occurs during the fifth spike around minute nine, indicating Round 5. *Node 1* exits, which results in dropping all its unacknowledged fragments. Shortly after, all remaining nodes detect the change in the topology and drop all unacknowledged messages for *Node 1*. This is clearly visible as a vertical decline being reported by all remaining nodes.

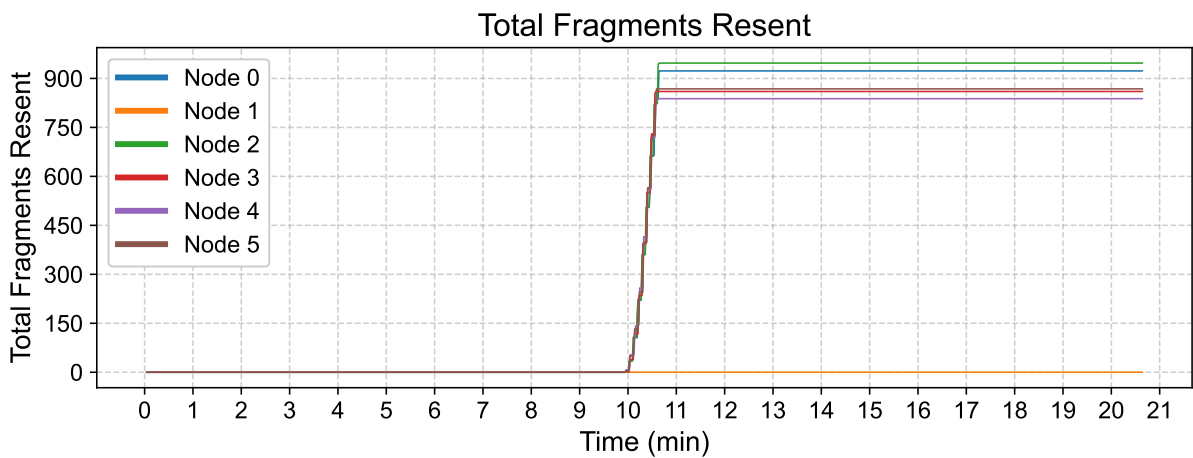


Figure 6.4: Number of Messages Retransmitted per Node in Topology

The steep decline is followed by a slowed-down decline in unacknowledged packets, which results from fragments being retransmitted that were intended to be routed over the exited *Node 1*. The sharp increase in retransmitted fragments after minute four, shared by all remaining nodes, is clearly visible in Figure 6.4.

Overall, this shows the reliable retransmission mechanism of the system. First, it demonstrates the seamless integration of a node joining the federation. Second, it shows the smooth handling of a node's premature shutdown. Peers react by retransmitting packages and eventually detect the shutdown. The network is reconfigured to exclude the

exited node.

6.2 Communication and Mixing

This section evaluates the networking and mixing properties of the framework, focusing on network overhead, latency, and the effects of system parameters. Additionally, the cryptographic properties of the mixnet are evaluated by analyzing the uncertainty of a link introduced by the mixing from a theoretical point of view. All experiments are conducted over a single training round ($R = 1$), as the observed behavior remains consistent across multiple rounds.

6.2.1 Network Metrics

Figure 6.5 illustrates the relationship between the average sending interval μ and the average round-trip time (RTT) and the average data rate.

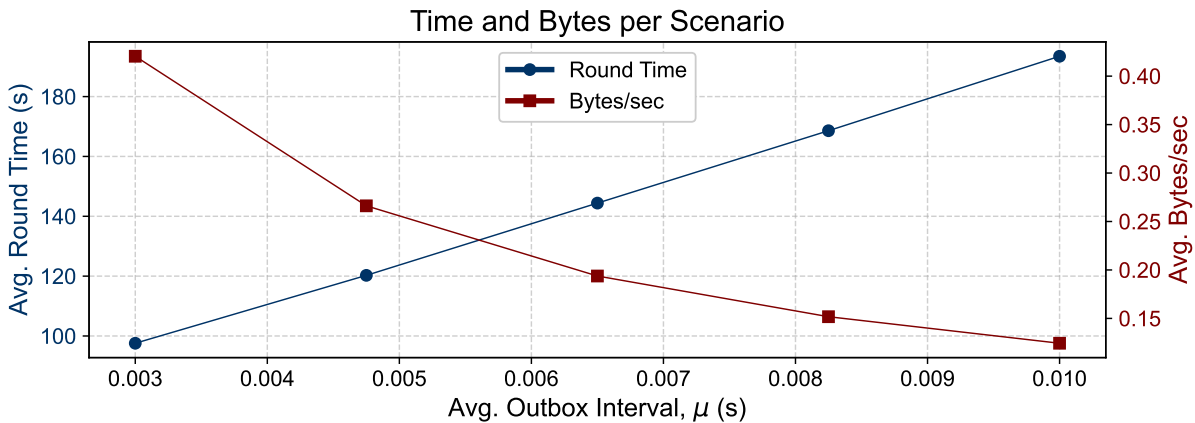


Figure 6.5: Effect of μ on Average RTT Compared to Its Effect on Traffic Rate

The average round time increases linearly with μ . This is because of larger sending intervals resulting in slower transmission rates. This leads to longer delays in completing a round due to the accumulation of the delays during the intermediate mixing hops. In turn, the average transmission rate in bytes per second decreases as μ increases. A low transmission rate reduced the load on the network due a lower traffic concentration and reduced number of cover packets.

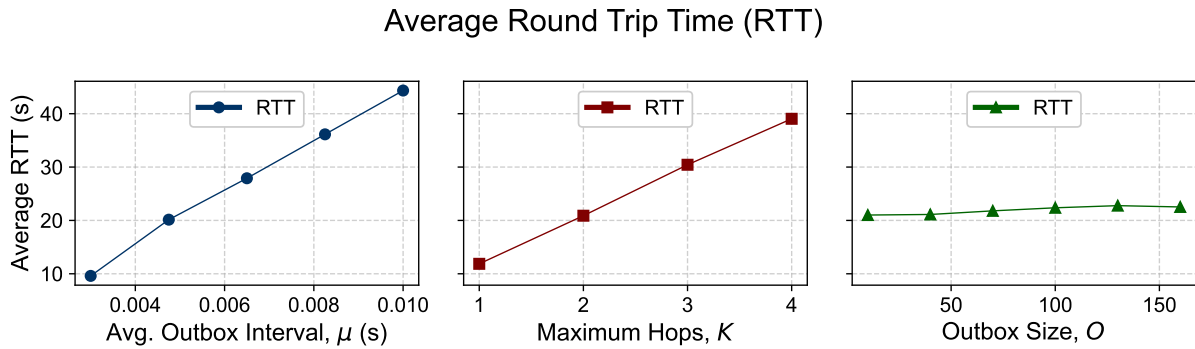


Figure 6.6: Effects of Mixnet Parameters on Average RTT

Overall, both metrics illustrate the major trade-off the mixing is faced with:

- **Low μ :**
 - **Pros:** Faster round times due to higher transmission rates.
 - **Cons:** Increased network overhead due to higher traffic, including dummy packets.
- **High μ :**
 - **Pros:** Lower network overhead with reduced traffic.
 - **Cons:** Slower round times, which may impact the system’s responsiveness.

Due to the stochastically constant output of each node, a high data rate also implies sending out dummy packets at a high rate. Therefore, the following experiments aim to analyze the trade-off between putting pressure on the network and keeping the framework fast and responsive.

Each Sphinx packet is set to have a size of 1253 bytes, with a payload of 1024 bytes. Those values were determined during development and held constant throughout all experiments. The configurable parameters of the mixnet are the average sending interval μ , maximum intermediate hops K , and outbox size O .

The average sending interval μ directly determines the stochastically uniform data rate. The number of maximum intermediate hops K determines the number of times a message is mixed, which mainly influences the RTT. Overall, all three configurable parameters influence the responsiveness and entropy of the mixnet. The outbox size O determines the number of messages, which are shuffled before being sent out in the new order. It mainly determines the computational costs of shuffling and the uncertainty (entropy) of the messages being sent in the same order as received. Furthermore, a large outbox size increases the variance of the RTT for each individual packet.

Figure 6.6 illustrates the average RTT as a function of the configurable parameters of the mixnet. The results show that the average RTT increases linearly with both μ and K . The RTT only slightly correlates with O , which can be explained by the individual packet having more variance in its position within the outbox, while the individual changes (being

randomly placed in the front or back of the outbox) cancel out on average. While the large effects cancel out, the responsiveness the mixer slightly decreases with O , due to dummy traffic in the outbox accumulated during the training phase, which first has to leave the outbox before fragments can leave the node. However, as has been stated, this effect is minor.

While the previous experiment illustrated the effect of the mixing parameters on the average RTT, Figure 6.7 illustrates their respective effects on the average output rate in bytes/sec. Notably, the output rate is solely determined by the parameter μ , as μ directly parametrized the average sending interval, and, in turn, configures the output rate. Therefore, the results also show that the rate has no correlation with K and O .

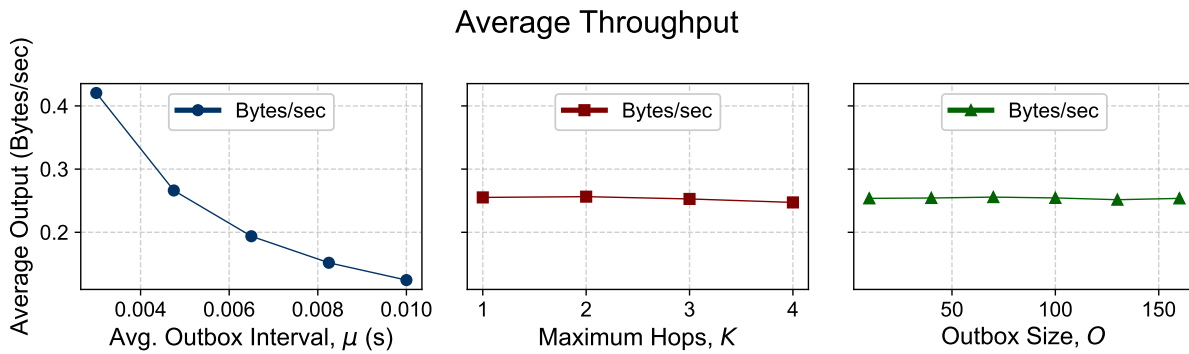


Figure 6.7: Effects of Mixnet Parameters on Average Output

Figure 6.8 illustrates the total sent payload in bytes as a function of the configurable parameters of the mixnet. The total bytes sent increase linearly with K , as longer paths involve more relays and consequently more traffic. Since the sending rate is fixed by μ , this additional network load results in longer RTT, as observed in the Figure 6.6, as well as longer training rounds.

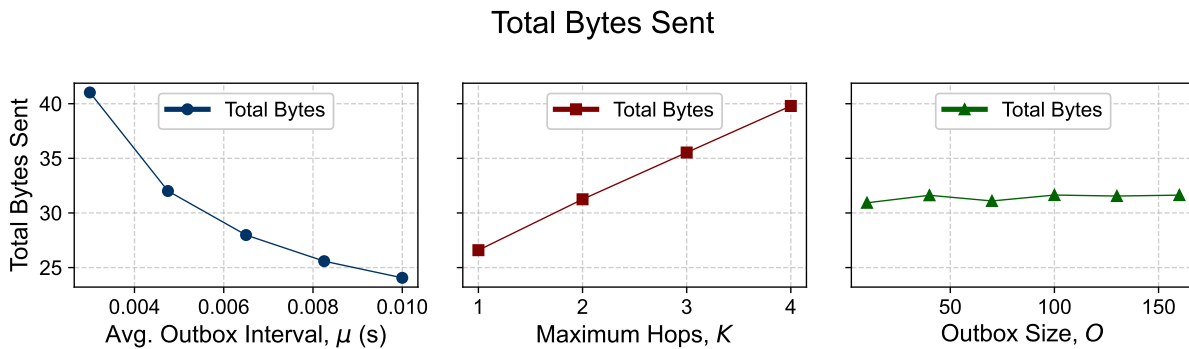


Figure 6.8: Effects of Mixnet Parameters on Total Bytes Sent

Both metrics, average bytes per second and total bytes sent, decrease as μ increases. The reduction in average bytes per second is a result of longer intervals between messages, while short intervals generate high amounts of dummy traffic.

Overall, one can again observe the two opposing effects that the parameter μ has on the system performance. The optimal parameter configuration is likely use-case dependent. K has linear influence on RTT, and therefore also on total training round duration. The parameter O has comparatively negligible effects on performance.

6.2.2 Entropy

To evaluate the system's properties in terms of unlinkability, a theoretical analysis of the information entropy was conducted. The entropy of a message's path reflects the stochastic uncertainty of linking packages. The uncertainty metric is evaluated in terms of Shannon entropy [54], which is defined as follows:

$$H(X) = - \sum_{x \in X} p(x) \log p(x)$$

Information entropy can be conceptualized as the average number of bits required to encode the state of a system, which in turn corresponds to the uncertainty of said system.

In this analysis, we differentiate between the local entropy, or relay entropy, and path entropy. Local entropy measures the uncertainty of linking an incoming package to an outgoing one while observing a single node, whereas path entropy is the accumulated entropy from sender to receiver. The entropy of a message is increased by the shuffling in the outbox, the routing over intermediary nodes, and the indistinguishability from dummy messages.

The basis of this analysis is given by a threat model that assumes a global passive adversary. This adversary can observe all network connections and has access to all incoming and outgoing packets of a node. Under these assumptions, the unlinkability is solely granted by the mixer architecture.

It is important to note that the outbox size O provides a baseline guarantee of mixing within the mixnet. The outbox ensures that incoming packets are shuffled with $O - 1$ other packets before being forwarded. This mixing process provides a minimum guarantee for entropy. In practice, incoming messages are not placed directly into the outbox but are queued for processing until they are forwarded to the outbox. As such, the calculated entropy values are lower bounds of the system's actual entropy.

Relay Entropy

Relay entropy, or local entropy, is assessed by computing the uncertainty of an incoming message being the same as an outgoing message focused on a single node n in the default experiment setup.

Figure 6.9 illustrates the relay entropy in bits dependent on the number of nodes, outbox size O , and the number of intermediate hops K . All other parameters of the default

scenario are held constant if not changed by the figure’s x-axis. As described in Chapter 4, the probability of successfully linking an incoming packet m_i to an outgoing packet m_o is bounded by: $P(m_o = m_i) < \frac{K-1}{K} \cdot \frac{1}{O}$.

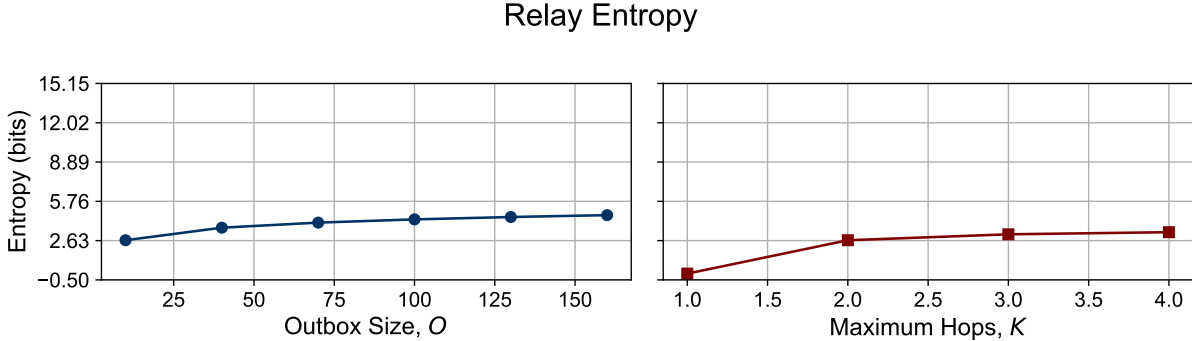


Figure 6.9: Relay Entropy as a Function of Outbox Size and Maximum Hops

An incoming packet m_i either terminates at node n or is shuffled and relayed. The probability of termination is $\frac{1}{K}$. If the packet is relayed, it is mixed with $O - 1$ other packets, resulting in a probability of $\frac{K-1}{K} \cdot \frac{1}{O}$ for each possible outgoing position.

Figure 6.9 shows that the entropy grows with O , but a relatively small increase with larger O , such as $O > 50$. Similarly, the curve flattens out after $K > 2$. This suggests that beyond a certain threshold, simply increasing the outbox size or the maximum hops does not significantly enhance the entropy on a local level. Thus, an optimal configuration for K is likely in the smaller range, possibly not exceeding $K = 3$, considering its effect on RTT shown in Figure 6.6. As O has a much smaller effect on performance, its tuning is not as critical, though its effect on entropy flattens out in a similar manner.

Path Entropy

Figure 6.10 illustrates the path entropy in bits dependent on the number of nodes N , outbox size O , and the number of intermediate hops K . All other parameters of the default scenario are held constant if not changed by the figure’s x-axis.

To analyze the path-entropy, the threat model is extended. In addition to the global passive network observation considered previously, the extended model includes a malicious node attempting to identify a fragment’s origin using the available global network information.

First, the fragment may have traversed K path lengths. Second, at each relay node, there are O incoming packages that could correspond to the fragment. Notably, the number of N does not affect the total entropy when the adversary has full access to global network information, as fragments can be traced between two nodes.

The total number of possible paths is given by:

$$\sum_{k=1}^K O^k$$

As all path lengths are equally likely and the shuffling is uniform, all of the given paths have the same probability. Therefore, the path entropy for this analysis is defined as follows:

$$H_{path} = \log\left(\sum_{k=1}^K O^k\right)$$

Similar to Figure 6.9, Figure 6.10 shows an increase of the total path entropy with the outbox size O and the number of maximum hops K . The results suggest that both a high O and K have a positive effect on path entropy; the correlation is logarithmic. Therefore, due to the diminishing returns, a more conservative choice for the parameter K is advisable, as it has been shown to have a high impact on runtime and latency. The parameter O has a much smaller impact on performance. These results align with the findings from the relay entropy analysis. Therefore, a large O and small K constitute an optimal configuration in terms of both entropy and performance.

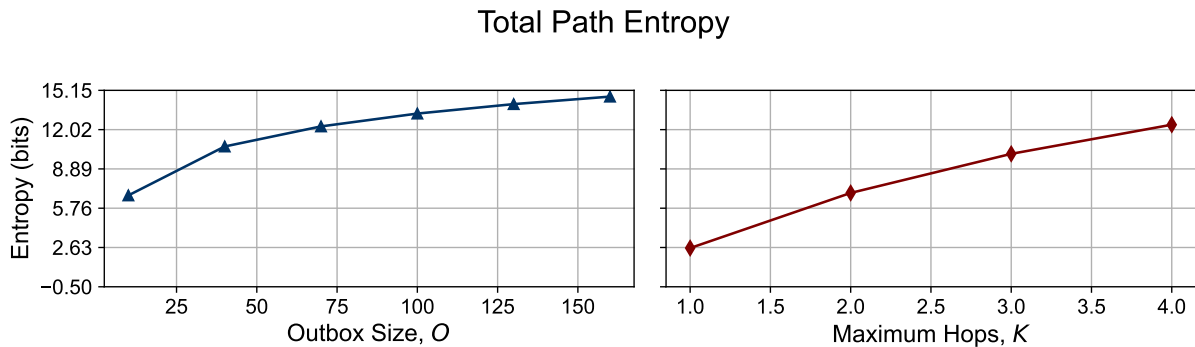


Figure 6.10: Path Entropy as a Function of Outbox Size and Maximum Hops

Discussion

Overall, the local entropy should be fine-tuned for performance by the size of the outbox and maximum hops, whereas both parameters quickly reach a point of moderate impact. This observation extends to the global path entropy as well. This implies that even a small network can gain substantially in privacy by moderately increasing the number of hops or the size of the outbox. However, the number of hops increases the time the packets spend in the network and the number of messages in the network at any given time. In contrast, increasing the outbox size has a less dramatic impact on performance.

Therefore, it is advisable to increase the entropy of the system by employing a large outbox size $O > 50$ and keeping the maximum hops small, possibly $K = 2$. This finding implies that this mixnet ought to rely more on the shuffling of the outbox in the mixes rather

than long paths, when performance is considered. It is important to note, however, that in the broader context of privacy and security, path length remains vital, as it mitigates the threat of potentially compromised nodes.

6.3 Computational Resources

Evaluating the computational resources and performance visualizes the trade-off between using resources for training and mixing. The following experiments analyze the time, CPU, and memory utilization depending on the scenario configurations. Resource usage of the individual containers is tracked as they progress through the stages of the DFL workflow. Due to the framework being implemented fully asynchronously, the mixing processes continue in the background at any stage. Memory and CPU-time are not measured per individual process but per stage of the workflow. The CPU time and memory are computed by averaging each node's total grouped by stage.

The computational metrics were periodically retrieved by the Manager using the Python Docker SDK. The measurements were analyzed in the context of the node's current stage. All experiments were conducted on an *Ubuntu 22.04* server with an *AMD EPYC 7502P 32-Core processor*, 125 GB memory, and 197 GB of storage. To directly observe the relative usage of resources for learning and communication, no dedicated GPU was used.

6.3.1 CPU-Time

Figure 6.11 illustrates the total and relative CPU-time at each individual stage for full scenarios depending on the number of nodes N or number of intermediate hops K . Figure 6.12 illustrates CPU-time depending on the number of intermediate hops K . All other parameters of the default scenario are held constant if not changed by the figure's x-axis.

Network Size Impact on CPU Consumption

As the number of nodes increases, each node receives a smaller fraction of training data, while the evaluation set stays constant. In theory, this would result in a decrease in the CPU-time for the training stage of each individual node.

Regarding the total CPU-time, Figure 6.11 shows the training time increases with the number of nodes. This results from the increased computational load used concurrently by the mixing process. This results in the training stage taking visibly longer with an increase in nodes, even if the theoretical computational load for training should be reduced due to the smaller training set.

However, the average total CPU-time used by a node is increasing with an increase in nodes. This shows that the background process of mixing during the training phase substantially affects the performance of the training, resulting in a prolonged training stage.

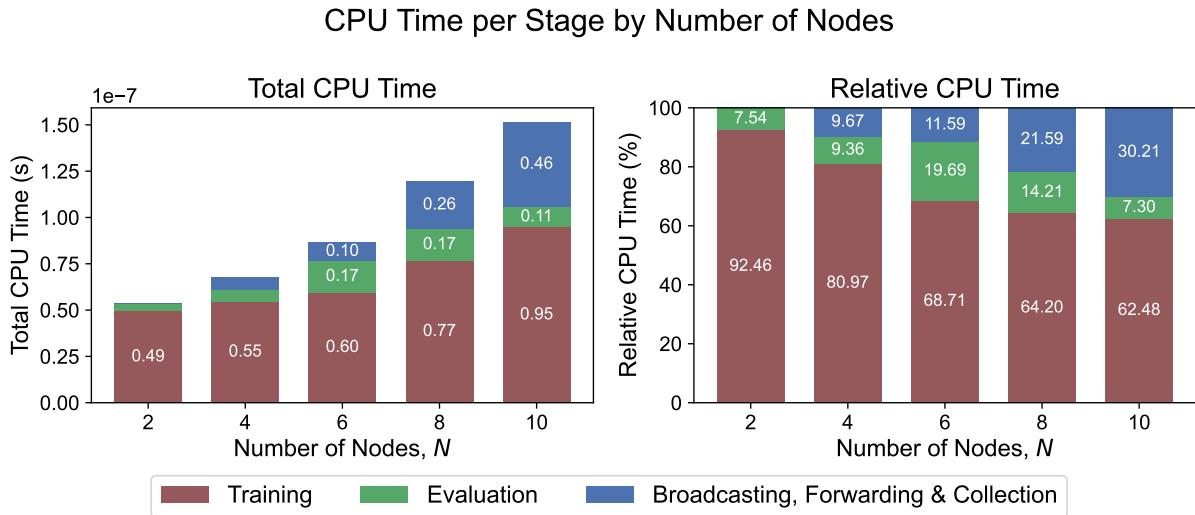


Figure 6.11: Impact of Network Size on CPU Consumption

At the small network size of eight nodes, CPU-time used by model exchange, either in the background or explicitly during the corresponding stage, is approximately equal to the total CPU-time used for training.

Similarly, the relative CPU-time in Figure 6.11 shows 100% of the available resources are used for learning at $N = 2$, while the model exchange accounts for less than 0.1% and therefore not shown in the plot. At $N = 10$, during model exchange alone consumes $\sim 30.21\%$ of CPU-time, illustrating its increased computational costs, even without accounting for the communication-related background processes during other stages.

Overall, the quadratic communication complexity in fully-connected DFL networks is further amplified. The mixnet not only increases the costs during model exchange, but also during any other stage of the DFL workflow, as it is a constant background process. Meaningful packets and dummy traffic are treated equally by the mixnet, which results in similar network loads during training and model exchange.

Intermediate Hops Impact on CPU Consumption

The CPU-time for the model exchange is expected to increase with the maximum hops K , as each node must mix more packets. An increased number of hops results in a more of packets in the network overall and therefore amplifies the computational load required at all times. In contrast, the number of model fragments being sent out and awaited stays constant.

Results in Figure 6.12 indicate an increase in the average total CPU-time per node with an increase in hops. This reflects how all stages are affected by the increased network load due to the constant background processing of traffic.

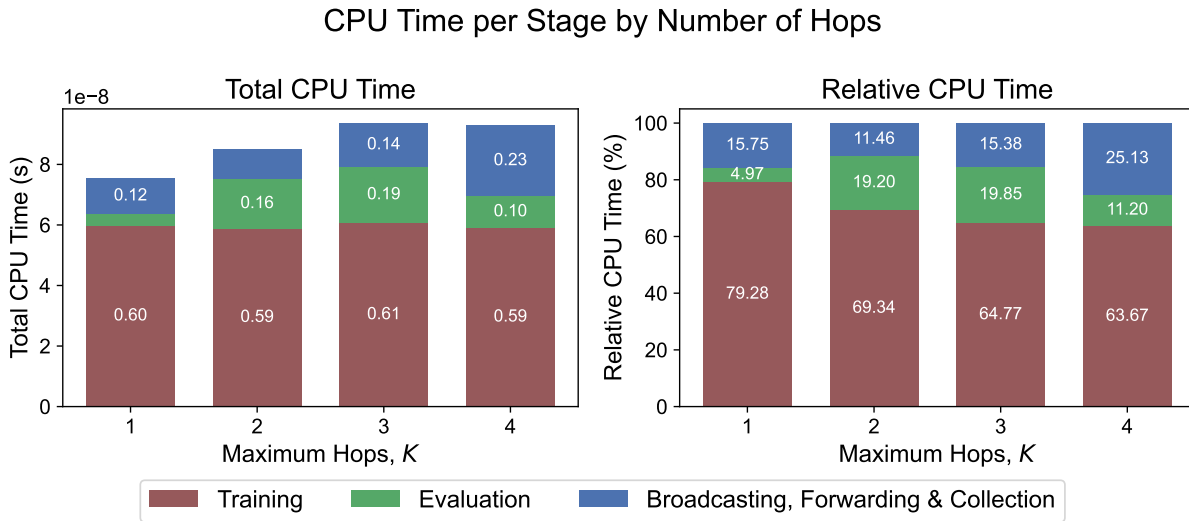


Figure 6.12: Impact of Intermediate Hops on CPU Consumption

Similarly, Figure 6.12 shows a trend for the relative CPU-time used with an increase in the number of intermediate hops. Relative computational costs for training decrease, while the other stages require increased resources.

Discussion

Overall, the CPU-time used during each stage shows the strain being put on all stages. With an increase in the number of hops, every fragment or dummy message is being processed by more nodes. With an increased load due to mixing in the background, training and evaluation are becoming increasingly inefficient and are competing for computational resources with the mixnet.

6.3.2 Memory

Figure 6.13 illustrates the average assigned memory at each stage over all nodes, analyzed over the duration of a training scenario, depending on the number of nodes N and number of maximum hops K .

In contrast to the CPU-time, memory shows no clear trend or differences during the different stages. Similarly, memory does not visibly increase or decrease with an increase in the number of nodes N or the number of intermediate hops K . No clear trends for memory could be observed in the range of the experimental setup.

Overall, this indicates that CPU-time is the limiting factor while memory stays more or less constant. However, the memory requirements of approximately 600 MB illustrates that the framework could be deployed on edge devices such as the *Raspberry Pi 4 Model B* [55], whereas using true micro-computing devices such as the *ESP32* [56] would most certainly significantly struggle with CPU and memory.

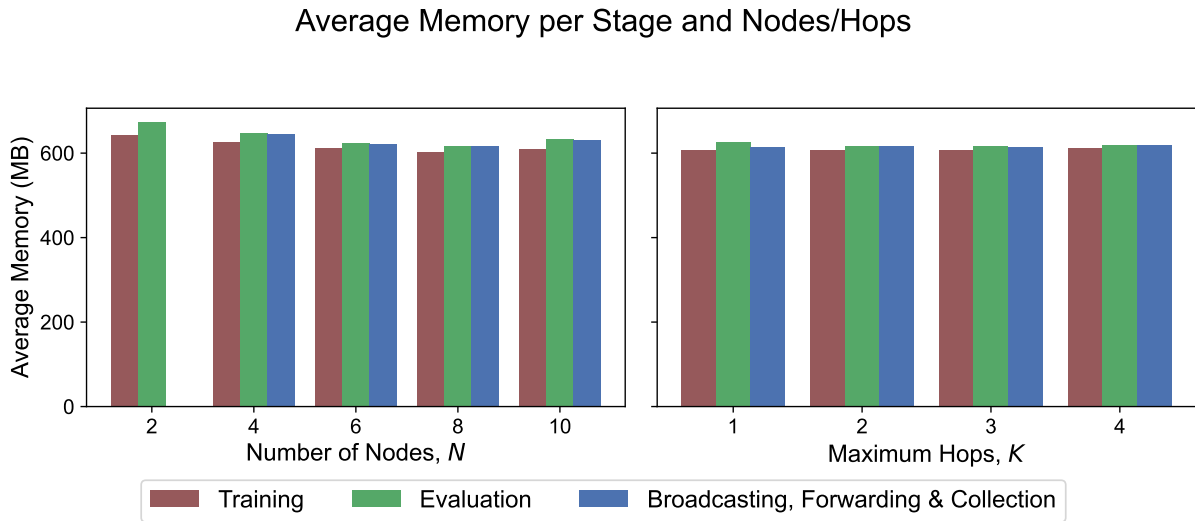


Figure 6.13: Impact of Network Size and Intermediate Hops on Memory

6.3.3 Time

Figure 6.14 illustrates the average time for a full round iteration, depending on the number of nodes N and number of maximum hops K . All other parameters of the default scenario are held constant if not changed by the figure's x-axis.

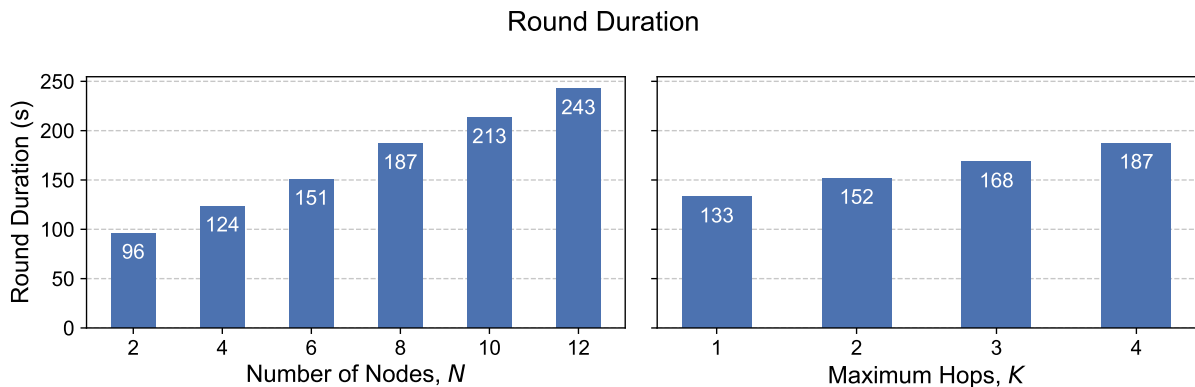


Figure 6.14: Impact of Network Size and Maximum Hops on Round Duration

Figure 6.14 shows that the round time increases approximately linearly with both the number of nodes N and maximum hops K . The increase in K directly affects the runtime, as it increases the average path length that a fragment has to traverse. Interestingly, N exerts a similar influence on runtime, even though it is not related to path length. However, a larger N substantially increases total network traffic, as each node has to send its fragments to every other node in the federation. Given the fixed sending rate of the nodes, the total volume of traffic scales linearly with N , leading to a corresponding linear increase in runtime. Furthermore, Figure 6.14 suggests that the increase in N has a more significant effect on runtime than K .

Overall, both parameters have a considerable impact on the round duration. Even a scenario with $N = 12$, representing a relatively small federation, already faces high round durations. This finding raises questions about real-world scalability.

6.4 Learning and Accuracy

Figure 6.15 shows both the average trained accuracy and the average aggregated accuracy over time. The accuracy metric demonstrates that the system reliably functions as a DFL network, alongside its enhanced privacy features. Trained accuracy represents the performance of the model obtained only through local training, with respect to the shared evaluation data. Similarly, aggregated accuracy results from evaluating the model after fragment aggregation, on the same dataset.

Experimental Setup

In addition to the default experimental setup, the framework’s default learning configuration was used:

- **Model: FedCNN newly created for framework**
 - **Layer 1:** Conv2D (1 input channel, 32 output channels, kernel size 3, padding 1, no bias) + BatchNorm2D + ReLU.
 - **Layer 2:** Conv2D (32 input channels, 64 output channels, kernel size 3, stride 2, padding 1, no bias) + BatchNorm2D + ReLU.
 - **Layer 3:** Conv2D (64 input channels, 128 output channels, kernel size 3, stride 2, padding 1, no bias) + BatchNorm2D + ReLU.
 - **Pooling:** Adaptive Average Pooling (output size 1).
 - **Classifier:** Fully connected layer (128 input features, 10 output classes) with log-softmax activation.
- **Dataset:** MNIST, partitioned non-IID using a Dirichlet distribution ($\alpha = 10$).
- **Training:** Adam optimizer ($lr = 1 \times 10^{-3}$, weight decay = 1×10^{-4}), cross-entropy loss, batch size 64, with data augmentation (random rotation, random cropping, normalization).
- **Aggregation:** fragment-based averaging, with model parameters transmitted in 512-byte fragments.

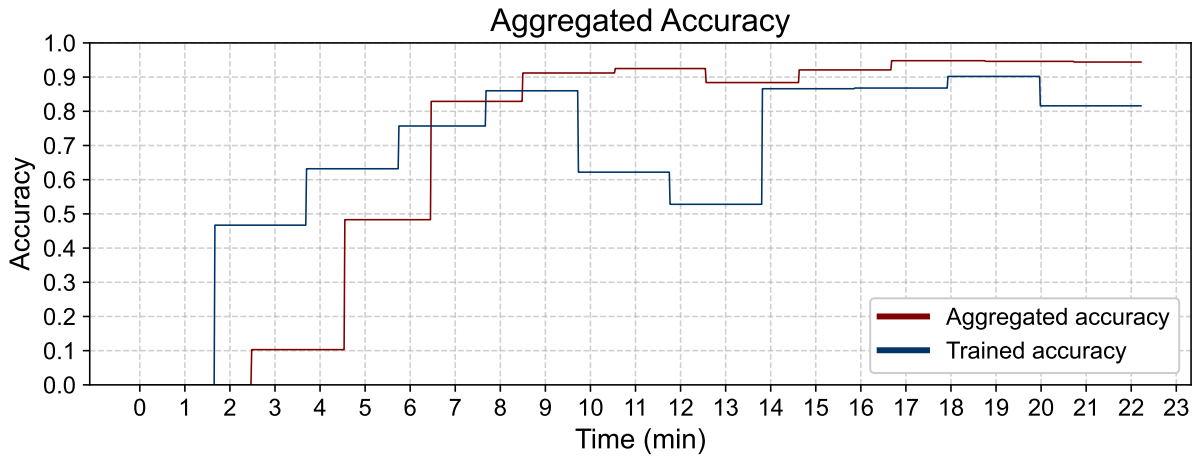


Figure 6.15: Average Local and Global Accuracy Across Nodes Over Time

Trained and Aggregated Accuracy

Although trained accuracy values shown in Figure 6.15 are averaged across all nodes in the federation, the result exhibits substantial fluctuations over time. In contrast, the aggregated accuracy is more stable, and while not strictly monotonic, it shows a clear and rapid upward trend. Importantly, the combination of parameter exchange and the fragment-based FedAvg algorithm yields better results than those obtained by individual nodes in isolation.

Figure 6.16 shows the average local and aggregated accuracies as a function of the number of nodes (N). The observed decrease in local accuracy with increasing N reflects the local disadvantage of the total training data being shared with an increasingly larger number of participants. As each node’s portion of the training data diminishes, its local accuracy correspondingly declines.

In contrast, Figure 6.16 shows the aggregated accuracy’s robustness against the reduction in local accuracy. Although there is a slight decrease in aggregated accuracy as N increases, it remains significantly more stable. This suggests that the fragment-based FedAvg can effectively synthesize new knowledge through the combination of local models, which is a characteristic feature of DFL. However, the modest decrease in aggregated accuracy also shows that this synthesized shared model cannot match the performance achieved by traditionally training a model on a single node having access to the full training set.

Discussion

Both Figure 6.16 and Figure 6.15 illustrate the framework’s successful core functionality for DFL. In addition, it shows the newly proposed anonymous fragment-based aggregation algorithm being a valuable extension to basic FedAvg. The framework’s robustness against networking fluctuations, which is achieved by dropping duplicated messages and

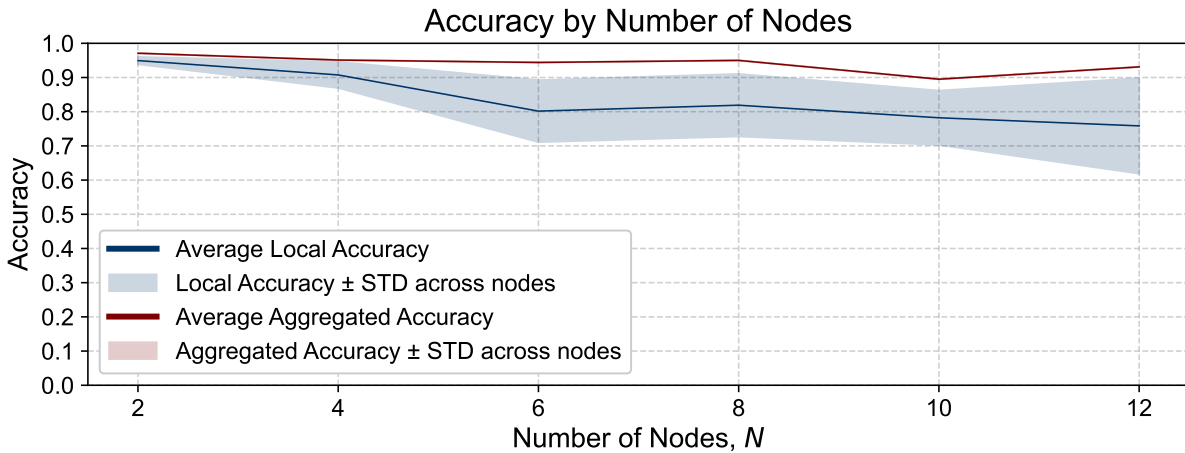


Figure 6.16: Average Local and Global Accuracy Across Nodes Number of Nodes (N)

guaranteed delivery through acknowledgments, does not come at a cost of reduced aggregated model accuracy. In theory, both fragment-based FedAvg and basic FedAvg yield the same results in a benign scenario.

Overall, the newly proposed fragment-based aggregation is robust to spontaneous changes in the topology during model collection. On the other hand, the algorithm opens up a variety of possibilities to attack the aggregation due to the completely anonymous fragments. Neither the sender nor its full model can be analyzed or tracked.

6.5 Mitigation of DFL Threats

In this section, a qualitative evaluation of the system with regards to its mitigation of known FL threats is conducted. The proposed system assumes a threat model of a global passive adversary and of honest-but-curious worker nodes, as well as a combination of the two.

Beyond unlinkability, a major concern of FL lies in attacks aimed at local training data [57]. These include malicious nodes providing deficient data, as well as third-party attacks that infiltrate and poison a worker node. Degraded local data can result in a fully corrupted global model. Poisoning attacks can be aimed both at local data or at the model parameters. The proposed system does not address poisoning attacks or related threats and assumes at least honest-but-curious participation from workers. Furthermore, it lacks mechanisms for detecting or mitigating actively adversarial behavior. Identifying malicious activity at the fragment level is especially challenging due to the anonymity of the fragments; therefore, this system is exposed to an increased threat of poisoning attacks.

Furthermore, privacy inference attacks constitute a major threat to privacy in FL. In these attacks, sensitive information is extracted from model parameters [57]. These are

the attacks that the proposed system aims to address primarily. The source of the threat is twofold, as either an eavesdropper or a worker can attempt to infer information. Eavesdroppers have no way of identifying the source of a fragment and cannot break the encryption of the package. Similarly, workers cannot link the received fragment to its source. Therefore, this system effectively mitigates inference attacks on worker nodes. However, [57] identify privacy inference attacks conducted during the predicting phase, with attackers using a public API of the finished global model. Membership attacks, where attackers infer a sample’s presence in the combined global dataset, are still equally viable, as are model attacks, where parameters of the global model are reconstructed. Importantly, these are global attacks and cannot leak sensitive information about individual workers.

In addition to global privacy inference attacks, evasion attacks constitute a further type of attack that is conducted during the prediction phase. Evasion attacks use adversarial samples, which are model queries that are engineered to trick the model. These types of samples can be constructed by analyzing model output, known as *black box attacks*, or with access to internal model parameters, known as *white box attacks*. Evasion attacks are not addressed by this work, and as all workers have full knowledge of the global model, the threat of *white box attacks* persists in the proposed system [57].

Overall, the system addresses privacy inference attacks aimed at extracting sensitive information about specific workers, but global inference attacks remain unaddressed. As the system assumes workers are at least honest-but-curious participants, poisoning attacks are not mitigated. Moreover, the unlinkability property increases this threat as it also precludes protective mechanisms such as reputation systems.

6.6 Comparison to Nebula

Initially, *Nebula* was considered as a foundation for implementing the proposed peer-based mixing approach for DFL. However, the framework was ultimately not built on *Nebula* but was instead developed from scratch. This decision arose after recognizing several limitations in *Nebula*’s networking module that made it unsuitable for the intended use case. These limitations are revisited and contrasted with the current implementation.

Nebula faces several limitations that impact its adaptability and robustness in dynamic federated learning environments. One key drawback is its lack of peer connection monitoring, which prevents it from actively adapting to changes in network topology, such as node failures or disconnections. Without active monitoring, *Nebula* cannot detect or respond to inactive peers. The assumption of a persistent network with healthy nodes is deeply rooted in *Nebula*’s implementation and is a major reason why it was not used as a base implementation. Instead, the current system does not rely on the assumption of stability and implements a peer monitoring system, which allows for increased fault-tolerance.

Another limitation is *Nebula*’s strict separation of bootstrapping the topology and the FL workflows. This design decision results in its inability to support the spontaneous joining of new nodes during runtime.

Additionally, Nebula depends on receiving a model from each peer, even if the peer is offline. It relies on long timeouts even for indefinitely closed TCP connections. This static behavior severely limits the system's ability to quickly overcome permanent changes in the topology. In contrast, the current system is entirely unaware of the fragment's senders and thus does not await a model from each peer, resulting in a smoother handling of exited nodes.

Furthermore, Nebula strictly requires a full set of parameters from each node to integrate its model, making it unsuitable for handling partial updates. Not being able to handle partial transmissions reduces Nebula's flexibility for constrained networks or devices where unreliable transmissions are a common problem. Comparatively, nodes in the current system do not depend on receiving a complete model from each peer, as it uses fragment-based aggregation. Therefore, the federation remains reactive to dynamic changes in the network.

Finally, Nebula relies on a single TCP connection between any two peers. Two nodes are required to coordinate their roles, as only one node is allowed to have the server's role. The coordination is managed by the strict ordering of the peers' IDs and multiple coordination messages. In contrast, the proposed framework adopts a more modular approach by using two separate TCP connections, one for reading and one for writing. This design decouples read and write operations, simplifying implementation and omitting any coordination efforts. Furthermore, host names are not required to be strictly sortable.

Overall, Nebula is a full-fledged framework, focusing on providing various scenarios, configurable in terms of datasets, algorithms, and malicious participants. On the other hand, its optimistic assumption of the network's state could be reconsidered.

While the newly proposed framework works out of the box for its default scenarios, it was built with a strong focus on networking. It illustrates a valid alternative approach to Nebula's communication module, as it is more resilient in non-optimal network conditions.

Chapter 7

Summary and Conclusions

This Master’s Project addresses the research gap in achieving anonymous DFL using mixnets. It proposes a novel DFL framework that integrates peer-based mixing for an anonymous, fragment-based model exchange and dynamic reconfiguration to support dynamically changing topologies.

The architecture contributes several key techniques. First, it introduces a DFL workflow that achieves automatic synchronization during the model exchange stage even in the presence of spontaneously joining or exiting participants. Second, it implements an anonymous fragment-based model exchange algorithm that splits, transmits, and aggregates model fragments without relying on a fragment’s sender identity, thereby increasing unlinkability and privacy. Third, the framework integrates a robust networking layer built on onion routing using Sphinx packets, featuring anonymous acknowledgments, resilient retransmission logic, and message duplicate handling for reliable and unlinkable communication. Fourth, the dual role of nodes as participants in the DFL scenario as well as peer-based mixnet leverages DFL’s decentralized nature without relying on external infrastructure. Fifth, the framework comes with continuous peer health monitoring, enabling the system to instantly react to unresponsive nodes or integrate newly joining participants seamlessly, ensuring robustness in dynamic topologies. Finally, it provides a fully integrated infrastructure for observing learning quality, communication behavior, computational resource consumption, and node synchronization, offering detailed real-time insights.

Experiments on the implementation have demonstrated that the system remains functional and stable while a node joins or exits the federation. Nodes immediately react to changes in topology and reconfigure their learning and mixing workflows without being disrupted or desynchronizing with their peers. Nodes joining late are integrated quickly and participate seamlessly in the peer-based mixing and DFL. The individual node’s assumptions on the other node’s behavior are minimal and result in a naturally synchronized DFL workflow, being independent of any explicit orchestration messages to its peers.

Experiments focusing on communication identified trade-offs regarding the responsiveness and network load. Short, stochastically uniform sending intervals keep the network responsive, while increasing the load onto the peers. RTT grows approximately linearly with the sending interval and intermediate hop count, while the size of the shuffled outbox

of the mixer only shows a minor increase in latency. A theoretical analysis of the entropy shows that both outbox size and maximum hops have a positive impact on entropy, but with diminishing returns. As the latency and runtime experiments showed, the maximum number of message hops has a much more significant impact on runtime than the outbox size. Therefore, it is advisable to increase the outbox size rather than the maximum hops. Despite this, it is important to remember further privacy concerns, such as potentially compromised nodes, when configuring scenarios with few intermediary hops.

Performance measurements identified the CPU-time as the limiting resource. Continuous mixing running in the background competes with learning across all stages of the DFL workflow. CPU-time rises with both the number of nodes and the number of hops. As the number of nodes increases, the individual nodes' dataset shrinks, but the packets being mixed per node increases. Consequently, all stages are increasingly slower and affected by the communication overhead. Increasing the number of hops only moderately increases the total CPU-time of a scenario. However, the increase in continuously mixed packets affects all stages of the learning workflow, increasing the variance of used resources even for unrelated stages such as model validation. Memory was observed to be moderate and approximately constant during all stages, even with an increased number of nodes or intermediate hops. Round duration increases approximately linearly with both the number of nodes and the number of intermediate hops.

Machine learning results confirmed that anonymous fragment-based aggregation yields a stable global model that outperforms the local models. This illustrates the newly proposed algorithm as a robust alternative performing producing results equal to FedAvg in the best case, while being robust to dynamic changes in the topology and partially transmitted models. In a qualitative comparison to Nebula, the proposed framework is robust to topology changes, handles partial transmissions seamlessly, supports spontaneous late joins, and simplifies peer communication by decoupling read and write connections without the need for any coordination messages or sortable IDs.

The framework demonstrates a practical approach to privacy-preserving decentralized federated learning in imperfect and dynamic networks.

7.1 Limitations

While the framework demonstrates robustness and practicality, several limitations persist. The implementation introduces significant computational overhead and high latency, potentially limiting scalability. The strong emphasis on unlinkability introduces vulnerabilities, such as susceptibility to DDoS attacks or disruptions caused by malicious nodes, neither of which is addressed in this work. The proposed system addresses the threat model of a global passive adversary and of honest-but-curious worker nodes. Due to the strong unlinkability property, it effectively mitigates the risk privacy inference attacks aimed at individual nodes. In contrast, global inference attacks, such as inferring a sample's global dataset membership, remain unaddressed. As the system assumes workers are at least honest-but-curious participants, poisoning attacks are not mitigated. Moreover,

strong anonymity can raise this risk because it rules out protective tools like reputation systems.

7.2 Future Work

Future research should explore mechanisms to improve scalability by optimizing resource usage, particularly CPU-time and network latency, to support larger networks. Extending the system to integrate dynamic reconfiguration for adaptive sending intervals or intermediate hops could further improve efficiency.

Additionally, researching mechanisms to detect and mitigate adversarial behavior is critical for improving the framework. Decentralized, anonymous, and fragment-based detection methods could leverage the framework's existing robustness while maintaining unlinkability.

Bibliography

- [1] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, “Reliable federated learning for mobile networks”, *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020. DOI: 10.1109/MWC.001.1900119.
- [2] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, “A survey on federated learning: The journey from centralized to distributed on-site learning and beyond”, *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5476–5497, 2020.
- [3] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, 2023.
- [4] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms”, *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.
- [5] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, “The secret sharer: Evaluating and testing unintended memorization in neural networks”, in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 267–284.
- [6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions”, *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [7] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions”, *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [8] A. Gholami, N. Torkzaban, and J. S. Baras, “Trusted decentralized federated learning”, in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 1–6. DOI: 10.1109/CCNC49033.2022.9700624.
- [9] L. Yuan, Z. Wang, L. Sun, S. Y. Philip, and C. G. Brinton, “Decentralized federated learning: A survey and perspective”, *IEEE Internet of Things Journal*, 2024.
- [10] V. Mothukuri, R. M. Parizi, S. Pouriye, Y. Huang, A. Dehghantanha, and G. Srivastava, “A survey on security and privacy of federated learning”, *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.
- [11] E. T. M. Beltrán, Á. L. P. Gómez, C. Feng, *et al.*, “Fedstellar: A platform for decentralized federated learning”, *Expert Systems with Applications*, vol. 242, p. 122861, 2024.

- [12] K. Sampigethaya and R. Poovendran, “A survey on mix networks and their secure applications”, *Proceedings of the IEEE*, vol. 94, pp. 2142–2181, Jan. 2007. DOI: 10.1109/JPROC.2006.889687.
- [13] O. Javidbakht and P. Venkitasubramaniam, “Delay anonymity tradeoff in mix networks: Optimal routing”, *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1162–1175, 2017.
- [14] J.-F. Raymond, “Traffic analysis: Protocols, attacks, design issues, and open problems”, in *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*, Springer, 2001, pp. 10–29.
- [15] D. Zhu, J. Tu, D. Cai, *et al.*, “Srfacs: A secure and robust framework for anonymous communication systems”, *PloS one*, vol. 19, no. 12, e0312817, 2024.
- [16] P. Venkitasubramaniam and V. Anantharam, “On the anonymity of chaum mixes”, in *2008 IEEE International Symposium on Information Theory*, IEEE, 2008, pp. 534–538.
- [17] X. Chen, Y. Gao, and H. Deng, “Aifl: Ensuring unlinkable anonymity and robust incentive in cross-device federated learning”, *IEEE Internet of Things Journal*, 2024.
- [18] Z. Xu, R. Zhang, W. Liang, *et al.*, “A privacy-preserving data aggregation protocol for internet of vehicles with federated learning”, *IEEE Transactions on Intelligent Vehicles*, 2024.
- [19] S. Rahmadika and K.-H. Rhee, “Unlinkable collaborative learning transactions: Privacy-awareness in decentralized approaches”, *IEEE Access*, vol. 9, pp. 65 293–65 307, 2021.
- [20] L. Lyu, J. Yu, K. Nandakumar, *et al.*, “Towards fair and privacy-preserving federated deep models”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2524–2541, 2020.
- [21] R. Dingledine, N. Mathewson, P. F. Syverson, *et al.*, “Tor: The second-generation onion router.”, in *USENIX security symposium*, vol. 4, 2004, pp. 303–320.
- [22] Y. Chen, Y. Su, M. Zhang, H. Chai, Y. Wei, and S. Yu, “Fedor: An anonymous framework of federated learning in internet of things”, *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 18 620–18 631, 2022.
- [23] X. Wang, J. Li, H. Lin, C. Dai, S. Garg, and G. Kaddoum, “Aefl: Anonymous and efficient federated learning in vehicle road cooperation systems with augmented intelligence of things”, *IEEE Internet of Things Journal*, 2024.
- [24] X. Ma, X. Sun, Y. Wu, Z. Liu, X. Chen, and C. Dong, “Differentially private byzantine-robust federated learning”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3690–3701, 2022.
- [25] C. Gu, X. Cui, X. Zhu, and D. Hu, “Fl2dp: Privacy-preserving federated learning via differential privacy for artificial iot”, *IEEE Transactions on Industrial Informatics*, vol. 20, no. 4, pp. 5100–5111, 2023.
- [26] Y. Huang, G. Xu, Q. Wang, X. Song, and X. Wang, “Efficient and privacy-preserving authentication for federated learning in industrial internet of things data sharing application”, *IEEE Internet of Things Journal*, 2024.

- [27] J. Zhang, J. Zhang, Z. Ma, T. Li, X. Li, and J. Ma, “Rupt-fl: Robust two-layered privacy-preserving federated learning framework with unlinkability for iov”, *IEEE Transactions on Vehicular Technology*, 2024.
- [28] D. Mazières and M. F. Kaashoek, “The design, implementation and operation of an email pseudonym server”, in *Proceedings of the 5th ACM Conference on Computer and Communications Security*, ser. CCS ’98, San Francisco, California, USA: Association for Computing Machinery, 1998, pp. 27–36, ISBN: 1581130074. DOI: 10.1145/288090.288098. [Online]. Available: <https://doi.org/10.1145/288090.288098>.
- [29] C. Gulcu and G. Tsudik, “Mixing e-mail with babel”, in *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*, 1996, pp. 2–16. DOI: 10.1109/NDSS.1996.492350.
- [30] P. Y. Ryan, P. B. Rønne, and V. Iovino, “Selene: Voting with transparent verifiability and coercion-mitigation”, in *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20*, Springer, 2016, pp. 176–192.
- [31] C. Killer, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, and B. Stiller, “Provo-tumn: Decentralized, mix-net-based, and receipt-free voting system”, in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022, pp. 1–9. DOI: 10.1109/ICBC54727.2022.9805556.
- [32] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, “Chvote system specification.”, *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 325, 2017.
- [33] G. Danezis and C. Diaz, “A survey of anonymous communication channels”, 2008.
- [34] C. Diaz, H. Halpin, and A. Kiayias, *The nym network*, <https://nymtech.net/nym-whitepaper.pdf>, Accessed 7 March 2025, 2021.
- [35] The Tor Project, Inc., *Tor browser*, version 14.5.5, Jul. 23, 2025. [Online]. Available: <https://www.torproject.org>.
- [36] P. Syverson, “Why i’m not an entropist”, in *International Workshop on Security Protocols*, Springer, 2009, pp. 213–230.
- [37] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms”, *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.
- [38] B. Adida, “Helios: Web-based open-audit voting.”, in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.
- [39] D. A. López-García, J. Pérez Torreglosa, D. Vera, and M. Sánchez-Raya, “Binary-tree-fed mixnet: An efficient symmetric encryption solution”, *Applied Sciences*, vol. 14, no. 3, p. 966, 2024.
- [40] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, “The loopix anonymity system”, in *26th usenix security symposium (usenix security 17)*, 2017, pp. 1199–1216.

- [41] S. A. Gaballah, L. Abdullah, M. Mühlhäuser, and K. Marky, “Let the users choose: Low latency or strong anonymity? investigating mix nodes with paired mixing techniques”, in *Proceedings of the 19th International Conference on Availability, Reliability and Security*, 2024, pp. 1–11.
- [42] M. Rahimi, “Malaria: Management of low-latency routing impact on mix network anonymity”, in *2024 22nd International Symposium on Network Computing and Applications (NCA)*, IEEE, 2024, pp. 193–202.
- [43] A. Blanco-Justicia, J. Domingo-Ferrer, S. Martínez, D. Sánchez, A. Flanagan, and K. E. Tan, “Achieving security and privacy in federated learning systems: Survey, research challenges and future directions”, *Engineering Applications of Artificial Intelligence*, vol. 106, p. 104468, 2021.
- [44] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data”, in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.
- [45] X. Cao, M. Fang, J. Liu, and N. Z. Gong, “Fltrust: Byzantine-robust federated learning via trust bootstrapping”, *arXiv preprint arXiv:2012.13995*, 2020.
- [46] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent”, *Advances in neural information processing systems*, vol. 30, 2017.
- [47] L. Wang, X. Zhao, Z. Lu, L. Wang, and S. Zhang, “Enhancing privacy preservation and trustworthiness for decentralized federated learning”, *Information Sciences*, vol. 628, pp. 449–468, 2023.
- [48] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, “Biscotti: A blockchain system for private and secure federated learning”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, 2020.
- [49] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, “{Batchcrypt}: Efficient homomorphic encryption for {cross-silo} federated learning”, in *2020 USENIX annual technical conference (USENIX ATC 20)*, 2020, pp. 493–506.
- [50] G. Danezis and I. Goldberg, “Sphinx: A compact and provably secure mix format”, in *2009 30th IEEE Symposium on Security and Privacy*, IEEE, 2009, pp. 269–282.
- [51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.
- [52] T. maintainers and contributors, *TorchVision: PyTorch’s Computer Vision library*, Nov. 2016. [Online]. Available: <https://github.com/pytorch/vision>.
- [53] Python Software Foundation, *secrets — Generate secure random numbers for managing secrets*, Accessed: 2025-07-04, 2025. [Online]. Available: <https://docs.python.org/3/library/secrets.html>.
- [54] C. E. Shannon, “A mathematical theory of communication”, *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [55] Raspberry Pi Foundation, *Raspberry pi 4 model b datasheet*, <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>, Accessed: 2025-08-08.

- [56] Espressif Systems, *Esp32 series datasheet v4.9*, Revision 4.9, Accessed: 2025-08-08, 2025.
- [57] P. Liu, X. Xu, and W. Wang, “Threats, attacks and defenses to federated learning: Issues, taxonomy and perspectives”, *Cybersecurity*, vol. 5, no. 1, p. 4, 2022.

Abbreviations

FL	Federated Learning
DFL	Decentralized Federated Learning
CFL	Centralized Federated Learning
ML	Machine Learning
FedAvg	Federated Averaging
MNIST	Modified National Institute of Standards and Technology (dataset)
CNN	Convolutional Neural Network
SDFL	Semi-distributed Federated Learning
DOS	Denial of Service
OPK	One-time Private Key
OTD	One-time Address
AODV	Ad-hoc On-demand Distance Vector (routing)
TOR	The Onion Router
GPOR	Greedy Geographic Routing with Path Optimization
VPN	Virtual Private Network
DL	Distributed Ledger
SURB	Single Use Reply Block
SSE	Server Side Event
RTT	Round-Trip Time
Conv2D	Two Dimensional Convolution
ReLU	Rectified Linear Unit
FIFO	First in first out

List of Figures

2.1	System Model of Mixnets	8
3.1	Grouped Onion routing [23].	11
4.1	High-Level View of Round Trip	20
4.2	Architecture Overview	22
4.3	Forward- and backward-path of a fragment through the mixnet.	25
5.1	Implementation Overview.	28
5.2	Node Overview in Frontend	29
5.3	Exemplary Selection of Metrics out of 33 Options in Frontend	30
5.4	Sequence Diagram of Learning and Communication Process	33
6.1	Self-Reported Number of Peer Connections per Node	39
6.2	Stage Information per Node in Topology	40
6.3	Unacknowledged Messages per Node in Topology	41
6.4	Number of Messages Retransmitted per Node in Topology	41
6.5	Effect of μ on Average RTT Compared to Its Effect on Traffic Rate	42
6.6	Effects of Mixnet Parameters on Average RTT	43
6.7	Effects of Mixnet Parameters on Average Output	44
6.8	Effects of Mixnet Parameters on Total Bytes Sent	44
6.9	Relay Entropy as a Function of Outbox Size and Maximum Hops	46
6.10	Path Entropy as a Function of Outbox Size and Maximum Hops	47
6.11	Impact of Network Size on CPU Consumption	49

6.12 Impact of Intermediate Hops on CPU Consumption	50
6.13 Impact of Network Size and Intermediate Hops on Memory	51
6.14 Impact of Network Size and Maximum Hops on Round Duration	51
6.15 Average Local and Global Accuracy Across Nodes Over Time	53
6.16 Average Local and Global Accuracy Across Nodes Number of Nodes (N) .	54

List of Tables

3.1	Example Mixnets in Comparison	14
3.2	Comparison of Related Work I	16
3.3	Comparison of Related Work II	17

.1 Source Code and Reproducibility

For transparency and reproducibility, we provide the complete framework alongside the recorded metrics data to generate the figures. The latest released versioned is available at [GitHub Release v1.0](#).

The release contains the following:

- Framework source code for the Manager, Nodes, and Frontend.
- Jupyter notebooks used to generate plots and insights for evaluation.
- Recorded metrics used for evaluation.
- Figures produced by the Jupyter notebooks.
- Slide decks (PowerPoint) for the intermediate and final presentations.
- PDF export of the underlying written work.