



University of
Zurich^{UZH}

Design and Implementation of a Scalable and Privacy-Aware Networking Architecture for Decentralized Federated Learning

*Sandrin Raphael Hunkeler
Zurich, Switzerland
Student ID: 18-253-815*

Supervisor: Chao Feng, Thomas Grübl
Date of Submission: July 05, 2026

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 All passages taken verbatim or in sense from published or unpublished writings are identified as such.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, June 23, 2026

Sandrin Hunkeler

Abstract

English

Decentralized federated learning (DFL) trains a shared model across distributed nodes without sharing their private and sensitive training data. The model exchange improves privacy but still links the participants through the communication link. Therefore, it is vulnerable to attacks targeting anonymity. Peer-based mixing hides these communication links by fragmenting the model update and routing the fragments through a mixnet that the DFL nodes run themselves. A previous version of the framework showed this is feasible, but only on a six-node full mesh with model compression.

This thesis refactors and extends that framework to scale anonymous DFL up while improving configurability and efficiency. The transport layer is refactored to use QUIC with mutual TLS for encryption and authentication. The model exchange extends the message compression with a 8-bit quantization on the fragment level. Moreover, a configurable gossiping algorithm controls the ratio of the model each peer receives. The existing framework's full mesh is extended by structured circulant topologies. Due to the sparse topologies, a node no longer has a transport link to every peer in the scenario. Therefore, each peer observes peer liveness through the overlay graph by analyzing its anonymous acknowledgments. In the case of node churn, the nodes reroute their traffic over alternative paths to maintain a reliable model exchange. Krum is added as a Byzantine-robust aggregation algorithm alongside FedAvg to defend against model poisoning from the now anonymized senders.

The framework is evaluated regarding the trade-offs between anonymity, computational resources, and learning quality. At the $N=32$ baseline, LeNet-5 reaches an aggregated accuracy of 0.92 on MNIST and 0.73 on Fashion-MNIST and only loses a few accuracy points when scaling up the scenario towards $N=64$. The baseline configuration of $K = 2$, $O = 50$ on a circulant topology reaches 13.3 bits of path entropy at a round-trip time (RTT) of 29.7 s. Higher anonymity settings with $K = 5$ reach 33.2 bits at 245 s RTT. This shows the cost of anonymity. Mixed rounds take about ten times the wall-clock of unmixed rounds. Moreover, each added hop or shorter send interval increases the per-round cost faster than the additionally gained entropy. The experiments determine the practical operating range at three hops or fewer to achieve the required path entropy. The remaining parameters should be used to adjust the framework for the available computational resources and the required RTT.

Deutsch

Bei Dezentralem föderiertem Lernen (DFL) wird ein ein Modell über dezentrale Knoten trainiert, ohne deren privaten und sensitiven Trainingsdaten zu teilen. Der Austausch der Modelle selbst verknüpft die Teilnehmer aber weiterhin über den Kommunikationsweg. Peer-basiertes Mixing anonymisiert diese Verbindungen, indem es das Modell fragmentiert und jedes Modellfragment durch ein Mixnet leitet. Das Mixnet selber wird ebenfalls durch die DFL-Knoten selbst betreiben. Ein früheres Framework zeigte, dass dies möglich ist, allerdings nur in einem Full Mesh aus sechs Knoten ohne optimierungen für Flexibilität und Skalierbarkeit.

Diese Arbeit überarbeitet und erweitert dieses Framework, um die Methoden und die Architektur von anonymem DFL effizienter und skalierbarer umzusetzen. Der Transportlayer wurde mit QUIC und gegenseitigem TLS für Verschlüsselung und Authentifizierung neu aufgebaut. Der Modellaustausch ergänzt nun die bestehende generische Kompression um eine 8-Bit-Quantisierung und einen Gossipp Algorithmus. Dies, um die Verteilung der Modell Fragmente auf stochastischer Basis auszutauschen. Das bisherige Full Mesh wird durch weniger verbundene zirkulante Topologien ergänzt. Da ein Knoten nicht mehr zu jedem Peer eine Transportverbindung hält, beobachtet jeder Knoten die Erreichbarkeit der Peers über den gemixten Overlay-Graphen. Bei einem unvorhergesehenen Exit eines Knotens, konfigurieren die Knoten die Onion-Pakete über alternative Pfade. Dies, um die Zuverlässigkeit des Austausches zu erhalten. Um sich vor anonymen Knoten mit der Absicht das Modell zu vergiften zu schützen wird Krum nebst FedAvg als Byzantine-robusten Aggregationsalgorithmus hinzugefügt.

Das Framework wird evaluiert über den Kompromiss zwischen Anonymität, Rechenressourcen und Lernqualität. Bei der Basiskonfiguration mit $N=32$ erreicht LeNet-5 eine Genauigkeit von 0.92 auf MNIST und 0.73 auf Fashion-MNIST und verliert bis zur maximal evaluierten Skalierbarkeit von $N=64$ nur wenige Punkte. Mit $N=32$, $K = 2$, $O = 50$ erreicht eine zirkulante Topologie 13.3 Bit Pfadentropie. Dies, bei einer Umlaufzeit der Nachrichten von 29.7 s, während Einstellungen für hohe Anonymität mit $K = 5$ bei 33.2 Bit eine Umlaufzeit von 245 s erreichen. Diese Anonymität hat ihren Preis. Gemixte Runden dauern etwa zehnmal so lange bei der Standardkonfiguration als ungemixte. Zusätzliche Hops oder ein kürzeres Sendeintervall erhöht die Kosten pro Runde schneller als die dabei gewonnene Entropie. Die Experimente legen für das Framework einen praktischen Arbeitsbereich von höchstens drei Hops um die Pfadentropie zu garantieren. Die übrigen Parameter sollten primär für die Feinjustierung der Computerressourcen und der Kommunikation verwendet werden.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Chao Feng. His consistent support, valuable feedback, and generous time investment have been crucial throughout this thesis.

Additionally, I am grateful to Prof. Dr. Burkhard Stiller for the opportunity to complete my Master's Thesis at the Communication Systems Group (CSG) of the University of Zurich. This experience has been personally enriching and has enhanced my 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 Contributions	2
1.3 Description of Work	2
2 Background	3
2.1 Federated Learning	3
2.1.1 Decentralized Federated Learning	4
2.1.2 Privacy and Security	4
2.2 Mixnets	4
2.2.1 Entropy	5
2.3 Model Exchange Process	6
3 Related Work	9
3.1 Communication Protocols	9
3.1.1 Distributed Networks	9
3.1.2 In-Memory and Local Networks	10
3.2 Model Payload and Compression	10

3.2.1	Payload	11
3.2.2	Model Fragmentation	11
3.2.3	Message Compression	12
3.2.4	Optimized Quantization	12
3.3	Temporal and Spatial Model Exchange	13
3.3.1	Topology	13
3.3.2	Gossiping	14
3.3.3	Synchronization	14
3.4	Linking Model Updates	14
3.5	Summary	15
3.6	Research Questions	16
3.7	Objectives	16
4	Architecture	19
4.1	System Assumptions	19
4.2	Overview	20
4.3	Communication Protocol	20
4.4	Model Payload and Compression	22
4.4.1	Quantization	22
4.4.2	Krum Aggregation	23
4.5	Temporal and Spatial Model Exchange	23
4.5.1	Spatial Model Exchange	23
4.5.2	Temporal Model Exchange	26
4.6	Adaptive Peer Liveness and Resend Control	26
4.6.1	Per-peer session	26
4.6.2	The liveness signal	27
4.6.3	Ack-gated resend control	27
4.6.4	Marking unreachable and natural recovery	27

<i>CONTENTS</i>	ix
4.6.5 Mesh-healthy bring-up	28
4.6.6 Liveness-aware first-hop selection and persistent reconnect	28
4.7 Summary	29
5 Implementation	31
5.1 Inherited Framework and Scope of Contribution	31
5.2 Scalable Transport	32
5.3 Bandwidth-Efficient Model Exchange	33
5.3.1 Stochastic Quantization	33
5.3.2 Fragment Encoding	33
5.3.3 Temporal Partial Exchange	34
5.4 Structured Topologies	34
5.4.1 Topology Generation	34
5.4.2 Topology-Aware Sphinx Routing	35
5.5 Aggregation	35
5.5.1 FedAvg with Partial Updates	35
5.5.2 Krum Selection	35
5.5.3 Byzantine Attack Simulation	36
5.6 Reliable Delivery under Anonymity	37
5.7 Adaptive Peer Liveness and Resend Control	38
5.8 Runtime Performance	40
5.9 Orchestration and Configurability	41
5.10 Summary	41

6	Evaluation	45
6.1	Setup	45
6.2	Scalability	46
6.2.1	Convergence at scale	46
6.2.2	Communication volume scaling	47
6.2.3	Resource scaling by degree	50
6.2.4	Model-size scaling	52
6.3	Anonymity	54
6.3.1	Relay and path entropy	54
6.3.2	Anonymity set by topology and N	55
6.3.3	Cover traffic	56
6.3.4	The cost of mixing	57
6.3.5	Content attack from a curious recipient	59
6.4	Joint optimization	61
6.4.1	Mixnet parameters	61
6.5	Convergence verification	66
6.5.1	Sparse topology	68
6.5.2	8-bit quantization	68
6.5.3	Partial-update ratio	69
6.5.4	Data heterogeneity	71
6.6	Stressors	71
6.6.1	Dynamic membership	71
6.6.2	Byzantine attacks	74
6.7	Comparison to previous work	76
6.7.1	Comparison against unlinkability frameworks	76
7	Summary and Conclusions	79
7.1	Limitations	80
7.2	Future Work	80

<i>CONTENTS</i>	xi
Bibliography	83
Abbreviations	87
List of Figures	88
List of Tables	92
A Source Code and Reproducibility	95

Chapter 1

Introduction

Decentralized Federated Learning (DFL) is used to train a shared model across distributed nodes without sharing sensitive training data but only locally trained model parameters [1]. Despite this, the model exchange itself still links participants to one another through the communication link. Being able to link a model exchange to participants exposes and undermines the anonymity and privacy of them [2]. Peer-based mixing addresses this by routing the fragmented model through a mixnet hosted by the DFL nodes themselves [3]. This thesis refactors and extends the existing peer-based mixing framework of [3] to scale higher and improving configurability. Finally, this work explores the optimization and trade-offs between anonymity, use of computational resources, and the learning quality.

1.1 Motivation

The predecessor framework [3] demonstrated that anonymous DFL is feasible, but only at a small scale with limited configurability. It runs on a six-node full mesh, where the number of connections grows quadratically with the number of nodes. It applies no model compression and broadcasts the full model deterministically to every peer in the mesh.

Related work shows that communication is the is the main constraint for scalability in both centralized and decentralized federated learning. Frameworks improve scalability through sparse topologies, highly compressed gradients, and reduced cryptographic overhead. However, peer-based mixing requires symmetric and uniform topologies without any network patterns, uniform packet sizes, and a constant output rate. All this is violated by design by the improvements of the related work.

This trade-off defines the research gap this work addresses, namely the joint optimization of the communication layer for scalable anonymous DFL. The goal is to push the setting from $N=6$ of [3] to a new $N=64$ limit. This, while keeping entropy, computational resources, and learning quality inside practical ranges.

1.2 Contributions

This thesis refactors and extends the peer-based mixing framework of [3]. It keeps the framework’s dual role, where every node is both a DFL participant and a mixing node. This, so the mixnet runs on the nodes themselves and without any external or centralized infrastructure. The goal is to preserve DFL’s decentralized nature. It also keeps the threat model of a global passive adversary and honest-but-curious peers. In addition to a general refactoring of [3], it makes the following contributions:

- It scales anonymous DFL from the six-node full mesh of [3] to an $N=64$ ceiling.
- It jointly optimizes the communication layer for scaling up mixed DFL through 8-bit quantization, a mixed partial-update ratio gossiping algorithm, and diameter optimized sparse circulant topologies.
- It replaces the transport layer with QUIC to ensure mutual TLS for encryption and authentication.
- It separates the authenticated and encrypted transport topology from the unlinkable mixed overlay topology.
- It observes peer liveness through the overlay topology instead of holding a transport link to every peer. This, to reroute over alternative paths to stay reliable under node churn while preserving the path entropy.
- It introduces a fragment-based Krum as a Byzantine-robust aggregation algorithm to mitigate model poisoning. This, due to the increased vulnerability of the nodes due to the high level of anonymity.

1.3 Description of Work

The next chapter covers key background concepts. A literature review then compares related DFL frameworks and their communication layers as well as techniques for unlinkable DFL. This is then followed by the research gap, the research questions, and the objectives of this work. The following chapters introduce the architecture and then the specifics of its implementation. The evaluation provides a comprehensive and detailed look into the trade-offs and performance of the framework. Finally, the work and its findings are summarized before having an outlook on the future work.

Chapter 2

Background

This chapter introduces the technologies and anonymity mechanisms that form the basis of this thesis. It first explains why federated learning (FL) emerged as an alternative to centralized machine learning and how different FL structures work in practice. Afterwards, it introduces mixing networks as a technique to disrupt traffic patterns.

2.1 Federated Learning

Machine learning traditionally depends on centralizing all training data in one location. While this simplifies data collection and training, this approach fully exposes sensitive information and makes the process vulnerable to failures or attacks targeting the central server [4]. Federated learning was introduced to overcome these limitations by letting each participant keep its data locally and only share its private model updates to maintain data privacy [5]. To aggregate multiple individual data owner's models into a single final model, various algorithms exist [1], [6].

The motivation behind FL is not limited to data privacy. Reducing the amount of transmitted data also lowers bandwidth requirements, and opens ML methods to use cases where ML methods are currently not performed due to their sensitive training data [4], [6]. However, distributing the training process also introduces new challenges and vulnerabilities. Due to the training being performed in private, malicious participants can attempt to tamper with their own model or attack other participants' models by applying malicious techniques to gain insights on the private training data [7]. These concerns show that while privacy is improved, authenticity becomes a major issue when using FL.

FL can be implemented in multiple ways. The most widely deployed structure still relies on a central server to collect and aggregate updates. This approach, known as centralized federated learning (CFL). CFL simplifies the coordination but keeps the aggregation step on a single trusted participant [7]. Although CFL avoids transferring private data, it inherits some of the risks of traditional centralized architectures such as their limited scalability [8].

2.1.1 Decentralized Federated Learning

A different strategy removes the weakness of having a central aggregator. In decentralized federated learning, aggregation is distributed across the participating nodes themselves [1]. This design eliminates the single point of failure and increases robustness, but it also generates more communication traffic and makes coordination more difficult [1]. Because the network structure and data distribution vary widely between participants, the training and aggregation become more dynamic and requires more careful alignment [9]. Balancing communication, local computation, and aggregation therefore becomes one of the main engineering challenges in DFL [9].

2.1.2 Privacy and Security

Privacy concerns in FL exist mainly at two levels. One layer is the communication between devices. The other is the training and aggregation process itself, where model updates may unintentionally leak information [2]. Removing the central server in DFL improves privacy at the training layer because no trusted authority is required to handle the updates [2].

However, neither CFL nor DFL are immune to privacy leakage on the communication layer. The parameters can still contain traces of sensitive information which result in unintentional memorization [10]. Furthermore, with increased privacy the vulnerability to attackers distributing poisoned updates is increasing. To mitigate poisoning attacks, each individual DFL aggregator is required to deploy some mitigation techniques such as anomaly detection [11].

2.2 Mixnets

Mix networks address a completely different problem from FL but share similar privacy goals. Introduced by Chaum in 1981 [12], mixnets are designed to hide who communicates with whom. Instead of protecting message content, they focus on preventing observers from linking senders and recipients, a challenge commonly named the traffic analysis problem.

A mixnet processes messages through a sequence of nodes, called mixes. Each mix applies various techniques to decouple incoming from outgoing messages [13]. To prevent linking messages, mixes usually pad messages to a uniform size [14]. Furthermore, messages can be randomly delayed or reordered to make time based analysis less effective [14]. The scheduling of output batches can follow fixed rules, randomized patterns, or even include dummy messages to hide and reduce traffic patterns [15].

These protections introduce overhead. Batching adds latency, and cryptographic transformations increase computational cost [13]. For this reason, mixnets are rather unsuitable for scenarios that require near-real-time responsiveness, such as interactive video. Another limitation is reliability, meaning if a mix becomes unavailable, messages within the node or being routed over the node can be lost [16].

Nevertheless, mixnets do not rely on centralized entities and are meant to be fully decentralized [12], making them naturally work well with DFL.

Threat Model

This work assumes a global passive adversary that can observe all network connections. This adversary can monitor incoming and outgoing packets at every node, trace their timing. However, the adversary cannot observe the internal mechanics of a mixer. In particular, the adversary cannot determine which incoming packet corresponds to which outgoing packet inside the shuffled batch. Therefore, the only anonymity in this model is sourced from the mixing of the individual nodes [17].

Overall, the topology and its parameters can impact anonymity [17]. If the topology is not uniform but has different node roles, inhomogeneous regions, irregular edges, or inconsistent mixing behavior, the adversary can exploit these asymmetries to reduce anonymity [17]. On the other hand, if all nodes behave identically and receive approximately the same number of messages, the adversary cannot gain additional information from the graph structure. Under these uniform conditions and the global passive adversary model, the effective anonymity comes only from the combined mixing of the hops. Meaning, the path entropy depends only on the shuffled outbox size O and the maximum number of hops K .

2.2.1 Entropy

Unlinkability between a sender and receiver can be expressed using Shannon entropy [18]. Entropy in a peer-based mixnet indicates the uncertainty the passive observer has about which message belongs to which node. If we assume X to be a discrete random variable over a limited candidate set, and with $p(x)$ being the probability that candidate $x \in X$ is the true one from the observer's perspective, the Shannon entropy of X is

$$H(X) = - \sum_{x \in X} p(x) \log p(x) \quad (2.1)$$

High entropy is similar to a distribution $p(x)$ that is close to uniform, showing high anonymity. Low entropy indicates probability mass concentrated around a few candidates and thus weaker anonymity. Measured in bits, entropy describes how uncertain the adversary is when identifying the true sender or receiver [17]. Although entropy increases with the number of hops and the size of shuffled messages, it is fundamentally bounded by the number of possible receivers, namely the anonymity set.

Relay Entropy

Relay entropy (local message entropy) measures the uncertainty of linking an incoming message on a single mixing node to one of its outgoing messages. It depends only on the

mixing behavior at the node and is independent of the network topology. This, as long as all nodes receive approximately the same amount of messages and behave uniformly [17].

An incoming message m_i at node n is either addressed to n or forwarded further. The probability of the message being addressed to n is $\frac{1}{K}$, where K is the number of possible receivers. With probability $\frac{K-1}{K}$, the message is forwarded and mixed with $O - 1$ other messages. Under these assumptions, each outgoing position in the shuffled batch has probability

$$\frac{K-1}{K} \cdot \frac{1}{O} \quad (2.2)$$

The relay entropy is defined as the Shannon entropy of this distribution and defines the local unlinkability provided by a single mix [17].

Path Entropy

Path entropy captures the relay entropy accumulated along a full end-to-end route from sender to receiver across multiple mixing hops. Because the adversary observes all network connections, the number of nodes in the topology does not increase the anonymity. This, as the adversary can trace all forwarding relations between nodes. Therefore, the remaining uncertainty results only from the sequence of mixed hops.

If a message traverses K hops, each hop contributes a factor of O possible shuffled outputs. The adversary must therefore consider all sequences of shuffles of length K , resulting in the total number of possible paths

$$\sum_{k=1}^K O^k \quad (2.3)$$

Assuming all path lengths are equally likely and the shuffling is uniform, all paths in this set are equal in probability. The resulting path entropy is

$$H_{\text{path}} = \log_2 \left(\sum_{k=1}^K O^k \right) \quad (2.4)$$

This defines the end-to-end entropy induced solely by the mixnet under the global passive adversary model.

2.3 Model Exchange Process

The communication process in DFL differs depending on if the updates are exchanged directly or through peer-based multi-hop mixing. The following summarizes both approaches.

Non-Mixed DFL

Non-mixed DFL uses direct peer-to-peer model exchange with minimal overhead and no anonymity guarantees. The overall procedure is similar across DFL frameworks such as [19], [20].

1. **Local training:** Each node trains its model on private data.
2. **Direct exchange:** Nodes transmit updates directly to a selected set of peers.
3. **Aggregation:** Peers aggregate received updates into their local model.
4. **Iteration:** The procedure repeats until convergence or round limits are reached.

Mixed DFL

Mixed DFL anonymizes model exchange using onion routing, multi-hop relaying, and unlinkable acknowledgements, following the MixDFL design [3].

1. **Local training:** Nodes compute local model updates.
2. **Path selection:** A forward path and independent reverse path are generated.
3. **Onion packaging:** Updates are padded, optionally compressed, and onion-encrypted.
4. **Relaying:** Intermediary peers forward, batch, shuffle, and delay packets to increase entropy.
5. **Anonymous delivery:** The receiver obtains the update without seeing the sender's identity.
6. **Aggregation:** The update is aggregated using sender-independent aggregation methods.
7. **Acknowledgement:** The onion-encrypted acknowledgement is returned along its reverse path to the anonymous initial sender.
8. **Iteration:** The process repeats until the termination criteria of the DFL scenario are reached.

Chapter 3

Related Work

This chapter provides an overview of existing FL frameworks with a focus on their communication layer and anonymizing techniques. Communication plays a central role for the scalability, robustness and unlinkability of FL workflows. This, because it controls how frequently, efficiently, anonymously, and reliably nodes exchange their model updates and metadata. The surveyed frameworks are analyzed along three primary dimensions: communication protocol, payload and compression, temporal and spacial model exchange.

3.1 Communication Protocols

The communication protocol defines how information is exchanged between participating nodes. Depending on the FL-framework’s implementation, the protocol communicates through the local network, in-memory networks, or fully distributed networks. Monitoring and management traffic is excluded from this discussion to maintain focus on the model exchange. Table 3.1 provides a summary over the protocols used by the reviewed FL-platforms.

3.1.1 Distributed Networks

Asynchronous TCP sockets are widely used in DFL frameworks, mainly because they support long-lived bidirectional channels [1], [21], [22]. In contrast, many CFL platforms rely on gRPC, which combines buffers with HTTP/2 multiplexing to reduce the overhead of manually controlling serialization and parallel HTTP requests [23], [24], [25], [26], [27], [28].

Lyra [20] uses REST APIs for its P2P network. REST offers a simple, high-level interface, but it also introduces additional latency due to payload overhead and periodic reconnections. This higher abstraction level limits room for optimizing performance, making REST less suitable for scalable DFL. Similarly, FedML [29] relies on MQTT and S3 (Amazon HTTP protocol). While MQTT is a lightweight publish/subscribe messaging protocol, it mainly

suited for CFL. Its dependence on a central broker does not align well with the decentralized nature of DFL.

[30] introduces a hybrid design that integrates Inter-Planetary-File-System (IPFS) and blockchain components. The use of IPFS reduces data transfer handling by publishing only content addressed references. In addition to improving scalability, the blockchain layer is used for tracking the progression of the network which adds trust in malicious scenarios.

In addition to common protocols, some systems adopt techniques originally developed for scalable distributed computing. FATE [31] proposes a custom communication engine named Open Site Exchange (OSX). OSX combines multiple transport mechanisms such as gRPC, HTTP/1.x, and TLS while supporting distributed data frameworks like Spark and EggRoll. Similarly, P2P-Fed [32] uses a Chord distributed hash table (DHT) protocol to build a structured P2P overlay network for its scalable peer discovery and to handle high node churn.

FedLab [33] is the only reviewed framework built on *torch.distributed*. While the underlying communication module was mainly designed for distributed ML, it has shown to also be reusable for FL, making use of broadcasting feature.

3.1.2 In-Memory and Local Networks

Some frameworks support both network-based simulation and in-memory transport. Others rely on communication mechanisms that operate entirely on the local machine. [32] uses Unix sockets in FL simulations to achieve high performance on local systems. Similarly, TensorFlow Federated [26] is designed mainly for research and comes with a communication abstraction built on top of gRPC. FLGo [34] also focuses on simplifying experimentation by simulating communication directly in memory. Although these approaches do not reflect real network behavior, they enable researching scalability without the unpredictability of actual network communication.

Several frameworks also adopt *Message Passing Interface* (MPI) due to its strong performance in distributed memory environments [29], [35]. MPI is a high-performance communication protocol for distributed memory systems. While MPI provides high throughput and low latency, it introduces complexity.

3.2 Model Payload and Compression

The transmitted payload, regardless of the communication protocol, is a key factor for scalability. Most reviewed frameworks exchange full model parameters for aggregation, while a subset supports alternative payloads such as gradients or logits. In addition, many frameworks apply generic or parameter-based compression techniques to further reduce transmission size. Table 3.2 provides a summarized comparison of the reviewed FL platforms regarding payload and compression properties.

Table 3.1: Communication protocol

Src.	Platform	DFL	Protocol
[21]	Fedstellar	Yes	socket (TCP)
[1]	Nebula	Yes	asyncio (TCP)
[23]	p2pfl	Yes	gRPC, Unix Socket
[22]	DecentralizePy	Yes	socket (TCP)
[32]	P2P-Fed	Yes	socket (TCP)
[35]	GossipFL	Yes	MPI
[29]	FedML	Yes	MPI, MQTT, S3
[30]	HXSu DFL	Yes	IPFS + Blockchain
[20]	FLyra	Yes	REST
[3]	MixDFL	Yes	asyncio (TCP)
[24]	Flower	No	gRPC
[25]	OpenFL	No	gRPC
[31]	FATE	No	OSX (TCP RPC)
[33]	FedLab	No	torch.distttributed
[26]	TFF	No	simulated gRPC
[27]	Fed. Scope	No	gRPC
[28]	EasyFL	No	gRPC
[34]	FLGo	No	Simulation

3.2.1 Payload

A frequently seen approach in the reviewed framework is to reduce communication payload is to exchange gradients instead of full model parameters. This, since gradients typically scale with the input dimension rather than the overall model size. This can heavily lower total transmitted payload, especially for large models. Additionally, gradient-based payloads are well suited for applying compression techniques such as sparsification or quantization.

However, transmitting only gradients restricts the use of aggregation methods that rely on full model parameters, such as model-based similarity measures for robust aggregation. This creates a tradeoff between communication efficiency and flexibility for aggregation and analytics. Several of the reviewed DFL and CFL frameworks expose configurable payload options, supporting the exchange of gradients or models, depending on the scenario [22], [27], [29], [33].

3.2.2 Model Fragmentation

Shatter [36] divides each model into chunks and lets each node run several virtual nodes that each send one chunk. Each virtual node has its own identity and hides which node produced a given chunk. Furthermore, it prevents any receiver from getting a full model.

[3] also introduced splitting up the model into fragments, but for a different reason. A Sphinx packet carries a fixed-length body, so a model larger than one body has to be split into fixed-size fragments. So one packet carries one fragments which is then routed trough

mixnet. Therefore, fragmentation is primarily a property of the transport mechanism of [3] and not a privacy or scaling mechanism. Compared to DivShare [37] and Shatter [36], the sender of a fragment is hidden from a passive network observer. Furthermore, fragments don't have any identity or pseudonym attached.

3.2.3 Message Compression

Nebula [1], Fedstellar [21], and MixDFL [3] apply the generic *Deflate* compression algorithm (zlib), which is not optimized for model or gradient payloads. Similarly, p2pfl [23] and DecentralizedPy [22] apply optimized but generic message compression techniques such as *LZMA* or *LZ4*. Other frameworks provide no dedicated compression step at all or simply rely on communication protocol features such as binary-serialization for their compression [20], [30], [31], [32], [34].

3.2.4 Optimized Quantization

Specialized FL quantization is often applied in combination with other pre-processing techniques. This, to improving the effect of basic quantization or to compress quantized parameters further. Quantization reduces the datatype of model or gradient values to smaller representations such as int8 or low precision floats. Various DFL and CFL frameworks provide generic or specialized quantization algorithms [23], [27], [29], [33], [35]. While quantization leaves the model architecture unchanged, several frameworks additionally apply model sparsification to meaningfully and permanently reduce model size [23], [29], [35].

TensorFlow Federated [26] provides tools for active zeroing, removing extreme parameter values to improve quantization. Moreover, [26] applies L2 norm clipping before quantization. FedLab [33] applies *Quantized Stochastic Gradient* (QSGD), which uses stochastic rounding to quantize each gradient component, providing a good balance between reduced communication cost and stable convergence [38]. In addition, FedLab [33] supports top-k gradient pruning to further reduce the transmitted payload. [24] provides a selection of Dasha algorithms. Dasha algorithms [39] apply gradient compressing methods such as top-k or random-k to compress the gradient.

DecentralizedPy [22] provides multiple lossy and lossless methods for compressing model or gradient payloads. Some techniques are based on *Elias* lossless compression [40], which reduces integer sizes significantly. For float values, lossy *Elias* quantization is applied by scaling up values before type casting.

Similarly, OpenFL [25] and EasyFL [28] include a range of advanced compression algorithms adapted to FL. Some of them are based on *Sparse Ternary Compression* (STC) [41]. STC combines top-k gradient sparsification, ternarization, and efficient Golomb encoding. Top-k sparsification keeps only the k largest gradient values and sets all remaining entries to zero. Ternarization maps each gradient to $\{+1, 0, -1\}$, similar to sign-based methods but with an added neutral value. Golomb encoding [42] then compresses the sparse top-k gradient vectors into indexed gradient vectors.

Table 3.2: Exchange payload and compression.

Src.	Platform	DFL	Exchanged	Compression
[21]	Fedstellar	Yes	Params	zlib
[1]	Nebula	Yes	Params	zlib
[23]	p2pfl	Yes	Params	Quant., Spars., Low-Rank Approx., Zlib, LZMA
[22]	DecentralizePy	Yes	Params/Grad	Elias (lossless, lossy, quantization), LZ4
[32]	P2P-Fed	Yes	Params	None
[35]	GossipFL	Yes	Params	Sparsification, Quantization, Sign-based
[29]	FedML	Yes	Params/Grad	Quantization, Sparsification
[30]	HXSu DFL	Yes	Params	None
[20]	FLyra	Yes	Params	None
[3]	MixDFL	Yes	Params	zlib
[24]	Flower	No	Params	Dasha
[25]	OpenFL	No	Params	STC, SKC, KC, Eden Quantization
[31]	FATE	No	Params	None
[33]	FedLab	No	Params/Grad	Top-k, QSGD
[26]	TFF	No	Params	Quantization
[27]	Fed. Scope	No	Params/Grad	Quantization, Gzip, deflate
[28]	EasyFL	No	Params	STC
[34]	FLGo	No	Params	None

3.3 Temporal and Spatial Model Exchange

The scalability of CFL and DFL networks depends on both the node topology and the temporal dynamics of the ongoing FL scenario. Prior work [43] has shown that topology strongly influences convergence time, communication overhead during model exchange, and the computational load placed on each node. Table 3.3 summarizes reviewed FL platforms regarding their temporal and spatial characteristics.

3.3.1 Topology

Scalability is influenced by the underlying topology. DFL platforms offer significantly more topology flexibility than CFL, which are limited almost exclusively to the star topology with its central aggregator. Star topologies are limited in scalability by the capabilities of the central node. In contrast, custom DFL topologies’ scalability depends more on the number of links of the individual node.

Many reviewed frameworks allow their topology to be configurable, rather than providing strict topologies. For example, Fedstellar [21] supports full mesh, star, ring, random, and custom configurations. Similarly, DecentralizePy [22] enables arbitrary graph topologies including ring and d-regular networks. Their evaluation has shown that d-regular graphs offer a good accuracy-to-communication tradeoff [22].

3.3.2 Gossiping

Fedstellar [21] and p2pfl [23] implement gossip-based communication. Fedstellar [21] applies dynamic forwarding strategies that adapt to current network conditions. GossipFL [35] follows a very lightweight approach, where each client communicates with only a single peer per round using compressed model updates.

The work in [44] formulates gossip communication as a graph optimization problem for fixed-size networks. It identifies topologies that maximize the convergence speed of gossip-based model averaging while keeping communication overhead low. With this strategy, communication cost scales with the node degree rather than the total number of nodes.

FedLay [45] focuses on overlay topologies and provides the first fully decentralized method for creating near-random regular topologies that remain robust under node churn. Empirical experiments show that gossip learning can achieve accuracy comparable to federated averaging while reducing communication by roughly 25% [43].

DivShare [37] splits the local model into parameter slices and sends them to a random set of peers. This allows a slow node to still contribute part of its model without delaying the global convergence. This gossiping algorithm allows the nodes to proceed in the rounds more asynchronously and increases efficiency.

3.3.3 Synchronization

Model exchange can be coordinated synchronously or asynchronously. Synchronous approaches trigger model exchanges at aligned time points, which can create short but intense communication bursts. Most reviewed FL frameworks operate synchronously, exchanging only in strict round-based time boundaries to coordinate the training process.

Synchronous coordination can rely on explicit stateful messages that define when a round begins and ends, as seen in [20]. However, it can also emerge naturally when nodes wait for peer updates before progressing, as in MixDFL, Nebula, and Fedstellar [1], [3], [21]. The asynchronous method has a reduced control message overhead but may struggle under heterogeneous network conditions or heterogeneous compute resources.

In contrast, asynchronous frameworks allow peers to exchange models independently and tolerate more timing variations without requiring strict global rounds. This enables more robust operation in dynamic or unreliable networks and reduces bottlenecks caused by slower participants. Examples include [29], [32], [34], which have mechanisms to support unaligned updates.

3.4 Linking Model Updates

Various frameworks apply mixing techniques to reduce linkability in FL. [46] routes updates through onion-encrypted paths and evaluates source and path anonymity when some relays

Table 3.3: Spatial and temporal communication architecture.

Src.	Platform	DFL	Topology	Gossip	Synchronization
[21]	Fedstellar	Yes	Any	No	Sync
[1]	Nebula	Yes	Any	No	Sync
[23]	p2pfl	Yes	Any	Yes	Async or Sync
[22]	DecentralizePy	Yes	Any	Yes	Async
[32]	P2P-Fed	Yes	Chord ring	Yes	Sync
[35]	GossipFL	Yes	Any	Yes	Async
[29]	FedML	Yes	Any	No	Sync or Async
[30]	HXSu DFL	Yes	Any	No	Sync
[20]	FLyra	Yes	Any	No	Sync
[3]	MixDFL	Yes	Full Mesh	No	Sync
[24]	Flower	No	Star	No	Sync
[25]	OpenFL	No	Star	No	No
[31]	FATE	No	Star	No	Sync
[33]	FedLab	No	Star	No	Sync
[26]	TFF	No	Star	No	Sync
[27]	Fed. Scope	No	Star	No	Sync
[28]	EasyFL	No	Star	No	Sync
[34]	FLGo	No	Star	Yes	Async

are compromised. Similarly, the Tor-based FL framework [47] focuses on cost of mixing and the anonymity depending on the share of malicious routers. [48] authenticates senders with an anonymous signature scheme and argues unlinkability through a security proof rather than an experimental measurement. [49] and [50] add a global shuffler in CFL that reorders the updates to hide the original sender of an update. However, their experiments report accuracy against the added noise for improved anonymity rather than the possibility to break its anonymity. Table 3.4 lists each approach by the level at which it hides a sender and how it evaluates linkability.

Table 3.4: Sender unlinkability: protection level and how it is evaluated.

Work	Mechanism	Level	Linkability evaluated
[46]	Onion routing (GPOR)	Node, path	Source and path anonymity
[47]	Onion routing, reputation routing	Node	Cost, malicious-router rate
[48]	Anonymous signatures	Node	Security proof, cost
[49]	Gradient noise, shuffling, Byzantine-robust	Gradient	Accuracy, Byzantine robustness
[50]	Gradient noise, shuffling	Gradient	Accuracy vs. noise
This work	Peer-based mixnet	Fragment	Fragment-linkage attack, anonymity set

3.5 Summary

Existing FL frameworks show large variation in how they handle communication. Reviewed frameworks successfully apply various techniques to the three independent dimensions of the

communication layer. Communication protocols range from low-level, long-lived channels such as TCP to high-level interfaces like REST or MQTT. Each of the protocols has distinct tradeoffs between performance, implementation, and flexibility in their topology. Moreover, the choice of payload and compression defines transmission efficiency, independent of the used protocol. While several reviewed frameworks include advanced quantization or sparsification algorithms, many still rely on generic lossless compressors. This limits the payload reduction but maintains convergence quality due to no loss of information. Temporal and spatial coordination strongly affects the convergence and scalability. CFL is limited to star topologies. In contrast, DFL benefits from flexible graph structures and gossip mechanisms, which balance communication load and robustness. Across these dimensions, the frameworks also differ in how they hide the sender of an update. Most reviewed frameworks rely on routing or added noise, while they focus on entropy and learning quality.

3.6 Research Questions

Related work has shown that the individual dimensions of the communication layer strongly affect scalability and can be optimized independently. However, existing frameworks differ widely in their protocols, payload types, compression techniques, and temporal or spatial coordination of their model exchange.

Related work does not show these dimensions being optimized for scalable anonymous DFL. The requirement to maintain high Shannon entropy reduces the available design space. This, as increased scalability typically benefits from spatially sparse topologies, small buffers, and low communication overhead. Moreover, it often does not apply heavy cryptographic mechanisms for a gain in scalability. In contrast, peer-based mixing needs exactly those mechanisms to preserve its high shannon entropy and protect its participants' identities.

Motivated by these observations, this thesis investigates how communication can be jointly optimized for large scale mixed DFL under anonymity constraints. Concretely, it addresses the following research question:

1. How do different choices of payload handling, and temporal or spatial model exchange affect scalability and convergence in anonymous decentralized federated learning with limited per node connectivity?
2. How can these dimensions be jointly configured to improve scalability while maintaining sufficient Shannon entropy in peer based mixed decentralized federated learning scenarios?

3.7 Objectives

Goal of this work is to refactor and extend the framework for peer-based mixing in DFL [3]. While the current framework provides the essential tools for anonymous DFL, it is not

suited for scenarios involving many nodes. It relies solely on a full mesh architecture, where the number of connections grows quadratically with the number of nodes.

Furthermore, the framework currently does not apply any specific compression methods and only supports sending full losslessly compressed models to every node in the mesh. It also only applies deterministic broadcast, not optimizing spatial or temporal node exchange strategies for scalability. In addition, the implementation is brittle and not designed for more advanced or extensible communication techniques.

The following objectives guide the refactoring and extension of the peer-based mixing framework of [3] for scalability:

1. Design an improved communication layer for the existing mixed DFL framework [3] which jointly optimized scalability, anonymity, and convergence.
2. Refactor the existing framework [3] to improve maintainability, resource usage, configurability, and flexibility for extension.

Chapter 4

Architecture

This chapter introduces the architecture for the newly improved mixed DFL. The system supports two forwarding modes, normal and mixed. Both work with a shared transport topology. The two modes differ in routing, anonymity, timing behavior, and forwarding. All design components are independent, so anonymity mechanisms can be enabled or disabled without changing the learning, aggregation, transport, or model exchange mechanism.

The architecture is based on [3] while it introduces various changes. Overall it's goal is to improve configurability and scalability while preserving the advantages of [3]. The main architectural contributions over [3] are the newly introduced entropy- and anonymity preserving observation of the DFL-peers without having a transport connection, the fragment-based model compression, fragment-based Krum for robust aggregation, and the unlinkable fragment-based gossiping.

4.1 System Assumptions

The design depends on a few assumptions, namely the implementation and the intentions of the participants. These assumptions also follow the mixing and threat model described in Chapter 2.

- Each node holds the transport certificates, the Sphinx keys, and the neighbor information before the DFL scenario starts.
- All peers are honest-but-curious by default and follow the DFL workflow principles. The threat model assumes a global passive adversary that observes the whole network. The adversary is not implemented, it is only a theoretical construct used to calculate the entropy.

4.2 Overview

The goal of this work is to design an improved communication layer for the existing mixed DFL framework [3]. The communication layer should jointly optimize scalability, anonymity, and convergence. Moreover, it is the goal to refactor the existing framework to improve maintainability, resource usage, configurability, and flexibility for extension.

To ensure that anonymity mechanisms, transport choices, and topology settings can vary independently, the architecture is structured into four layers.

- **Topology layer (transport topology)** defines the topology spanned by the direct peer-to-peer transport connections. This layer provides secure, authenticated, and reliable transport connections to all directly connected peers.
- **Overlay topology layer** defines a logical routing graph that is constructed on top of the transport layer. It is used only in mixed DFL to enable anonymous multi-hop routing with high path-entropy.
- **Forwarding layer** performs either direct single-hop forwarding using the transport topology (normal DFL) or multi-hop anonymous forwarding using Sphinx on the overlay topology (mixed DFL).
- **Model layer** handles learning, quantization, fragmentation, serialization, malicious behavior, and compression of model updates. In addition, it performs aggregation of received fragments using either FedAvg or Byzantine-robust Krum.

Not visible in the layered architecture is the adaptive peer-liveness and resend mechanism. This algorithm keeps the model exchange reliable even under node churn while preserving anonymity and efficiency. Figure 4.1 shows the updated architecture of the underlying MixDFL framework [3]. Updates to the manager are mainly about configurability and do not affect the overall architecture of the nodes themselves.

4.3 Communication Protocol

The communication protocol establishes the transport topology and provides reliable messaging between directly connected peers. Reviewed DFL frameworks commonly use TCP or gRPC. Interestingly, no reviewed DFL framework appears to use QUIC. However, QUIC is well suited for this purpose and comes with several useful properties.

- built-in TLS with encryption and mutual authentication,
- reliable delivery with congestion control,
- message-oriented communication via independent streams,
- low setup overhead

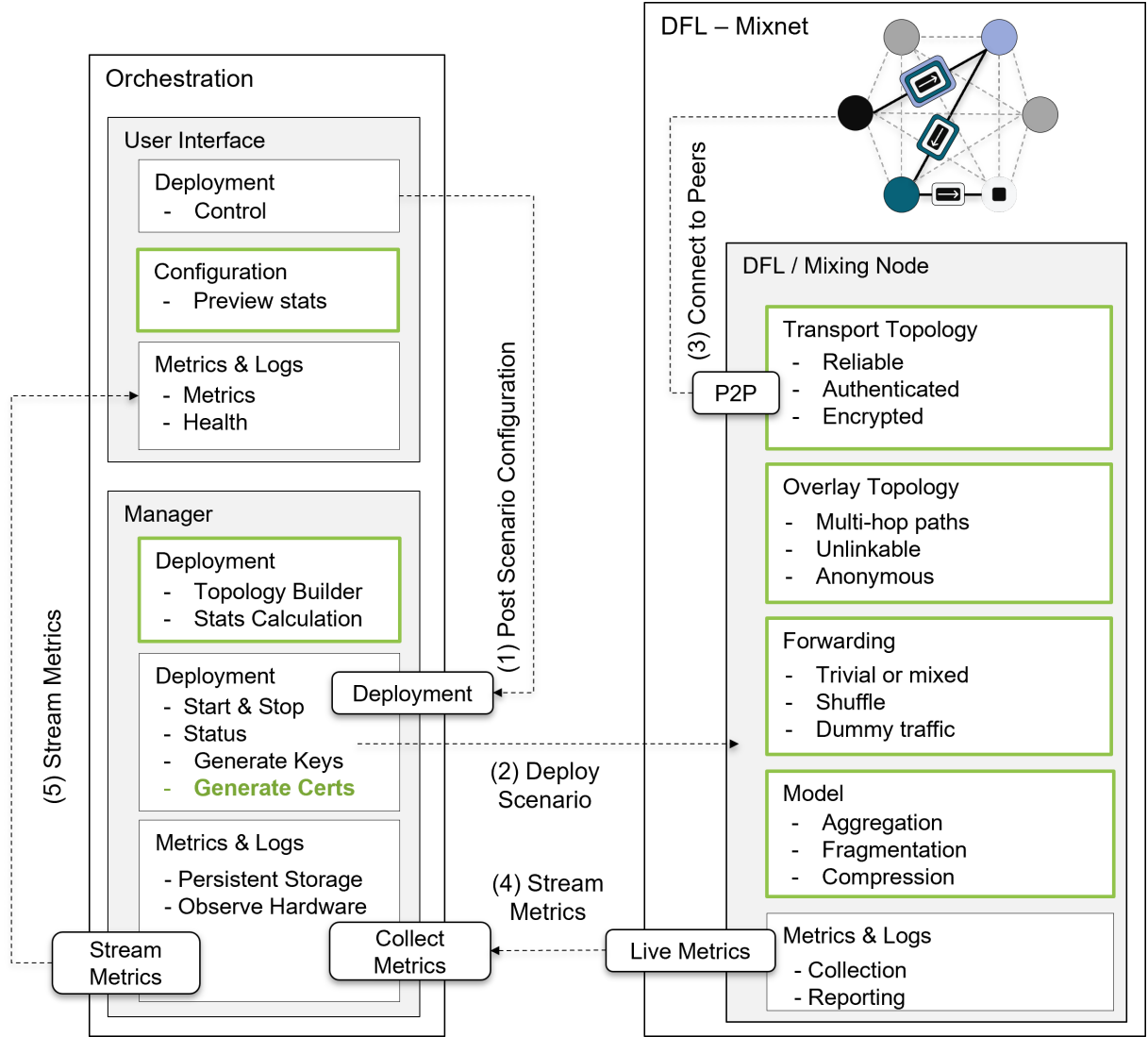


Figure 4.1: Edited Architecture overview from [3].

- no head-of-line blocking.

The inbuilt properties are suited for [3] and a good and secure replacement over the existing TCP module. In general, the transport layer must provide several functionalities to be used in the newly proposed architecture.

- authenticated and encrypted connections,
- reliable and ordered message delivery,
- multiplexed bidirectional streams,
- compatibility with high-frequency model transmission.

The new architecture strictly separates anonymity and security. Mutual authentication and per-hop encryption are located entirely in the transport layer, while anonymity comes only

from the forwarding layer. Each neighbor transport link is a single authenticated QUIC connection whose certificates are generated for the individual peers during deployment. So the transport topology is a set of certificate-authenticated point-to-point links with exactly one connection per neighbor pair. In contrast, [3] used two independent transport connections per peer and used unauthenticated and unencrypted TCP byte streams.

4.4 Model Payload and Compression

The current implementation in [3] flattens the model into a continuous `float32` vector, slices it into fixed-size fragments, and compresses each fragment using generic lossless compression. After compression, each fragment is sent as a single onion-encrypted Sphinx packet. At the receiver, fragments are aggregated using FedAvg-style element-wise averaging. The local model is always included as the first contribution. The node flattens its own parameters, initializes an accumulator and per-position counter, and then merges all received fragments.

As discussed in Chapter 3, many FL frameworks optimizing scalability apply advanced gradient compression techniques to reduce transmission cost. However, gradient-based compression becomes unstable when models diverge due to asynchronous round progression, non-IID data, or node churn.

4.4.1 Quantization

The mixed DFL framework of [3], and this extended design, assume independent nodes, ongoing churn, and no control messages for synchronous round progression. To preserve the robustness of [3] to node churn, the newly proposed architecture still sends full model payloads. However, its design reduces payload size through a combination of quantization and generic message compression.

To reduce the payload size, thereby decreasing the number of fragments and Sphinx packets needed, each fragment is quantized using an unbiased 8-bit stochastic quantizer. This approach benefits from noise smoothing when many fragments are aggregated. For each fragment, the quantization is defined as:

$$v_{\min} = \min_i w_i, \quad v_{\max} = \max_i w_i, \quad (4.1)$$

$$q_i = \text{round}_{\text{stoch}} \left(255 \cdot \frac{w_i - v_{\min}}{v_{\max} - v_{\min}} \right), \quad (4.2)$$

$$\hat{w}_i = v_{\min} + \frac{v_{\max} - v_{\min}}{255} \cdot q_i. \quad (4.3)$$

The resulting 8-bit representation substantially reduces fragment size. This also reduces the number of fragments to be exchanged as more parameters fit into a fixed-size fragment. The quantization bit width is configurable to 8-bit, while a 32-bit float setting is kept for comparison and lossless model exchange. Without quantization, only the generic message compression is used.

4.4.2 Krum Aggregation

In anonymous mixed DFL, sender identities are hidden by the mixnet. This does rule out any identity-based defenses. Robustness must therefore operate entirely at the data level. Krum [51] addresses this by selecting the single fragment closest to the honest majority, making it work well in scenarios where Byzantine participants cannot be identified or tracked by identity. The architecture introduces an adaptation of Krum algorithm that operates on fragments rather than over the full model vector. This aligns well with the existing implementation that already operates on a fragment level.

4.5 Temporal and Spatial Model Exchange

As discussed in Chapter 3, many DFL frameworks rely on gossip mechanisms and sparse topologies to improve scalability. While such topologies reduce communication overhead, they also slow down information diffusion, which affects model propagation. Moreover, sparse topologies additionally weakens the level of anonymity by reducing the entropy of the routing graph. High entropy requires uniform and symmetric node behavior, conditions that sparse or irregular network structures trade-off for scalability.

[3] supports neither sparse topologies nor gossip-based exchange. The architecture therefore introduces mixed gossiping and optimized sparse uniform topologies. Both are compatible with the anonymity requirements of mixed DFL.

4.5.1 Spatial Model Exchange

As shown in Chapter 2, entropy in peer-based mixing highly depends on uniformity. Nodes must behave identically with traffic handling, routing roles, and mixing behavior. This requires topologies that are symmetric, uniform, and regular. This, so all nodes have identical degree, betweenness, and connectivity patterns.

The transport topology determines several properties.

- the number of transport-layer connections per node (transport-degree),
- the number of overlay neighbors reachable for model exchange (overlay-degree),
- the network diameter, which influences convergence speed, and synchronization,
- and the achievable path entropy under multi-hop routing.

The architecture introduces several new topology options for the existing framework. They preserve the symmetry and regularity required for mixed DFL. On the other hand, they improve scalability in both standard and mixed modes. With all topologies, the degree is configurable but must satisfy the constraints of the chosen topology type to keep the uniformity and therefore maintain the entropy in mixed DFL.

Circulant graph. A circulant graph is a deterministic, even-degree, fully symmetric topology where each node connects to neighbors at fixed offsets. All nodes share identical structural properties, and the degree can be increased by adding more offset values. With a well-chosen offset, circulant graphs give a low diameter while keeping full uniformity. (Figure 4.2)

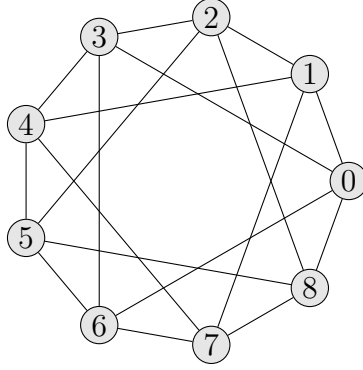


Figure 4.2: Circulant graph $C_9(\{1, 3\})$ with degree 4 and diameter 2. The offset of 3 was chosen as it results in a minimal diameter for the degree and number of nodes.

The offsets must meet two conditions. The first is connectivity, which requires that the greatest common divisor of n and the offsets is one. The second only applies when n is even. If every offset is odd, each edge flips the parity of the node index. This means that the graph splits into two equal halves with no edges between them. Mixing odd and even offsets avoids this. Within these conditions, the offset set with the smallest diameter is searched during the scenario generation by an optimizer.

Ring lattice. A ring lattice connects each node symmetrically to its k nearest neighbors on both sides. It is deterministic, uniform, and even-degree. The degree is configurable by selecting the neighborhood size, while the regular ring structure guarantees identical connectivity for all nodes. (Figure 4.3)

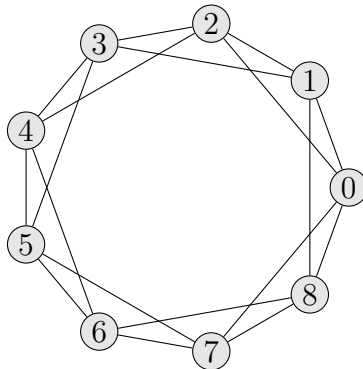


Figure 4.3: Ring lattice graph $C_9(\{1, 2\})$ with degree 4 and diameter 2. Compared to Figure 4.2 the diameter cannot be optimized as the offset must be consecutive.

Hypercube. For forming a hypercube topology, each node is assigned a binary identifier. Then the topology connects nodes whose identifiers differ by exactly one bit. It is highly symmetric, low-diameter, and uniform. The degree equals the dimension of the cube and therefore scales as $\log_2 N$. Hypercubes offer high uniformity and low diameter but require the number of nodes to be a power of two which highly limits flexibility during scenario configuration. (Figure 4.4)

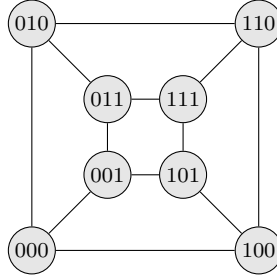


Figure 4.4: Hypercube graph Q_3 with $N = 8$, degree 3, and diameter 3. Each node is labeled by a 3-bit identifier. Two nodes are connected if their identifiers differ in exactly one bit.

Overlay Topology

In mixed mode, the overlay topology defines the P2P network on the DFL-level and uses the underlying transport-topology for multi-hop routing. Depending on the transport topology, fixed-length paths can revisit nodes or edges, but this does not weaken anonymity under the global passive adversary model 2. The overlay topology therefore is defined by all neighbors reachable in exactly the configured number of hops. Figure 4.5 illustrates this for $C_{12}(\{1, 5\})$ at $K=2$.

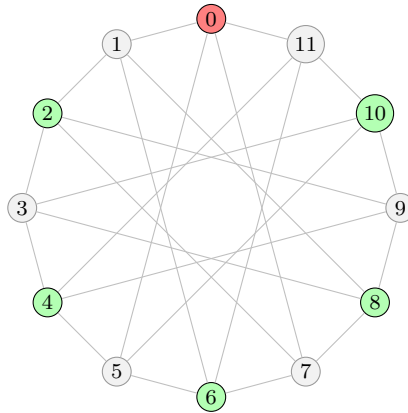


Figure 4.5: Overlay neighborhood of node 0 in the circulant graph $C_{12}(\{1, 5\})$ at $K = 2$ hops. The origin (red) and the exchange peers reachable in exactly two hops (green) form the local overlay topology. The edges show the underlying transport topology.

4.5.2 Temporal Model Exchange

The implementation of [3] performs deterministic broadcasting. Each node sends all fragments to all peers in every round. This results in a complete model transmission but also has a high communication overhead. To increase scalability, the new architecture adds a randomized partial model gossip feature. A new parameter controls the probability with which each fragment is transmitted to a given peer. This is similar to the approach of DivShare [37] for improved scalability and asynchronous round progression.

Since the aggregation mechanism already supports averaging over arbitrary subsets of fragments, partial updates integrate well into the existing workflow. This temporal randomization lowers the required number of fragments being exchanged in each round. Moreover, each node naturally receives fragments from all peers over time, enabling the system to scale to much larger networks in both normal and mixed DFL settings.

4.6 Adaptive Peer Liveness and Resend Control

In asynchronous DFL with node churn, peers may join late or exit early. Without a detection mechanism, nodes waste bandwidth on departed destinations and accumulate unacknowledged fragments addressed to peers that will never reply. Each node knows only its own scheduled join and exit rounds. The transitions of other peers are not known and must be inferred from observable behavior.

Transport connections exist only between direct graph neighbors, the relay edges of the topology. Overlay exchange peers are reached purely through multi-hop mixnet paths, and a received fragment never reveals its origin. A returned SURB acknowledgment is therefore the only obtainable end-to-end liveness signal for an overlay peer, since there is no other channel on which that peer’s reachability could be observed without breaking anonymity.

A separate per-peer session class turns that single SURB based liveness signal into a reactive controller that manages resend, unreachable-marking, and recovery while having node churn.

4.6.1 Per-peer session

Each node get a per-peer view of the overlay and transport topology over time. One session per peer owns the transport link state, the pending fragments addressed to that peer, and the liveness state. The liveness state is always either *unconfirmed*, *live*, or *unreachable*. A peer starts *unconfirmed*, meaning it should exist by the configuration but has not shown yet that traffic completes a round trip to it. Concentrating the analysis in one object lets the resend and aggregation logic reuse a single aggregated state instead of the spread reachability logic as in [3].

4.6.2 The liveness signal

Incoming fragments are reaching a node via different mix paths and reveal nothing about their origin. This makes incoming messages unusable as a per-peer liveness signal. The returned SURB acknowledgment is the only end-to-end signal that must have been sent by the peer. Compared to [3] which solely used transport link state, the newly introduced sparse topology prevent deducting liveness from the transport layer. A peer can have an active transport link while a distant overlay path to it is dead, or have an inactive transport link while still getting traffic from it over a different path. While fragment and SURBs use completely independent paths over the mixnet, SURBs are the only signal which can proof reachability. However, the liveness signal cannot differentiate between a live node having exited or just unreachable due to an exit of an intermediate hop node.

4.6.3 Ack-gated resend control

A stale unacknowledged fragment is resent only when three conditions hold at the same time. First, a peer's acknowledgment has minimum once been received. Second, no acknowledgment has arrived since this payload's last send. And last, the awaiting node has already received sufficient other fragments in the round the SURB is being awaited in. The rule of having ever received a SURB is used to detect late joining nodes which don't acknowledged at all so far. Waiting with the resend until sufficiently many fragments have arrived adapts the resending rate to the capacity of the network and the natural round-trip-time instead of using a static threshold.

If all conditions are met, a single resend is scheduled. Reliability on the overlay level is achieved by best-effort as fragments arriving too late would not be aggregated anyway by the receiver as they are probably arriving way too late. If no SURB was received in near time, all tracked resendings are paused until the node has proven to be reachable by receiving any SURB from it. All this helps to minimize traffic while maintaining meaningful reliability.

4.6.4 Marking unreachable and natural recovery

Marking a peer as *unreachable* follows two conditions: a fragment must have been resent once without having received a SURB for it and no SURB has been received by the peer lately. This reduced false positives by only marking peers as unreachable if the evidence for a node being absent is strong.

While *unreachable* peers are excluded from the fragment broadcast, each such peer is instead sent exactly one acknowledgment probe per round. This probe's only purpose is to give the peer a possibility of proving its reachability by returning the SURB. In contrast to cover traffic, the probe carries a SURB and can be acknowledged. Probes are the newly introduced efficient anonymous recovery path, which matches the asynchronous, node churn-robust architecture of the system.

4.6.5 Mesh-healthy bring-up

Before the DFL loop starts, each node waits until it has a healthy transport connection to all scheduled peers of the transport topology. If a transport connection cannot be established, a bounded timeout is awaited. Because it is not strictly blocking, a slow or late joining neighbor only delays but never blocks the start of a scenario.

4.6.6 Liveness-aware first-hop selection and persistent reconnect

Two different questions are answered by two different signals, and the architecture keeps them intentionally separated. Whether a transport neighbor can *relay* traffic and is usable as a first hop is decided by its transport link health. Whether a peer is a healthy *destination* for a model exchange is decided by the overlay reachability state tracking. The transport link health is never used for the overlay reachability tracking. This, so a short link fluctuation cannot mark a peer unreachable, and a reconnected neighbor is reconnected as a relay immediately. Using the overlay probing for transport peers would require to send a probe and await its acknowledgement.

The router uses the transport link signal at path-selection time. A precomputed path pool holds, for each source-destination pair, complete paths that each reach the destination in the configured hop count. When a node selects a path for its own outgoing traffic, it draws uniformly only among those pool entries whose first hop currently has a live transport link. The topology is static, so an exited peer is a reachability change and not a permanent topology change.

Reconnection is a continuous per-peer background loop triggered by connection loss. Any peer that flips *unreachable* mid-round drops out of the possible paths to select from so an exited peer does not halt round progression. This approach enables flexible reconfiguration depending on the sensed state of the network with a lightweight filtering approach instead of a managed topology state.

Figure 4.6 summarizes the per-peer session as a state machine.

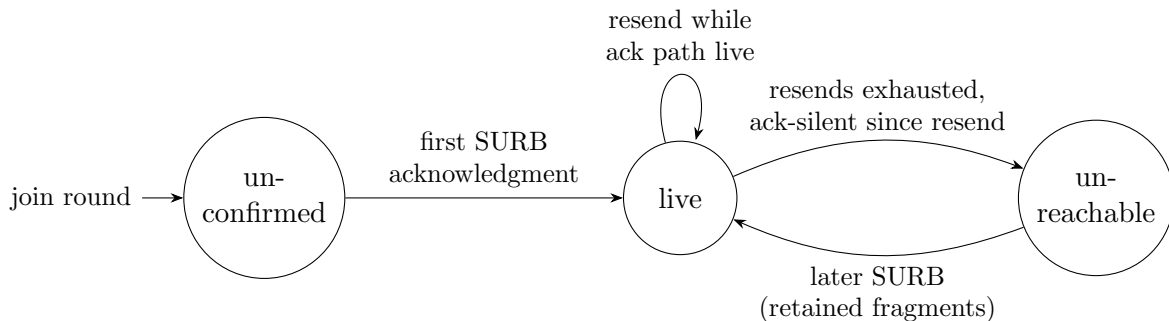


Figure 4.6: Per-peer session state. A peer is sent to while *unconfirmed* or *live* and excluded only while *unreachable*. The flip to *unreachable* requires resend exhaustion with no acknowledgment since the resend, and a later acknowledgment recovers the stored fragments.

4.7 Summary

The architecture introduced in this chapter provides a modular and scalable communication layer for both normal and mixed DFL. Each dimension of the architecture aims to preserve the entropy while enabling to scale scenarios up to have more nodes. A newly introduced peer-liveness and resend control mechanism replaces the inherited transport-based reachability method of [3]. To keep track of the network's state, a newly introduced per-peer session class is the single point of truth. The class is used to aggregate liveness signals and to adapt to the information in communication behavior. Overall, the communication system is designed for a reliable fragment exchange under node churn while ensuring quick round progression.

By separating transport, routing, anonymity, and model handling into clear layers, the new architecture keeps the responsibilities independent. This means mixing can be switched on or off without changing the learning workflow. Moreover, sparse, symmetric and uniform topologies allow the framework to jointly optimize for scalability, entropy, and convergence. Moreover, to increase scalability further, partial temporal exchange gossiping and quantization were newly introduced. All of this is achieved while keeping the flexibility for model-based algorithms, staying robust under malicious peers, and without introducing messaging patterns.

In summary, the architecture forms an improved basis for the mixed DFL framework. It scales higher, is open for extension, preserves anonymity, and remains robust in dynamic scenarios.

Chapter 5

Implementation

This chapter describes the implementation of the architecture. It builds on the existing codebase of [3]. Inherited workflows and components are described for completeness, even though they are not the contribution of this work. However, they are explicitly mentioned as such to distinguish them from the new contributions of this work.

5.1 Inherited Framework and Scope of Contribution

The framework of [3] is the starting point for this work's implementation. Table 5.1 summarizes which components are inherited and slightly refactored, which were substantially rewritten, and which are new contributions.

Table 5.1: Status of each implementation component relative to the inherited framework [3]. Inherited components are reused without major changes. Refactored components mainly keep their external interface but were substantially rewritten. New components are contributions of this work.

Component	Status	Notes
Sphinx onion / SURB (sphinxmix)	Inherited	Onion packet, SURB construction, decryption
FastAPI manager skeleton	Inherited	Public interface preserved
React dashboard	Refactored	Similar design, simplified implementation
Metrics push and SSE pipeline	Inherited	Mostly used as-is, more efficient
Learner round loop	Refactored	High-level structure kept; fragment-await reworked
Docker deployment	Refactored	FullNodeConfig injection replaces env wiring, optimized multi-stage build for smaller images
ModelHandler split	Refactored	Training, aggregation, partitioning, quantization, registry
MessageStore / ResendManager	Refactored	Payload-hash linking, round binding
SphinxRouter (path generation)	Refactored	Path pool, fixed-length enforcement
Manager routes	Refactored	Public-key, indicators, topology generation
Topology generation	New	Four topologies, indicator-driven
Aggregator interface	New	Injectable; Krum added
ChunkCodec	New	Stochastic 8-bit quantization
Byzantine attack simulator	New	label_flip , gaussian_noise
QUIC transport (aioquic , mTLS)	New	Replaces asyncio TCP framing
PeerSession (overlay liveness)	New	SURB-driven assessment
IndicatorService	New	Pre-launch warnings
FullNodeConfig (Pydantic)	New	Replaces static manager config

5.2 Scalable Transport

The transport layer is replaced with the QUIC protocol. QUIC is implemented with the **aioquic** library, replacing the plain **asyncio** TCP framing of the previous implementation. QUIC provides built-in TLS encryption, multiplexed streams without head-of-line blocking, and long-lived low-footprint connections. Each transport message is sent on a separate unidirectional stream, so fragments can be transmitted concurrently without blocking each other.

Peer authentication is now ensured cryptographically rather than being based on IP addresses as with [3]. Each node holds an RSA-2048 certificate issued and injected by the manager during the deployment. The manager acts as a shared trusted certificate authority and regenerates the certificates for each scenario. Both peers verify the other

peer’s certificate during the mutual TLS handshake. Each peer identity is written in the *Common Name* field of the certificate. Moreover, the transport topology holds now exactly one authenticated connection per neighbor pair, instead of two as with [3].

The long lived and sparse QUIC connection uses less resources but is prone to timing out. A terminated QUIC connection instantly removes the connection record and triggers a background reconnect. Due to the rare communication flow when mixing is deactivated, a periodic keepalive PING is sent to all open QUIC connections.

5.3 Bandwidth-Efficient Model Exchange

5.3.1 Stochastic Quantization

The stochastic quantization as described in Chapter 4 is applied on a fragment-level. For each fragment, the minimum and maximum values define coefficients that map the float weights into the 8-bit integer range of $[0, 255]$. The receiver of a fragment reverts the quantization using the coefficients to recover the original weights. The coefficients are stored per-fragment $v_{\min}$ and $v_{\max}$ together with the uint8 data. This results in a roughly fourfold payload reduction. This, as the amount of data grows with the fragment size while the size of the metadata stays constant. The quantization bit width is itself configurable. The 8-bit stochastic quantizer is the default, while a 32-bit float is only processed by lossless gZip compression. The full encoding is shown in Figure 5.1.

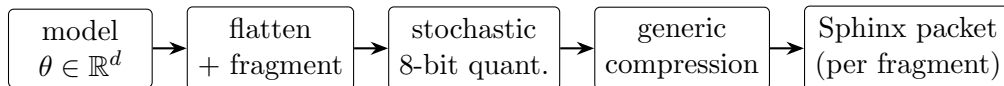


Figure 5.1: Per-round encoding of a model update. The parameter vector is flattened, sliced into fixed-size fragments, quantized to 8 bits per element with stochastic rounding, generically compressed, and wrapped one fragment per Sphinx packet for transmission on the overlay.

5.3.2 Fragment Encoding

A separate class `ChunkCodec` flattens the model weights into a single vector. While the existing logic was reused, it is now majorly refactored. The class then splits the vector into fragments to fit into the limited Sphinx packet body. At the default body length, a quantized fragment carries roughly ten thousand weights together with the quantization metadata. The high-level layout of one fragment is shown in Figure 5.2.

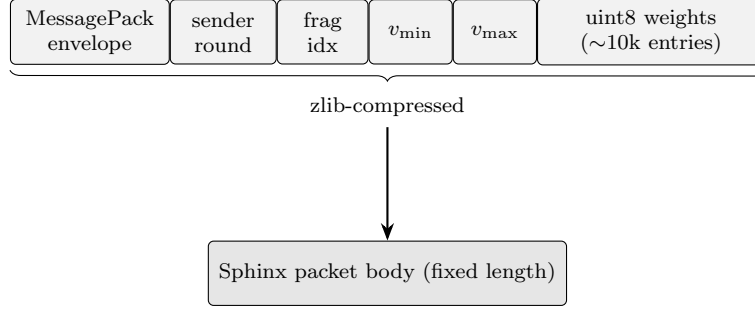


Figure 5.2: High-level layout of a single fragment. The MessagePack envelope, sender round, fragment index, per-fragment $v_{\min}$ and $v_{\max}$, and the uint8 weight payload are zlib-compressed together and placed inside a fixed-length Sphinx packet body.

The receiver collects the flattened fragments without gaining any information about the original sender. Cover packets reuse the same fragment format but carry a field labeling the package type. Unlike a real fragment, a cover packet carries no return path and is never acknowledged. This reduces back-pressure for the receiver and gives the nodes much more control over the congestion of the network and its own outbox. Because the threat model of Chapter 2 treats every message as independent, not sending that acknowledgment does not create a pattern and therefore does not reduce the entropy.

5.3.3 Temporal Partial Exchange

To reduce the number of fragments being exchanged, a `partial_update_ratio` parameter $r \in (0, 1]$ controls the fraction of peers that receive a given fragment at each round. The original sender selects $\max(1, \text{round}(p \cdot r))$ peers uniformly at random from all active peers p . Depending on whether the mixing is activated, the active peers are chosen from the transport or overlay topology. Both, the existing FedAvg and the newly implemented Krum aggregation algorithm already work well with fragment-based partial updates, so the mechanism does not affect aggregation. While the reduced number of fragments does not automatically result in less bandwidth with the mixing activated, it results in shorter rounds which results in reduced total payload transmitted.

5.4 Structured Topologies

5.4.1 Topology Generation

The four topologies introduced in Chapter 4 are generated in the manager using the `networkx` library. The full mesh, ring lattice, and hypercube are implemented by direct library calls. The circulant offset set is selected by enumerating offset combinations under the two conditions described in Chapter 4 and picking the candidate with the smallest diameter.

Independent of the topology, the resulting adjacency is serialized as JSON and injected into each container together with the node’s neighbor list. The same functions are used by the `IndicatorService` during configuration in the front-end. This, so violations such as odd-degree with ring lattice graphs or a hypercube with non-power-of-two n are already reported as being invalid to the user during configuration.

5.4.2 Topology-Aware Sphinx Routing

Due to the newly gained flexibility in the topology configuration compared to the previous full-mesh, the routing and path generation are completely refactored. The new `SphinxRouter` computes a path pool at initialization. At runtime, paths are randomly drawn from the pool, so the path selection does not leak information to a passive observer. When a node selects a path for a message, pool is first filtered to those paths whose first hop currently has a live QUIC link. Due to the deployed topology being static, filtering is much more performant than regenerating the pool. Peers which late-joined the scenario can simply be marked as reachable and are automatically considered for routing in the next path selection. Only the first hop can be filtered this way, as it is the only relay the node directly observes through the QUIC connection. Intermediate nodes are not observed and simply drop messages addressed to unreachable next hops, while still reporting the dropped message to the metrics system.

5.5 Aggregation

In addition to the FedAvg aggregator in the existing framework, this work introduces a fragment-based Krum. This, since the strong anonymity provided by the mixnet breaks any identity-based robust aggregation algorithm.

5.5.1 FedAvg with Partial Updates

While the FedAvg algorithm was already part of the existing framework, it was improved for speed and flexibility. The FedAvg class initializes an accumulator with the flattened local model weights and a per-position counter starting at one. Each received fragment is added at its position range while the corresponding index counters are increased by one. The final flattened model is the per-index average $w_i = \text{acc}_i / \text{count}_i$. This approach makes the algorithm robust to receiving few partial updates as it always has the node’s own model.

5.5.2 Krum Selection

Fragment-based Krum, as newly introduced in this work, provides Byzantine robustness by selecting the single fragment closest to the honest majority. Within each collection of

fragments over the same indexes, the pairwise squared Euclidean distances are computed,

$$d_{ij}^2 = \sum_k (w_{ik} - w_{jk})^2, \quad (5.1)$$

and each fragment's score is the sum of the distances to its $n - f - 2$ nearest neighbors. To ensure that $n \geq 2f + 3$, $f = \lfloor f_{\text{byz}} \cdot n \rfloor$ is clamped. Finally, the minimum-score fragment is selected to be the new global model. When a fragment group holds at most two vectors Krum cannot run. That index group is then averaged instead, as in FedAvg.

Algorithm 1: Fragment-based Krum aggregation

input : n fragment vectors $f_{g,1}, \dots, f_{g,n}$ for each chunk $g = 1, \dots, G$; Byzantine fraction β
output : aggregated model $\hat{w}$

```

1  $m \leftarrow \lfloor \beta n \rfloor$ 
2 for  $g = 1, \dots, G$  do
3   if  $n \leq 2$  then
4      $\hat{w}_g \leftarrow \frac{1}{n} \sum_{i=1}^n f_{g,i}$ 
5   else
6     for  $i = 1, \dots, n$  do
7        $s_i \leftarrow$  sum of the  $n - m - 2$  smallest values of  $\|f_{g,i} - f_{g,j}\|^2$  for  $j \neq i$ 
8      $\hat{w}_g \leftarrow f_{g,i}$  for the  $i$  that minimises  $s_i$ 
9 return  $\hat{w}_1, \dots, \hat{w}_G$  concatenated

```

The degree of a node directly affects the effectiveness of the Krum algorithm. With a topology where nodes have degree- d and a partial-update ratio r , the expected fragment count per position is approximately $d \cdot r$. For Krum to be effective in a Byzantine scenario this must exceed $2f + 3$.

5.5.3 Byzantine Attack Simulation

To assess the robustness of the Krum aggregation algorithm, this work introduces a new attack module. The `attack_type` parameter selects between three attacks. The `label_flip` attack replaces each training label y with $(C - 1) - y$, where C is the class count. Therefore, the Byzantine node trains toward the inverted class. The second attack is `gaussian_noise`, which adds independent noise to each weight before fragmentation. The noise standard deviation is set to $\sigma = \text{attack_noise_sigma} \cdot \text{std}(w)$, so the noise scales with the current model rather than overwriting it. The third attack `sign_flip` simply flips the sign on all weights. To make the attack scenarios reproducible, the noise is drawn from a deterministic per-(node, round) seed. The honest nodes do not know which peers are Byzantine. Their only defense is the Krum aggregation algorithm.

5.6 Reliable Delivery under Anonymity

Compared to the previous framework [3], where each node has a transport connection to all peers in the scenario, reliable delivery is much more complex in the newly introduced sparse topologies with node churn. While this work partially reuses high-level approaches of the previous work, it introduces stability improvements making scenarios more robust when deploying many nodes.

The **MessageStore** and the **ResendManager** were majorly refactored to handle the retransmission. The inherited framework treated each retransmitted SURB as a separate outstanding message. The new implementation links every retransmission of the same payload to one logical delivery through a SHA-256 reverse-index. This, so a single SURB acknowledgment clears every linked retransmission for that payload at once (Figure 5.3).

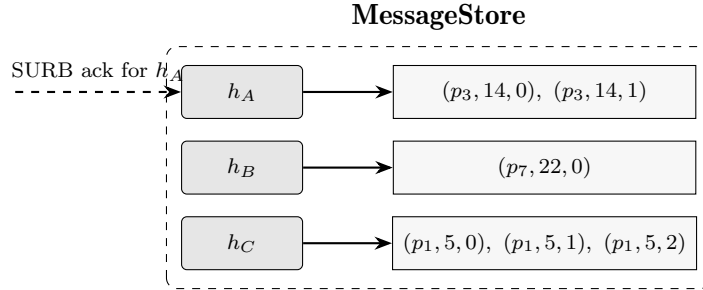


Figure 5.3: Reverse-index inside the **MessageStore**. Each payload hash h_X (SHA-256 prefix) keys the set of retransmissions of the same payload. Each is recorded as a (peer, fragment_idx, attempt) tuple. A single returned SURB acknowledgment for h_A clears every tuple under h_A at once. This, so one successful return is enough, regardless of how many retransmissions are still on the way.

The **ResendManager** periodically rechecks fragments that remain unacknowledged past a threshold and re-queues them through the mixer with a new randomized Sphinx path. Using a new path is required, so that an observer cannot link a resend to the original message. In contrast to [3], the resend decision is managed specifically per peer rather than simply by time. The specific gates for resendings are described in Chapter 4.

In addition to the messages being linked with the new implementation, they now also contain the round in the message. The inherited framework did not send the information about the rounds in the model fragments. While this reduced the payload size, it resulted in destabilizing node progression when increasing the number of nodes. The resend is now made dependent on whether the fragment's round has already received valid fragments to aggregate, so the sender keeps trying to resend fragments for rounds it has already past. This, to help a slower receiver complete its awaiting fragments and reach up with faster nodes in round progression.

Similarly, the round is now also considered for incoming fragments. The **MessageStore** stores arriving fragments by round but only if they were trained in a round greater or equal to the local current round. This lets fragments from a peer that has already advanced beyond the local round survive in the buffer until the local node reaches that round, instead

of being silently dropped. Fragments arriving for a round that has already been closed for aggregation are discarded but still acknowledged.

Compared to the previous implementation, rounds are not awaiting until they received all their sent fragments' SURBs. The fragment-collection phase has two terminal conditions. One is that the node has received the expected number of fragments by overlay-peers and fragments per update. The expected number of fragments is a stochastic expected value by multiplying the partial-update-ratio, the overlay peer count, and the fragments per model. The second terminating mechanism starts after having received 50 % of the expected fragments. After the level has been reached, the node waits for a time window (30 seconds by default) after having received a valid fragment for the current round. The window timer resets with each incoming valid fragment (Figure 5.4). This replaces the hard timeout as implemented by [3] and is much more robust to node churn as well as naturally adapted to the scenario's RTT.

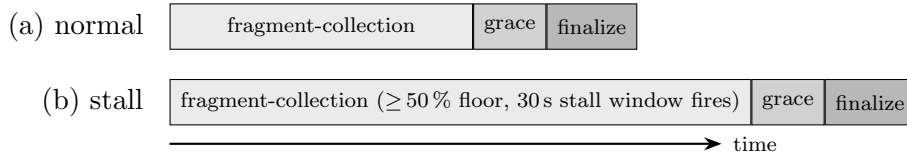


Figure 5.4: Round-phase timing under (a) normal completion and (b) stall escape. The fragment-collection phase ends as soon as the expected fragment count is reached (a) or, once the 50 % completeness floor is met, when the 30-second stall window fires without further intake (b). A small 5-second grace window is awaited before the round finalizes.

5.7 Adaptive Peer Liveness and Resend Control

Overlay liveness is tracked per peer but separately from the QUIC transport state. The `PeerSession` class accumulates the pending fragments addressed to that peer and has one of three states: *pending*, *reachable*, or *unreachable*. A session starts *pending*. The first SURB acknowledgment from the peer proves that it is reachable and marks it as *reachable*. A later acknowledgment also recovers a peer back out of *unreachable*. The linkability state is linked to returned SURB acknowledgments rather than on the QUIC link since a node can be disconnected on the transport level while being reachable over a multi-hop mixed path. If the mixed path is broken and cannot be reached while maintaining the entropy, it can still have an active transport connection and be used for sending out packets. The design is explained in more detail in Chapter 4, Section 4.6.

The unreachable state is set by the `ResendManager` through `record_resend_exhausted`. Once a fragment has used its bounded (default one) resend count and has not received any acknowledgment since the issued resend, it is marked unreachable. The state change keeps the pending fragments. This retain and recover behavior allows nodes to become reachable again without having discarded all information meant for them by the other nodes. Fragments are sent to *pending* and *reachable* peers and only *unreachable* ones are excluded. The expected fragments being awaited for aggregation is calculated from that

same realized send-set. This, so a peer that flips *unreachable* mid-round drops out of the expected fragment count and the round finalizes instead of stalling.

For path selection, the first hop is filtered by the transport link state. So outgoing traffic is routed around a dead direct transport neighbor while keeping the path length and destination unchanged. A terminated QUIC connection starts a per-peer background reconnect loop, so a short mid-round drop is repaired immediately.

The cover traffic is generated continuously and placed into a queue to not delay the constant mixing output. The mixer pops a pre-generated cover packet from a stash whenever its outbox is empty. Cover messages are fire-and-forget since they are not tracked and not acknowledged. This prevents the network from being overwhelmed with covers and improving the flexibility of nodes to prioritize meaningful traffic in their local queue.

To sense nodes being reachable again, once per round every *unreachable* peer is sent one probe packet that carries, unlike cover messages, a SURB. Its returned acknowledgment is what changes the peer back to *reachable*. Algorithm 2 states the combined bring-up, acknowledgment handling, resend scheduling, first-hop selection and release filter, reconnect, and artificially managed node churn.

Algorithm 2: Adaptive per-peer liveness, ack-gated resend, and self-controlled churn

input : graph neighbors N ; own join/exit rounds $r_{\text{join}}, r_{\text{exit}}$; per-peer assessment $\text{ST}(p) \in \{\text{unconfirmed}, \text{live}, \text{unreachable}\}$; resend window τ ; resend budget K ; mixer rate μ

- 1 **Bring-up (once, before the round loop)**
- 2 wait until every $p \in N$ whose join round has passed has a transport link, or until timeout, then proceed
- 3 **On SURB acknowledgment from p**
- 4 mark the payload's fragments acknowledged; $\text{ST}(p) \leftarrow \text{live}$; stamp the last-ack time
- 5 **On resend scan of a stale unacknowledged fragment f for p**
- 6 **if p acknowledged within τ and no acknowledgment since f 's last send and the round had intake then**
- 7 **if** *retry count* $< K$ **then** resend f ; increment its retry count
- 8 **else** $\text{ST}(p) \leftarrow \text{unreachable}$ (keep f pending)
- 9 **else** defer f (budget untouched, no state change)
- 10 **On path selection to target t (own outbound)**
- 11 restrict the equal-length path pool for t to entries whose first hop has a live link; draw one uniformly
- 12 **if no such entry then** drop and count (length and target are never relaxed)
- 13 **On mixer release with first hop p**
- 14 **if the link to p is live then** forward
- 15 **else** drop silently and count; continue releasing at μ
- 16 **On connection loss to an owned link p**
- 17 re-dial p with bounded backoff until restored, independent of rounds
- 18 **Round r start**
- 19 **if $r < r_{\text{join}}$ then** idle
- 20 **if $r > r_{\text{exit}}$ then** terminate

5.8 Runtime Performance

Each node runs on a single `asyncio` event loop that coordinates the QUIC transport callbacks, the mixer send loop, retransmission, and the round-level training loop. CPU-bound work that would otherwise stall this loop is offloaded to the worker threads at five locations: Sphinx forward and reply construction, Sphinx packet decryption on incoming messages, training-steps, accuracy evaluation, and model aggregation.

The mixer's μ output rate holds only if send events are triggered on schedule. The send loop compares elapsed wall time against the next scheduled sending event. If less wall clock time has passed than the μ , the algorithm simply waits. If more time elapsed than μ , the `drift_resets` metric is increased and the logs show a warning in the observable shell. Without this guard, a single training spike on a busy host would silently desynchronize the traffic-rate from the configured rate.

Two further technical configurations keep the training load separated from the main loop. PyTorch’s worker running training and evaluation is moved to the Linux `SCHED_IDLE` scheduling policy so it gives back the CPU whenever the mixer needs to act. This helps keeping the event loop responsive on the small Docker hosts, where the training would otherwise starve the mixing.

Shutdown is now also controlled. The manager provides each node’s `current_round` on the API endpoint `/nodes/status` so exiting the scenario can be awaited by the nodes. This prevents slower nodes from losing the mixnet too early and not being able to finish the last round.

5.9 Orchestration and Configurability

A typed Pydantic model, `FullNodeConfig`, replaces the previous static configuration. The full node configuration is serialized to an environment dictionary in the manager and injected at container startup, so that adding or changing an experiment parameter no longer requires source-code edits on the node side.

The manager itself is a FastAPI application that orchestrates the experiments through the Docker SDK. At startup, it generates the Sphinx private keys from elliptic-curve keypairs and the TLS chain from a self-signed RSA-2048 CA. All keys and certificates are issued freshly per-node for every scenario. Each node runs in a separate Docker container with read-only volume mounts for the secrets and the application code.

The exit and join scheduling is deterministic. Late-joining nodes are assigned to round $\lceil 0.3 \cdot R \rceil$ and early-exiting nodes to round $\lfloor 0.7 \cdot R \rfloor$, where R is the total round count. Each node receives only its own schedule and acts on it locally without the other nodes knowing about it. Followingly, all other nodes must act to the event through the liveness feature. The manager also caches the latest values per node and streams them to the React dashboard through server-sent events. At the end of an experiment, the accumulated metrics are exported to a timestamped CSV.

During configuration, before any container starts, the `IndicatorService` in the manager precomputes the scenario from the configuration. The `IndicatorService` is used to test configurations at config-time and to predict various KPIs of the possible scenario. The indicators are exposed through an HTTP endpoint so that the dashboard can warn against invalid settings before the experiment launches.

5.10 Summary

The transport layer is replaced with QUIC, adding TLS, multiplexed streams, and cryptographic peer authentication through certificates issued by the manager. The bandwidth is reduced through stochastic 8-bit quantization of the fragmented model weights and a partial update ratio gossip algorithm that sends each fragment to only a random subset of

Configuration

Ready

Learning

Model
LeNet-5 (~62K params)

Dataset
MNIST (handwritten digits)

Training Rounds
10

Batch Size
64

Torch Threads
1

Dirichlet Alpha
10

Aggregation
FedAvg

Quantization
8-bit (~4x smaller)

Attack Type
None

Network

Number of Nodes
5

Topology
Full Mesh (all-to-all)

Late-Joining Nodes
0

Early-Exiting Nodes
0

Mixnet

☒ Enable Mixnet

Mix Interval (s)
0.1

Max Hops
2

Outbox Size
10

Partial Update Ratio
1

Figure 5.5: The form at the top sets the configuration and the topology graph previews the mesh.

Scenario Indicators			
TOPOLOGY		ANONYMITY	
Transport connections	4		
Transport diameter	1	Path entropy	6.8 bits
Overlay degree	4	Relay entropy	2.7 bits
		Anonymity set	4
		COMMUNICATION	
		Model params	62K
		Sphinx payload	10.2 KB
		Sphinx packet	10.5 KB
		Fragments	7
		Msgs / node / round	28
		Bytes / round	1.4 MB
		Partial update ratio	100%
		MIXNET TRAFFIC	
		Send rate / node	10 pkt/s
		Bandwidth / node	104.7 KB/s
		Mix cycle time	1.0 s
		RTT lower bound	4.0 s
		Packets / node / round	30
		Total bytes / round	3.1 MB

Figure 5.6: The `IndicatorService` panel precomputes the scenario KPIs before any container starts: transport connections and diameter, overlay degree, path and relay entropy, anonymity-set size, fragments and bytes per round, and the mixnet send rate.



Figure 5.7: The node table and the metric charts below stream live values over server-sent events once the run is underway.

the peers. Moreover, new uniform sparse topologies are added for a good balance between entropy and scalability. The refactored **SphinxRouter** now draws random paths from a precomputed pool and filters nodes' states for efficient path selection.

To evaluate attacks leveraging the high level of anonymity, the framework has now a fragment-based Krum aggregator and a Byzantine attack simulator. Moreover, various improvements regarding the reliable delivery and efficiency are added. Examples are linking retransmissions through a reverse index, embedding round information in messages, and replacing the old hard timeout with an adaptive timeout window that adjusts to network conditions. The new peer liveness is tracked per node via returned acknowledgments rather than by the transport state as with the previous framework. Moreover, new probe packets are introduced to probe unreachable peers for their recovery. Improvements regarding the configurability include a new injected configuration to each node and supporting endpoints on the manager to catch invalid configurations before deployment.

Chapter 6

Evaluation

This chapter evaluates the framework against the two research questions. RQ1 asks how payload handling as well as spatial and temporal model exchange affect the scalability and the convergence in anonymous DFL under limited per-node connectivity. RQ2 asks how those same dimensions can be jointly optimized to improve the scalability while maintaining sufficient anonymity in peer-based mixed DFL.

The following chapter is organized around the three main boundaries of a scalable peer-based and mixed DFL scenario. These are the size of the network that the framework supports, the entropy guarantee that justifies the mixnet, and the price that the mixing layer costs to deliver that anonymity.

6.1 Setup

Unless specified otherwise, the experiments use the baseline scenario summarized in the following Table 6.1. Differences from this baseline are stated in the individual experiments.

The experiments were run on an *Ubuntu* server with an *AMD EPYC 7502P* processor, 50 visible cores, 94 GiB memory, and 8 GiB swap. No dedicated GPU was used, and each node was limited to one core for improving the repeatability. Scenarios above 50 nodes share the maximum available cores.

Each experiment was run five times with the identical seeds. Figures and numerical results are the average over all five experiments. The bands reported in the figures are the 5/95-th percentiles across all nodes for the parameter sweeps, and the min/max across nodes for the single-run scenarios, like the node churn.

Parameter	Value
N	32
μ	0.1 s
K	2
O	100
Rounds	10
Sphinx body	10240 bytes
σ	0.001
Aggregation	FedAvg
Dataset	MNIST, Dirichlet $\alpha = 10$
Model	LeNet-5
Mixing	activated
Quantization	8 bit
Topology	Circulant
Transport degree	4
Overlay degree	8
Partial-update-ratio	1

Table 6.1: Baseline experimental scenario.

6.2 Scalability

This section evaluates the contribution of the framework over its previous version by [3]. The framework runs anonymous DFL at the $N=32$ baseline of Table 6.1 and up to an $N=64$ maximum. The question is whether the learning, the per-round communication overhead, the CPU-time, and the cost of a heavier model all stay inside practical limits.

6.2.1 Convergence at scale

The following experiment ran the baseline configuration at $N \in \{16, 32, 64\}$ on MNIST and Fashion-MNIST, ten rounds. In addition, there is a small- N anchor at $N=6$ on full mesh (MNIST only) for comparison to the previous version of [3]. This experiment isolates whether the learning works reliably when increasing the node count and arrange the nodes in a sparse topology.

Figure 6.1 shows the three circulant sizes converging in the same shape on both datasets, a steep climb over the first three rounds that converge towards a plateau by rounds eight to ten. Round-ten accuracy declines with network size. This, as with an increase in N each peer receives a thinner non-overlapping partition of the dataset, so one local step improves the accuracy less and the first FedAvg averages over noisier local models. The datasets show a difference in the accuracy spread. The spread shows a 12.0 pp spread on MNIST and a tighter 7.3 pp on the harder Fashion-MNIST. The per- N values are reported in Table 6.2.

The average local validation accuracy (dashed line) sits below the average aggregated one at every round. This shows DFL working as intended, as the aggregated models

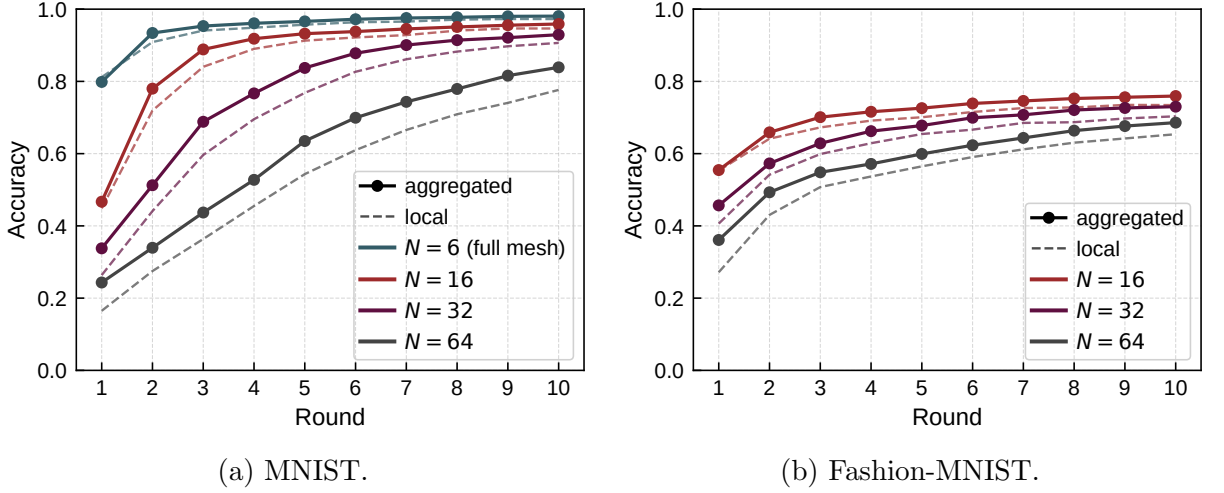


Figure 6.1: Accuracy convergence across N at the baseline. Solid lines with markers show aggregated accuracy after each round’s combined update; dashed lines show local accuracy after the local training step, before aggregation. Lines are the mean across nodes. The $N=6$ (MNIST panel) is run on full mesh as a small- N anchor, $N \in \{16, 32, 64\}$ are circulant.

	MNIST		Fashion-MNIST	
N	round 1	round 10	round 1	round 10
6 (full mesh)	0.799	0.981	–	–
16	0.467	0.959	0.555	0.759
32	0.338	0.929	0.457	0.730
64	0.243	0.839	0.361	0.686

Table 6.2: Aggregated accuracy at round one and round ten across N on both datasets, at the baseline of Table 6.1. $N=6$ is the full-mesh small- N anchor (MNIST only); $N \in \{16, 32, 64\}$ are circulant. Mean across nodes; five runs per point at $N \in \{6, 64\}$, single run at $N \in \{16, 32\}$.

outperform the average local model.

Table 6.2 shows that the accuracy of the previous work’s $N=6$ full-mesh shows a strictly better accuracy over all sparse circulant topologies. This reflects that a small- N results in both the larger per-peer training-set partition and the complete aggregation of all models in each round. From the circulant’s $N=32$ baseline to the $N=64$ ceiling, the final accuracy at round 10 differs only minimally over the full-mesh. The curves at the largest N reach the same plateau region the baseline does. Scaling the node count does not collapse the learning signal for the evaluated sparse circulant topology.

6.2.2 Communication volume scaling

The communication overhead of fragment-based mixed DFL is the second limitation the evaluation assesses. The number of fragments per model follows the fixed Sphinx packet

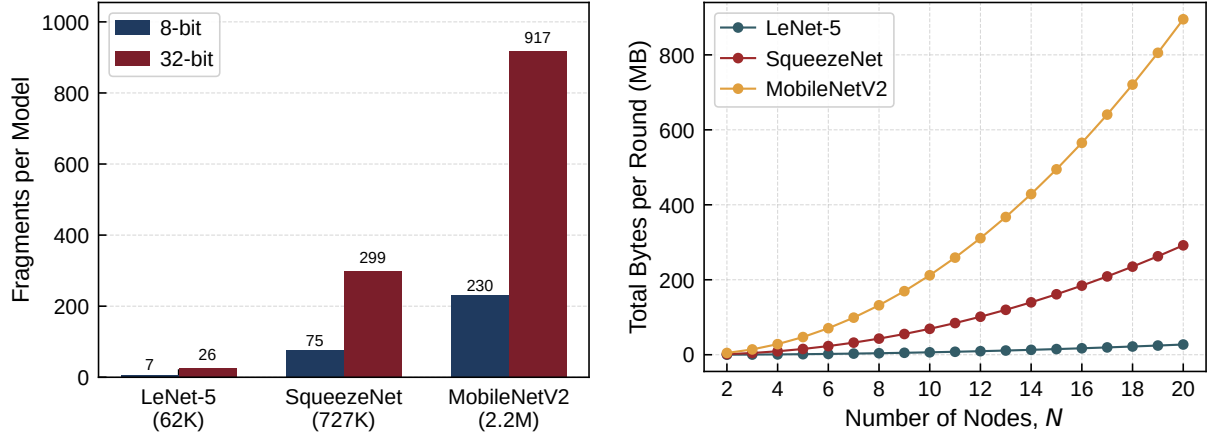
size. The quantization implementation reserves meta=8 bytes per fragment for the per-fragment scaling and one byte per parameter (uint8), while the unquantized payload uses four bytes per parameter (float32). Fragments per model are then $n_{\text{frag}} = \lceil P/n_{\text{frag_capacity}} \rceil$ for P parameters, and per-round bytes scale as $n_{\text{frag}} \cdot \text{round}(p \cdot r) \cdot N \cdot B$. p is the overlay-peer count, r the partial-update ratio, and B the byte-per-fragment constant.

Figure 6.2a shows the fragment count for LeNet-5, SqueezeNet, and MobileNetV2 under both the 8-bit and 32-bit encodings. The quantization saving is visible in the fragment count a given peer has to exchange with peers. At LeNet-5 the fragment count drops from 26 at 32-bit to 7 at 8-bit. This is a $\div 3.71$ ratio that approaches the ideal $\div 4$ of the quantization alone.

Figure 6.2b shows a sweep of a theoretical full-mesh to assess the per-round bytes over N at the baseline of $K=2$ hops. LeNet-5 stays in the tens of megabytes per round across the full $N \in \{6, \dots, 64\}$ range. However, SqueezeNet sits already roughly ten times above LeNet-5 across the same range and MobileNetV2 approaches a gigabyte per round at $N=64$. The more-than-linear increase in exchanged bytes with an increase in model size and N are a clear limitation for practical and highly scaled scenarios.

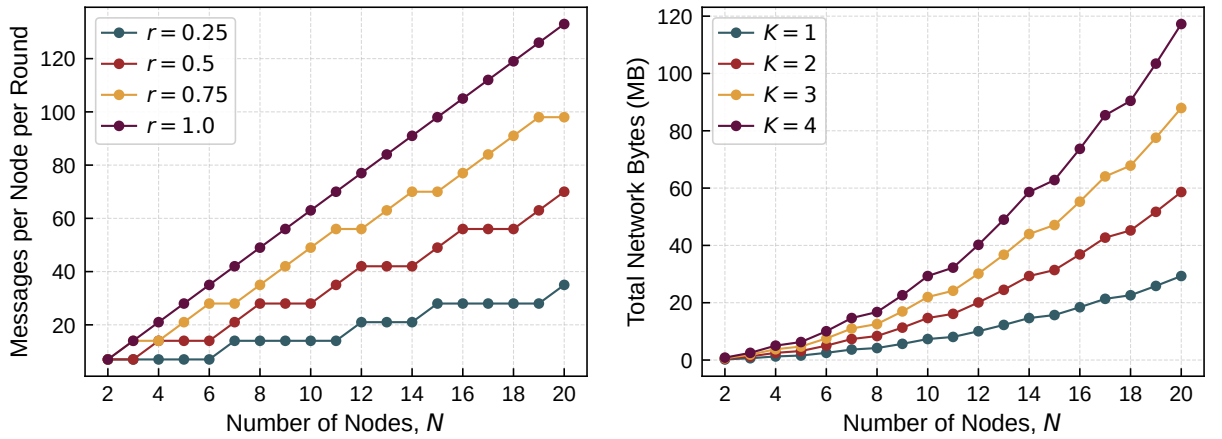
Figure 6.3a sweeps the partial-update ratio $r \in \{0.25, 0.5, 0.75, 1.0\}$ over N . The fragment volume scales linearly in $\text{round}(p \cdot r)$. The visible steps in the curve are the result of the rounding to whole messages. The per-node bandwidth is fixed at $1/\mu$ due to the constant-rate stream required by the mixer. So the partial-update ratio shrinks the time it takes to exchange meaningful messages but does not reduce bandwidth. Figure 6.3b shows the aggregated per-node packet counts summed to the network level.

The total bytes scale linearly in both N and K . While the partial-update ratio r can be used to linearly reduce the number of fragments to be exchange, increasing k results in an increase of the the number of bytes more-than-linearly. Communication volume is therefore more bounded by the model and the number of hops. The per-round budget grows only linearly in N while network parameter K and a larger model are more limiting factors for scalability.



(a) Fragments per model under the 8-bit and 32-bit encodings. (b) Total bytes per round as N grows, by model.

Figure 6.2: Communication volume by model class.



(a) Messages per node per round vs. N for several partial-update ratios. (b) Total network bytes per round vs. N for several K , LeNet-5.

Figure 6.3: Per-node and aggregate communication volume as N grows.

6.2.3 Resource scaling by degree

Similar to k , N , and the model size, this subsection assesses the effect of an increase in node degree. Figures 6.4 and 6.5 sweep node degree $d \in \{4, 16, 32, 48, 62\}$ at $N=64$, $n_{\text{rounds}}=5$, while excluding the warm-up round. The figures show the absolute mean CPU time per round and the mean phase share of CPU time during the rounds.

The phase of collecting fragments for aggregation clearly dominates the spent resources over all degree configurations. Concurrent to all phases is the continuous mixing process, routing packets and generating cover traffic. The evaluation uses more resources than the training due to the validation set being constant, while the training set is non-overlapping between the nodes. The mean CPU time is relatively low at $d=4$ but quickly stagnates above 30 seconds for higher degrees (Figure 6.4b). The relative CPU-time also shows a plateau being reached already at low degree (Figure 6.4a). Table 6.3 shows the per-degree CPU and memory during the same scenario.

While CPU-time plateaus at higher degrees does memory climb linearly with degree. This follows the increase in information to be held for more connections and the increase in peers to cache and exchange models. Similarly, other implemented techniques increase memory requirements with an increase in degree due to tracking the larger in-flight packet set for resendings and SURBs.

The circulant and ring lattice are similar to each other across the sweep except at $d=16$, where the circulant has a distinct CPU peak the ring lattice never shows. This is not a fluctuation and can be reproduced on both replicates and on SqueezeNet. The reason for the outlier is reachability. Because the circulant offsets minimise diameter, a node’s two-hop set already covers the whole graph at $d=16$. In contrast the ring lattice still reaches only half of it (Figure 6.6). So, $d=16$ is the smallest degree reaching everyone for the circulant. Therefore, it is also where each link carries the most of the topology’s traffic, 3.9 reachable overlay peers per connection against 1.0 at the densest degree.

The send rate is fixed by μ , so the QUIC transport moves the same number of fragments at every degree, but at $d=16$ it sends them over the fewest connections and its per-packet send and acknowledgement work grows with the load each connection carries. Per-node CPU therefore peaks at $d=16$ and falls toward its floor as a higher degree spreads the traffic over more links, while memory climbs the other way as each added connection is tracked.

Per-node CPU time is dominated by the broadcast and collect phase at every degree the corpus runs, with training only being a small fraction of it. Away from the $d=16$ peak the two topologies result in very similar metrics. So outside this specific configuration, the topology choice is more determined by convergence and anonymity rather than resources.

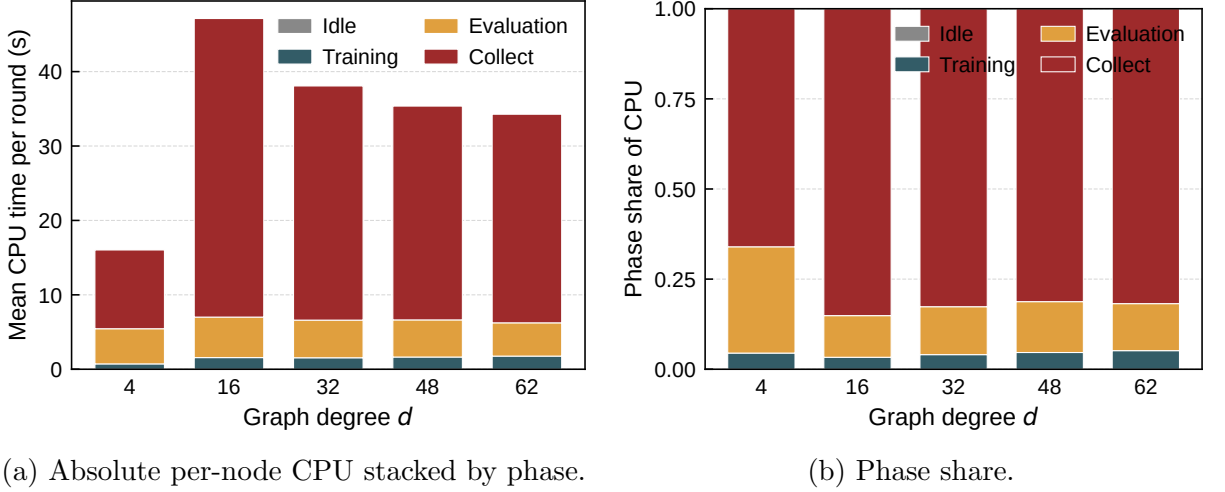


Figure 6.4: Per-node CPU broken down by phase across graph degree $d \in \{4, 16, 32, 48, 62\}$ at $N=64$ on MNIST, circulant topology, $n_{\text{rounds}}=5$, averaged over all nodes and the five runs with the warm-up round excluded. Left: absolute total CPU per node, stacked by phase. Right: the same data normalised to phase shares.

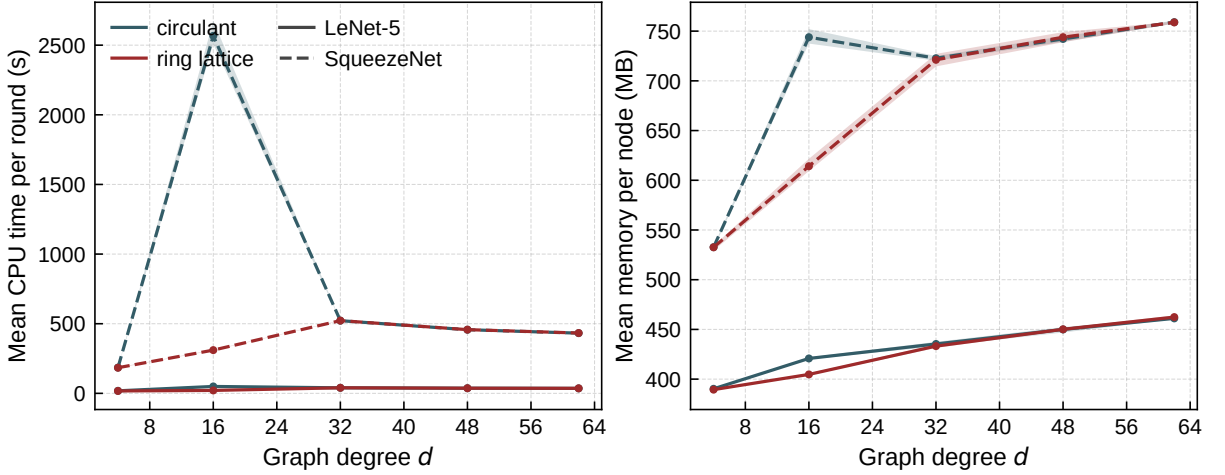


Figure 6.5: Mean per-node CPU time per round (s) and mean memory (MB) across graph degree $d \in \{4, 16, 32, 48, 62\}$ at $N=64$, $\mu=0.1$ s, $K=2$. Circulant and ring lattice are plotted as lines, the mean over five runs per point.

Degree d	CPU/round (s)	Memory (MB)
4	18.2	390.3
16	49.6	420.8
32	39.8	435.4
48	36.9	450.0
62	35.7	461.1

Table 6.3: Per-node CPU time per round and mean memory for the circulant topology across graph degree d , LeNet-5 at $N=64$, $\mu=0.1$ s, $K=2$, mean over five runs per point. The CPU peak at $d=16$ is the circulant run.

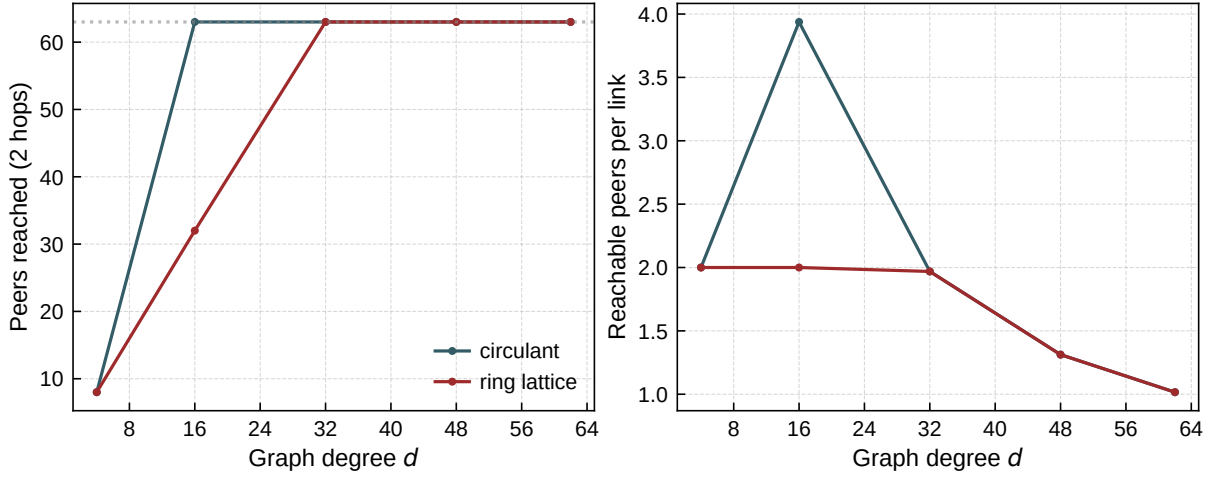


Figure 6.6: Two-hop reachable set and its per-link ratio across graph degree d at $N=64$, circulant versus ring lattice. Left: peers reached within two hops, the set a node aggregates over and the anonymity set its fragments are hidden in. The dotted line marks the whole graph $N-1$. Right: reachable peers per connection (reach/d), the share of the graph’s traffic each link carries. The diameter-minimising circulant reaches the whole overlay graph at $d=16$ and peaks in per-link concentration there, while the ring lattice reaches half the graph at $d=16$ and saturates at $d=32$. Both panels are deterministic from the topology generator.

6.2.4 Model-size scaling

To isolate the model size from the topology and degree, the degree sweep is repeated with SqueezeNet ($\sim 727\text{K}$ parameters), to compare against the LeNet-5 ($\sim 60\text{K}$) baseline at $N=64$. Figure 6.7 compares the two architectures across the degree sweep. At the matched dense degrees the mean round duration is about 158 s on LeNet-5 against about 1024 s on SqueezeNet. This is a $6.5\times$ larger average round wall-clock time for the same network and topology. Time-averaged memory shows about $1.66\times$ the LeNet-5 figure at the same degrees. Similarly, SqueezeNet shows a $13\times$ larger demand in absolute CPU-time per round compared to LeNet-5.

Figure 6.8 shows the per-node CPU time split by phase for SqueezeNet. In contrast to LeNet-5, are training and evaluation more resource demanding, together accounting for about 50% of CPU at $d \geq 16$ rather than about 17%. Fragment collection therefore converges to 50% on SqueezeNet against the roughly 82% LeNet-5 plateau.

Overall, this indicates that model-size heavily influences the round wall-clock time and the total CPU-time.

In summary, learning slows down with increasing N but still shows monotone accuracy improvement. Quantization and partial-update reduce the number of fragments to be sent linearly, while the number of hops grow it more than linear. Per-node CPU-time heavily depends on the model size, much less on the degree. In contrast, memory increases with degree while the model-size determines the offset. The round wall-clock time similarly has a large offset depending on model size but plateaus with increase in degree. This, due to

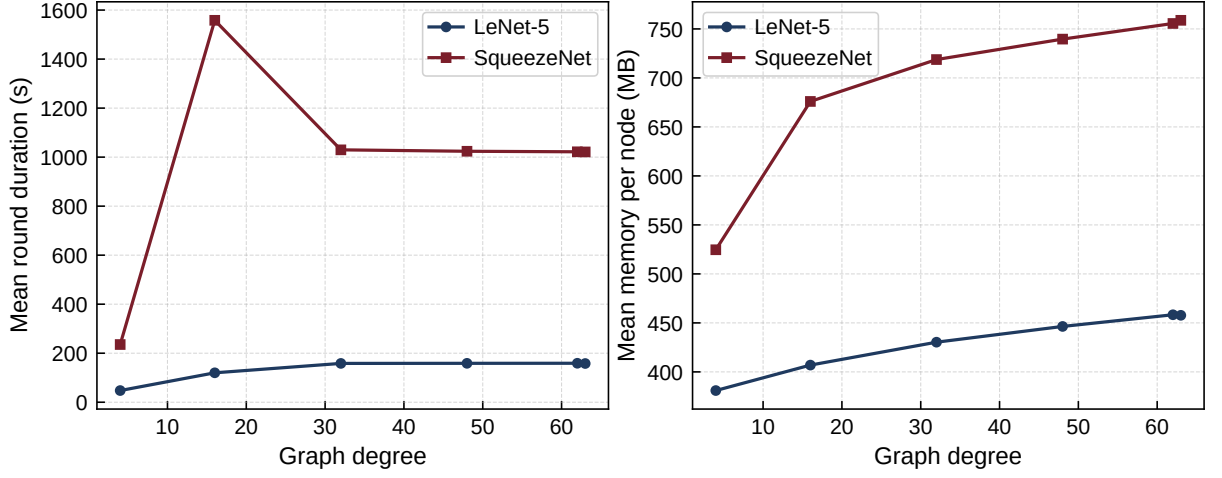


Figure 6.7: LeNet-5 versus SqueezeNet mean round duration and memory across the degree sweep at $N=64$, $\mu=0.1$ s, $K=2$, $O=100$, MNIST, 8-bit quantization. SqueezeNet spans $d \in \{4, 16, 32, 48, 62\}$ and the full mesh. LeNet-5 covers the dense degrees $\{32, 48, 62\}$ and the full mesh. Each marker is the per-degree mean over the available topologies and five runs each.

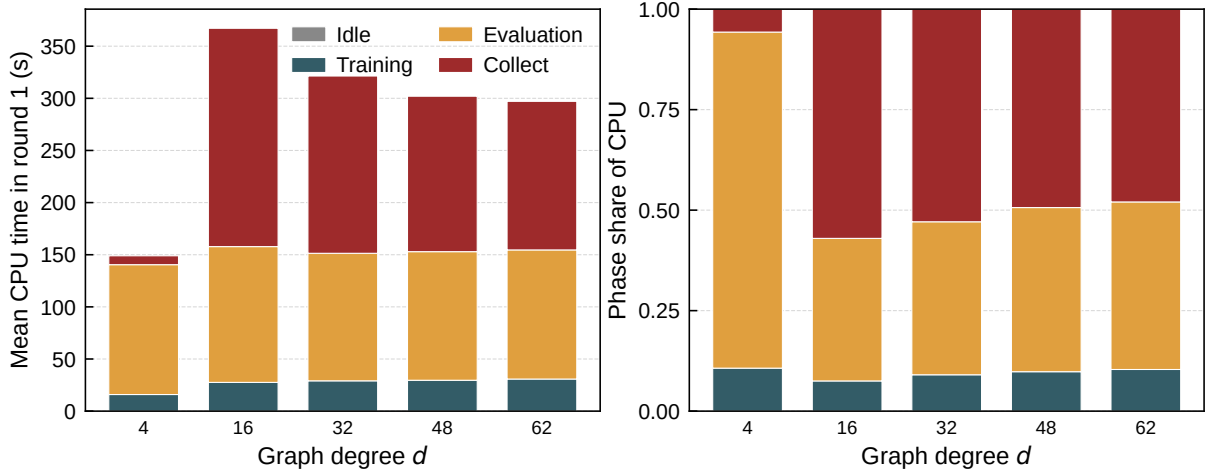


Figure 6.8: SqueezeNet per-node CPU broken down by phase across circulant degrees $\{4, 16, 32, 48, 62\}$ at $N=64$. Layout matches Figure 6.4.

the reachable nodes being limited. the fragment collection is compared to training an is an enormous investment in wall-clock time and CPU-time.

6.3 Anonymity

Anonymity is the fundamental constraint that improving scalability has. This framework's threat model is a global passive adversary that observes every link. The observer tries to find out the original sender and final receiver from packet timing and routing. The mixnet defends against it by routing each packet of uniform size through K hops, mixing it at each relay in an outbox of capacity O , and maintaining a constant output rate with some jitter.

The entropy of the mixnet is bounded in two ways. First, by the path entropy an the end-to-end overlay path can offer and secondly by the anonymity set of the overlay topology. Any global passive adversary would assess both. This section is about the cost of anonymity.

6.3.1 Relay and path entropy

The first anonymity metric is the relay entropy at a single mix. With a probability of $p_{\text{fwd}} = (K-1)/(KO)$ across O output slots and the probability of a message having reached its target $p_{\text{abs}} = 1/K$. Shannon's formula gives $H_{\text{relay}} = -\sum_x p(x) \log_2 p(x)$. Figure 6.9a shows that Shannon entropy saturates quickly. Beyond $O \approx 50$, the entropy gain is small and the curve flattens above $K=2$. The shuffled outbox therefore provides a strong baseline entropy but has diminishing returns past a moderate size.

The second quantity is the path entropy from sender to receiver,

$$H_{\text{path}} = \log_2 \left(\sum_{k=1}^K O^k \right), \quad (6.1)$$

the uncertainty that a global observer faces in linking a sender to a receiver across a path of K hops with a shuffled outbox of O at each hop. Figure 6.9b sweeps $K \in \{1, \dots, 5\}$ and $O \in [5, 200]$. The dominant term in the sum is O^K , so the expression grows logarithmically in O and approximately linearly in K . At $O=100$, the analytical entropy is 6.64 bits at $K=1$, 13.30 bits at $K=2$, 19.95 bits at $K=3$, 26.59 bits at $K=4$, and 33.23 bits at $K=5$, an approximate 6.6-bit-per-hop step. Each unit of K therefore buys more entropy than each unit of O . However, this is payed with a higher latency cost, as visible in Section 6.4.1.

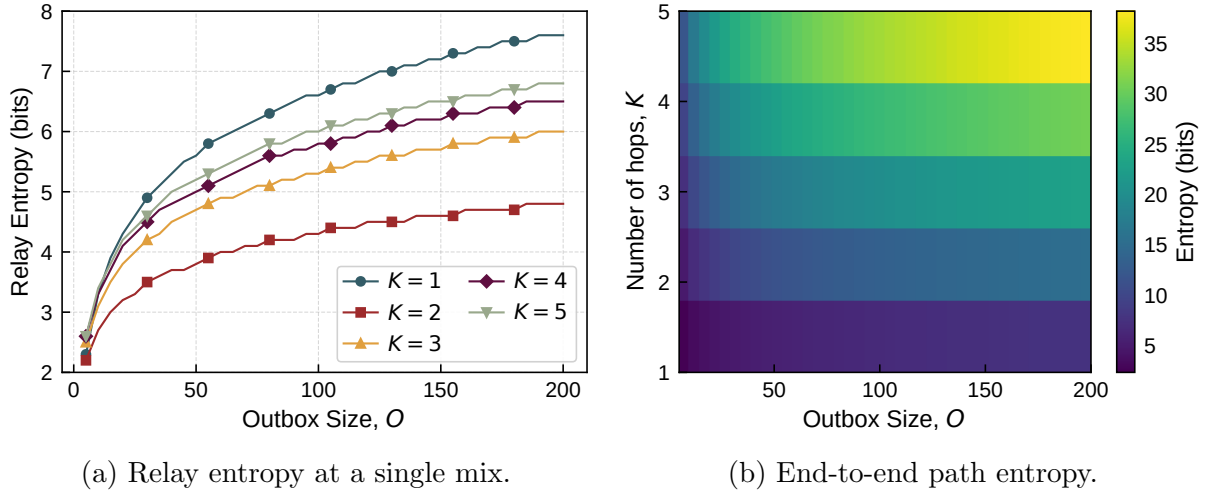


Figure 6.9: Relay entropy and end-to-end overlay path entropy as the outbox size O varies, one line per hop count K .

6.3.2 Anonymity set by topology and N

The path-entropy does depend on K and O , but the anonymity set depends on the topology and N . Figure 6.10 shows the per-node anonymity-set size for different topologies at $K=2$ and degree of four. The full mesh holds at $N-1$ for any K , since every peer is one hop away. The two structured topologies fall below it once N outgrows the K -hop reachable nodes at the configured degree. This means not all nodes are reachable anymore for every topology other than the full mesh.

At the baseline $N=32$, $K=2$, $d=4$, the set of reachable peers is 8 for the circulant against $N-1=31$ on the full mesh. This is a $3.9\times$ increase in the anonymity set. The ring lattice, not being optimized for minimal diameter, has a higher or equal anonymity set at every N compared to the circulant. This is already a visible trade off between the minimal diameter compared to the anonymity set with the same number of nodes and degree.

Figure 6.6 evaluates the anonymity set with a sweep over the degree at $N=64$. While the anonymity set of the circulant is slightly smaller than the ring lattice's at $d=4$, the circulant reaches the whole graph already at $d=16$ against $d=32$ for the ring lattice. However, this anonymity set at low degree shows a clear peak in the per-node CPU-time in Section 6.2.3. This, because reaching everyone through the fewest links concentrates the most traffic on each connection. Increasing the degree slightly for the circulant reduces the resource usage while keeping the anonymity set and Shannon entropy at the same level.

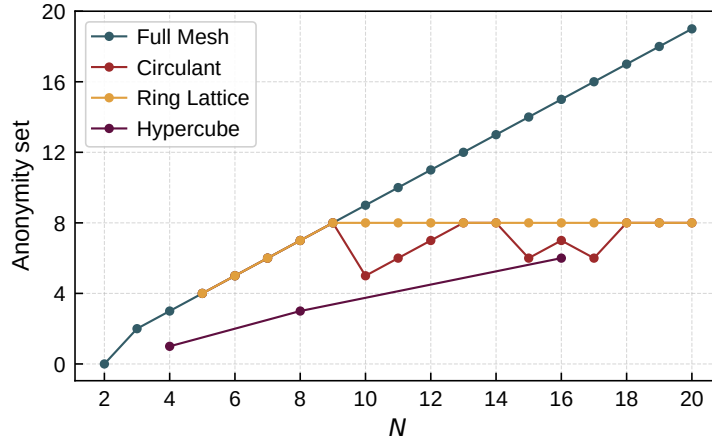


Figure 6.10: Anonymity-set size as a function of N for each topology of Section 5.4.1, $K=2$, target degree four. The full-mesh anonymity set equals $N-1$ at any K . The structured topologies fall below $N-1$ once N exceeds the K -hop reach at the configured degree.

6.3.3 Cover traffic

The constant-rate output of a mixing node uses cover messages have have a constant rate of μ in the case of having not sufficient meaningful messages to shuffle. This, so any external observer cannot gain information from the output rate. The cover fraction measures how much of the mixed messages were generated by the node itself compared to meaningful messages. Other node's covers cannot be distinguished from meaningful packets and are treated as meaningful messages as they have to be relayed.

Across K at the baseline $\mu=0.1$ s, $N=32$, $O=100$ the fraction decreases monotonically, from $K=1$ to 0.79 at the baseline $K=2$ and a ~ 0.51 plateau from $K=4$ on (Figure 6.11). The decrease is driven by the amount of meaningful messages in the network. Each added hop increases the exchange-peer set on the circulant topology and increases the per-node real-fragment count as more fragments are being sent. This effect decreases the fraction until the K -hop reachability plateaus at $K=4$ where all nodes are in the reachable set. Similarly, the constant sending rate μ has a strong effect on the number of cover messages required to be sent which is visible in Section 6.4.1.

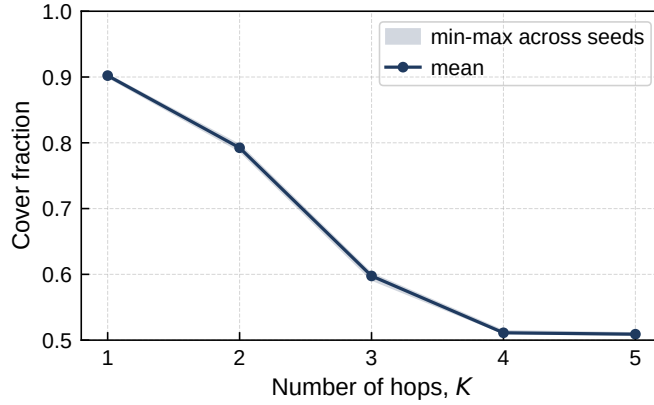


Figure 6.11: Cover fraction across the K sweep at $\mu=0.1$ s, $N=32$, $O=100$, mean over five runs, $K \in \{1, \dots, 5\}$.

6.3.4 The cost of mixing

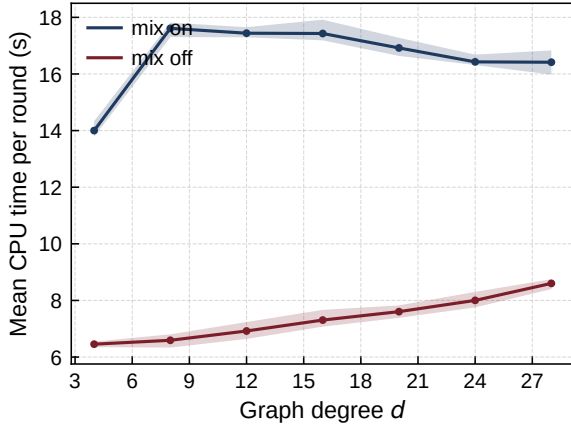
The constant-rate stream is required by the mixnet for breaking patterns. However, it adds CPU work on every node and when being increased, it increases the RTT and therefore the per-round wall-clock time. To isolate this trade-off, the following experiment shows the the baseline configuration being run twice at every swept degree, once with `mix_enabled` and once without, at $N=32$, $\mu=0.1$ s, $K=2$, $O=100$, $n_{\text{rounds}}=5$.

The two configurations are not comparable in their broadcast and aggregation workload. With mixing off, an update reaches only the direct neighbors by degree, while the mixed overlay reaches the K -hop set. Similarly, the mean CPU time per node normalized over round time is more of an indicator and not strictly comparable.

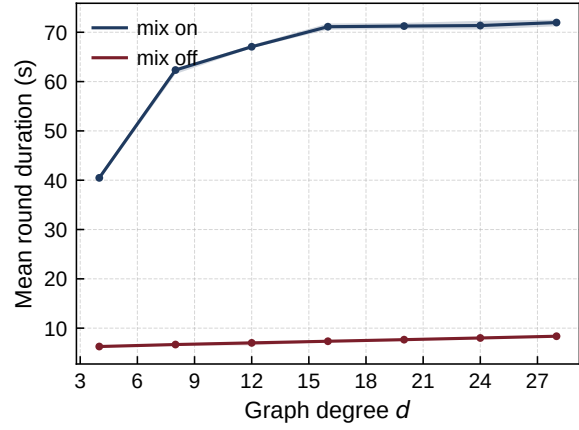
The per-round CPU panel (Figure 6.12a) compares mixing being activated and deactivated. For the mean CPU-time per round normalized over time, mixing shows using +7 to +11 s per round more on CPU-time. The strong incline and peak near $d=8$ is due to the degree resulting in many more nodes being overlay-reachable for model exchange.

Figure 6.12b comparing the wall-clock time shows main difference for running experiments from the user's perspective. Mix-on rounds run approximately ten times longer than mix-off rounds at every degree. This due to the high overhead in exchanging models compared to simple transport links. Similarly, the CPU-time delta shows not only the round time is longer but also the workload on the CPU normalized over time is substantially higher.

The sweep across N at the baseline degree four (Figure 6.13) gives a high-level intuition on the resource usage of running the experiment from a user's perspective. The number of nodes in the scenario does not have an a direct DFL impact on the mixed-off scenario. As with mixed-off, the number of exchanging peers is constant and defined by the transport degree and not increasing due to the k -hop reachability. For the mixing-on scenario, each added node can lead to new overlay paths for model exchange and effects the scenario as a whole. Nevertheless, the focus of this experiment lies mainly on the difference, not the general trends of the individual curves.



(a) Mean CPU time per round.



(b) Mean round duration.

Figure 6.12: Mixing cost across graph degree $d \in \{4, 8, 12, 16, 20, 24, 28\}$ at $N=32$, $K=2$, $O=100$, $n_{\text{rounds}}=5$. Only `mix_enabled` differs between the two arms. The per-round CPU delta is one dimension of the anonymity cost, the wall-clock per-round delta the other.

Degree d	CPU/round (s)		Round time (s)	
	mix on	mix off	mix on	mix off
4	14.0	6.5	40.5	6.3
8 (peak)	17.6	6.6	62.3	6.7
28	16.4	8.6	72.0	8.4

Table 6.4: Mixing cost at the sparse, CPU-peak, and dense configuration of the degree sweep, $n_{\text{rounds}}=5$, at the baseline of Table 6.1.

The per-round CPU time shows a delta of around +8 s across the whole N -axis. This is approximately double the effort in the same time window. Similarly, the CPU-time of the degree sweep shows an offset of +7.6 s, so the cost is roughly constant. Therefore, the mixing costs a constant multiplier in CPU-time, regardless of how many peers the framework runs.

The mean round duration shows a similar pattern for the sweeps in d and N as in Figure 6.13 and Figure 6.12b. Mixed scenarios take longer to finish by roughly a constant factor. Figure 6.13 shows the cost of the constant output rate of the mixnet, while the mixing-off scenario only communicates on demand.

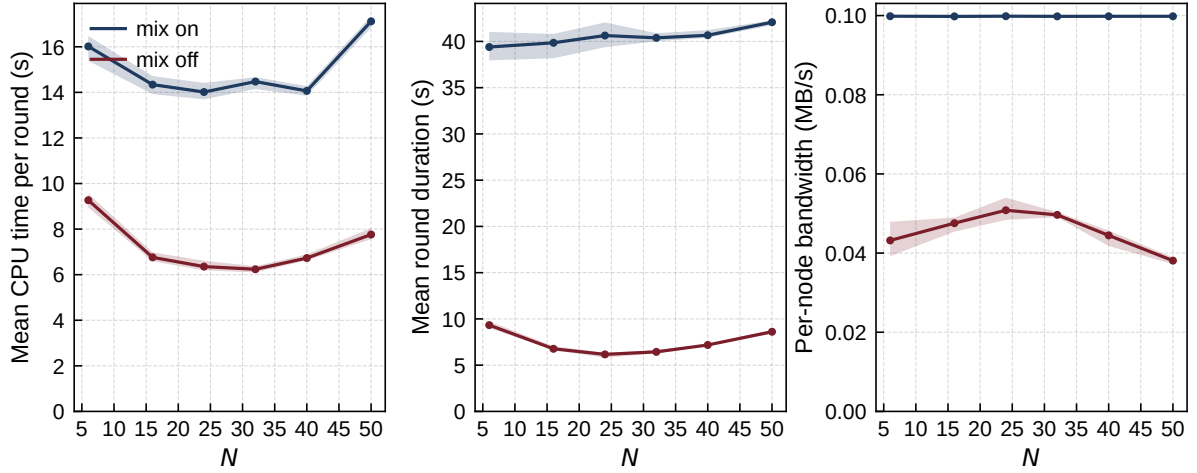


Figure 6.13: Mixing cost across network size $N \in [6, 50]$ at the baseline of Table 6.1 with degree four. The per-round CPU delta is similar to the $d=4$ point of Figure 6.12, around $+7.6$ s/r.

6.3.5 Content attack from a curious recipient

The mixing addresses an attacker observing the network, but it does not stop a curious DFL node from having a detailed look on the models. The threat model of Chapter 4 assumes peers are honest-but-curious. Any author of a fragment is hidden inside the same 2-hop reachable set of eight peers on the $N=32$, $d=4$ baseline. So a fragment belongs to an overlay peer with probability $1/d$.

This subsection studies a node that tries to identify the author of the fragments it receives, or at least link them to the same sender. The experiment introduces two simplifications which give the attacker an advantage. First, the attacker gets a full model update in the first aggregation round as a basis for later on assigning new updates to previous updates, independent of the partial-update-ratio. Second, all nodes other than the attacker perform a full aggregation, only the attacker receives fragments according to the partial-update-ratio.

The attacker keeps buckets to store fragments in. Beginning in round two, the attacker subtracts any received fragment from its own local model. The two attacks then compare these fragment differences by the cosine distance to the latest received fragments at the top of all buckets. The attacker then simply tries to place each received fragment into one of the buckets.

The evaluation shows two metrics. The tracking accuracy reports the accuracy of correctly assigned fragments into the anonymous senders's individual buckets. A high tracking accuracy indicates, that the attacker can track an anonymous sender's contributions by the similarity of its fragments over multiple rounds. The linking accuracy assigns each bucket an identity from which it has the most fragments placed into. A high linking accuracy indicates, that even if not all updates were placed correctly, the majority of the fragments still correctly uncover a specific sender. Therefore, if the attacker would gain any additional information, it could possibly uncover the information. A random guess scores $1/8$ over the eight overlay peers and a perfect attack scores 1.

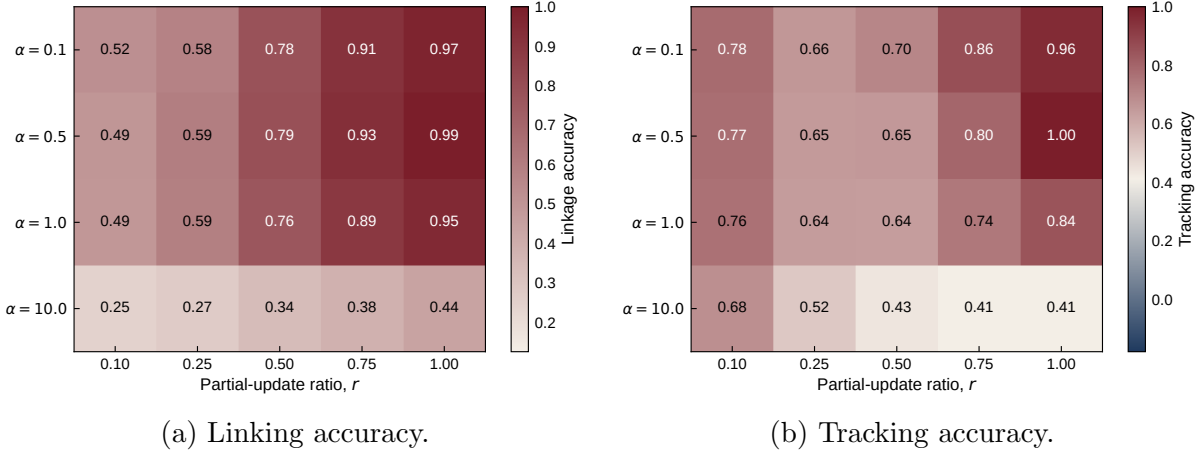


Figure 6.14: Curious-recipient content attack on the heterogeneity α against partial-update ratio r grid. Each cell shows its accuracy against a $1/8$ chance, and is the mean over repeated draws. (a) Linkage, the accuracy of assigned identities to the buckets. (b) Tracking, the share of fragments being placed in the correct bucket.

Linkage is increasing along both axes. With full update the attacker uncovers the sender correctly for all heterogeneity levels up to $\alpha=1$. Here the linking reaches an accuracy between 0.95 and 0.99 against the $1/8$ of a random guess. It drops to 0.44 only at the near-uniform $\alpha=10$. This shows that a heterogeneous dataset fingerprints each model fragment enough to track the fragments. The reference for each comparison is only performed to the latest received fragment placed in a bucket. So linkage becomes stochastically easier and increases with r , from 0.52 at $r=0.10$ to 0.97 at full update for $\alpha=0.1$.

Tracking finds out the most probable author of each model bucket by majority vote. At full update it correctly uncovers the sender for all heterogeneity levels up to $\alpha=1$. The accuracy is high between 0.84 and 1.00 against a baseline $1/8$ chance. Furthermore, it still reaches 0.41 at $\alpha=10$. This shows that an attacker could uncover a peer with high chance if it would gain additional information about some fragments' authors. Linkage is very strong where the data is heterogeneous and at least half of each update arrives.

The partial-update ratio mitigates both attacks but cannot prevent any of them. Furthermore, the heterogeneity that increases anonymity also hurts convergence as shown in Section 6.5.4.

Similarly, Shatter [36] performed experiments at the fragment level. The attacker holds every node's training data and computes the loss of each received chunk on each candidate. The assignment of model to sender is done by assigning the model to the dataset with the lowest loss. Shatter achieves a near uniform assignment even with this strong attack of an attacker holding all the data or very similar shadow data.

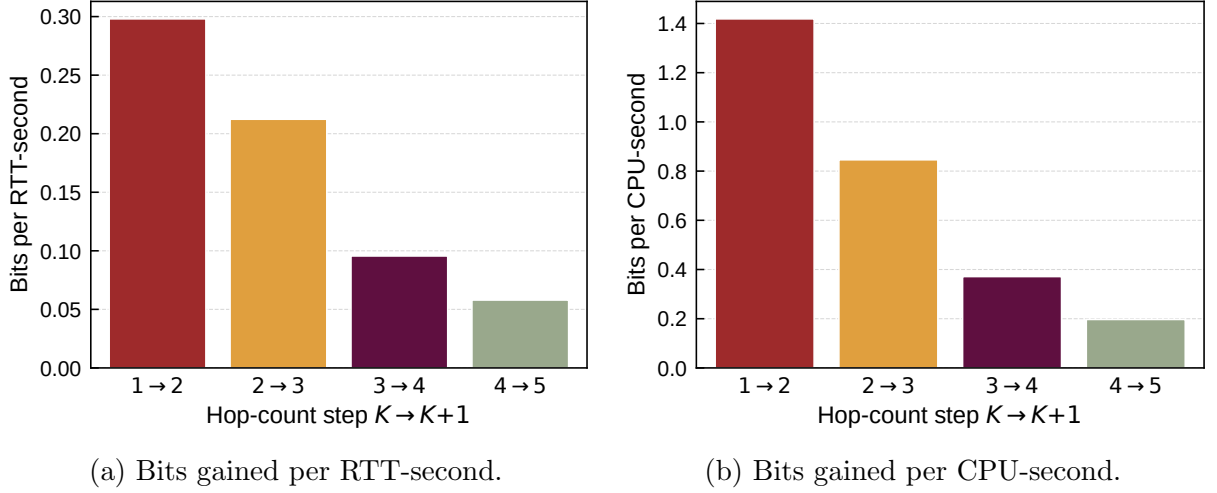


Figure 6.15: Marginal entropy gain per K-step at the baseline of Table 6.1. Each K-step buys $\log_2 O \approx 6.64$ bits. The bar height is that gain divided by the RTT or CPU difference between consecutive K values. μ contributes zero bits per RTT-second across its sweep, and O moves entropy only within a given K , so neither appears as bars.

6.4 Joint optimization

The mixnet is mainly defined by the three parameters: d , k , O . This section discusses the trade offs to answer the joint optimization of scalability and anonymity asked by RQ2. As discussed in Section 6.3.1, each step of k generates roughly $\log_2 O$ bits of entropy but at a rising RTT and CPU overhead per bit (Figure 6.15). On the other hand, μ and O affect the RTT and the entropy more fine grained as shown in Section 6.4.1. Choosing a practical operating configuration therefore reduces to primary defining K to achieve the required anonymity, with μ and O being for fine grained adjustment and to reduce the RTT.

6.4.1 Mixnet parameters

Each of the main three mixnet parameters creates a different combination of bandwidth, latency, cover-traffic share, CPU, memory, and mixing backlog queue. Figure 6.15 visualizes each K-step in RTT and CPU, while this subsection quantizes the cost on each axis individually at $N=32$ Table 6.5 shows the RTT, CPU, and memory endpoints of all three.

Change in μ

The mixer rate μ changes output rate without changing the entropy. So along the μ axis, path entropy stays at 13.30 bits for the default configuration. It is the user's primary option for adjusting RTT once K and O are chosen. Per-node bandwidth shows the analytical S/μ reference is very close to the measures bandwidth. The theoretical RTT has its upper bound at $2KO\mu$, which implies that each incoming fragment is placed at the

Parameter	Value	RTT (s)	CPU/round (s)	Mem (MB)
μ (s)	0.04	12.0	20.0	393
	0.10 (baseline)	29.7	14.2	393
	0.32	94.4	11.8	393
K	1	7.4	9.5	390
	2 (baseline)	29.7	14.2	393
	5	245.1	73.7	405
O	10	12.7	10.0	391
	100 (baseline)	29.7	14.2	393

Table 6.5: Cost endpoints for each mixnet parameter at $N=32$, with the other two parameters held at the baseline of Table 6.1. RTT is the median value. CPU and memory are per-node means per round.

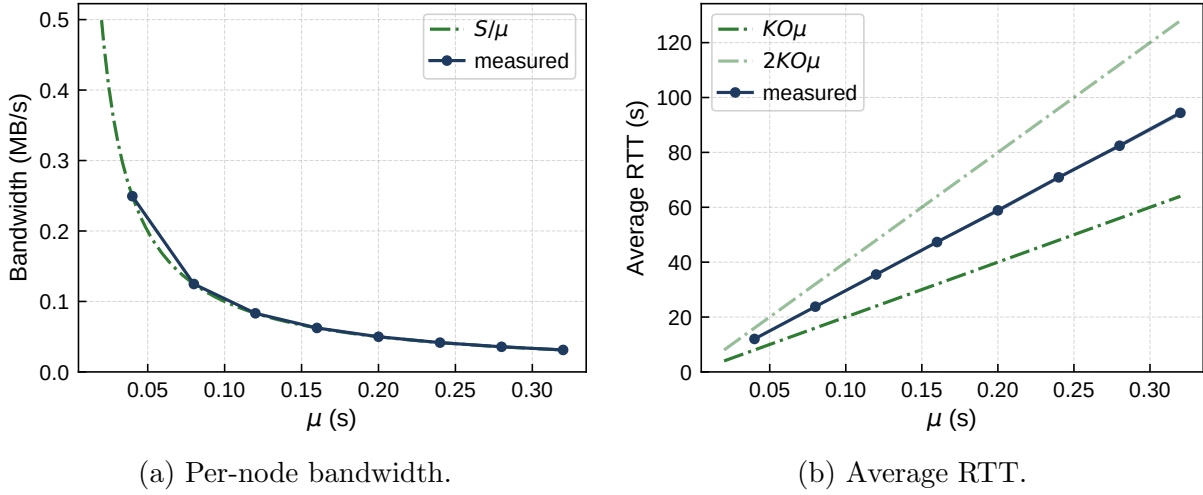
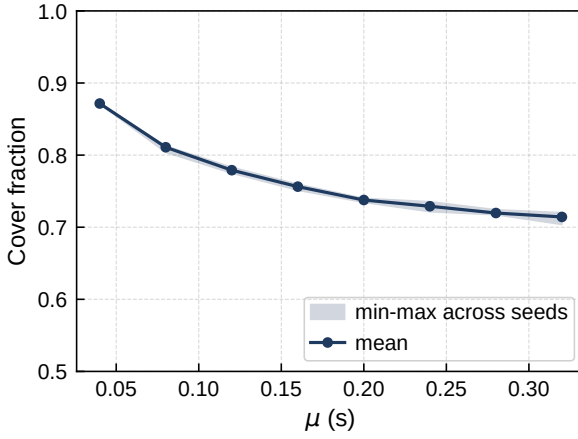


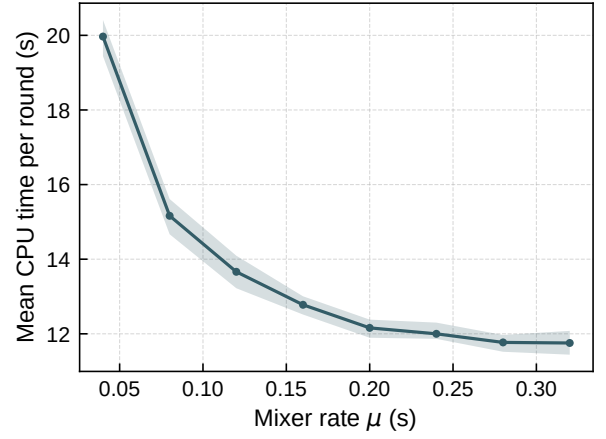
Figure 6.16: Measured bandwidth and RTT as a function of μ at $N=32$, $K=2$, $O=100$. Shaded bands are the min-max across all runs at each μ . The line is the mean across seeds.

end of the outbox after shuffling. The expected RTT of $KO\mu$ assumes each fragment is placed in the middle of the outbox after shuffling. It shows, that the measured RTT tends to converge towards the expected value but is on average higher in the evaluated ranges.

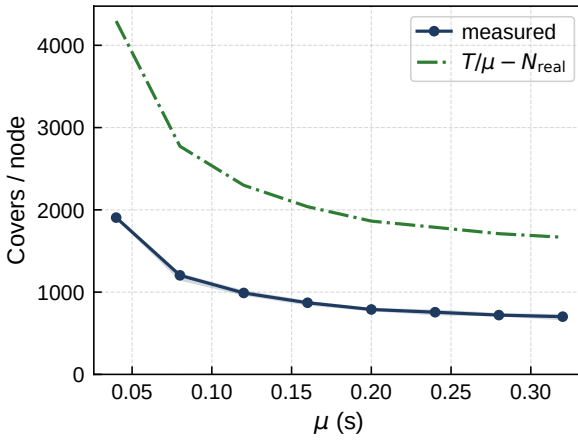
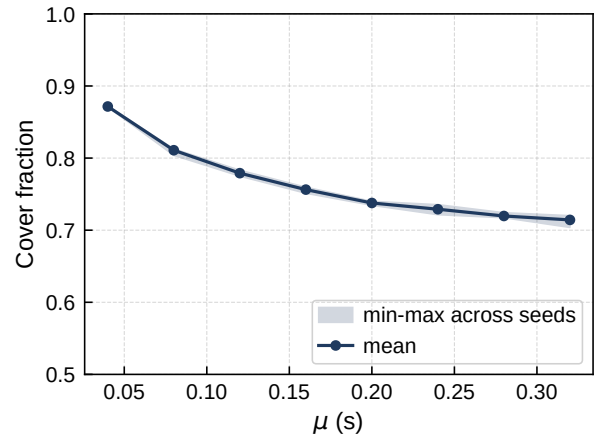
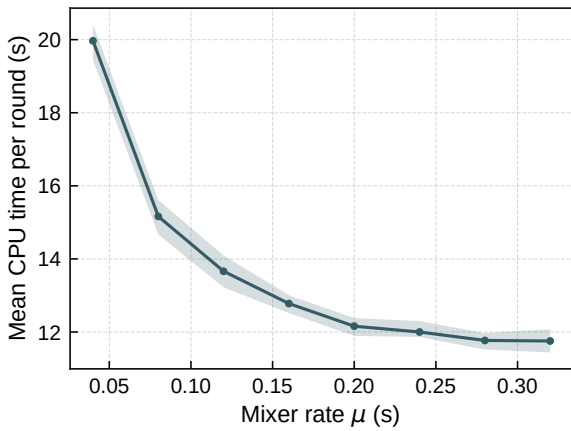
As for the computational resources, lower μ puts more pressure on each node and requires much more cover traffic to be generated as the meaningful traffic is independent of μ . Figure 6.17 shows the cover fraction and mean CPU-time rising as μ shrinks. In contrast, memory stays approximately constant with changing μ .



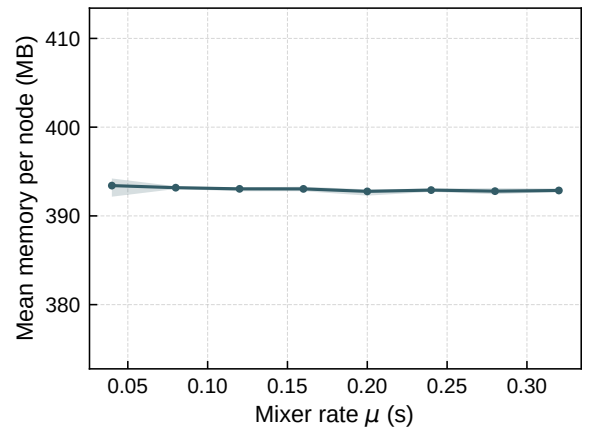
(a) Cover fraction.



(b) Mean CPU time per round.

Figure 6.17: Cover fraction and per-round CPU time as μ varies at the baseline of Table 6.1.(a) Cover packets per node across the μ sweep, against the analytical $T_{\text{active}}/\mu - N_{\text{real}}$ reference.(b) Cover fraction across the μ sweep.Figure 6.18: Cover-traffic detail at $N=32$, mean over five runs per point.

(a) Mean CPU time per round.



(b) Mean memory per node, widened axis.

Figure 6.19: Per-node CPU time per round and memory across the μ sweep at the baseline of Table 6.1.

Change in K

K is the biggest leverage for entropy and computational resources. Each added hop adds roughly $\log_2 O$ bits of path entropy but is expensive in measured RTT, per-round CPU, and queue backlog. RTT increases $2\text{--}4\times$ per step, a much larger increase than the expected RTT following $KO\mu$. The main reason is the congestion as the packets stay in the network for longer while the μ stays constant.

Figure 6.20 shows this more-than-linear increase in RTT with increase of k . Similarly, Figure 6.21b shows the increase in mean CPU-time per node following a similar more-than-linear increase as RTT. Figure 6.21a shows, similar to μ , a decrease in cover ratio with an increase in k . The cover fraction decreases to its plateau at $K = 4$ at approximately 50%.

Figure 6.22a shows the increased congestion due to an increase in K . This, as each node's queue for feeding the outbox queue grows while its outgoing rate μ is constant. Outstanding fragments per node at round end grow from zero to ~ 190 at $K=5$, while the mean round duration is increasing with them (Figure 6.22).

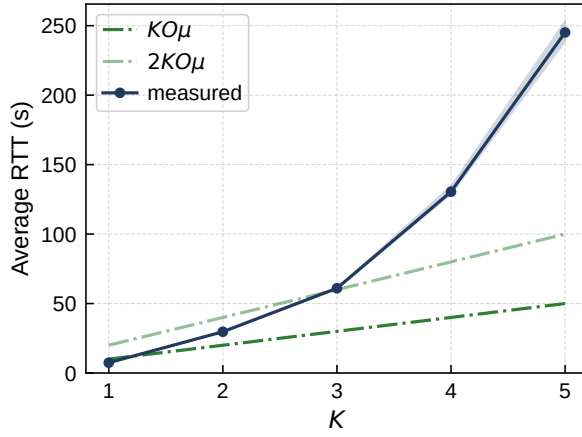
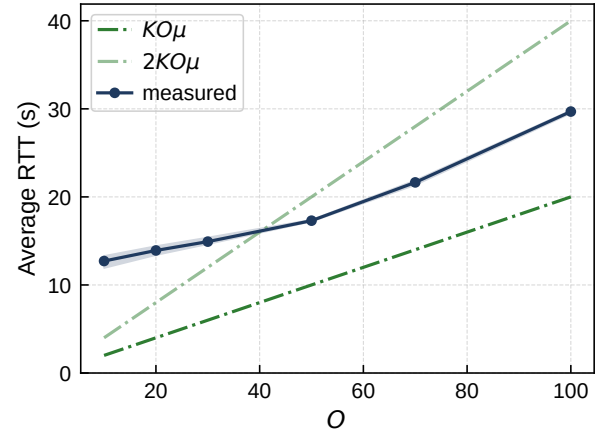
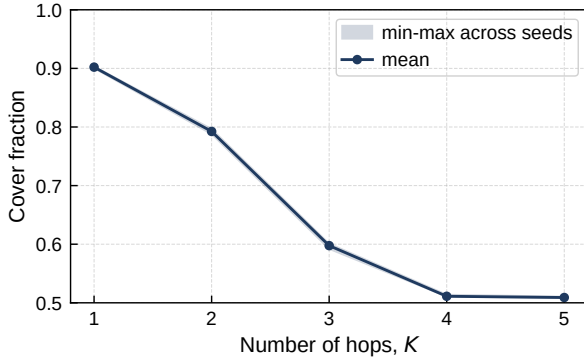
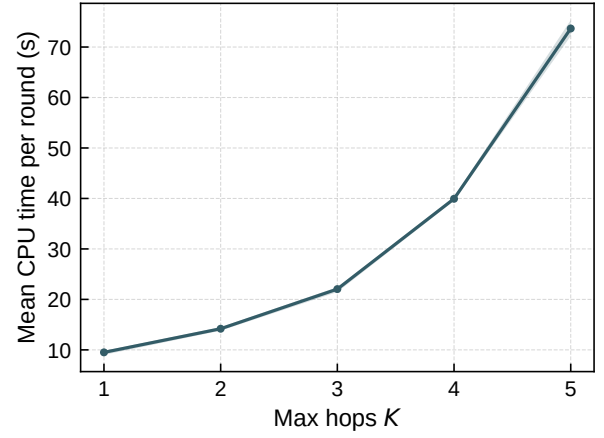
(a) vs maximum hops K .(b) vs outbox size O .

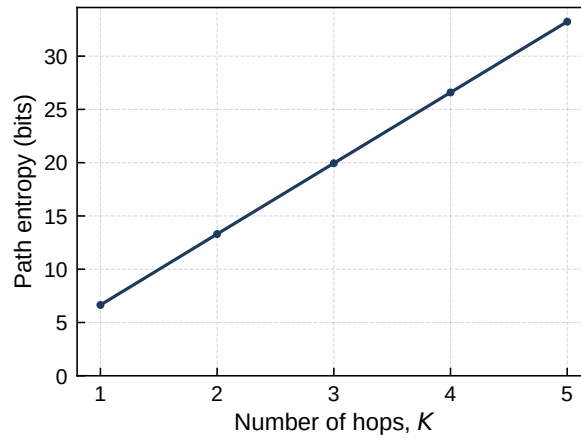
Figure 6.20: Measured average RTT as K and O vary. Shaded bands are the min-max across seeds at each parameter value. The line is the mean across seeds.



(a) Cover fraction.



(b) Mean CPU time per round.



(c) Analytical path entropy $\log_2(\sum_{k=1}^K O^k)$ at $O=100$.

Figure 6.21: Cover fraction, per-round CPU time, and analytical path entropy as K varies at the baseline of Table 6.1.

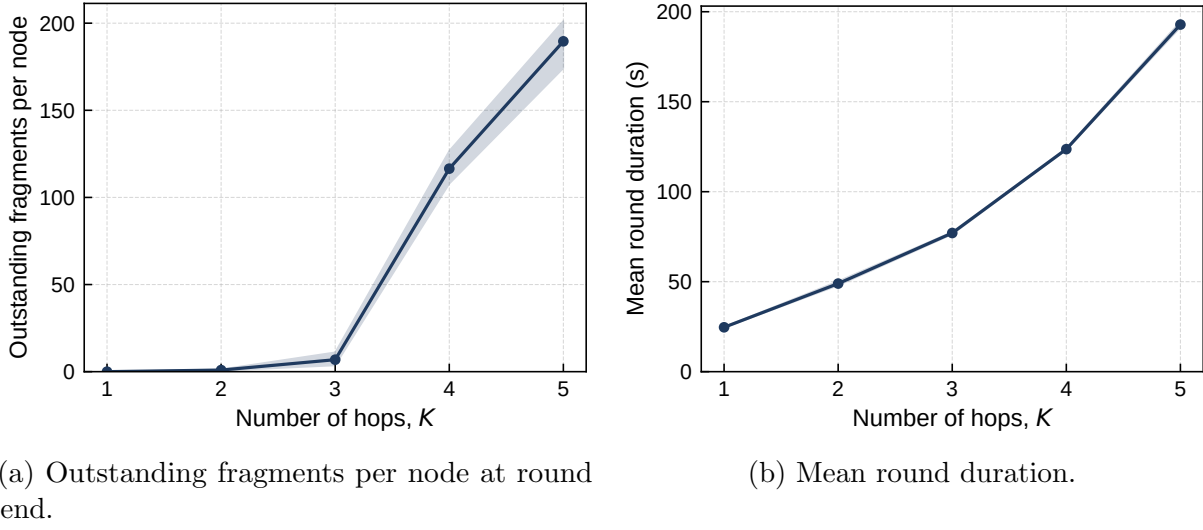


Figure 6.22: Backlog and round-duration response across the full $K \in \{1, \dots, 5\}$ range at $N=32$, $\mu=0.1$ s, $O=100$. The $K=5$ point runs at $n_{\text{rounds}}=5$, comfortably longer than the per-direction half-cycle $KO\mu=50$ s. Outstanding fragments and round duration grow monotonically with K across the swept range.

K and O

Figure 6.23 shows per-round CPU and memory across O and K on a common scale at the baseline $\mu=0.1$ s, $N=32$. A ten fold increase in the shuffled outbox size moves the mean round CPU-time by only $1.40\times$ and the memory approximately constant. This indicates that O is not that relevant when considering computational resources.

In contrast, CPU and memory are increasing steeply with an increase K . However, at the same time an increase of $\sim 7.7\times$ in CPU also provides a ~ 26 -bit rise in Shannon entropy. Memory shows a slight increase, following the intuition that the longer paths require a larger queue due to much higher congestion.

Together with the μ sweep, this links each K -step on Figure 6.15 to a measurable cost. An increase in K pays in CPU, RTT, and backlog queue size. Increasing μ pays in RTT and cover-stream CPU. O pays in CPU. The operating point that satisfies both anonymity and performance is the smallest K at which the path entropy requirement is met, with μ and O being used to bring RTT down to the required level of responsiveness.

6.5 Convergence verification

Convergence with data privacy is a major goal of DFL. However, in this work it is mainly a validating metric as improving the learning quality is not part of this work's contribution. Each parameter sweep the previous sections discussed, is tested here again against the round-ten accuracy. This, so the scalability statement in Section 6.2 can be explored further. This includes the sparse topology of Section 6.3.2, the 8-bit quantization of

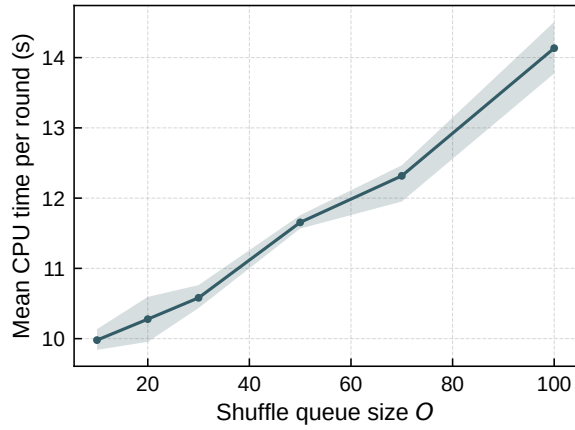
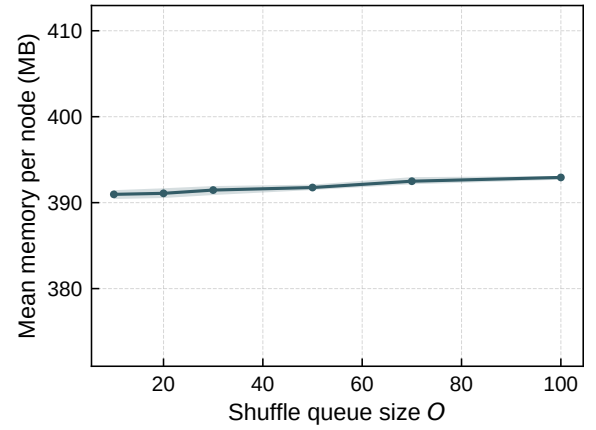
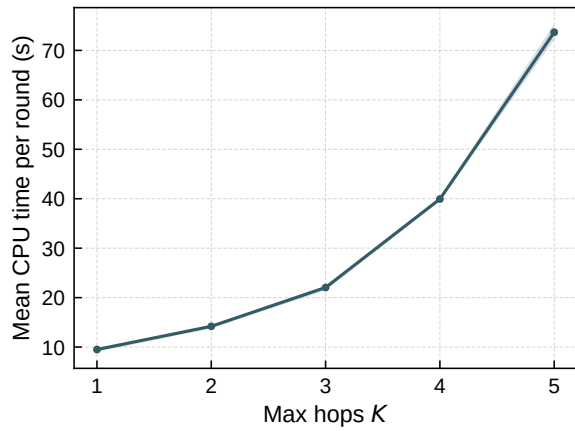
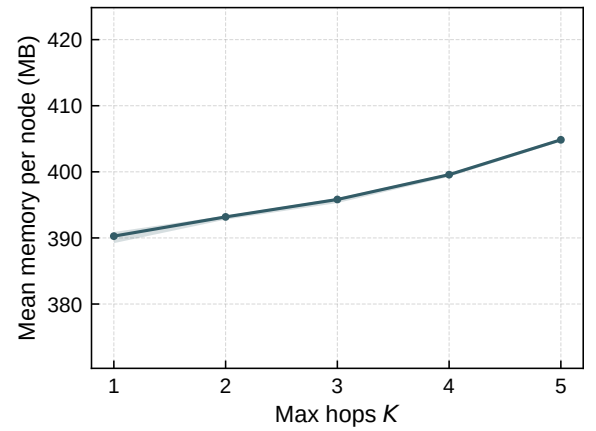
(a) Mean CPU time per round across O .(b) Mean memory across O .(c) Mean CPU time per round across K .(d) Mean memory across K .

Figure 6.23: Per-round CPU and memory across the O sweep (top row) and the K sweep (bottom row) at the baseline of Table 6.1.

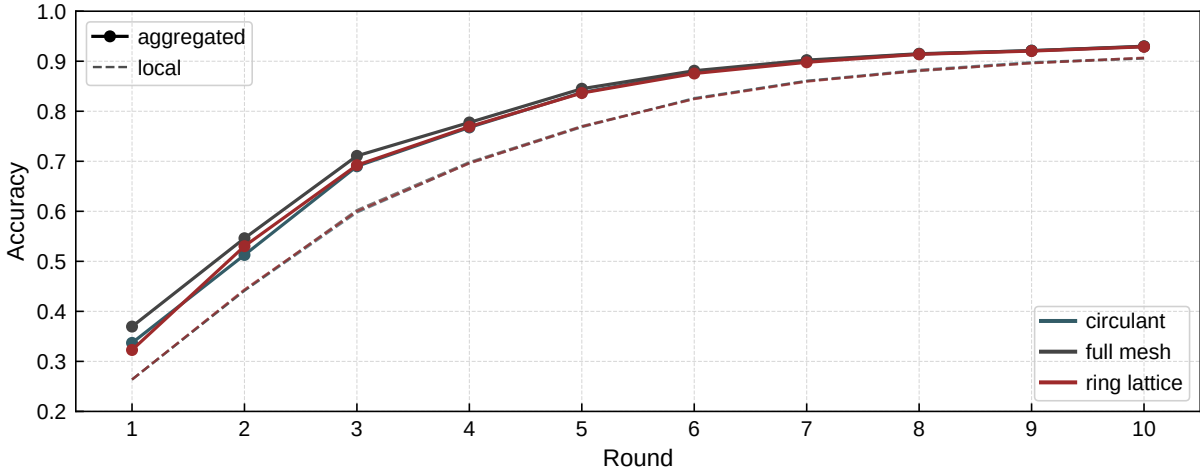


Figure 6.24: Accuracy convergence by topology at $N=32$, target degree four (full-mesh degree $N-1$). Solid lines with markers show aggregated accuracy after each round’s combined update. The dashed lines show local accuracy after the local training step, before aggregation. Lines are the mean across nodes and the five runs.

Section 6.2.2, the partial-update ratio, the cover stream of Section 6.3.3, and Dirichlet. The only question each of the following experiment answers is, whether the learning is robust to the choice in the chosen parameter ranges.

6.5.1 Sparse topology

The initial result in Section 6.2.1 showed, that the learning quality is mainly unaffected at the default configuration. This subsection verifies that the convergence quality also remains across various changes in mixing, topology, and learning parametrization.

Figure 6.24 shows the round-by-round accuracy. The full mesh takes an early lead, whereas the ring lattice and circulant topology have a 4.7 pp spread. However, the gap closes quickly and the three curves become almost indistinguishable early on. At round-ten the topologies have a spread of 0.11 pp which can be considered to be minimal. It shows that with the evaluated parameters, the choice between sparse topologies at the default configuration does not make a major difference.

6.5.2 8-bit quantization

The calculated reduction in Section 6.2.2 shows a $\div 3.7$ reduction in fragment-count when quantizing to 8-bit. This subsection verifies that the round-ten accuracy is roughly unchanged between the 8-bit and 32-bit settings at *dirichlet* = 10 for both, MNIST and Fashion-MNIST using LeNet-5.

Figure 6.25 overlays the two experiments. The 8-bit and 32-bit aggregated accuracies increase similarly within 0.16 pp of one another at every round on MNIST. On Fashion-MNIST they are off by approximately 0.06 pp. The $\div 3.7$ volume saving of Section 6.2.2

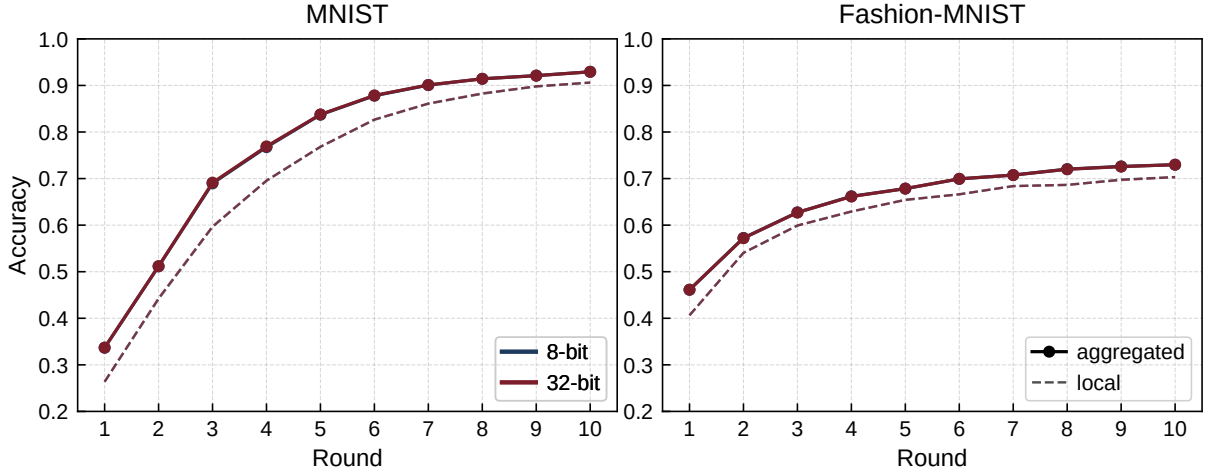


Figure 6.25: Accuracy convergence under 8-bit and 32-bit quantization at $N=32$, $n_{\text{rounds}}=10$, five runs, on MNIST (left) and Fashion-MNIST (right). Solid lines with markers show aggregated accuracy after each round’s aggregated update. The dashed lines show local accuracy after the local training step, before aggregation.

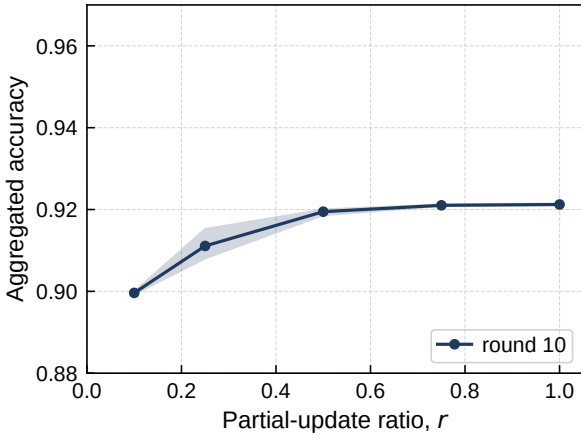
therefore comes at no measurable round-ten accuracy reduction using LeNet-5 and dirichlet of 10.

6.5.3 Partial-update ratio

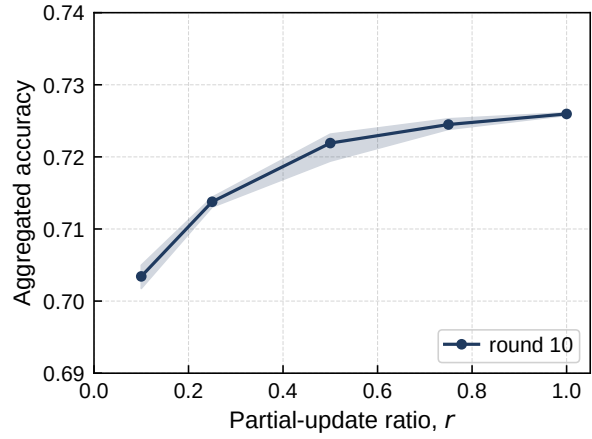
The constant-rate cover stream keeps the per-node send rate to $1/\mu$ regardless of the partial-update ratio r . This reduction does not have any implication on anonymity. This subsection sweeps $r \in \{0.10, 0.25, 0.50, 0.75, 1.00\}$ at $N=32$, $n_{\text{rounds}}=10$ on circulant degree four, with the same grid repeated on full mesh. This, to isolate how the learning signal responds to the partial updates.

Figure 6.26 reports round-ten accuracy across r . On MNIST it rises monotonically from 89.96 % at $r=0.10$ to 92.12 % at $r=1.00$. With Fashion-MNIST, the curve has a similar shape over a 2.2 pp, but shows a larger effect. Regarding the per-round development visible in Figure 6.27, the gap between the $r=0.10$ and $r=1.00$ curves peaks at 10.5 pp at round four for MNIST. For Fashion-MNIST, it stays approximately constant.

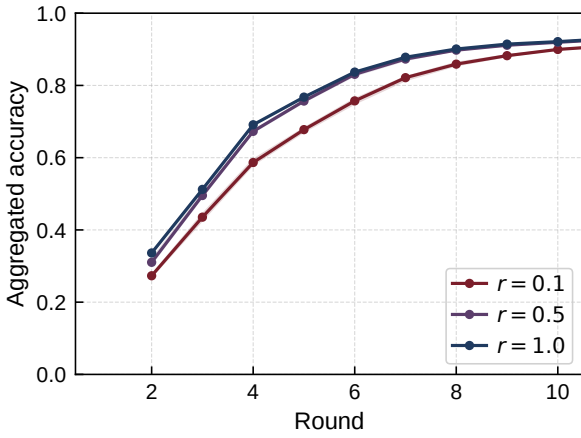
Figure 6.28 shows the same r -sweep using a full mesh topology. The round-ten spread reduces to 0.36 pp across the entire r range. This is approximately an order of magnitude tighter than the spread with the circulant topology. On a sparse topology each peer only aggregates with its K -hop neighbors each round. So even with the same dataset distribution among nodes, the slower diffusion and limited aggregation are visibly affecting the convergence with a lower r .



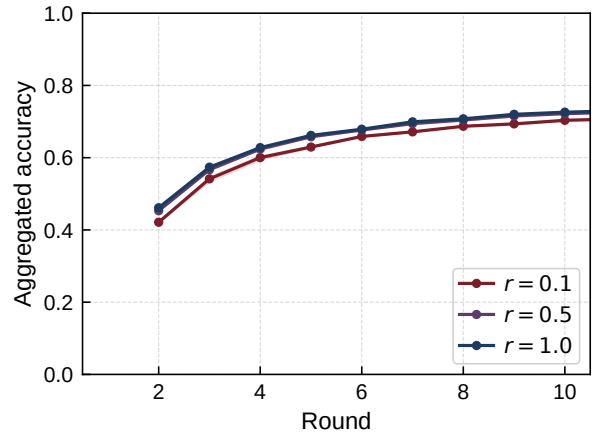
(a) MNIST.



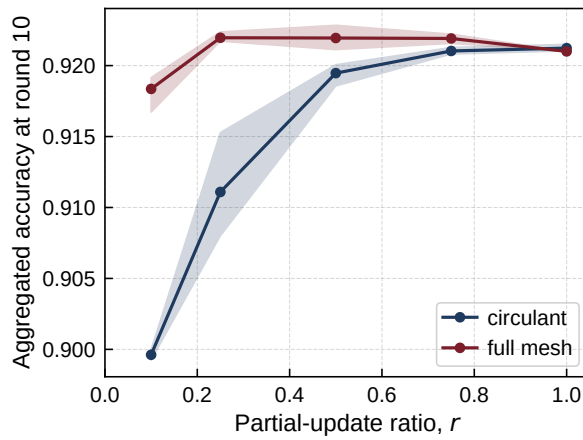
(b) Fashion-MNIST.

Figure 6.26: Round-ten accuracy across r at $N=32$, mean over five runs.

(a) MNIST.



(b) Fashion-MNIST.

Figure 6.27: Per-round accuracy trajectory at $r \in \{0.10, 0.25, 0.50, 0.75, 1.00\}$, $N=32$, mean over five runs.Figure 6.28: Round-ten accuracy across r on MNIST at $N=32$, circulant versus full mesh, mean over five runs.

6.5.4 Data heterogeneity

The partial-update sweep of Section 6.5.3 isolated the effect on the learning quality with a near-IID partitioning ($\alpha=10$). This subsection stresses the same mechanism under non-IID partitioning over $\alpha \in \{0.1, 0.5, 1.0\}$ against the same r grid at $N=32$ on circulant degree four.

Figure 6.29 shows the round-ten accuracy across the (α, r) grid. Against the near-IID anchor at $\alpha=10, r=1.0$ (92.12 %), the negative effect is increasing along both dimensions. Low data heterogeneity at $\alpha=1$ costs little at full model exchange. However, strong non-IID at $\alpha=0.1$ reduces the round-ten accuracy to 38.28 %, even at full exchange with $r=1.0$. It shows that heterogeneity and a low partial-update add up and reduce the overall learning quality drastically.

Figure 6.30 shows the learning over time at full update ($r = 1$). The mild IID curve at $\alpha=1$ shows an initially lower learning quality but catches up to within 3.6 pp of the near-IID default configuration by round ten (88.51 %). In contrast, $\alpha=0.1$ is still increasing at round ten. This indicates that the FedAvg averaging is mainly slowed by heterogeneity, but not broken.

The three design choices of the architecture to increase entropy-preserving scalability, namely the sparse topology, the 8-bit quantization, and the partial-update ratio, each still preserve the learning quality. The round-ten plateau and its progress remain mainly intact for most of the evaluated configurations. This shows that the techniques applies to increase the scalability do not break the learning quality but can drastically slow it down. The heterogeneity of the distribution has shown to be the only real axis limiting the learning quality.

6.6 Stressors

The techniques for scalability and anonymity of Sections 6.2 and 6.3 were evaluated only with stable and honest peers. This section drops that assumption. A single peer leaving and one arriving mid-run are evaluated. Moreover, we evaluate a fraction of the peers acting Byzantine.

6.6.1 Dynamic membership

For node churn the sparse circulant topology is the limitation. At $N=32$, degree four, $\text{max_hops}=2$ the max two-hop neighborhood of any single peer covers twelve peers. This includes four direct transport neighbors and eight overlay peers reachable only through them. The remaining 19 peers are unaffected and unaware of the node exiting or joining. Figure 6.31a shows the topology of the exit scenario

Figure 6.31b shows the fraction of received fragments which they can maximally receive, named the completeness. The unaware class never notices the event. Therefore, their

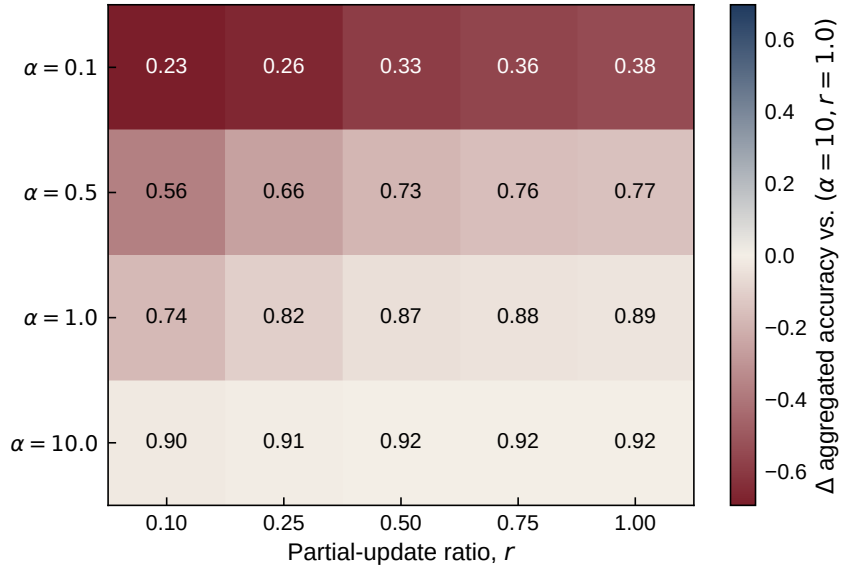


Figure 6.29: Round-ten aggregated accuracy across the heterogeneity sweep at $N=32$, centred on the near-IID anchor ($\alpha=10, r=1.0$). The $\alpha=10$ row reuses the Section 6.5.3 corpus.

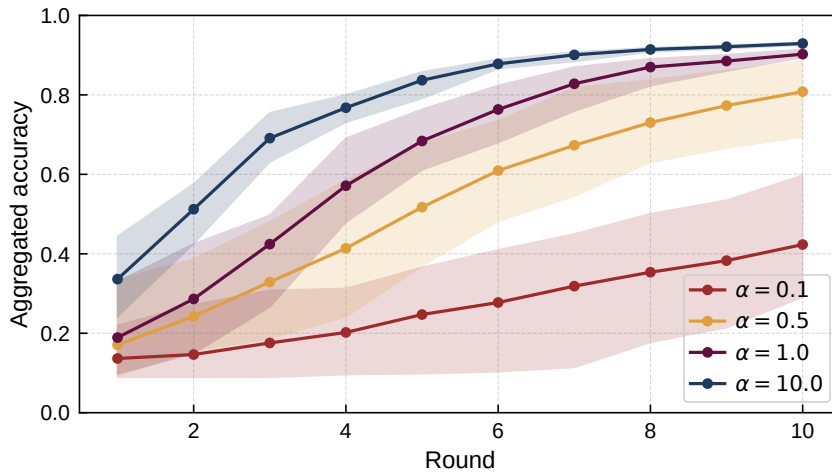


Figure 6.30: Per-round aggregated accuracy by Dirichlet α at fixed $r=1.0$, $N=32$, MNIST, FedAvg, circulant degree four. Lines are means across nodes. The bands are min/max across nodes.

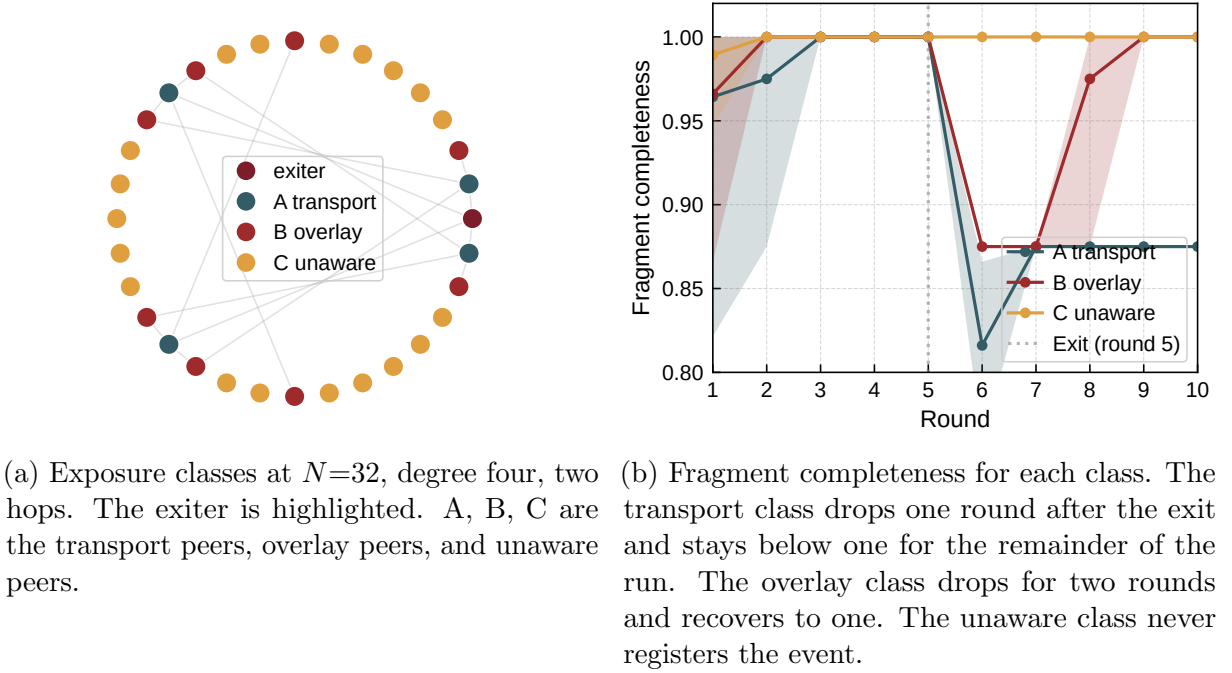
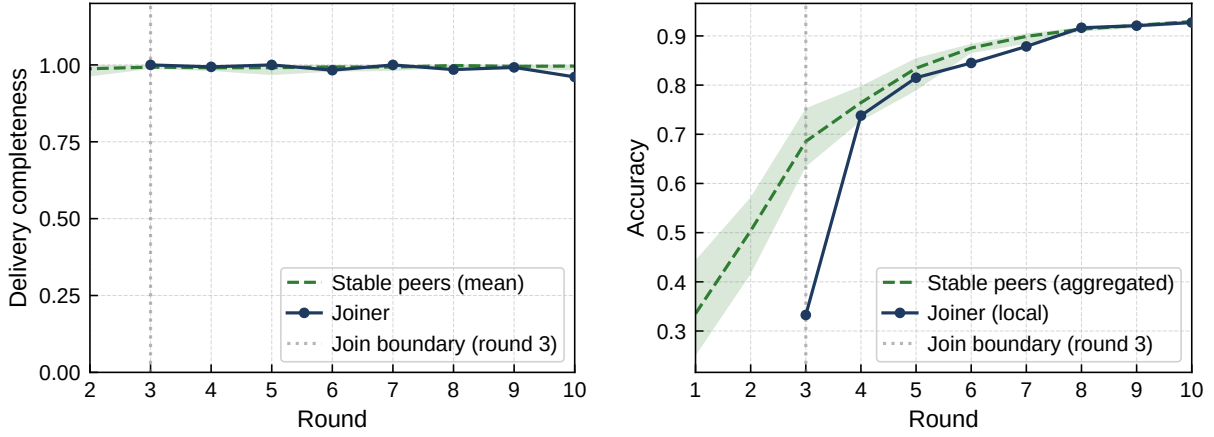


Figure 6.31: Exit scenario topology partition (left) and per-class fragment completeness (right). One peer leaves after round five at $N=32$, degree four, two hops, $n_{\text{rounds}}=10$, single run. Lines are within-class medians. The grey dotted vertical in the right panel marks the round-five exit.

completeness stays stable at full completeness. The eight overlay peers detect the exit of their overlay peers. The overlay topology is self healing and again at a stable level after two rounds. This shows the successful rerouting of their fragments over intact paths and the liveness feature of the framework. The four transport neighbors, which lose a whole group of overlay peers through the missing transport connection, cannot recover. Their completeness is stabilizing near 0.88 through round ten. This, as no alternative path can reach their overlay peers without the lost transport connection and while adhering to the K hops.

Figure 6.32a shows the late joiner's per-round fragment completeness. Already in the first round, the newly joined peer received all fragments of its overlay peers. This shows a quick integration into the transport and mixed overlay topology. Figure 6.32b shows the joining node quickly reaching the performance level of its peers. This, since FedAvg aggregates with the more refined models immediately and makes the joined node quickly align with its peers' models.



(a) Joiner per-round fragment intake as a fraction of the stable peers' mean.

(b) Joiner local accuracy catching up to the stable plateau within two rounds of arrival.

Figure 6.32: Join scenario at $N=32$, circulant degree four, $n_{\text{rounds}}=10$, single run, with one peer arriving at round three. Solid lines are the late joiner. the green dashed reference is the mean over the stable peers. Left: per-round fragment intake as a fraction of the stable peers' mean. Right: the joiner's local accuracy against the stable peers' aggregated mean. The joiner is excluded from the stable set. Its samples begin at round three. The grey dotted vertical marks the join event at round three.

6.6.2 Byzantine attacks

The Krum aggregator of the framework is the only guard against anonymous Byzantine peers. Its safety guarantee assumes the Byzantine count satisfies $n > 2f + 2$. Using Krum's default of $f = 0.33$ and $N=20$, this makes the limit of byzantine nodes to be able to defend against $b \leq 8$. If the number of byzantine nodes remains below this cap, Krum is expected to make the output seem like only aggregating with honest peers. However, with more malicious peers this breaks. This subsection sweeps $b \in \{0, \dots, 10\}$ at $N=20$ on a full mesh across three attack types at five runs per point. These are `label_flip` (full class label flipping), `sign_flip` (flipping sign of the model weights), and `gaussian_noise` (additive Gaussian on the model with $\sigma=10 \cdot \text{std}$).

Figure 6.33 and Table 6.6 compare the two aggregators against each other across the sweep. The pattern is similar for all three attacks. Krum holds through $b=9$. Under the two directed attacks, `label_flip` and `sign_flip`. Meanwhile, FedAvg degrades smoothly as each corrupted update poisons the average a little further. Under the undirected `gaussian_noise` attack, FedAvg does not degrade smoothly but collapses immediately at $b=2$. This, because one high-magnitude noise vector dominates the mean in naive averaging.

The empirical experiment shows, that the proposed fragment-based Krum reaches even one count beyond its theoretical limit on the directed attacks and just holds with the undirected one. This indicates, that for anonymous aggregation under standard attacks, fragment based krum can achieve robust aggregation.

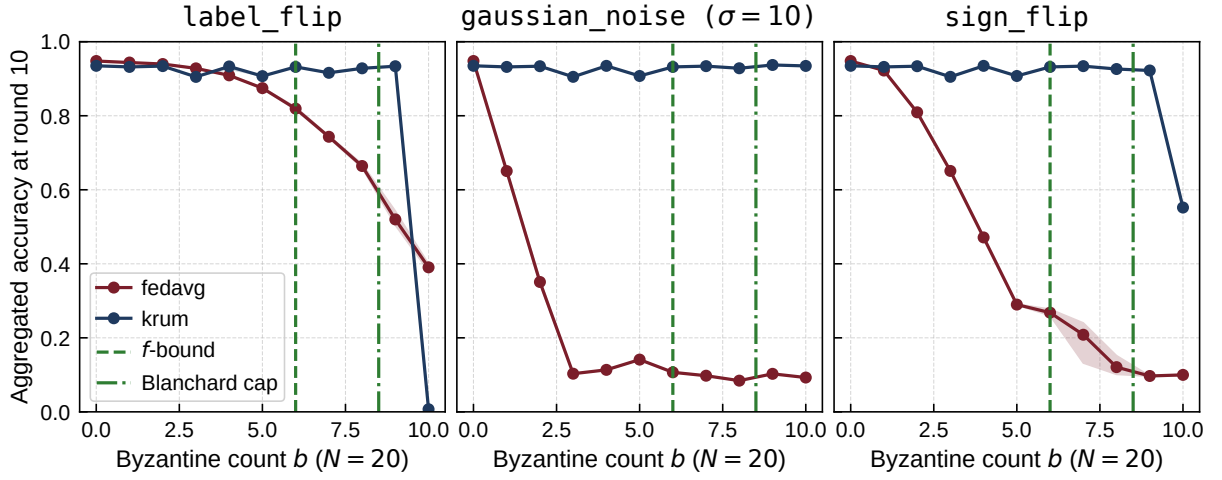


Figure 6.33: Round-ten accuracy versus Byzantine count b at $N=20$, full-mesh, $n_{\text{rounds}}=10$, mean over five runs per point. Panels left to right: **label_flip**, **gaussian_noise** ($\sigma=10$), **sign_flip**. Reference lines: f -bound at $b=\lfloor f \cdot N \rfloor = 6$ (dark green dashed) and Krum's limit between $b=8$ and $b=9$ (dark green dash-dot). FedAvg in **bordeaux**, Krum in **navy**.

Attack	Aggregator	Clean ($b=0$)	Round-ten accuracy across $b=1 \dots 10$
label_flip	FedAvg	0.948	smooth decay, 0.94 at $b=1$ to 0.39 at $b=10$
	Krum	0.935	within 3 pp through $b=9$, 0.007 at $b=10$
gaussian_noise ($\sigma=10$)	FedAvg	0.948	collapses at $b=2$ (0.35), below 0.15 after
	Krum	0.935	stays in the 0.90–0.94 band through $b=10$
sign_flip	FedAvg	0.948	0.92 at $b=1$, 0.29 at $b=5$, ~ 0.10 from $b=8$
	Krum	0.935	within 3 pp through $b=9$, 0.55 at $b=10$

Table 6.6: Round-ten accuracy under FedAvg and Krum across the three attacks at $N=20$, full mesh. Clean is the shared $b=0$ baseline per aggregator. Krum's limit $n > 2f + 2$ caps at $b \leq 8$.

6.7 Comparison to previous work

The previous sections describe the newly proposed framework on its own. Table 6.7 compares the learning and capabilities of the improved framework against its predecessor.

Dimension	Predecessor	This work	Delta
N	6	up to 64	+58 max
Topology	full mesh	circulant deg 4	anon-set 31 $\rightarrow$ 8
Packet body	1253 B	10240 B	$\times 8.2$
Quantization	none	8-bit	$\div 3.7$ fragments
Partial update	none	$r \in [0.1, 1]$	2.2 pp over $r \in [0.1, 1]$
Krum aggregator	none	$f=0.33$	survives $b=9/20$
Heterogeneity	not reported	$\alpha \in \{.1, .5, 1, 10\}$	$\alpha=1$ within 3.6 pp of IID
Dynamic membership	full mesh	sparse deg 4	1-round catch-up
Transport	paired TCP	QUIC mTLS	bootstrap
Mixing cost	not reported	+8.2 s/r CPU	baseline cost

Table 6.7: Configurable properties of this framework being compared against the predecessor work of Chapter 3. Predecessor values are quoted from the predecessor’s evaluation. This-work values are taken at the baseline description of Table 6.1.

The first four rows carry the scalability contribution. Network size, packet body, quantization, and transport are the changes that scale anonymous DFL further than the previous version. The wider Sphinx body and 8-bit quantization make the bytes per round smaller and more efficient. The QUIC mTLS rebuild of Section 5.2 makes a larger scenario have a low footprint as well as fully encrypted and authenticated.

The next two rows are the price of that scale. The sparse circulant topology gives a smaller anonymity set per fragment than the previous version’s full mesh. The constant-rate cover stream that protects the sender adds a per-round CPU cost on every node. The previous work did not evaluate or report any of both.

The last four rows are parameters the framework adds for scalability and anonymity. The previous version’s full-mesh, FedAvg, near-IID, stable-peer configuration didn’t evaluate any of them. Each parameter is evaluated in its approximate cost or behavior to assess the new joint optimization.

Depending on the data heterogeneity, the partial-update ratio trades a small accuracy cost for a drastic reduction in per-round fragments. Krum holds the round-ten accuracy even a node past its theoretical Byzantine limit. A single peer joining or leaving mid-run only costs a short disruption but is managed by the nodes themselves without any explicit coordination message.

6.7.1 Comparison against unlinkability frameworks

The table above compares this framework against its predecessor. AEFL [46] reports node and path anonymity when some relays are not honest. FL2DP [50] reports how hard it

is for a curious server to guess the sender. The other three [47], [48], and [49] argue the link is hard to break and report mainly cost or accuracy. None of the five measures the link at the level of the single fragment that crosses the network. This work measures it there. On the baseline circulant topology each fragment hides among 8 peers, and on a full mesh among 31. The content attack of Section 6.3.5 tests specifically on how well a curious node could reduce linkability on the fragment level. Shatter [36] does measure linkability directly on the fragment level by computing the loss over the origin’s training data. However, this does not prevent network level tracking.

Many related frameworks outsource their updates or their checks through a central server. AEFL [46] and FedTor [47] use a server that registers the nodes and generate the routing data. EPPAFL [48] uses a central server that issues pseudonyms to the nodes. DPBFL [49] and FL2DP [50] use a central shuffler that reorders the updates before the aggregation. This work uses a peer-based mixnet with no central server. It keeps running when a peer joins or leaves mid-run, over a sparse and changing mesh. However, the design has two costs. Namely, the sparse mesh gives a smaller anonymity set per fragment than a full mesh or a centralized shuffler can provide. Moreover, the constant-rate output adds continuous workload onto the node.

Chapter 7

Summary and Conclusions

This work refactored and extended the peer-based mixing DFL framework of [3] to scale anonymous DFL from six nodes in a full mesh to higher scaled $N=64$ scenarios. The transport layer was rebuilt on QUIC with mutual TLS for encryption and authentication. The model exchange process was upgraded to use a wider Sphinx body for increased efficiency, 8-bit quantization, and a configurable partial-update ratio gossiping algorithm. In addition to the full mesh topology, the new framework added sparse circulant topologies. While the previous framework relied on a transport connection to handle node churn, the liveness is now observed through the overlay graph. This makes the framework adaptable to node churn on any topology while still preserving the path entropy and unlinkability. Moreover, the newly proposed algorithm ensures a meaningful level of reliability while supporting stragglers to catch up. Krum was added as a Byzantine-robust aggregation option alongside FedAvg to mitigate various attacks. This, as DFL becomes increasingly vulnerable to poisoning attacks due to the anonymity of the model sender.

Regarding scalability, LeNet-5 has shown to converge to a round-ten accuracy of 0.92 on MNIST and 0.73 on Fashion-MNIST at the $N=32$ baseline. Higher scaled experiments showed only losing a few points towards the $N=64$ ceiling, even with the very small per-node datasets. The transmitted per-round bytes stay under tens of megabytes at LeNet-5 but approach a gigabyte at MobileNetV2 by $N=64$. However, with 8-bit quantization the fragment count was shown to be reduceable by approximately $3.7\times$ without a major loss in the model accuracy for the evaluated models.

On anonymity, the baseline operating point reaches 13.3 bits of path entropy at a RTT of approximately 29.7 s. High-anonymity settings can reach 33.2 bits at 245 s RTT at $K=5$. However, the cost of anonymity are growing fast and are becoming unpracticable quickly. Furthermore, moving away from the predecessor’s full mesh to a sparse circulant topology of transport degree four, trades the anonymity-set size for lower local connectivity. However, depending on the topology, transport connections and number of hops, sparse topologies can reach a anonymity set sizes equal to the full mesh. With $N=64$, the circulant has the same anonymity set size as the full mesh but with a local degree of $d = 16$ instead of $d = 31$.

However, anonymity is costly. The constant-rate cover stream the mixer requires an additional 7 to 9 s of CPU per node each round. However, this depends heavily on the

number of hops K , the overlay reachability defined by transport degree d , and model size. Comparing a mixed scenario against an un-mixed shows, that mixed rounds run approximately ten times the wall-clock of unmixed rounds. This is the price that the mixing layer pays for unlinkability.

Overall, the scalability and anonymity techniques have shown to largely preserve the learning quality. The partial-update ratio trades a small loss in accuracy. For LeNet-5 and MNIST the loss was approximately 2.2 pp for a roughly ten-fold reduction in fragments. Using Krum to mitigate poisoning attacks has shown to result in a stable learning performance. On the directed attacks Krum even held one malicious node past its theoretical Byzantine limit, while just holding the limit with the undirected attack. The framework’s overlay liveness feature ensures reliable model exchange by resending over diverse paths and reconfiguring themselves quickly. Experiments have shown that the nodes are adapting successfully after one round without disturbing the network after a random exit event. A random joining node has shown to quickly catch up and being used as a relay node quickly while getting its own fragments delivered reliably already in the joining round.

Across the joint optimization, the practical operating range of the framework is $K \leq 3$. K has a drastic effect on RTT while O increases at approximately half the analytically worst case slope. So when the same path-entropy is configured by K and O , the evaluation suggest a large O and a small K . The constant output rate μ does not affect the entropy but it can be used to reduce the RTT and make the framework more responsive. However, it should be adapted to the computational resources as lower μ results in high amounts of cover traffic which results in high CPU-time.

7.1 Limitations

The cost of mixing in CPU and RTT limits the practical anonymity level. This, as each added hop, larger shuffled outbox, and a shorter send interval raise cost faster than they increase the entropy. Moreover, the gained anonymity and entropy make the nodes vulnerable to attacks making use of sender anonymity and prevent mechanisms such as reputation systems by design. In addition, the evaluation has shown that a curious recipient can group and track anonymous update very well.

7.2 Future Work

Future work could make the mixnet parameters K , O , and μ adapt automatically to the computational resources of the node or the live network conditions. Furthermore, reducing the CPU time used for the mixing process would improve the efficiency of the continuous background process and free more resources for training and scaling up the scenarios. Moreover, new decentralized and anonymous detection techniques which would help identify malicious behavior could greatly improve the robustness of the framework.

Moreover, actively improving the anonymity of the model fragments themselves would preserve the unlinkability gained through the mixnet, which is otherwise partially lost.

Bibliography

- [1] E. T. M. Beltrán et al., “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.
- [2] 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.
- [3] S. Hunkeler and L. Spitz, *Peer-based mixing for decentralized federated learning*, Zürich, Switzerland: Communication Systems Group, Department of Informatics, Sep. 2025. [Online]. Available: https://files.ifi.uzh.ch/CSG/staff/feng/external/theses/master-project%5C_sandrin-hunkeler%5C_linn-spitz.pdf.
- [4] 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.
- [5] 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.
- [6] M. Moshawrab, M. Adda, A. Bouzouane, H. Ibrahim, and A. Raad, “Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives”, *Electronics*, vol. 12, no. 10, p. 2287, 2023.
- [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] 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.
- [11] V. Mothukuri, R. M. Parizi, S. Pouriyeh, 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.

- [12] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms”, *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] D. Zhu et al., “Srfacs: A secure and robust framework for anonymous communication systems”, *PloS one*, vol. 19, no. 12, e0312817, 2024.
- [17] I. BEN GUIRAT and M. C. Diaz Martinez, “Mixnet optimization methods”, *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 3, pp. 456–477, 2022.
- [18] C. E. Shannon, “A mathematical theory of communication”, *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [19] E. T. M. Beltrán et al., “Fedstellar: A platform for decentralized federated learning”, *Expert Systems with Applications*, vol. 242, p. 122 861, 2024.
- [20] T.-T. Näscher, “Design and implementation of a lightweight decentralized federated learning platform”, M.S. thesis, Universität Zürich, Zürich, Switzerland, Aug. 2025. [Online]. Available: <https://files.ifi.uzh.ch/CSG/staff/feng/external/theses/MA-timothy-naescher.pdf>.
- [21] E. T. M. Beltrán et al., “Fedstellar: A platform for decentralized federated learning”, *Expert Systems with Applications*, vol. 242, p. 122 861, 2024.
- [22] A. Dhasade, A.-M. Kermarrec, R. Pires, R. Sharma, and M. Vujasinovic, “Decentralizepy: A framework for studying decentralized learning”, in *Proceedings of the 2nd European Workshop on Machine Learning and Systems (EuroMLSys)*, ACM, 2023, pp. 34–41.
- [23] G. Syros, G. Yar, S. Boboila, C. Nita-Rotaru, and A. Oprea, *P2pfl: Peer-to-peer federated learning*, Online, Accessed: 2026-01-20, 2023. [Online]. Available: <https://github.com/p2pfl/p2pfl>.
- [24] D. J. Beutel et al., “Flower: A friendly federated learning research framework”, *arXiv preprint arXiv:2007.14390*, 2020.
- [25] P. Foley et al., “Openfl: The open federated learning library”, *Physics in Medicine & Biology*, vol. 67, no. 21, p. 214 001, 2022.
- [26] Google TensorFlow contributors, *Tensorflow federated: A framework for machine learning and other computations on decentralized data*, Online, Accessed: 2026-01-20, 2025. [Online]. Available: <https://github.com/google-parfait/tensorflow-federated>.

- [27] Z. Wang et al., “Federatedscope-gnn: Towards a unified, comprehensive and efficient package for federated graph learning”, in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4110–4120.
- [28] W. Zhuang, X. Gan, Y. Wen, and S. Zhang, “Easyfl: A low-code federated learning platform for dummies”, *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13 740–13 754, 2022.
- [29] C. He et al., “Fedml: A research library and benchmark for federated machine learning”, *arXiv preprint arXiv:2007.13518*, 2020.
- [30] HXSu, *Decentralized federated learning framework based on smart contract and ipfs*, Online, Accessed: 2026-01-20, 2025. [Online]. Available: <https://github.com/HXSu/DFL-Framework>.
- [31] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, “Fate: An industrial grade platform for collaborative learning with data protection”, *Journal of Machine Learning Research*, vol. 22, no. 226, pp. 1–6, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-815.html>.
- [32] A. Lim, S. Jang, and J. Lee, “P2p-fed: A decentralized federated learning platform on structured peer-to-peer systems”, in *IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, 2025.
- [33] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu, “Fedlab: A flexible federated learning framework”, *Journal of Machine Learning Research*, vol. 24, no. 100, pp. 1–7, 2023.
- [34] Z. Wang et al., “Flgo: A fully customizable federated learning platform”, *arXiv preprint arXiv:2306.12079*, 2023.
- [35] Z. Xie, Z. Song, B. Yang, and Z. Zhang, “Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 909–924, 2023.
- [36] S. Biswas et al., “Noiseless privacy-preserving decentralized learning”, *arXiv preprint arXiv:2404.09536*, 2024.
- [37] S. Biswas, A.-M. Kermarrec, A. Marouani, R. Pires, R. Sharma, and M. De Vos, “Boosting asynchronous decentralized learning with model fragmentation”, in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 685–696.
- [38] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, *Qsgd: Communication-efficient sgd via gradient quantization and encoding*, 2017. arXiv: 1610.02132 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1610.02132>.
- [39] A. Tyurin and P. Richtárik, *Dasha: Distributed nonconvex optimization with communication compression, optimal oracle complexity, and no client synchronization*, 2022. arXiv: 2202.01268 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2202.01268>.
- [40] P. Elias, “Universal codeword sets and representations of the integers”, *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 194–203, 1975. DOI: 10.1109/TIT.1975.1055349.

- [41] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, “Robust and communication-efficient federated learning from non-i.i.d. data”, *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, pp. 1–14, Nov. 2019. DOI: 10.1109/TNNLS.2019.29444481.
- [42] R. Rice and J. Plaunt, “Adaptive variable-length coding for efficient compression of spacecraft television data”, *IEEE Transactions on Communication Technology*, vol. 19, no. 6, pp. 889–897, 1971. DOI: 10.1109/TCOM.1971.1090789.
- [43] I. Hegedűs, G. Danner, and M. Jelasity, “Decentralized learning works: An empirical comparison of gossip learning and federated learning”, *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021.
- [44] H. Nguyen, H.-T. Nguyen, P. K. Donta, S. Pirttikangas, and L. Lovén, “Graph-based gossiping for communication efficiency in decentralized federated learning”, *arXiv preprint arXiv:2506.10607*, 2025.
- [45] Y. Chen et al., “Towards practical overlay networks for decentralized federated learning”, *arXiv preprint arXiv:2409.05331*, 2024.
- [46] 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.
- [47] Y. Chen, Y. Su, M. Zhang, H. Chai, Y. Wei, and S. Yu, “Fedtor: 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.
- [48] 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.
- [49] 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.
- [50] 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.
- [51] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent”, in *Advances in Neural Information Processing Systems*, I. Guyon et al., Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

Abbreviations

API	Application Programming Interface
CA	Certificate Authority
CFL	Centralized Federated Learning
CPU	Central Processing Unit
CSV	Comma-Separated Values
Dasha	Distributed Nonconvex Optimization with Communication Compression
DFL	Decentralized Federated Learning
DHT	Distributed Hash Table
FL	Federated Learning
GPU	Graphics Processing Unit
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
IID	Independent and Identically Distributed
IP	Internet Protocol
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
LZ4	Lempel–Ziv 4
LZMA	Lempel–Ziv–Markov Chain Algorithm
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MPI	Message Passing Interface
MQTT	Message Queuing Telemetry Transport
mTLS	Mutual Transport Layer Security
OSX	Open Site Exchange
P2P	Peer-to-Peer
QSGD	Quantized Stochastic Gradient
QUIC	Quick UDP Internet Connections
REST	Representational State Transfer
RPC	Remote Procedure Call
RSA	Rivest–Shamir–Adleman
RTT	Round-Trip Time
S3	Simple Storage Service
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SSE	Server-Sent Events
STC	Sparse Ternary Compression

SURB	Single-Use Reply Block
TCP	Transmission Control Protocol
TLS	Transport Layer Security

List of Figures

4.1	Edited Architecture overview from [3].	21
4.2	Circulant graph $C_9(\{1, 3\})$ with degree 4 and diameter 2. The offset of 3 was chosen as it results in a minimal diameter for the degree and number of nodes.	24
4.3	Ring lattice graph $C_9(\{1, 2\})$ with degree 4 and diameter 2. Compared to Figure 4.2 the diameter cannot be optimized as the offset must be consecutive.	24
4.4	Hypercube graph Q_3 with $N = 8$, degree 3, and diameter 3. Each node is labeled by a 3-bit identifier. Two nodes are connected if their identifiers differ in exactly one bit.	25
4.5	Overlay neighborhood of node 0 in the circulant graph $C_{12}(\{1, 5\})$ at $K = 2$ hops. The origin (red) and the exchange peers reachable in exactly two hops (green) form the local overlay topology. The edges show the underlying transport topology.	25
4.6	Per-peer session state. A peer is sent to while <i>unconfirmed</i> or <i>live</i> and excluded only while <i>unreachable</i> . The flip to <i>unreachable</i> requires resend exhaustion with no acknowledgment since the resend, and a later acknowledgment recovers the stored fragments.	28
5.1	Per-round encoding of a model update. The parameter vector is flattened, sliced into fixed-size fragments, quantized to 8 bits per element with stochastic rounding, generically compressed, and wrapped one fragment per Sphinx packet for transmission on the overlay.	33
5.2	High-level layout of a single fragment. The MessagePack envelope, sender round, fragment index, per-fragment $v_{\min}$ and $v_{\max}$, and the uint8 weight payload are zlib-compressed together and placed inside a fixed-length Sphinx packet body.	34

5.3	Reverse-index inside the <code>MessageStore</code> . Each payload hash h_X (SHA-256 prefix) keys the set of retransmissions of the same payload. Each is recorded as a(<code>peer</code> , <code>fragment_idx</code> , <code>attempt</code>) tuple. A single returned SURB acknowledgment for h_A clears every tuple under h_A at once. This, so one successful return is enough, regardless of how many retransmissions are still on the way.	37
5.4	Round-phase timing under (a) normal completion and (b) stall escape. The fragment-collection phase ends as soon as the expected fragment count is reached (a) or, once the 50% completeness floor is met, when the 30-second stall window fires without further intake (b). A small 5-second grace window is awaited before the round finalizes.	38
5.5	The form at the top sets the configuration and the topology graph previews the mesh.	42
5.6	The <code>IndicatorService</code> panel precomputes the scenario KPIs before any container starts: transport connections and diameter, overlay degree, path and relay entropy, anonymity-set size, fragments and bytes per round, and the mixnet send rate.	42
5.7	The node table and the metric charts below stream live values over server-sent events once the run is underway.	43
6.1	Accuracy convergence across N at the baseline. Solid lines with markers show aggregated accuracy after each round's combined update; dashed lines show local accuracy after the local training step, before aggregation. Lines are the mean across nodes. The $N=6$ (MNIST panel) is run on full mesh as a small- N anchor, $N \in \{16, 32, 64\}$ are circulant.	47
6.2	Communication volume by model class.	49
6.3	Per-node and aggregate communication volume as N grows.	49
6.4	Per-node CPU by phase across graph degree	51
6.5	Mean per-node CPU time per round (s) and mean memory (MB) across graph degree $d \in \{4, 16, 32, 48, 62\}$ at $N=64$, $\mu=0.1$ s, $K=2$. Circulant and ring lattice are plotted as lines, the mean over five runs per point.	51
6.6	Two-hop reachable set and its per-link ratio across graph degree d at $N=64$, circulant versus ring lattice. Left: peers reached within two hops, the set a node aggregates over and the anonymity set its fragments are hidden in. The dotted line marks the whole graph $N-1$. Right: reachable peers per connection (reach/d), the share of the graph's traffic each link carries. The diameter-minimising circulant reaches the whole overlay graph at $d=16$ and peaks in per-link concentration there, while the ring lattice reaches half the graph at $d=16$ and saturates at $d=32$. Both panels are deterministic from the topology generator.	52

6.7	LeNet-5 versus SqueezeNet mean round duration and memory across the degree sweep at $N=64$, $\mu=0.1$ s, $K=2$, $O=100$, MNIST, 8-bit quantization. SqueezeNet spans $d \in \{4, 16, 32, 48, 62\}$ and the full mesh. LeNet-5 covers the dense degrees $\{32, 48, 62\}$ and the full mesh. Each marker is the per-degree mean over the available topologies and five runs each.	53
6.8	SqueezeNet per-node CPU broken down by phase across circulant degrees $\{4, 16, 32, 48, 62\}$ at $N=64$. Layout matches Figure 6.4.	53
6.9	Relay entropy and end-to-end overlay path entropy as the outbox size O varies, one line per hop count K	55
6.10	Anonymity-set size as a function of N for each topology of Section 5.4.1, $K=2$, target degree four. The full-mesh anonymity set equals $N-1$ at any K . The structured topologies fall below $N-1$ once N exceeds the K -hop reach at the configured degree.	56
6.11	Cover fraction across the K sweep at $\mu=0.1$ s, $N=32$, $O=100$, mean over five runs, $K \in \{1, \dots, 5\}$	57
6.12	Mixing cost across graph degree $d \in \{4, 8, 12, 16, 20, 24, 28\}$ at $N=32$, $K=2$, $O=100$, $n_{\text{rounds}}=5$. Only <code>mix_enabled</code> differs between the two arms. The per-round CPU delta is one dimension of the anonymity cost, the wall-clock per-round delta the other.	58
6.13	Mixing cost across network size $N \in [6, 50]$ at the baseline of Table 6.1 with degree four. The per-round CPU delta is similar to the $d=4$ point of Figure 6.12, around $+7.6$ s/r.	59
6.14	Curious-recipient content attack on the heterogeneity α against partial-update ratio r grid. Each cell shows its accuracy against a $1/8$ chance, and is the mean over repeated draws. (a) Linkage, the accuracy of assigned identities to the buckets. (b) Tracking, the share of fragments being placed in the correct bucket.	60
6.15	Marginal entropy gain per K-step at the baseline of Table 6.1. Each K-step buys $\log_2 O \approx 6.64$ bits. The bar height is that gain divided by the RTT or CPU difference between consecutive K values. μ contributes zero bits per RTT-second across its sweep, and O moves entropy only within a given K , so neither appears as bars.	61
6.16	Measured bandwidth and RTT as a function of μ at $N=32$, $K=2$, $O=100$. Shaded bands are the min-max across all runs at each μ . The line is the mean across seeds.	62
6.17	Cover fraction and per-round CPU time as μ varies at the baseline of Table 6.1.	63
6.18	Cover-traffic detail at $N=32$, mean over five runs per point.	63
6.19	Per-node CPU time per round and memory across the μ sweep at the baseline of Table 6.1.	63

6.20	Measured average RTT as K and O vary. Shaded bands are the min–max across seeds at each parameter value. The line is the mean across seeds. . .	65
6.21	Cover fraction, per-round CPU time, and analytical path entropy as K varies at the baseline of Table 6.1.	65
6.22	Backlog and round-duration response across the full $K \in \{1, \dots, 5\}$ range at $N=32$, $\mu=0.1$ s, $O=100$. The $K=5$ point runs at $n_{\text{rounds}}=5$, comfortably longer than the per-direction half-cycle $KO\mu=50$ s. Outstanding fragments and round duration grow monotonically with K across the swept range. . .	66
6.23	Per-round CPU and memory across the O sweep (top row) and the K sweep (bottom row) at the baseline of Table 6.1.	67
6.24	Accuracy convergence by topology at $N=32$, target degree four (full-mesh degree $N-1$). Solid lines with markers show aggregated accuracy after each round’s combined update. The dashed lines show local accuracy after the local training step, before aggregation. Lines are the mean across nodes and the five runs.	68
6.25	Accuracy convergence under 8-bit and 32-bit quantization	69
6.26	Round-ten accuracy across r at $N=32$, mean over five runs.	70
6.27	Per-round accuracy trajectory at $r \in \{0.10, 0.25, 0.50, 0.75, 1.00\}$, $N=32$, mean over five runs.	70
6.28	Round-ten accuracy across r on MNIST at $N=32$, circulant versus full mesh, mean over five runs.	70
6.29	Round-ten aggregated accuracy across the heterogeneity sweep at $N=32$, centred on the near-IID anchor ($\alpha=10$, $r=1.0$). The $\alpha=10$ row reuses the Section 6.5.3 corpus.	72
6.30	Per-round aggregated accuracy by Dirichlet α at fixed $r=1.0$, $N=32$, MNIST, FedAvg, circulant degree four. Lines are means across nodes. The bands are min/max across nodes.	72
6.31	Exit scenario topology partition (left) and per-class fragment completeness (right). One peer leaves after round five at $N=32$, degree four, two hops, $n_{\text{rounds}}=10$, single run. Lines are within-class medians. The grey dotted vertical in the right panel marks the round-five exit.	73
6.32	Join scenario at $N=32$, circulant degree four, $n_{\text{rounds}}=10$, single run, with one peer arriving at round three. Solid lines are the late joiner. the green dashed reference is the mean over the stable peers. Left: per-round fragment intake as a fraction of the stable peers’ mean. Right: the joiner’s local accuracy against the stable peers’ aggregated mean. The joiner is excluded from the stable set. Tts samples begin at round three. The grey dotted vertical marks the join event at round three.	74
6.33	Round-ten accuracy versus Byzantine count	75

List of Tables

3.1	Communication protocol	11
3.2	Exchange payload and compression.	13
3.3	Spatial and temporal communication architecture.	15
3.4	Sender unlinkability: protection level and how it is evaluated.	15
5.1	Status of each implementation component relative to the inherited framework [3]. Inherited components are reused without major changes. Refactored components mainly keep their external interface but were substantially rewritten. New components are contributions of this work.	32
6.1	Baseline experimental scenario.	46
6.2	Aggregated accuracy at round one and round ten across N on both datasets, at the baseline of Table 6.1. $N=6$ is the full-mesh small- N anchor (MNIST only); $N \in \{16, 32, 64\}$ are circulant. Mean across nodes; five runs per point at $N \in \{6, 64\}$, single run at $N \in \{16, 32\}$	47
6.3	Per-node CPU time per round and mean memory for the circulant topology across graph degree d , LeNet-5 at $N=64$, $\mu=0.1$ s, $K=2$, mean over five runs per point. The CPU peak at $d=16$ is the circulant run.	51
6.4	Mixing cost at the sparse, CPU-peak, and dense configuration of the degree sweep, $n_{\text{rounds}}=5$, at the baseline of Table 6.1.	58
6.5	Cost endpoints for each mixnet parameter at $N=32$, with the other two parameters held at the baseline of Table 6.1. RTT is the median value. CPU and memory are per-node means per round.	62
6.6	Round-ten accuracy under FedAvg and Krum across the three attacks at $N=20$, full mesh. Clean is the shared $b=0$ baseline per aggregator. Krum's limit $n > 2f + 2$ caps at $b \leq 8$	75
6.7	Configurable properties of this framework being compared against the predecessor work of Chapter 3. Predecessor values are quoted from the predecessor's evaluation. This-work values are taken at the baseline description of Table 6.1.	76

Appendix A

Source Code and Reproducibility

The framework, evaluation scripts, and data needed to reproduce the results are available at GitHub Release v1.0, which contains:

- Framework source code (Manager, Nodes, and Frontend).
- Raw evaluation data and the Jupyter notebooks that process it.
- Figures generated by those notebooks.
- Slides for the intermediate and final presentations.
- A PDF export of this thesis.