Evaluating GossipSub for Data Availability Sampling Under Ethereum Consensus Deadlines

Master's by Research: Computer Science

Student Name: Hoang Ky Trinh
Student ID: 36753291
Supervisor: Dr Onur Ascigil

School of Computing & Communications | Lancaster University

School of Computing and Communications
Lancaster University
Dec 2025

Declaration

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted work, I consent to this being stored electronically and copied to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Name: Trinh Hoang Ky

Date: 28/02/2025

Abstract

This thesis investigates how GossipSub configuration choices influence data dissemination for a Data-Availability Sampling (DAS) workload under strict consensus time bounds. Using a configurable PeerSim-based simulator, we model a FullDAS-like setting in which a block producer erasure-codes a blob into a 2D extended matrix of share segments, partitions row/column segments into topics (custody-style sharding), and executes a two-phase workflow: *seeding*, where share segments are disseminated over topic meshes, and *sampling*, where validators must retrieve uniformly random share segments within a $T_{\mathrm{DAS}} = 4\,\mathrm{s}$ deadline. We systematically vary topic granularity (TOPICS), segmentation (segment amount, SA), replication (K-copies), bandwidth caps, and omission fault rate α, and measure phase success rates, completion-time distributions (with emphasis on tail latency), bandwidth consumption, and duplication overhead. The results show that segmentation and replication dominate performance and overhead: increasing SA from coarse to moderate values reduces duplication with diminishing returns beyond SA $\approx$ 1-16, while larger K increases redundancy and overhead and mainly provide a robustness margin under adverse conditions. Seeding completes quickly and remains resilient for SA≥ 4 even at high omission, whereas sampling is tail-latency dominated and degrades more sharply as α increases, leading to widespread deadline misses near $\alpha = 0.5$. Based on these findings, we adopt TOPICS= 256, SA= 8, $K = 4$, and a conservative per-node bandwidth cap of 60 Mbit/s for faulted multi-slot experiments to isolate GossipSub dynamics from bandwidth saturation.

Contents

1 Introduction

Blockchains face a fundamental scalability bottleneck because each validating node must independently execute the transactions in every block to check that the proposed state transition is correct. In other words, when a block claims that applying its transactions yields a new state, nodes do not accept this claim on trust. They rerun the transaction logic locally and verify that the resulting state matches the block. This repeated execution across many nodes limits throughput, since block size and block frequency must remain within what typical nodes can process and propagate through the network.

Throughout this dissertation, *Layer 1* (L1) refers to the base blockchain protocol that provides consensus, settlement, and the canonical record of blocks and data commitments. In Ethereum-style proof-of-stake systems, L1 consensus is maintained by *validators*: nodes that check proposed blocks and publish attestations, or votes, within fixed time intervals called *slots*. *Layer 2* (L2) refers to protocols, especially rollups, that execute many transactions outside the L1 execution path and then publish commitments and sufficient transaction data back to L1. This separation reduces the amount of execution work performed directly by L1 validators, but it does not remove the need for public data: if the transaction data behind an L2 state update is unavailable, other participants cannot reconstruct, verify, or challenge that update.

To reduce the L1 execution burden, many systems therefore scale through L2 rollups. A rollup executes transactions off-chain, computes a state update, and posts a commitment to that update on the L1 chain for settlement [2]. Crucially, the rollup must also make the underlying transaction data available. If the rollup operator kept this data private, other participants would be unable to reconstruct the rollup state and independently verify or challenge the claimed transition. As rollups scale, the main bottleneck therefore shifts from executing transactions to ensuring that large volumes of rollup data are disseminated and retrievable in time. This is the *data availability* problem: data may exist somewhere, but the system is only secure if honest verifiers can retrieve enough of it when needed.

This shift motivates Ethereum's roadmap towards higher throughput data channels, such as blob-carrying transactions, which provide dedicated bandwidth for rollup data without permanently increasing the execution layer state [3]. However, increasing data throughput intensifies the networking challenge, since blob data must be disseminated quickly and widely enough that validators and other verifiers can retrieve it within strict time budgets.

A key challenge with large blobs is that it is unrealistic for every validator to fully download the data for every block. Data Availability Sampling (DAS) avoids this full download by combining erasure coding with probabilistic checking. Thereby, the blob is erasure-encoded (often via a two-dimensional Reed Solomon style extension) into many small share segments, such that the original data can be reconstructed from any sufficiently large subset of share segment[17][21].

Validators then sample: instead of downloading the entire blob, each node requests only a small number of randomly chosen share segments. If a producer withholds a meaningful fraction of the data—i.e., some share segments are not disseminated or are not served upon request—then honest validators are likely to observe missing samples, and the blob can be

treated as unavailable.

Crucially, this sampling must be complete before the consensus vote deadline. In a slot-based consensus protocol, validators must broadcast their attestation/vote within a strict time window of the slot so that it can propagate through the network and be considered in the next consensus step. This creates an effective DAS deadline: the peer-to-peer layer must disseminate share segments widely and answer sample requests quickly enough that validators can form and publish their availability vote on time [17].

Modern blockchain networks therefore rely on gossip-based broadcast rather than client-server distribution. Gossip protocols scale naturally because the dissemination load is segmented across peers, and messages can reach many recipients over multiple hops without requiring a single high-capacity server [11]. In Ethereum's networking stack, the primary pub/sub protocol for dissemination is GossipSub [19]. GossipSub combines two complementary strategies. First, it uses an eager push path over a stable per-topic mesh. Nodes maintain a mesh of size D and forward full messages immediately to mesh neighbours to achieve low latency propagation. Second, it uses a lazy pull repair path outside the mesh. Nodes advertise compact metadata indicating which messages they have seen, and peers that learn about a message this way can request the full content if they missed it on the eager path. This design reduces unnecessary full message flooding while still enabling recovery from missed deliveries.

Because GossipSub is an open peer-to-peer protocol where any node can participate, it must tolerate faulty behaviour. In this thesis, we focus on omission faults, where a fraction of nodes do not forward share segments and do not serve requested share segments. We study how GossipSub configuration choices, including topic granularity, mesh degree parameters, and lazy repair fanout, affect DAS-related outcomes such as success rate, completion latency, bandwidth cost, and duplication overhead under these fault conditions.

1.1 Background

From execution bottlenecks to data availability bottlenecks. Traditional blockchains face an *execution bottleneck* that limits throughput because validators must re-execute and verify the transactions in each block in order to validate it. Layer-2 rollups reduce this cost by executing transactions *off-chain* (outside the Layer-1 chain) and then submitting a compact commitment to the result back to Layer-1. However, moving execution away from Layer-1 introduces a new verification problem: if the rollup operator kept the transaction data private, other parties could not independently check whether the state transition was correct. To preserve *independent verification* (i.e., anyone can validate the rollup's correctness without relying on a trusted intermediary), rollups publish the transaction data—or sufficient call data to Layer-1, enabling others to reconstruct the rollup state, verify proofs, and detect invalid behaviour [2].

As rollups scale, the limiting factor shifts from *computing* transactions to *distributing* large volumes of rollup data across the network. This is the data-availability (DA) bottleneck. Ethereum reduces the long-term storage burden using blob-carrying transactions: blocks reference ephemeral *blobs* that remain available for a limited time rather than being stored

permanently in the canonical state [3][18]. In this model, the security requirement is no longer that every validator re-executes everything; instead, it becomes critical that the underlying *peer-to-peer* network can disseminate blob data quickly and broadly so that any validator who needs to check availability can retrieve it within the protocol's deadline.

Why gossip broadcast? To make data availability practical at scale, systems first apply erasure coding (e.g., Reed-Solomon with a two-dimensional extension) to transform a blob into many small *share segments* (Figure 1). Coding makes availability *recoverable*: the original blob can be reconstructed from any sufficiently large subset of share segments, so availability depends on whether share segments are broadly retrievable from the network rather than whether a single party serves the entire blob.

At a high level, this thesis separates the DAS workflow into two phases. In the *seeding phase*, the block producer injects erasure-coded share segments into the peer-to-peer network so that custodians can store and forward them. In the *sampling phase*, validators request randomly chosen share segments and decide whether enough requested data was retrievable before the deadline. Section 3.5 formalises this two-phase workflow, but the distinction is introduced here because it motivates why the dissemination layer must provide both fast initial spread and reliable repair for missing shares.

A client-server distribution model is a poor fit for this setting. Concentrating dissemination on a small set of servers creates throughput bottlenecks, increases the impact of failures or censorship, and performs poorly under churn because servers must continuously track and serve a large, dynamic audience. In contrast, gossip-based peer-to-peer (P2P) dissemination spreads the load across the network: each node forwards to a limited number of peers, and fanout over multiple hops yields rapid, probabilistic coverage. This provides path diversity—multiple independent routes to obtain the same share segment—and improves robustness, since share segments can be retrieved from alternative peers rather than a single origin [11].

The main caveat is that gossip introduces *redundancy*: forwarding the same items along multiple paths can inflate bandwidth usage. Modern protocols therefore aim to *control* redundancy by limiting fanout and using metadata-driven pull for recovery, rather than flooding full payloads to everyone. GossipSub follows this approach, targeting low latency while keeping duplication bounded through a stable forwarding mesh and controlled metadata exchange.

Protocol Design and Faulty Models. In open P2P networks, the broadcast layer must operate correctly despite unreliable participants. In this thesis, we evaluate robustness under a *restricted fault model* that matches our experiments: (i) *crash/offline* faults, where nodes stop participating, and (ii) *omission* faults, where faulty nodes do not forward received data and do not respond to share segment requests during sampling. These behaviours directly stress DA dissemination because missing forwards reduce coverage and increase the probability that samplers cannot retrieve requested share segments within the deadline. We do *not* model identity-based attacks (e.g., Sybil) or content-spam attacks; the focus is specifically on how omission and non-participation affect dissemination efficiency and DA-

related outcomes [5][13].

Blockchain Networking Basics Blockchain systems run publicly on peer-to-peer (P2P) networks, which are inherently dynamic environments in which nodes can join, leave, or fail at any time. Consequently, networking protocols need to be able to function in challenging environments. Two factors typically control performance: latency (propagation time) and bandwidth (data throughput). As a result, the network layer's design goals for block and blob distribution are to minimise end-to-end latency to meet DAS time constraints, control message duplication to save bandwidth, and ensure mesh and data resilience against Byzantine errors. This network layer has emerged as one of the most critical components of modern rollup-centric systems for ensuring data availability [3][11].

Gossip Protocol Foundations The Gossip protocol was designed to meet the demands for increased throughput, reduced latency, and improved consensus mechanisms. In Demers *et al.* [4], it was demonstrated that periodic, pairwise data exchange with *anti-entropy* (pull) or *rumour-mongering* (push) could maintain data consistency over replication, thus increasing resilience [4]. Later work formalised the "random phone call" model, demonstrating that with a small amount of fanout, continuous push-pull could deliver messages to n nodes in $O(\log n)$ cycles. This established the basic timing and communication exchanges that served as the foundation for later designs [10]. Eugster *et al.* [6] classifies this design space, arguing that gossip protocols are scalable and resilient to node outages but at the cost of higher data distribution, duplication of data, and a slight increase in latency [6].

GossipSub Overview Traditional epidemic push-pull gossip is robust but can be bandwidth-inefficient because it tends to create many redundant deliveries. GossipSub, which is used in Ethereum's consensus-layer networking stack, improves efficiency by combining a low-latency *eager-push* path with a bandwidth-saving *lazy-pull* recovery path [19][20]. For each topic, each node maintains a relatively stable forwarding *mesh* with a target degree D, and eagerly forwards full messages to its mesh neighbours. To keep the mesh stable, nodes enforce bounds D_{low} and D_{high}: if the mesh falls below D_{low} the node, it *grafts* new peers, and if it grows above it, D_{high} it *prunes* peers.

Outside the mesh, nodes exchange compact metadata (e.g., message identifiers) to advertise recently seen messages. Peers that learn about a message via metadata can then request the full content if they missed it on the eager path, implementing a lazy pull-based repair mechanism. To reduce the impact of unreliable neighbours, GossipSub also includes a peer-scoring system: peers that consistently help propagate valid messages retain mesh membership, while peers that appear unhelpful are pruned over time [12].

Proposer-Builder Separation (PBS) Proposer-builder separation (PBS) was established as part of the new transition from Ethereum Proof-Of-Work (PoW) to Proof-Of-Stake (PoS), which split the usual validators, which previously served as both block builders and validators. PBS aimed to decrease the enormous load on centralised networks, particularly

during block creation, while maintaining high transparency [8][9][15]. Because integrating complete PBS systems into the main protocol was too complicated during the transition, the Ethereum community opted for an off-protocol variant known as Maximal Extractable Value Boost (MEV-Boost), which was used for Builder. Validators will experience less stress as they no longer need to create the block but rather sign and offer it to the network [9].

Data-Availability Sampling (DAS) Data-Availability Sampling (DAS) was introduced by Al-Bassam *et al.* [1] in 2018 to let validators assess *data availability* without downloading an entire blob. The main idea is to combine erasure coding with random sampling. First, the blob is split into equal-sized pieces (share segments/cells) and arranged as a 2D matrix. A two-dimensional Reed-Solomon erasure code is then applied to extend the matrix by adding parity in both dimensions, producing a larger *extended* matrix of share segments [1][21]. Figure 1 illustrates this extension: the original data occupies the top-left region, while the remaining cells are additional erasure-coded share segments.

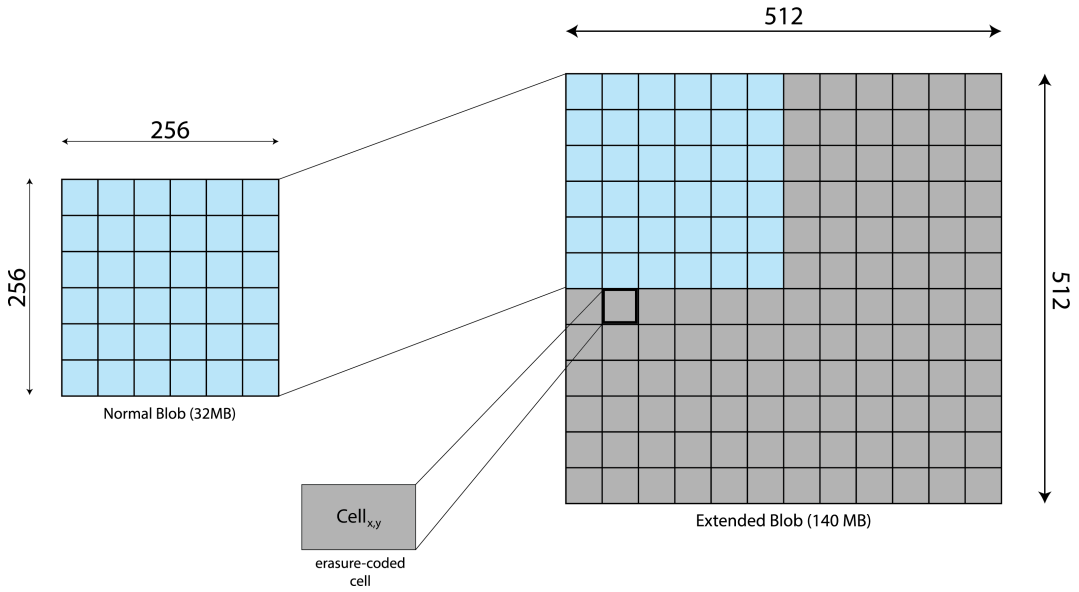


Figure 1: A $k \times k$ blob matrix extended to a $2k \times 2k$ matrix using the 2D Reed-Solomon construction $E = GBG^\top$. The original data occupies the systematic region, and the remaining cells are parity share segments.

The reason for erasure coding is *recoverability*. Reed-Solomon coding provides a threshold property: for a row (or column) that is extended from k to $2k$ share segments, any k available share segments in that row (or column) are sufficient to reconstruct the missing share segments. Intuitively, this means reconstruction is possible as long as fewer than half of the shared segments in a row/column are missing. Figure 2 visualises this intuition: when too many extended share segments are missing in the same region (e.g., beyond the 50% threshold), the original data cannot be reconstructed.

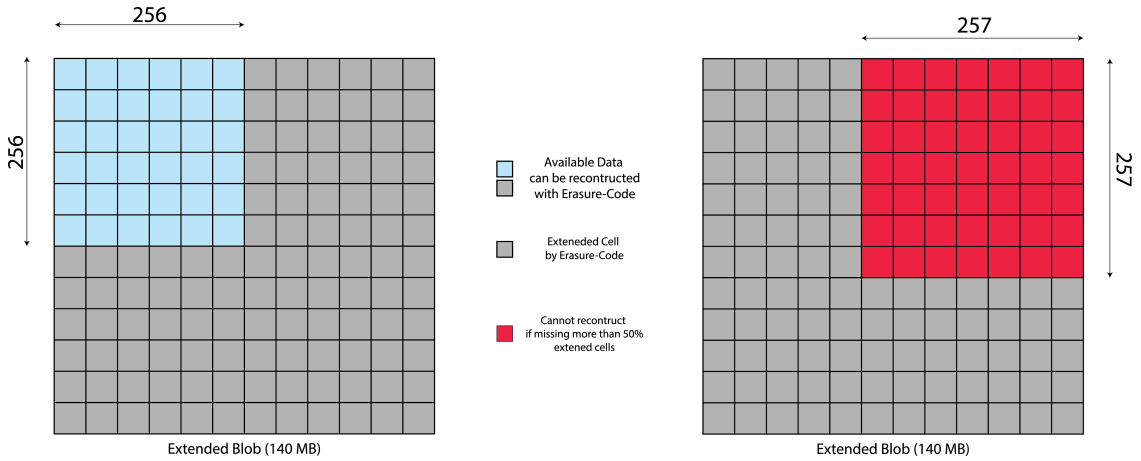


Figure 2: Reconstruction intuition under 2D erasure coding: recovery succeeds when enough share segments are available but fails if too many share segments are missing (e.g., more than 50% of the extended share segments for $k \to 2k$ extension).

After encoding, validators perform *sampling* rather than a full download. Instead of fetching all share segments from the extended matrix, each validator requests only a small number of uniformly random cells and treats repeated success as evidence that the blob is widely retrievable. Figure 3 shows an example where a validator samples 73 random cells. If a block producer (or faulty peers) withholds a meaningful fraction of share segments, then a validator's random requests become more likely to hit missing cells, causing sampling failures. In this way, DAS provides a probabilistic guarantee: increasing the number of samples reduces the chance that withheld data remains undetected. Because validators must reach an availability decision within the protocol's time budget, the peer-to-peer dissemination layer becomes critical: it must distribute share segments broadly and respond to sample requests quickly enough for validators to complete sampling on time.

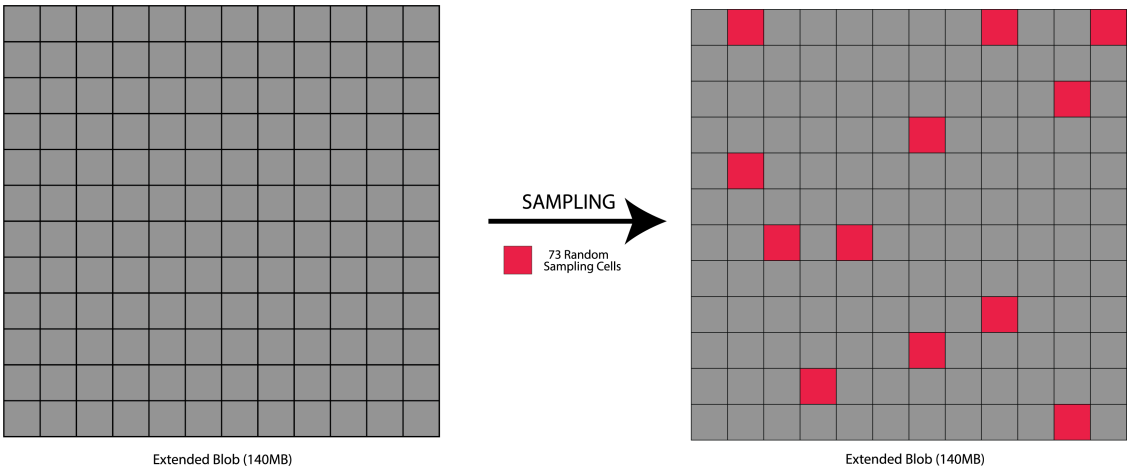


Figure 3: Data-Availability Sampling (DAS): a validator queries a small set of uniformly random cells (e.g., 73 samples) from the extended blob.

Why 12-second slot and ∼4-second DAS deadline? Ethereum progresses in fixed time steps called *slots*, each lasting 12 seconds. In each slot, a new block is proposed and needs to spread through the network, and then many validators vote to confirm what they saw. Validators cannot wait until the end of the full 12 seconds to vote, because the votes themselves must also travel through the network and be combined before the next slot begins. In practice, this means the network expects validators to make their decision and produce a vote early in the slot – roughly within the first third of the 12-second window (about 4 seconds). Data Availability Sampling (DAS) is part of that decision: a validator samples small pieces of the block's data to gain confidence that the full data is actually retrievable. Figure 4 summarises this timing model: the producer begins disseminating share segments at the start of the slot, seeding provides the initial spread of data, and validators must complete sampling before the approximate 4 s DAS deadline shown by the red marker.

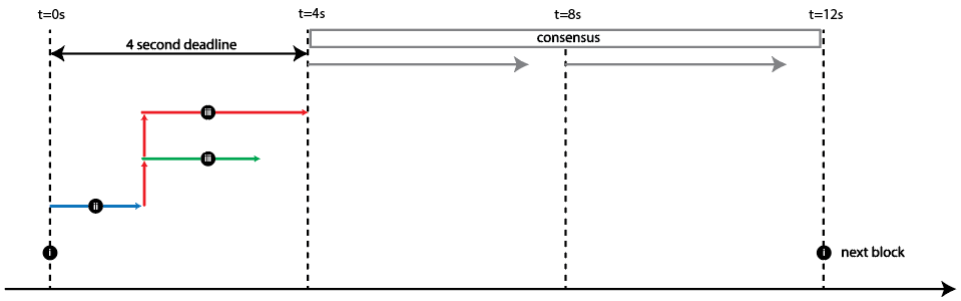


Figure 4: The timeline is in a slot, starting at (i) where the block producer distributes the segments to a set of validators at t=0s, after, at (ii) it will start the seeding phase and request the missing segment at (iii, green line). Along with since the first seed arrive at (ii), it also start sampling 73 random cells at (iii, red line)

GossipSub under Faulty Situation. GossipSub is a multi-layered pub/sub protocol that combines two complementary dissemination mechanisms to achieve reliable propagation while controlling bandwidth. For each topic, every node maintains a relatively stable *mesh* of peers with a target size D. Messages are delivered on this mesh using an *eager-push* strategy: when a node receives a new message, it immediately forwards the full message to its mesh neighbours, which minimises propagation delay along the primary forwarding paths.

To avoid broadcasting full messages to every peer, GossipSub complements eager-push with a *lazy-pull* mechanism for peers outside the mesh. Instead of sending the entire payload, nodes periodically gossip compact metadata (e.g., message identifiers) indicating which messages they have recently observed. Peers that learn about a message through this gossip can then *pull* it by requesting the full content if they have not already received it via the mesh. This two-tier design spreads messages quickly through the mesh while allowing additional peers to recover missing data on demand, reducing redundant transmission and saving bandwidth compared to indiscriminate flooding.

But it also causes a problem when there are faulty nodes in the mesh. GossipSub uses a peer scoring system for solving this issue. Through the use of a dynamic reputation mechanism,

nodes in the mesh compute scores for neighbour peers over time in response to their actions [19]. Further, more reliable nodes (honest nodes) can be grafted (added) to, while low-scoring or unreliable nodes are then pruned (dropped) from the mesh. Despite the robustness of this scoring system, it can still be exploited. For instance, faulty/omission nodes can flood the topic with spam messages or omit messages [12].

The research will also evaluate the resilience of these designs against particular, clearly defined faulty attacks. We particularly use and study forward omission attacks (both full and partial) to mimic faulty peers withholding data. Next, we look at how the attacks affect the success rate of Data Availability Sampling (DAS), how the protocol parameters (D, D_Low, and D_High) affect the lazy gossip fanout (D_Lazy), and how the lazy gossip fanout (D_Lazy) affects the success rate of DAS. The goal is therefore not to prove that GossipSub is sufficient for all real deployments but to isolate how its configurable mechanisms affect DAS-relevant behaviour in a controlled simulator. The results should be read as evidence about dissemination dynamics under the stated assumptions, not as a full deployment-level security proof.

1.2 Problem Statement

Layer 2 rollups increase the amount of data that must be disseminated through the base layer networking stack. In particular, rollup blobs and their erasure coded share segments must reach a large set of validators quickly enough for Data Availability Sampling (DAS) to complete within its time budget. This makes the peer to peer broadcast layer a critical dependency for security, since DAS assumes that honest validators can retrieve arbitrary share segments on demand, even when some peers are faulty and do not cooperate.

GossipSub is widely deployed as the pub sub layer for Ethereum style dissemination, but its behaviour under DAS motivated workloads depends heavily on configuration. In practice, parameters such as the mesh degree targets, the mesh maintenance bounds, the lazy repair fanout, and topic granularity determine how quickly share segments spread, how much redundant traffic is created, and how resilient dissemination remains when some peers omit forwarding. Because these parameters interact in non trivial ways, it is not obvious which operating region provides a good balance between meeting DAS deadlines and keeping bandwidth overhead manageable.

This thesis therefore studies the following problem. Given a network of n peers running GossipSub, where a fraction α of peers follow an omission fault model, what parameter settings provide reliable and timely dissemination of erasure coded share segments, while keeping network overhead and duplication bounded, and how do these outcomes change as the network scales.

Central question How do key GossipSub parameters affect DAS relevant outcomes under omission faults, in terms of performance, overhead, and scalability.

(i) **Performance**: How often honest validators complete the seeding and sampling objectives within the configured DAS deadline, and how completion time varies across the

network.

(ii) **Overhead**: How much bandwidth is consumed per node, and how much duplication is generated due to redundant deliveries and request retries.

(iii) **Scalability**: How performance and overhead change as the number of peers n increases under the same workload and fault level.

Fault model considered To keep the evaluation aligned with the simulator and the experiments reported in this thesis, we focus on omission style faults only. A fraction α of nodes fails to forward received share segments, and may also fail to serve requested share segments during sampling. We do not model identity based attacks or spam flooding in the experiments. The goal is to understand how omission alone stresses dissemination and how GossipSub configuration can mitigate its impact.

1.3 Threat Model

This thesis evaluates GossipSub under an *omission-fault* model. The purpose of the threat model is to define the faulty behaviour used in the simulator; it is not intended to prove a consensus-level Byzantine fault-tolerance threshold.

Network and timing model. We model the peer-to-peer dissemination layer as a discrete-event, bounded-delay simulation. Honest nodes communicate through GossipSub topic meshes, and message delivery is scheduled according to the simulator's configured transport delay, bandwidth limits, heartbeat period, request timeouts, and retry rules. This is therefore not a fully asynchronous model with an adversarial message scheduler. Instead, it is a bounded-timing evaluation model designed to test whether dissemination and sampling complete within an Ethereum-style slot budget. The main timing quantities are the 12 s slot and the DAS decision deadline $T_{\mathrm{DAS}} = 4$ s.

Fault placement and adversary timing. For each simulation run, a fraction α of validators is selected as faulty before the execution begins. The faulty set remains fixed during the run; hence, the experiments use a *static* adversary rather than an adaptive adversary that can corrupt new nodes after observing the execution. Faulty nodes may coordinate their omission behaviour, but they do not gain special network privileges and they participate in the same topic and mesh-selection process as honest nodes.

Adversary knowledge. The adversary is assumed to know the protocol, the public configuration parameters, the topic layout, and which nodes it controls. Faulty nodes observe the messages and requests they receive locally. However, the adversary is not modelled as knowing future honest random choices before they are made, such as a validator's future sampling targets before those requests are issued. The adversary also does not control the simulator's network scheduler or the underlying routing between honest nodes.

Adversarial capabilities. A faulty validator may perform two omission behaviours:

1. *Forward omission:* the node receives a DATA message but does not forward it to mesh neighbours.
2. *Serve omission:* the node receives a request for a share segment but does not serve the corresponding DATA response, causing the requester to wait for a timeout and retry elsewhere.

Faulty nodes may apply these omissions whenever the relevant event occurs. They do not create invalid share segments, forge signatures, equivocate about data contents, or manipulate identities.

Adversarial limitations. The adversary cannot break cryptographic primitives, cannot forge valid data, cannot create Sybil identities, cannot directly choose honest nodes' neighbours beyond the ordinary GossipSub mesh process, and cannot eclipse a victim by controlling all of its network connections. We also do not model topic flooding or spam attacks in the main experiments.

Fault-rate interpretation and limitation. The omission rate α is an experimental stress parameter, not a proven fault-tolerance threshold. In particular, this work does not require or prove that $\alpha < 0.5$ is a theoretical boundary for omission faults. The simulator could be configured with larger omission fractions, but the reported experiments focus on $\alpha \leq 0.5$ to study the region where GossipSub begins to lose deadline satisfaction under the chosen workload. Therefore, the observed degradation near $\alpha = 0.5$ should be interpreted as an empirical limitation of the evaluated GossipSub-DAS configuration, not as a formal Byzantine fault-tolerance guarantee.

1.4 Research Objective & Questions

This dissertation experimentally assesses the *GossipSub* protocol as a dissemination mechanism for a data-availability sampling (DAS) workload. The primary aim is to quantify how *GossipSub parameter configurations* affect seeding and sampling outcomes under time constraints, limited bandwidth, and increasing omission fraction α.

This primary aim is to measure the security-performance-complexity frontier: to verify which *parameter configurations* give the greatest DAS success probability and the lowest latency under time constraints, along with restricted bandwidth and duplication.

From this purpose, we construct the following research questions:

RQ1: Performance: How do the parameters such as K-Copies, Segment Amount, etc. affect the raw performance of GossipSub?

RQ2: Overhead: What is the bandwidth and duplication cost of the GossipSub protocol under the dissemination task?

RQ3: Resilience and scalability: How do the above metrics degrade as the faulty fraction α increases?

This research focuses on the impact of each strategy on the performance of GossipSub when distributing and sampling for each specific time slot of 4 seconds to catch up with real-world systems [17][19].

Scope of claims. This dissertation evaluates GossipSub empirically under the stated simulator assumptions. It does not prove consensus safety, prove a cryptographic data-availability guarantee, or establish a theoretical fault-tolerance threshold. Instead, it measures how GossipSub's dissemination and repair mechanisms behave for a DAS-like workload under controlled timing, bandwidth, topic-layout, and omission-fault parameters. This scope is important because the main results are performance and robustness measurements, not formal security proofs.

Table 1: Terminology used throughout the dissertation.

Term	Meaning in this thesis
Blob / block data	The rollup data payload disseminated in a slot
Erasure-coded matrix	2D extended matrix produced by erasure coding
Share	Atomic unit transmitted and requested (one coded piece)
Row/column segment	Group of shares assigned together for custody/topic mapping
SA (segment amount)	Number of share parts per row/column segment
Topic	GossipSub channel carrying one (or more) row+column segments
Omission fraction α	Fraction of nodes that drop forwards / ignore requests

2 Methodology

This chapter describes how we answer the research questions in Section 1.4 by evaluating how *GossipSub configuration parameters* affect dissemination (seeding) performance and sampling success under increasing omission fraction α.

From the research questions in Section 1.4, we test the following hypotheses:

- **H1:** In a healthy network ($\alpha = 0$), appropriate GossipSub configurations achieve near-100% DAS success within the 4 s deadline.
- **H2:** As omission increases, more aggressive redundancy/repair settings improve DAS success up to higher α, at the cost of increased bandwidth and duplication.
- **H3:** When bandwidth is the bottleneck, increasing per-node capacity reduces tail latency and duplicated deliveries, improving deadline satisfaction.

2.1 Evaluation Approach

We use discrete-event simulation (PeerSim) to study how GossipSub configuration choices affect dissemination and sampling in large networks (thousands of nodes) under controlled fault scenarios. The simulator focuses on *dissemination dynamics*-topic membership, mesh formation and maintenance, eager forwarding, metadata advertisement, request/repair behaviour, and retries—while abstracting away cryptographic verification and data reconstruction (e.g., signature checks, commitments, and erasure-code decoding). This allows us to attribute performance changes directly to networking behaviour and protocol parameters rather than to CPU-bound cryptographic costs.

Each experiment run follows the two protocol phases in Section 3.5. At $t = 0$, a block producer seeds share segments into their assigned topics. Validators relay segments according to GossipSub's eager-push mesh and lazy-pull repair mechanisms and then perform data-availability sampling within the configured deadline. We repeat each configuration over multiple random seeds to mitigate stochastic effects from random neighbour selection, topic assignment, and sampling choices. We report aggregate metrics including phase-specific success rates (Section 3.7), completion time distributions, bandwidth usage, and duplication overhead.

Relation of metrics to prior work. The metrics in this thesis are chosen to connect three related but distinct evaluation perspectives. First, DAS theory and DAS networking studies commonly focus on whether enough randomly sampled shares can be retrieved to give confidence that the underlying data is available. Second, peer-to-peer networking studies often measure propagation delay, delivery ratio, bandwidth consumption, and resilience under churn or faulty peers. Third, GossipSub-specific evaluations emphasise duplicate deliveries, control-plane traffic, mesh maintenance, and attack resilience. Our evaluation combines these perspectives: seeding success captures whether custodians receive enough share coverage for their assigned topics; DAS success captures whether validators retrieve the required number of random samples before T_{DAS}; completion-time CDFs expose deadline-relevant tail latency; and bandwidth plus duplication quantify the network cost of achieving that reliability. We do not measure cryptographic verification cost, decoding throughput, or consensus reward/penalty effects, because the simulator intentionally isolates dissemination and repair behaviour.

2.2 Experimental Factors

Independent variables. In the results chapter, we evaluate a single dissemination protocol (GossipSub). Accordingly, we vary a focused set of GossipSub and workload parameters that correspond directly to the result sections (baseline behaviour, bandwidth sensitivity, latency under faults, and overhead under faults):

- **Healthy vs. faulty setting (Faulty rate).** We vary the fraction of faulty validators, denoted by $\alpha \in [0, 0.5]$, that perform the omission behaviour defined in the threat model. In the results, α is reported as the *Faulty rate (MR)* and is the primary axis for the "Impact of Faulty/Omission Nodes" sections.
- **Bandwidth cap.** To quantify the data-availability bottleneck under constrained networking, we sweep the per-node bandwidth limit (e.g., 10, 20, and 30 Mbit/s). This variable is used in the "Impact of Bandwidth on Performance" subsection and affects both completion time and duplication/overhead.
- **Topic granularity (sharding).** We vary the number of GossipSub topics and the mapping of row/column segments to topics (fine-grained sharding vs. coarser grouping), as defined in Section 3.4. This directly controls how much data each topic carries and therefore influences both seeding latency and network overhead.
- **Segmentation and replication data.** We vary the segment amount SA (how many share segment items a row/column segment is cut into) and, when enabled, the replication level (how many copies are injected per segment). These parameters determine message volume and redundancy and are reflected in both the latency results (seeding/sampling completion) and the overhead results (duplication and bandwidth usage).
- **Mesh and repair parameters.** We sweep GossipSub's mesh degree controls ($D, D_{\mathrm{low}}, D_{\mathrm{high}}$) and the lazy repair fanout (metadata advertisement / pull-based repair). These parameters shape how quickly share segments spread (latency) and how much redundant

traffic is generated (overhead), and they are analysed primarily in the "Impact of Faulty Nodes on Performance (Latency)" and "Impact of Faulty Nodes on Network Overhead" subsections.

Dependent variables. We report metrics that match the two-phase workflow (seeding then sampling) and the overhead-focused results:

- **Seeding success rate:** fraction of honest validators that, for every topic they custodian, receive at least half of the corresponding row/column segment share segments (the threshold depends on SA as defined in Section 3.7).
- **DAS success rate:** fraction of honest validators that obtain at least $S_{\text{target}} = 73$ distinct sampled share segments within the sampling deadline $T_{\text{DAS}} = 4$ seconds.
- **Latency distributions:** seed arrival times and sampling completion times per node, reported using distribution summaries (e.g., median and tail percentiles such as p95/p99) to capture both typical and worst-case behaviour.
- **Network overhead:** per-node bandwidth cost (bytes sent/received) and **duplication** (redundant deliveries of already-seen share segments), which quantify the efficiency cost of achieving low latency and robustness.
- **Control-plane overhead:** where instrumented, the volume of protocol-control traffic (e.g., metadata advertisements and request/retry messages) to distinguish data-plane load from coordination overhead.

2.3 Statistical Treatment

Each experimental configuration is executed over multiple independent runs with different random seeds. The seed controls stochastic elements such as topic assignment, mesh neighbour selection, message ordering, and the random choice of sampled share segments. Repeating runs therefore reduces sensitivity to any single "lucky" or "unlucky" topology realisation and allows us to report stable aggregate trends.

For each metric, we report *distributional* summaries rather than only means. In particular, we use the median to represent typical behaviour and tail percentiles (e.g., p95/p99) to capture worst-case effects that are especially relevant for DAS deadlines. For success metrics (seeding success and DAS success), we report the fraction of honest nodes meeting the phase-specific success criteria within the configured time budget.

Where applicable, we also compute confidence intervals across repeated runs by aggregating the per-run summary statistic (e.g., median completion time or overall success rate) and then reporting uncertainty over seeds. All figures and aggregate values are derived from the simulator's per-node logs (Chapter 5), which record message deliveries, duplicate receptions, request/retry events, and per-node completion times for both the seeding and sampling phases.

3 System Model & Assumptions

Symbol	Meaning
n	Number of validators/peers in the simulated network
α	Fraction of faulty validators following the omission-fault model
MR	Faulty/omission rate used in experiment labels; equivalent to α
TOPICS	Number of GossipSub topics used for row/column custody mapping
SA	Segment Amount; number of parts into which each row/column segment is cut
K	Number of copies/replicas injected per share segment during seeding
D	Target GossipSub mesh degree
D_{low}	Lower mesh-degree bound; below this, peers are grafted
D_{high}	Upper mesh-degree bound; above this, peers are pruned
D_{out}	Minimum outbound degree / bounded fanout for outside-topic advertisements
T_{DAS}	DAS decision deadline, set to 4000 ms in the experiments
S_{target}	Number of distinct samples required for DAS success, set to 73
T_i	Set of topics for which validator i is a custodian
C_i	Cache of distinct share segments held by validator i

Table 2: Notation used throughout the dissertation.

3.1 Entities and Roles

Our simulator models the network as a set of peer-to-peer (P2P) *nodes* running GossipSub. Each node is assigned a *role* that determines (i) whether it originates data, (ii) how it participates in dissemination, and (iii) how it behaves during the sampling process. To isolate the impact of GossipSub mechanics (topic granularity, mesh formation, and gossip-based repair) under controlled assumptions, we keep the role model intentionally minimal.

We model the following roles:

- **Block producer:** Creates one block per simulation run, erasure-codes it into a 2D extended matrix of share segments, segments the matrix, and seeds share segments by publishing them into GossipSub topics.
- **Correct validators ($V_1, \ldots, V_n$):** Participate as recipients and forwarders in dissemination, and during the sampling phase request random share segments to assess data availability.
- **Faulty validators (V_f):** A fraction α of validator nodes that follow the adversarial behaviour described in Section 1.3.

Figure 5 gives the high-level dissemination model used in the simulator: the producer injects share segments into GossipSub topics, honest validators relay data through topic meshes, and faulty validators may appear as ordinary mesh participants while omitting forwarding or serving actions.

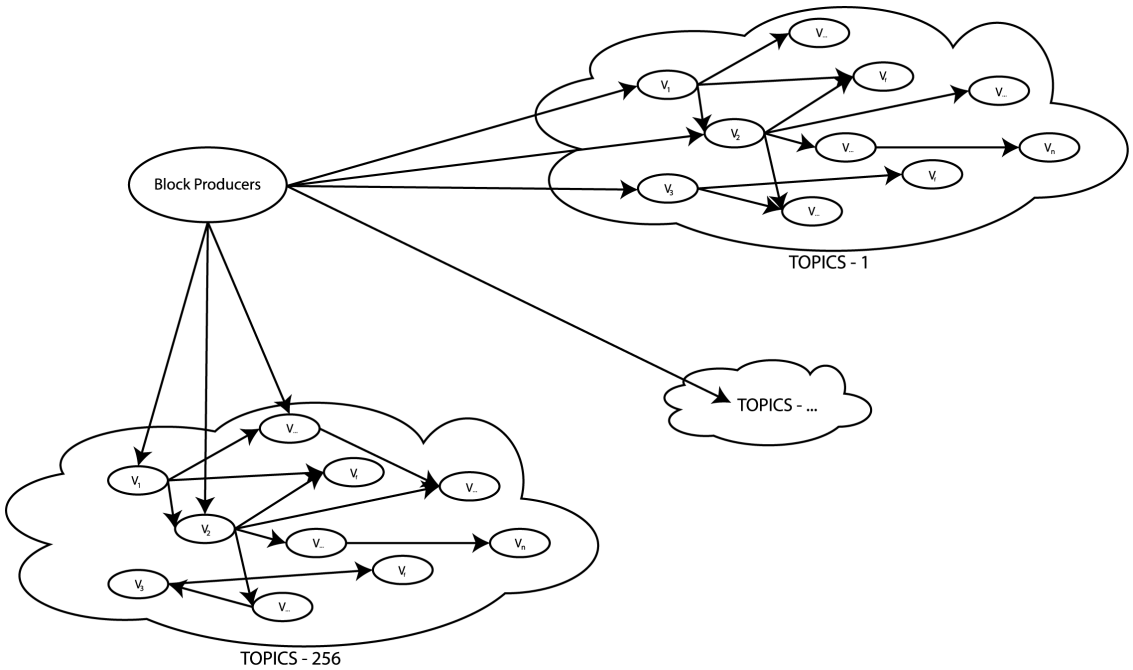


Figure 5: High-level dissemination model. The block producer seeds share segments into multiple GossipSub topics. Within each topic, validators form a forwarding mesh and propagate shared segments via multi-hop gossip. $V_1, \ldots, V_n$ denotes correct validators and V_f denotes a faulty validator.

Block producer. Each run has a single source of fresh data: the block producer. At the beginning of the simulation, it creates a block payload, arranges it as a 2D matrix, and applies 2D Reed-Solomon erasure coding to obtain an *extended* matrix of share segments (Section 1.1). The producer then partitions the extended matrix by cutting each *row* and *column* into a configured number of segments (denoted by SA in our experiments). These segments define the dissemination units used by GossipSub: each segment is mapped to a topic that is responsible for disseminating the shared segments that belong to that row/column segment (or both, depending on the experiment configuration).

A topic may represent (i) a *row segment* (a slice of a particular row), (ii) a *column segment* (a slice of a particular column), or (iii) a combined *row/column* assignment where a topic covers segments in both dimensions under the chosen partitioning scheme. After assigning share segments to topics, the producer *seeds* the network by publishing each share segment into its topic. From that point onwards, GossipSub handles multi-hop propagation within each topic mesh (Figure 5), so the producer does not deliver each share segment directly to every validator.

Correct validators ($V_1, \ldots, V_n$). Most nodes are validators. They perform two coupled tasks: (i) acting as *relays* in the GossipSub overlay and (ii) acting as *samplers* that assess availability within a time window. As relays, validators subscribe to relevant topics, maintain topic-specific meshes, forward received share segments to mesh neighbours, and use

GossipSub metadata exchange to advertise recently seen messages (e.g., via message identifiers). They also deduplicate and cache share segments, which helps when a share segment is missed on the eager forwarding path.

Once dissemination begins, validators enter the sampling phase. Each validator chooses a set of random coordinates in the extended matrix and attempts to obtain the corresponding share segments. If a requested share segment has already arrived via dissemination, it is served from the local cache; otherwise, the validator triggers on-demand retrieval from peers. This couples "push" (mesh forwarding) and "pull" (repair/retrieval) dynamics: sampling success depends on whether share segments were seeded broadly enough and whether the overlay can respond quickly to missing-share segment requests. In our evaluation, we measure per-validator completion time (when all sampled share segments are obtained) and whether sampling completes within the configured deadline, while validators continue forwarding and serving share segments throughout the run.

Faulty validators (V_f). Faulty nodes are modelled as a subset of validators (a fraction α) that deviate from the *correct* behaviour, rather than as a separate class with different connectivity or special privileges. This reflects the operational setting where faulty participants can be selected as mesh neighbours, subscribe to the same topics, and appear as ordinary peers in the overlay (Figure 5). Their impact therefore comes from *how* they act during dissemination and request servicing.

Section 1.3 specifies the precise adversarial behaviour. At a high level, faulty validators aim to degrade availability-related outcomes by reducing effective propagation and/or delaying retrieval (e.g., by not forwarding received share segments and not serving requested share segments). Modelling faulty validators as "validators with modified actions" allows us to measure how resilient the GossipSub-based dissemination and sampling pipeline is as α increases, without changing the underlying topology or topic configuration.

3.2 Network and Membership Model

Our simulator's dissemination operates according to GossipSub, where each node has a "mesh" of peers for each topic that it uses to quickly send messages. A node tries to keep about eight active mesh neighbours for each topic it is subscribed to. This is called a mesh degree of $D = 8$. We also set the standard lower and upper bounds, D_{low} and D_{high}, to keep the mesh stable. The node "grafts" more peers into the mesh (adds new neighbours for eager forwarding) if the mesh size drops below D_{low}. If the mesh gets bigger than D_{high}, the node "prunes" extra peers (removes neighbours) to cut down on extra traffic. Mesh maintenance happens on a regular basis, during heartbeats, which check the current mesh degree and start GRAFT/PRUNE actions as needed. This model shows the main control loop in GossipSub: keep the mesh in range of D_{low} and D_{high}, try to settle close to D for reliable propagation, but limit fanout to avoid duplication and save bandwidth.

Model for sending messages (push + metadata). We use GossipSub's eager push along the mesh and lazy gossip outside the mesh to model how messages spread. When a

node gets a share segment it hasn't seen before, it sends it to its mesh neighbours who are also interested in that topic. Also, nodes let non-mesh peers know about share segments (the segments cut from row/column) they have recently seen by sending compact `IHAVE`-style metadata (message IDs). This lets peers who are interested ask for missing items using `IWANT`. Our simulator does not model RTT distributions or per-link latency, which is important because it abstracts away fine-grained transport timing. Instead, after a peer has learned the metadata for a share segment, we treat the next request/response as an immediate logical transfer event (unless the simulator's bandwidth and processing limits are turned on for that experiment). This choice shows what we are most interested in: the main question is "Which peers learn about which share segments and through which GossipSub mechanisms?" not the exact millisecond-level delivery delay on each link.

3.3 Data Model

2D Reed–Solomon erasure extension. We model each block as a square matrix of symbols over a finite field. Let the original block be

$$B \in \mathbb{F}^{k \times k}$$

where each entry of B is a data symbol or share cell. Let

$$G \in \mathbb{F}^{2k \times k}$$

be a systematic generator matrix for an $[2k, k]$ Reed-Solomon code. In systematic form, the first k encoded symbols correspond to the original data symbols, and the remaining k symbols are parity symbols. Equivalently, G may be viewed as a Vandermonde-style evaluation matrix, transformed into systematic form, such that any k rows of G are linearly independent.

The two-dimensional extension is obtained by applying this code first across rows and then across columns. Row extension maps the original matrix to

$$B_{\text{row}} = BG^{\top} \in \mathbb{F}^{k \times 2k}.$$

Column extension then maps this row-extended matrix to

$$E = GBG^{\top} \in \mathbb{F}^{2k \times 2k}.$$

The matrix E is the erasure-extended block used by the simulator. Each entry $E_{r,c}$ is a share segment identified by its matrix coordinate (r, c). In the systematic layout, the original $k \times k$ data appears in the top-left region, while the remaining cells are parity shares produced by the row and column codes.

The relevant Reed-Solomon property is that any k available symbols from an encoded row or encoded column are sufficient to interpolate the corresponding degree $< k$ polynomial

and reconstruct the missing symbols in that row or column. Thus, for a $k \to 2k$ extension, reconstruction remains possible for a row or column if at least half of its encoded symbols are available. This threshold property motivates using random sampling as evidence of retrievability: if a significant fraction of the extended matrix is withheld, uniformly random sample requests are increasingly likely to encounter missing share segments [21].

In the simulator, we do not perform full Reed-Solomon decoding or polynomial interpolation. Instead, the extended matrix E provides the coordinate space for dissemination and sampling. GossipSub disseminates individual share segments as pub/sub DATA messages, and validators request individual coordinates during sampling. This abstraction is sufficient for the networking question studied here: whether share segments become widely retrievable before the DAS deadline under different GossipSub configurations and omission rates.

Amount of segments per row and column. The parameter SA (segment amount, also called shard amount) lets you control how the block is split up for distribution. Intuitively, it SA tells the producer how many share segment items to make for each row/column segment when they "cut" the extended matrix for distribution. When SA is bigger, each row and column is split into more (smaller) pieces. This adds more share segment messages to GossipSub. This can make parallelism better by letting different peers get different segments at the same time, but it also makes the protocol more complicated because it has to keep track of more things, advertise them through metadata, and send them through the mesh. When SA is smaller, there are fewer items that need to be sent, which means less metadata and forwarding overhead. However, each item takes up more space in the block, which could make each transfer more expensive.

We can directly study this trade-off by changing SA in the evaluation. For a given overall block size, SA affects the total number of share segment messages, the amount of duplicated delivery caused by eager forwarding, and the workload for sampling (because validators ask for share segments at random over the matrix). So, SA is a key parameter that links the data model to measurable results of dissemination, like how long it takes to finish and how much bandwidth it uses.

3.4 Topic and Custody Abstraction

We approximate custody-style dissemination by *co-locating* one row segment and one column segment within the same GossipSub topic. Intuitively, each topic represents a small "responsibility unit" of the erasure-extended matrix: members of that topic collectively relay and serve the share segments belonging to the assigned row and column.

Topic construction. In the ideal (high-topic) configuration, each GossipSub topic is mapped to exactly:

- **One row segment** of the matrix, and
- **One column segment** of the matrix.

When the configured number of topics is smaller than the number of row/column segments, we *merge* responsibilities by mapping multiple row segments and multiple column segments into the same topic. Hence, fewer topics imply a larger per-topic workload, because each topic carries more row/column data. This abstraction allows us to study how "topic granularity" impacts dissemination overhead and completion time: many topics correspond to finer sharding, while fewer topics correspond to coarser aggregation.

Membership and custody. Nodes subscribe to the topics to which they are assigned. Within a topic, all subscribed validators are treated as *custodians* for that topic's assigned row/column segments: they are expected to receive, forward, and serve share segments from those segments to other members. Unlike a global-broadcast design where every node attempts to relay every share segment, this custody abstraction limits each node's primary responsibility to the subset of rows/columns covered by its subscribed topics, which reduces unnecessary forwarding across unrelated data.

Intra-topic dissemination and completion criterion. For each topic, the block producer injects the share segments belonging to the topic's assigned row and column segments into the pub/sub overlay. Dissemination then proceeds inside that topic through GossipSub's forwarding mechanisms. We model dissemination as continuing until *all* validators subscribed to the topic have received sufficient coverage of the topic's data. A validator is considered *locally complete* for a topic once it has obtained at least half of the share segments from the topic's row segment and at least half of the share segments from the topic's column segment (i.e., a 50% coverage threshold per dimension). The topic is considered *complete* only when every validator in the topic satisfies this condition for every row/column segment assigned to that topic (in the merged case). This stopping rule captures the idea that the topic continues to "push" and "repair" dissemination until all its members have enough data from the rows and columns represented in that topic, aligning the dissemination objective with the sampling-style requirement of obtaining partial but sufficient coverage.

3.5 Protocol Phases (PANDAS-like [17])

Each simulation run follows a PANDAS-like two-phase structure that separates *data injection and dissemination* from *data-availability sampling*. This separation helps attribute performance changes either to how efficiently the network spreads share segments (seeding) or to how effectively validators can retrieve missing share segments under a fixed time budget (sampling). A key element of our methodology is *topic granularity*: we partition the erasure-coded 2D matrix into *segments* and map these segments to GossipSub topics.

Figure 6 illustrates the segmentation scheme with an example where $SA = 4$. Given an extended matrix of size $N \times N$, each row and each column is cut into SA contiguous segments, so each segment spans N/SA share segments along that dimension (assuming SA divides N). A share segment at coordinates (x, y) belongs to exactly one row segment and one column segment. Importantly, this is not an overlap of data: the same share segment is simply associated with two segment identities, one induced by its row position and one induced by

its column position, which enables different topic-mapping choices in our experiments.

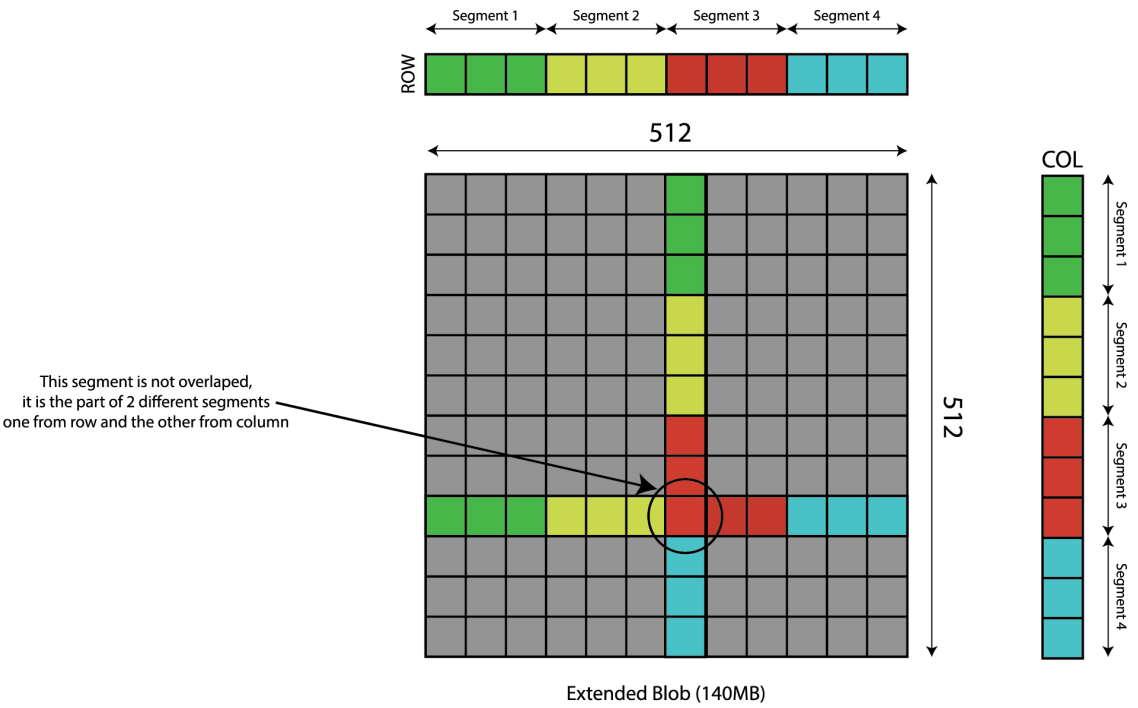


Figure 6: Example segmentation with $SA = 4$. Each row and each column of the extended matrix is partitioned into four contiguous segments. The circled share segment belongs to one row segment and one column segment.

1. **Seeding (dissemination phase).** The run begins with a single block producer generating an erasure-coded, extended matrix and then publishing share segments into GossipSub topics according to the segmentation and topic model (Section 3.4). After that, share segments are assigned to the topic(s) responsible for their row segment, their column segment, or both (depending on the experiment configuration). Validators subscribed to a topic act as *custodians* for that segment: they receive the producer's publications and relay shared segments to other members of the same topic. Under GossipSub, dissemination proceeds primarily via *mesh-based eager forwarding* (full-share segment push to mesh neighbours) and is complemented by *lazy repair* through metadata advertisements and subsequent on-demand requests. Conceptually, this phase "warms up" the network by spreading a broad base of shared segments so that subsequent sampling can succeed with minimal additional traffic.

2. **Sampling (availability phase).** After seeding, each validator initiates Data Availability Sampling by requesting a set of randomly chosen share segments across the extended matrix. Sampling targets are drawn uniformly over the matrix and therefore span multiple row and column segments (and multiple topics). If a requested share segment is already present in the local cache due to seeding, it is counted immediately; otherwise, the validator attempts retrieval from peers within the relevant topic(s), using GossipSub's metadata-driven repair path when available (or falling back to other

eligible topic peers when metadata is insufficient). Requests may be retried according to the protocol control logic to tolerate unresponsive or faulty neighbours. A validator declares *success* once it collects at least $S_{\text{target}} = 73$ *distinct* share segments before the sampling deadline. We use this criterion to compute sampling success rates and completion times across runs and to evaluate how dissemination quality in the seeding phase translates into practical retrievability during sampling.

3.6 Timing Model

Our timing model is designed to reflect the tight time constraints of Data-Availability Sampling (DAS) in slot-based blockchains, while still allowing enough time for protocol maintenance mechanisms (in particular GossipSub scoring) to evolve across multiple heartbeats.

Slot structure and DAS budget. Ethereum advances in fixed *slots* of 12 seconds [9]. In each slot, a block must be proposed, propagated, and then attested to by validators before the next slot begins. Validators cannot wait until the end of the full 12-second window to make their decision, because attestations themselves must also propagate and be aggregated in time. For this reason, DAS inherits a much smaller effective budget within each slot. We model this by setting the sampling deadline to

- **Sampling deadline:** $T_{\text{DAS}} = 4000$ ms,

which approximates the "early-slot" time available for a validator to fetch samples and gain confidence that the data is retrievable.

Protocol timers. Within this budget, protocol-specific timers determine how quickly nodes repair missing data and react to unresponsive neighbours. We configure these timers for *GossipSub*, including:

- **GossipSub heartbeat:** periodic mesh maintenance and metadata gossip dissemination.
- **Request timeouts:** how long a node waits before deeming a peer unresponsive.
- **Retry intervals:** how frequently failed share segment requests are re-issued to alternative peers.

These timers directly affect both completion time and bandwidth: shorter timeouts can improve responsiveness under faults but may increase redundant requests, while longer timeouts reduce overhead but risk missing the deadline.

Multi-slot runs for scoring dynamics. Although a single DAS decision is bounded by T_{DAS}peer scoring and mesh adaptation, GossipSub requires multiple heartbeats to meaningfully differentiate good and bad neighbours. To capture these dynamics, each simulation run

spans **five consecutive slots** (i.e., $5 \times 12 = 60$ seconds of simulated time). A single block is seeded at $t = 0$ and sampling is evaluated within its T_{DAS} window, but the run continues across subsequent slots so that score updates, GRAFT/PRUNE actions, and mesh reconfiguration can be observed over time. This design allows us to measure not only whether sampling succeeds within 4 seconds but also how quickly the overlay "self-heals" under faults as the scoring system accumulates evidence and adjusts peer selection.

3.7 Success and Measurement Definitions

To align our evaluation with the two-phase workflow in Section 3.5, we define success separately for the *seeding* and *sampling* phases and then derive network-wide metrics from per-node outcomes. Throughout, we count only *distinct* share segments (duplicates do not increase progress), since repeated deliveries primarily contribute to bandwidth and processing overhead.

Seeding-phase success (custody completion). During seeding, each validator is assigned a set of topics as described in Section 3.4, and is therefore a custodian for the row/-column segments carried by those topics. Let $\text{cov}_{i,\tau}^{\text{row}}$ and $\text{cov}_{i,\tau}^{\text{col}}$ denote the number of *distinct* share segments of the designated row and column segment(s) that node i has obtained in topic τ by the end of the seeding phase. The total number of share segments per segment is determined by the segmentation parameter SA (Section 3.3). We say that node i achieves *custody completion* for topic τ if it has obtained at least half of the share segments for the topic's designated row segment and at least half for the designated column segment:

$$\text{seed\_ok}_{i,\tau} = \begin{cases} 1 & \text{if } \text{cov}_{i,\tau}^{\text{row}} \geq \frac{SA}{2} \quad \wedge \quad \text{cov}_{i,\tau}^{\text{col}} \geq \frac{SA}{2} \\ 0 & \text{otherwise.} \end{cases}$$

A node i is considered successful in the seeding phase if it satisfies this condition for *all* topics it subscribes to:

$$\text{seed\_success}_i = 1 \iff \forall \tau \in \mathcal{T}_i,\ \text{seed\_ok}_{i,\tau} = 1,$$

where $\mathcal{T}_i$ is the set of topics for which node i is a custodian. The *network-wide seeding success rate* is then the fraction of honest nodes with $\text{seed\_success}_i = 1$. Algorithm 5 gives the corresponding implementation-level check used by the simulator to convert per-topic row/column coverage into a per-node seeding success outcome.

Sampling-phase success (DAS target). During sampling, each validator independently requests a fixed number of randomly chosen share segments across the block. Let $\text{samples}_i(T_{\text{DAS}})$ be the number of distinct sampled share segments successfully obtained by node i by the sampling deadline T_{DAS}. Node i succeeds in the sampling phase if:

$$\text{sample\_success}_i = \begin{cases} 1 & \text{if } \text{samples}_i(T_{\text{DAS}}) \geq S_{\text{target}} \\ 0 & \text{otherwise,} \end{cases} \qquad \text{where } S_{\text{target}} = 73.$$

We set $S_{\text{target}} = 73$ because it provides a practical sampling workload that is large enough to be statistically meaningful while remaining feasible within a 4-second DAS budget. Intuitively, each successful sample is an independent "probe" of data retrievability; increasing the number of probes reduces the probability that a validator incorrectly concludes "available" when many share segments are actually missing. In our experiments, using 73 samples yields a stable success-rate estimate across runs (low variance) without dominating the bandwidth budget of the sampling phase.

The *network-wide sampling success rate* is the fraction of honest nodes with:

$\text{sample\_success}_i = 1$.

Measured outcomes. In addition to per-phase success rates, we record: (i) **latency** metrics, including seed-share segment arrival times and per-node sampling completion times; (ii) **bandwidth usage**, measured as bytes sent/received per node; and (iii) **duplication overhead**, measured as the volume of redundant (already-seen) share segment deliveries. These measurements allow us to quantify the trade-offs between fast completion, low duplication, and robustness under faults.

4 Protocol Design

This chapter specifies the protocol mapping used in our simulator to model an *Ethereum-style* data-availability sampling workflow over topic-based gossip. We do *not* propose a new protocol. Instead, we study a *FullDAS-like* setting on Ethereum's DA roadmap: a regime where blobs are large enough that validators cannot practically download everything and where validators rely on sampling and on-demand retrieval over a P2P dissemination layer. In current designs, Ethereum is expected to progress from committee/custody-based approaches (often referred to as *PeerDAS* [16]) toward broader, more general sampling (*FullDAS*[7]). Our simulator therefore focuses on the networking and dissemination requirements that become dominant in the FullDAS regime and evaluates how well GossipSub can support seeding and sampling under omission-style faulty behaviour.

Thereby, we describe (i) the message types and identifiers we track, (ii) how GossipSub is mapped to custody-style row/column segments and their associated topics, and (iii) the end-to-end workflow that combines seeding and sampling within a slot-aligned time budget. Throughout, the simulator focuses on dissemination and repair dynamics (forwarding, metadata, requests, retries, and mesh maintenance) and abstracts away cryptographic verification and decoding, since our goal is to isolate the impact of GossipSub configuration on availability-related performance under faults.

4.1 Terminology and Message Model

A *share segment* is the atomic data unit disseminated and requested. Each share segment is identified by a tuple (`topicId`, `row/col`, `index`, `part`), where `topicId` selects the GossipSub topic, `row/col` indicates whether the share segment belongs to the topic's row or column segment, `index` identifies the row/column number, and `part` denotes the segment part number induced by SA.

We model four message categories:

- **DATA (share segment) messages:** full share segment payloads disseminated during seeding and served in response to requests.
- **IHAVE advertisements:** metadata-only announcements indicating that a peer has seen a given share segment identifier.
- **IWANT requests:** point-to-point requests for a specific share segment identifier, typically triggered after receiving IHAVE.
- **Mesh control:** `GRAFT` and `PRUNE` control messages are used to maintain per-topic meshes.

Each node maintains (i) a *message cache* for recently received DATA so it can avoid re-processing duplicates and can satisfy later IWANT requests and (ii) a short-lived *ephemeral cache* for recently seen identifiers that may be re-advertised via IHAVE during subsequent heartbeats. The simulator instruments per-node delivery timestamps, duplication counters

(reception of already-seen share segments), request/response events, and phase-completion times.

4.2 GossipSub-DAS Adaptation

Topic design (row/column custody). We implement custody-style dissemination by mapping matrix segments into topics as described in Section 3.4. In the most fine-grained setting, each topic carries exactly **one row segment and one column segment**. When the configured number of topics is smaller than the number of segments, multiple row segments and multiple column segments are grouped into the same topic, increasing the per-topic workload. Validators subscribe to the topics for which they are designated as custodians; within each subscribed topic they are responsible for receiving, forwarding, and serving share segments belonging to that topic's row/column segment(s).

Mesh formation and maintenance. For each subscribed topic, a node maintains a (directed) forwarding mesh with target degree D and bounds D_{low} and D_{high}. New peers are `GRAFT`ed when the mesh falls below D_{low}, and peers are `PRUNE`d when the mesh exceeds D_{high}. Mesh maintenance is executed periodically on a heartbeat timer.

We explicitly track outbound and inbound neighbours per topic and enforce a minimum outbound degree D_{out} so that each node retains sufficient eager-push connectivity even when the mesh is directed. In addition, D_{out} is used to bound *cross-topic* metadata exposure: when a node advertises recently seen messages, it may send those advertisements to up to D_{out} peers that are *outside* the topic. This provides limited "bridging" across topics without flooding the entire network. We set $D_{\text{out}} \approx \min(D_{\text{low}} - 1,\ D/2)$, and enforce $D_{\text{out}} \geq 1$.

Dissemination (eager-push + lazy-pull repair). During seeding, the producer publishes each share segment as a `DATA` message to its corresponding topic. Nodes forward `DATA` messages *eagerly* along the topic mesh to minimise propagation latency. To support repair beyond the mesh, nodes also emit `IHAVE` advertisements (metadata only) for recently seen share segments segment. In our model, `IHAVE` is sent to (i) peers in the same topic to help fill mesh gaps and (ii) a bounded number of peers outside the topic, limited by D_{out}, to improve discoverability across topics and reduce the chance that a share segment remains confined to a single segment community.

Peers that learn about a missing share segment via `IHAVE` can issue an `IWANT` request to pull the full `DATA` from an advertising peer. In our simulator, we do not model link-level RTT distributions; instead, once metadata for a share segment is known, the subsequent `IWANT`/`DATA` exchange is treated as a logical request/serve event, subject to any configured bandwidth caps and protocol timeouts. This keeps the focus on overlay behaviour: which nodes learn about which share segments, how quickly missing share segments are discovered, and how effectively the protocol repairs dissemination gaps under omission faults.

Peer scoring and fault adaptation. GossipSub's peer scoring is used to mitigate low-quality neighbours over time. On each heartbeat, nodes update peer scores based on observed behaviour (e.g., contribution to message delivery within topics) and apply threshold-based actions: peers that fall below configured thresholds can be deprioritised or removed from meshes via PRUNE, often with a backoff interval to prevent immediate re-grafting. We run simulations over multiple slots (Section 3.6) so that scores can accumulate and mesh updates can reflect sustained behaviour rather than a single transient event.

Configuration knobs. All major parameters are exposed via PeerSim configuration to support sensitivity analyses, including mesh degree bounds ($D, D_{\text{low}}, D_{\text{high}}$), the IHAVE fanout policy (either a fixed lazy fanout or a gossip-factor fraction), heartbeat period, advertisement history limits, message TTLs for re-advertisement, request timeouts/retry intervals, and workload parameters such as SA, number of topics and replication settings (when enabled).

4.3 End-to-End DAS Workflow

Each simulation run executes a single block dissemination and sampling episode. The complete high-level sequence is summarised in Algorithm 1, which connects producer encoding, seeding, validator sampling, and metric collection.

1. **Producer encoding and partitioning.** The producer constructs the 2D erasure-extended matrix and partitions each row/column segment into SA share segment parts. The corresponding producer-side loop over topics and assigned row/column segments is given in Algorithm 2.

2. **Seeding.** The producer publishes row/column share segments into their mapped topics. Validators relay DATA on the mesh (eager push) and advertise recent identifiers via IHAVE to enable pull-based repair. The node-side handlers for receiving DATA, advertising IHAVE metadata, issuing IWANT requests, and retrying after timeouts are shown in Algorithm 3.

3. **Sampling.** Each validator selects a fixed number of target share segments uniformly at random across the block (across all topics) and issues IWANT requests for any missing samples. A node declares sampling success once it obtains at least S_{target} distinct sampled share segments before the DAS deadline $T_{\text{DAS}} = 4$ seconds. The validator sampling loop, including target selection, cache checking, missing-sample requests, and deadline evaluation, is shown in Algorithm 4.

Although sampling is evaluated within the 4-second window, runs span multiple 12-second slots so that mesh maintenance and peer scoring can evolve across several heartbeats, allowing us to observe whether the overlay adapts to faults over time.

4.4 Pseudocode Summary of the Simulated Protocol

This section collects the pseudocode referenced throughout the protocol description. Algorithm 1 gives the end-to-end execution, Algorithm 2 specifies producer-side seeding, Algorithm 3 gives the validator event handlers for eager forwarding and lazy repair, Algorithm 4 gives the sampling loop, and Algorithm 5 gives the seeding-completion check. Together, these algorithms make explicit how the simulator connects topic mapping, deduplication, metadata-driven repair, request retries, and phase-specific stopping conditions.

Algorithm 1 End-to-end run (producer seeding + validator sampling)

Input: topics $\mathcal{T}$, segment amount SA, deadline T_{DAS}, target S_{target}
The producer constructs erasure-extended share segment matrix M
The producer partitions each row/column segment into SA share segments
$t \leftarrow 0$
ProducerSeed$(M, \mathcal{T})$
ValidatorsSample$(T_{\text{DAS}}, S_{\text{target}})$
Record: success rates, completion times, bandwidth, duplication

Algorithm 2 Producer seeding into custody topics

procedure ProducerSeed$(M, \mathcal{T})$
    for all topic $\tau \in \mathcal{T}$ **do**
        $(R_\tau, C_\tau) \leftarrow$ row/column segment(s) assigned to τ
        for all share segment $s \in (R_\tau \cup C_\tau)$ **do**
            PublishDATA(τ, s)
        end for
    end for
end procedure

Algorithm 3 Validator handlers (eager-push + metadata-driven lazy-pull)

State at node i: seen share segments $\mathcal{C}_i$, recent ids $\mathcal{H}_i$, mesh $\mathcal{M}_{i,\tau}$, pending $\mathcal{Q}_i$

procedure OnReceiveDATA(τ, s)
  if $s \in \mathcal{C}_i$ **then**
    duplications += 1; **return**
  end if
  $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{s\}$ ▷ store segment share segments
  $\mathcal{H}_i \leftarrow \mathcal{H}_i \cup \{\mathrm{id}(s)\}$ ▷ store id for gossip
  UpdateSeedingCoverage(τ, s)
  for all $p \in \mathcal{M}_{i,\tau}$ **do**
    SendDATA(p, τ, s) ▷ eager-push on mesh
  end for
end procedure
procedure OnHeartbeat
  for all subscribed topic τ **do**
    MaintainMesh(τ) ▷ enforce $D, D_{\text{low}}, D_{\text{high}}$
    $A \leftarrow$ select recent ids from $\mathcal{H}_i$ for topic τ
    for all peer p in lazy-gossip fanout for τ **do**
      SendIHAVE(p, τ, A)
    end for
  end for
end procedure
procedure OnReceiveIHAVE(τ, A)
  for all $m \in A$ **do**
    if $m \notin \mathcal{C}_i$ and m is needed (seeding repair or sampling) **then**
      $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{(m, \tau)\}$
    end if
  end for
  IssueIWANT
end procedure
procedure IssueIWANT
  while $\mathcal{Q}_i \neq \emptyset$ and request budget allows **do**
    pick (m, τ) from $\mathcal{Q}_i$
    pick peer p that advertised m (or is subscribed to τ)
    SendIWANT(p, τ, m)
    start timeout timer for (m, τ)
  end while
end procedure
procedure OnTimeout(m, τ)
  re-enqueue (m, τ) into $\mathcal{Q}_i$; wait `retry_interval`
  IssueIWANT
end procedure

Algorithm 4 Sampling at validator i

procedure ValidatorsSample($T_{\text{DAS}}, S_{\text{target}}$)
  for all validator node i **do**
    $Targets_i \leftarrow$ choose S_{target} distinct share segments uniformly across all topics
    $Collected_i \leftarrow Targets_i \cap \mathcal{C}_i$
    $t \leftarrow 0$
    while $t < T_{\text{DAS}}$ and $|Collected_i| < S_{\text{target}}$ **do**
      enqueue missing ids from $Targets_i \setminus Collected_i$ into $\mathcal{Q}_i$
      IssueIWANT
      advance time to next event (DATA / heartbeat / timeout)
      update $Collected_i$ upon OnReceiveDATA
    end while
    $\text{sample\_success}_i \leftarrow \mathbb{I}[|Collected_i| \geq S_{\text{target}}]$
  end for
end procedure

Algorithm 5 Seeding-phase success check (per topic custody)

procedure CheckSeedingSuccess(i)
  for all custodian topics $\tau \in \mathcal{T}_i$ **do**
    $\text{seed\_ok}_{i,\tau} \leftarrow \mathbb{I}\left[\text{cov}_{i,\tau}^{row} \geq \frac{SA}{2} \ \wedge \ \text{cov}_{i,\tau}^{col} \geq \frac{SA}{2}\right]$
  end for
  $\text{seed\_success}_i \leftarrow \mathbb{I}\left[\forall \tau \in \mathcal{T}_i : \text{seed\_ok}_{i,\tau} = 1\right]$
end procedure

4.5 Design Trade-offs

The protocol design balances three competing objectives:

- **Robustness:** Eager forwarding plus lazy-pull repair improves tolerance to omission and uneven dissemination, while scoring and PRUNE back off help reduce prolonged exposure to low-quality neighbours.
- **Efficiency:** Larger meshes and aggressive repair can reduce tail latency but increase duplicate deliveries and control-plane traffic (IHAVE/IWANT). Topic granularity and SA also directly shapes message volume and metadata overhead.
- **Tunability:** Exposing mesh, gossip, and timeout parameters enables systematic exploration of how configuration affects seeding completion, sampling success, and overhead under different fault levels.

These trade-offs motivate the parameter sweeps and overhead/latency analyses presented in Chapter 7.

5 Implementation in PeerSim

This chapter describes how the protocol design (Chapter 4) is realised in PeerSim. We focus on (i) how nodes, topics, and meshes are initialised, (ii) how GossipSub is implemented as an event-driven state machine, and (iii) how the simulator instruments per-node logs used in the results chapters.

Figure 7 summarises the implementation architecture used in the simulator. Configuration files instantiate the network, topic layout, bandwidth limits, and fault settings; the initialisation components construct validators, topics, and meshes; the GossipSub protocol component processes dissemination, sampling, timeout, and heartbeat events; and the observer exports the CSV logs used in the results chapter.

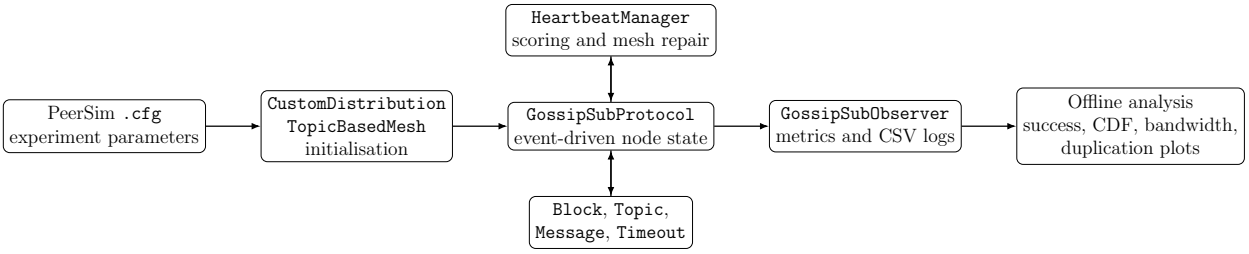


Figure 7: PeerSim implementation architecture. Configuration files define the experiment parameters; initialisation components create validators, topics, and meshes; `GossipSubProtocol` handles DATA, IHAVE/IWANT, timeout, heartbeat, and sampling events; supporting classes represent block data and messages; and `GossipSubObserver` exports the per-node logs used for analysis.

5.1 Code-base Overview

Implementation novelty. The implementation contribution of this thesis is not a new cryptographic erasure code and not a new variant of GossipSub. The novelty lies in the simulator integration required to evaluate GossipSub under a DAS-style workload. In particular, the code adds a DAS-specific data model, a custody-style topic mapping, a two-phase seeding/sampling workflow, omission-fault behaviours, and per-node instrumentation on top of an event-driven GossipSub simulation.

Concretely, the simulator extends a standard pub/sub dissemination setting in five ways. First, it represents a block as a 2D erasure-extended coordinate space, and maps share segments to row/column responsibilities. Second, it assigns validators to GossipSub topics as custodians so that each topic carries a controlled subset of the extended matrix rather than a single global broadcast stream. Third, it models the DAS workflow explicitly: a producer seeds share segments, validators cache and forward them, and validators later issue random sample requests subject to a $T_{\mathrm{DAS}} = 4$ s deadline. Fourth, it implements omission-specific faulty behaviour at both forwarding and request-serving points, allowing the same GossipSub configuration to be tested under increasing α. Fifth, it logs DAS-specific metrics, including seeding coverage, sampling completion, request timeouts, duplicate DATA deliveries, and bandwidth usage, which are not produced by a generic GossipSub broadcast simulator.

This means the simulator's novelty is at the level of workload mapping, adversarial evaluation, and measurement instrumentation. It provides a controlled way to ask how existing GossipSub mechanisms behave when the workload is not ordinary message broadcast but deadline-constrained dissemination and retrieval of erasure-coded share segments.

The simulator is implemented in the `peersim.GossipSub` package as a set of PeerSim protocol components (`EDProtocol`) and global controllers/observers (`Control`). The main components are:

- **Initialisation and membership:** `CustomDistribution.java` assigns each node a unique identifier, selects a block producer, samples faulty nodes according to `MALICIOUS_RATE`, creates the topic set and assigns custody/topic membership for validators. It also builds the initial per-topic mesh using `TopicBasedMesh.java`.
- **Protocol logic:** `GossipSubProtocol.java` implements the core event handler (`processEvent`) and maintains per-node state: topic subscriptions, meshes, caches, request/timeout bookkeeping, and counters for duplication and bandwidth.
- **Heartbeat, caches, and scoring:** `HeartbeatManager.java`, together with `PeerScoreInfo.java` and `GossipScoringConfig.java`, implement periodic maintenance: expiring ephemeral message state, re-advertising metadata, updating peer scores, and triggering mesh updates (GRAFT/PRUNE decisions).
- **Data and identifiers:** `Block.java` represents the underlying 2D share segment matrix;
  `Topic.java` stores topic membership; `Message.java` defines the event payloads (DATA and control messages); and `Timeout.java` models request timeout events.

- **Instrumentation:** `GossipSubObserver.java` collects per-node metrics and writes CSV outputs used in Chapter 7.

All parameters (network size, topic layout, mesh degrees, timers, bandwidth caps, fault rate, and sampling target) are provided through PeerSim `.cfg` files (e.g., `GossipConfig.cfg`), allowing the same code to be executed across large parameter sweeps.

5.2 Message Types and Event-driven State Machines

PeerSim executes our protocol as a discrete-event system. Each node runs an instance of `GossipSubProtocol` that reacts to incoming events in `processEvent`. Conceptually, the node behaviour is organised into three interacting state machines:

Producer state. A single node is designated as the block producer at initialisation. When it receives the block-proposal trigger event (in the implementation, `MSG_BLOCK_PROPOSER`), it generates a fresh `Block` instance and initiates seeding by publishing share segments into the corresponding topics. The seeding strategy is selected via configuration (e.g., `DISTRIBUTION_STRATEGY`) and determines how the block is partitioned and injected (row/column-based sharding, optional replication/copies, etc.). The producer then relies on the overlay to propagate the injected share segments to all subscribed validators.

Dissemination state (GossipSub). Validators process two classes of messages:

- **DATA messages** carry full share segment payloads and are forwarded eagerly along the topic mesh.
- **Control messages** implement lazy repair and mesh maintenance, including metadata advertisements (`IHAVE`), requests (`IWANT`), and mesh control (`GRAFT`/`PRUNE` where enabled by the maintenance logic).

Each node maintains a *message cache* (to deduplicate and to serve later requests) and an *ephemeral cache* used by the heartbeat to re-advertise recently seen message identifiers. Deduplication is explicitly counted: repeated deliveries of an already-seen share segment increment duplication counters but do not increase progress toward seeding or sampling.

Sampler state. Each validator maintains a sampling controller implemented in `GossipSubProtocol`. Sampling uses a fixed target size (`SAMPLE_AMOUNT`) and is bounded by the DAS deadline. Requests are driven by metadata: upon receiving an `IHAVE` for an unseen share segment, the node issues an `IWANT` to the advertising peer and schedules a `Timeout` event. If the timeout fires before the share segment arrives, the request is retried to an alternative eligible peer (subject to the retry policy and parallelism limits). This models the repair behaviour required under omission: progress depends not only on eager forwarding but also on whether metadata leads to responsive data sources.

Heartbeat and periodic maintenance. A per-node heartbeat (configured to run periodically, e.g., every 1 second in the implementation) triggers three essential maintenance tasks: (i) expiration of old ephemeral entries, (ii) re-advertisement of recently seen message identifiers (bounded by a per-heartbeat history limit), and (iii) peer score update and mesh adjustment. Scores are decayed over time and recomputed per peer using the configured scoring parameters; peers below thresholds may be pruned from meshes, while the mesh is repaired to satisfy degree constraints. To observe how scoring evolves, simulation runs extend over multiple slots (Section 3.6) even though DAS success is evaluated within the 4-second sampling budget.

5.3 Configuration Mapping Parameter

The PeerSim `.cfg` configuration maps directly to the system model (Chapter 2) and the protocol knobs (Chapter 4). The most important mappings parameter are:

- **Network scale and faults:** `NUMBER_OF_VALIDATORS` (network size n) and `MALICIOUS_RATE` (fault fraction α / MR). Faulty behaviour is realised by enabling omission on selected nodes, causing them to drop forwarding/serving actions according to the threat model.
- **Topic layout and custody:** `NUMBER_OF_TOPICS`, `NUMBER_OF_VALIDATORS_PER_TOPIC`, and row/column grouping parameters define how many segments are packed into a topic and how validators are assigned as custodians.
- **Segmentation and load:** `SHARD_AMOUNT` corresponds to SA (share segments per row/column segment). Optional replication is controlled by `SHARD_COPIES` (when enabled), increasing redundancy at the cost of higher bandwidth and duplication.
- **GossipSub mesh knobs:** `DEGREE` (D), `MIN_DEGREE` (D_{low}), and `MAX_DEGREE` (D_{high}) configure mesh size targets and bounds. Lazy repair fanout is configured via `D_LAZY` and/or `GOSSIP_FACTOR`, depending on the experiment.
- **Timers and sampling target:** heartbeat period and advertisement history/TTL parameters govern how often metadata is exchanged and how long it remains relevant; `SAMPLE_AMOUNT` sets the sampling target (S_{target}).
- **Bandwidth caps:** `INTERFACE_BANDWIDTH` and `BLOCK_PRODUCER_BANDWIDTH` control per-node transmission capacity. Message delivery is scheduled using a bandwidth-based serialisation delay and the configured transport latency.

This mapping allows us to reproduce the experimental factors in Chapter 2.2 by varying only the configuration file between runs.

5.4 Logging and Metrics Collection

All reported results are derived from per-node counters and timestamps recorded during simulation. Measurement is implemented through a combination of in-protocol counters

(maintained inside `GossipSubProtocol`) and a global observer (`GossipSubObserver`) that periodically snapshots state and writes CSV outputs.

At a minimum, we record:

- **Phase progress and success:** number of distinct share segments received for seeding custody, and number of distinct samples obtained before T_{DAS}.
- **Latency traces:** seed arrival times and sampling completion times (and, where applicable, request/timeout-driven delays).
- **Bandwidth usage:** bytes sent/received per node, computed from message sizes and the simulated send schedule.
- **Duplication and control overhead:** duplicate DATA receptions (already-seen share segments) and duplicate control-plane receptions (e.g., repeated advertisements), enabling the overhead analysis in the results chapter.

CSV outputs are then aggregated offline to produce success-rate curves, CDFs of completion time, and overhead plots (Chapters 7 onward).

6 Experimental Methodology

This chapter describes the empirical methodology used to evaluate our simulated DAS-style workload over **GossipSub**. Rather than comparing multiple broadcast protocols, our experiments focus on how *GossipSub configuration parameters* and *workload partitioning choices* affect (i) seeding completion, (ii) DAS sampling success under a 4-second deadline, and (iii) network overhead (bandwidth and duplication). We define the simulation scenarios and parameter sweeps, specify the dependent metrics collected from per-node logs, and outline the data collection and aggregation process.

6.1 Scenarios and Parameter Sweeps

We design simulation scenarios based on the configuration parameters in `GossipConfig.cfg` [**GossipConfig.cfg**]. Unless otherwise stated, we use a **baseline network** of $n = 8192$ validators and a **scale-out scenario** of $n = 16384$ validators (`NUMBER_OF_VALIDATORS`) to assess how trends change with network size. Each run follows the two-phase workflow in Section 3.5: the producer seeds at $t = 0$, validators perform sampling with a deadline of $T_{\text{DAS}} = 4000$ ms, and the simulation continues across **five 12-second slots** so that GossipSub's peer scoring and mesh adaptation can evolve over multiple heartbeats (Section 3.6).

Fault stress scenario (omission). To study robustness under faults, we vary the omission fault fraction `MALICIOUS_RATE` (α) from 0% (healthy network) up to 50%. Faulty nodes deviate according to the omission behaviour in the threat model (e.g., failing to forward and/or serve shared segments). The resulting curves quantify how seeding and sampling degrade as α increases and identify the operating region where GossipSub remains effective.

Bandwidth sensitivity scenario. To quantify how network capacity constrains dissemination and repair, we sweep the per-node bandwidth cap (e.g., 10, 20, and 30 Mbit/s) while holding the remaining protocol and workload parameters constant. This scenario is used to relate overhead (duplication and bytes transferred) to completion time and sampling success under an explicit capacity constraint.

Protocol/workload sensitivity scenarios. We then conduct controlled parameter sweeps to isolate the impact of key knobs that determine dissemination parallelism and redundancy:

- **Propagation redundancy (k-copies):** vary `SHARD_COPIES` (e.g., 1, 2, 4, 8) to study how additional seeding replication trades bandwidth for robustness and latency.
- **Topic granularity:** vary `NUMBER_OF_TOPICS` and/or `NUMBER_ROWS_OR_COLS_PER_TOPIC` to move between fine-grained sharding (more topics, less data per topic) and coarse grouping (fewer topics, more data per topic), following the topic model in Section 3.4.
- **Data segmentation:** vary `SHARD_AMOUNT` (SA) to control how many share segment items each row/column segment is cut into (Section 3.3).

- **Mesh and repair knobs:** vary GossipSub's mesh degree controls ($D, D_{\text{low}}, D_{\text{high}}$) and the lazy repair fanout (e.g., `D_LAZY` or a gossip-factor setting) to study the latency-overhead trade-off under both healthy and faulty conditions.

Across all sweeps, we hold non-target parameters fixed to attribute changes in outcomes to the variable under study.

Network timing abstraction. Our simulator abstracts away detailed RTT modelling and focuses on overlay dynamics (mesh forwarding, metadata propagation, and request/repair logic). Where link delays are configured (e.g., `MINDELAY`/`MAXDELAY`), they are kept fixed across scenarios so that observed effects are driven primarily by GossipSub configuration and fault behaviour rather than by heterogeneous latency.

6.2 Metrics

We collect dependent variables that match the two-phase success definitions (Section 3.7) and the overhead-focused results analysis:

- **Seeding success rate:** fraction of honest validators that, for every topic they custodian, receive at least half of the corresponding row and column segment share segments (threshold depends on SA).
- **DAS success rate:** fraction of honest validators that obtain at least $S_{\text{target}} = 73$ distinct sampled share segments within $T_{\text{DAS}} = 4000$ ms. We use a fixed target to keep sampling load comparable across scenarios and to align with the PANDAS-style sampling abstraction (Section 3.5) [17].
- **Latency distributions:** per-node seed arrival times and sampling completion times, summarised using median and tail percentiles (e.g., p95/p99) to capture worst-case behaviour relevant to deadlines.
- **Network overhead:** per-node bytes sent/received and duplication (redundant deliveries of already-seen share segments). Where instrumented, we additionally report control-plane overhead (e.g., metadata advertisements and request/retry traffic) to separate coordination cost from data-plane cost.
- **Resilience curves:** we report how the above metrics change as a function of α (MR), highlighting degradation trends and identifying parameter settings that widen or shrink the safe operating region.

6.3 Data Collection and Analysis

Each configuration is executed for multiple independent runs using different random seeds (five runs per configuration in our evaluation). The random seed affects topic assignment,

mesh neighbour selection, message ordering, and the random selection of sampled share segments. For each run, the simulator logs per-node events and counters (Chapter 5), including delivery timestamps, distinct-share segment counts, duplicate receptions, request/retry events, and byte counters.

We aggregate results across runs by computing distributional summaries (median and tail percentiles) for latency metrics and averaging success rates across seeds. Where reported, confidence intervals are computed over the repeated runs using the per-run aggregate statistic (e.g., sampling success rate or median completion time). All plots in the results chapter are generated from these CSV logs using the same aggregation pipeline, ensuring that comparisons across scenarios reflect identical measurement definitions and stopping criteria.

7 Results

This chapter presents the empirical results of the PeerSim experiments described in Chapter 6. We first evaluate baseline performance in a healthy network, then examine degradation under increasing omission behaviour.

7.1 Baseline Performance in a Healthy Network

To establish a baseline, we analyse protocol performance in an ideal scenario with 8,192 honest nodes and no adversaries. The primary goal is to determine whether the protocol can reliably meet the 4000 ms DAS deadline.

Figure 8 reports the average duplication per shard (`dup_per_shard_mean_final`) as a function of Segment Amount (SA) and replication factor (K-copies/K) across different numbers of GossipSub topics (128, 256, 512, and 1024). Overall, duplication is primarily driven by SA and K, while the number of topics has a smaller secondary effect.

Across all topic configurations, increasing SA from 1 to 4 substantially reduces duplication, indicating that coarse segmentation leads to higher redundancy during dissemination. However, further increasing SA from 4 to 8 results in only minor additional reductions, and moving to SA=16 provides little to no improvement (and in some settings slightly increases duplication). This behaviour shows diminishing returns beyond SA ≈4-8.

For every SA and topic setting, duplication increases consistently as K-copies per segment (K) increases (K=1 < K=2 < K=4 < K=8). This is expected because higher K replicates each segment to more peers, directly increasing redundancy during transmissions. The effect of K is pronounced across all subplots, with K=8 producing the highest duplication levels regardless of topic count.

Varying the number of topics changes the absolute duplication level modestly but does not alter the main trend: the relative ordering across SA and K remains consistent for 128, 256, 512, and 1024 topics. In the practical regime (SA $\in \{4, 8\}$ and $K \in \{2, 4\}$), the curves across topic counts are close, suggesting that topic scaling has limited impact compared to segmentation and replication.

We select 256 topics as a baseline for our custody/sharding abstraction [14], subsequent experiments focus on 256 topics and the parameter region SA $\in \{4, 8\}$ with $K \in \{2, 4\}$, since this range captures most of the reduction from segmentation while avoiding the high duplication overhead associated with larger K.

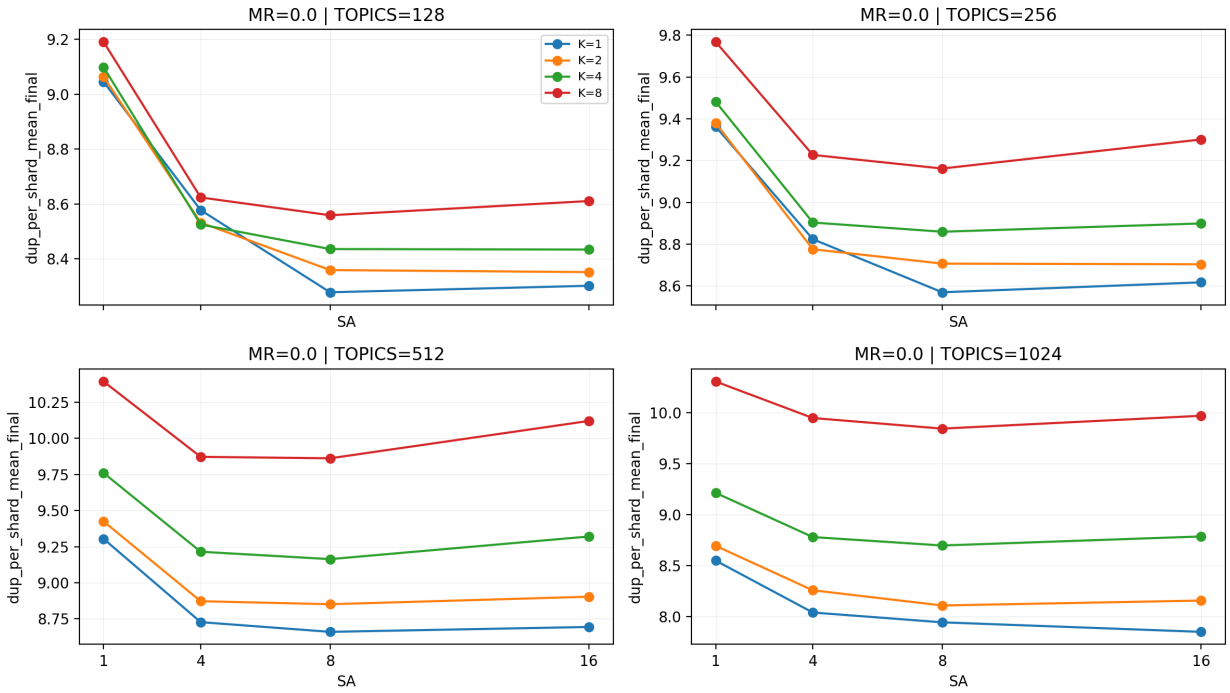


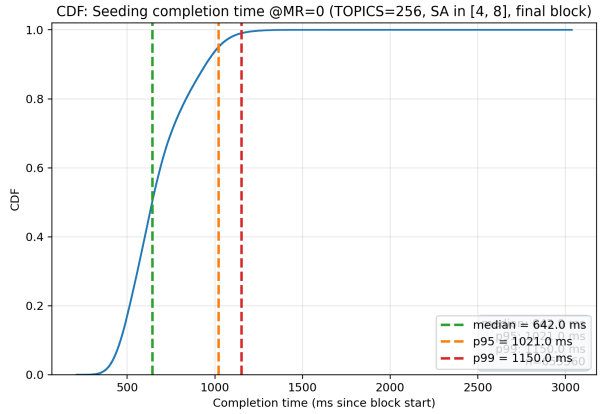
Figure 8: Mean of Duplication per Segments under 0% faulty rate.

Baseline latency in a healthy network (CDF view). To complement the duplication analysis above, we next characterise the *time-to-completion* distribution in the healthy setting ($\alpha = 0$) under the baseline configuration used throughout the thesis (TOPICS= 256, and the practical segmentation range $SA \in \{4, 8\}$). Figure 9a reports the cumulative distribution of *seeding completion time* per validator (time since block start until the node reaches the seeding criterion for its custodian topics), while Figure 9b reports the *DAS completion time* per validator (time since block start until the node finishes collecting the target number of samples). These CDFs provide a more informative view than averages because they highlight tail behaviour, which is critical under the 4 s DAS budget.

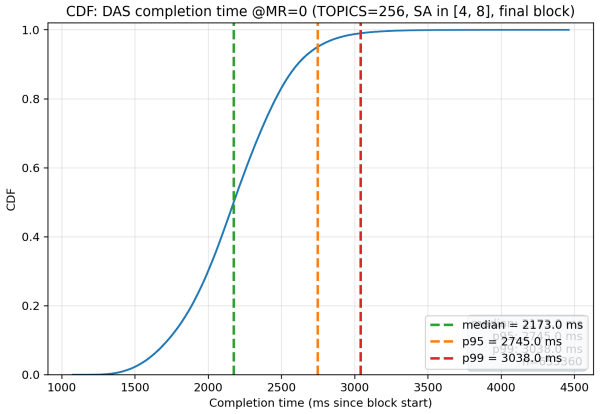
Seeding completes quickly, leaving headroom for sampling. Seeding completion is consistently fast in the healthy network. As shown in Figure 9a, the median seeding completion time is 642 ms, with p95 at 1021 ms and p99 at 1150 ms. This indicates that, under $\alpha = 0$ and with topic sharding at 256 topics, the custody topics become "sufficiently filled" for almost all validators within roughly the first second after the block is injected. Practically, this means the seeding phase rarely consumes the majority of the DAS time budget; instead, it acts as a rapid dissemination warm-up that populates caches and metadata, enabling the sampling phase to mostly retrieve remaining missing share segments via repair.

DAS completion is dominated by repair and remains within the 4 s budget for (almost) all nodes. Figure 9b shows that DAS completion is slower than seeding, reflecting

the additional work of locating and pulling specific randomly chosen share segments. Nevertheless, the distribution remains well within the 4 s sampling deadline in the healthy setting: the median DAS completion time is 2173 ms, with p95 at 2745 ms and p99 at 3038 ms. In other words, at least 99% of validators complete sampling in about 3.0 s or less, leaving a substantial slack relative to the 4 s budget. This baseline headroom is important for later sections: it provides room for performance degradation under omission faults before deadline misses become common, and it helps interpret whether parameter changes primarily affect the *median* experience or mainly inflate the *tail* of the completion-time distribution.



(a) CDF of seeding completion time under $\alpha = 0$ (TOPICS= 256, $SA \in \{4, 8\}$, final block). Dashed lines indicate median, p95, and p99.



(b) CDF of DAS completion time under $\alpha = 0$ (TOPICS= 256, $SA \in \{4, 8\}$, final block). Dashed lines indicate median, p95, and p99.

Figure 9: CDF of completion time baseline.

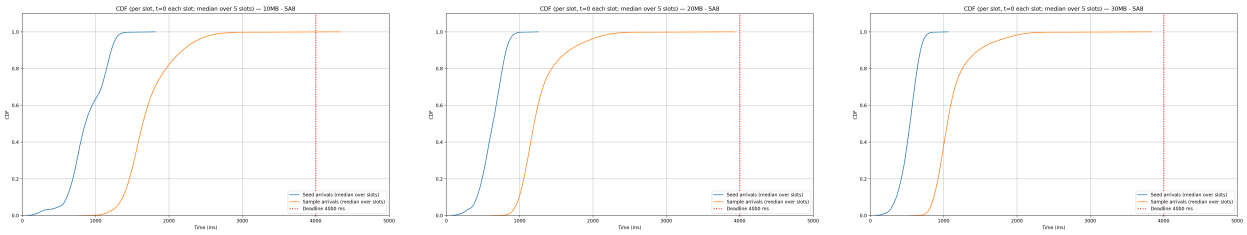
7.2 Impact of Bandwidth on Performance

To quantify how network capacity constrains dissemination and DAS under our *final baseline workload*, we repeat the sharding-based experiment under three per-node bandwidth caps (10, 20, and 30 Mbit/s), while keeping the remaining parameters fixed (TOPICS= 256, and **SA**= 8 as selected from the data-model sweep). We report two complementary timing views: (i) the *seed-part arrival time* (completion of the seeding criterion for the custodian topics) and (ii) the *sampling completion time* (time until a validator collects the target number of samples), both measured relative to the start of each slot. Since our runs span five consecutive slots, we summarise the timing distributions using the *median CDF over 5 slots* to reduce slot-to-slot noise while still reflecting steady-state behaviour.

Figures 10a–10c plot the CDFs of seed arrivals (blue) and sample completion (orange), with the red dashed line marking the 4 s DAS deadline. Across all bandwidth limits, seeding completes well within the deadline and is only weakly bandwidth-sensitive: increasing bandwidth shifts the seed CDF left (faster completion), but even at 10 Mbit/s the seed distribution converges long before 4 s. This indicates that, under SA= 8, the network can disseminate a sufficient fraction of the custodian row/column segments quickly, leaving most of the time budget for sampling and repair.

In contrast, sampling completion is noticeably more bandwidth-sensitive. At 10 Mbit/s, the sampling CDF shifts right and exhibits a longer tail that reaches (and can slightly exceed) the 4 s threshold, implying that a non-trivial fraction of validators may miss the deadline under sustained low capacity. At 20 Mbit/s and 30 Mbit/s, the sampling tail remains below the deadline, and the distribution becomes tighter, indicating that validators can complete the required pulls/repairs reliably within the allotted time. The improvement from 20 Mbit/s to 30 Mbit/s is mainly an additional safety margin in the tail rather than a large change in the median.

Based on these CDFs, we fix **SA**= 8 for subsequent experiments and adopt a per-node bandwidth of **at least 30 Mbit/s**. In particular, we configure later trials with **more than 30 Mbit/s per validator** to maintain a conservative safety margin for the 4 s DAS deadline. This choice is motivated by the tail behaviour: while 20-30 Mbit/s keeps sampling completion comfortably below 4 s in the healthy setting, any additional traffic from retries, longer runs (multiple slots), or higher omission fractions can shift the tail right. Setting the cap above 30 Mbit/s reduces the likelihood that bandwidth saturation becomes the dominant cause of deadline misses, ensuring that subsequent results primarily reflect GossipSub parameter effects rather than an artificially tight network bottleneck. The final-configuration bandwidth distributions used in the multi-slot omission experiments are reported in Appendix A. Figure 14 shows the healthy case ($\alpha = 0.0$), while Figures 19, 24, 29, 34, and 39 show the corresponding distributions for $\alpha = 0.1$ through $\alpha = 0.5$. These plots confirm whether the final experiments are bandwidth-saturated or whether the observed failures are more strongly associated with omission and repair behaviour.



(a) 10 Mbit/s (b) 20 Mbit/s (c) 30 Mbit/s

Figure 10: CDF of seeding (blue) and sampling completion (orange) per slot (median over 5 slots) under SA= 8 across bandwidth limits. The red dashed line marks the 4 s DAS deadline.

7.3 Impact of Faulty Nodes on Performance (Latency)

We next evaluate resilience under omission faults by sweeping the faulty fraction α, where faulty nodes may receive share-segment data but fail to forward or serve it.

The results show two practical regimes as α increases:

- **Stable regime ($\alpha = 0.0 \rightarrow 0.4$):** Seeding remains near-perfect for SA ≥ 4 with only mild degradation at SA $= 1$. Over the same range, DAS decreases steadily with

α, indicating a gradual increase in deadline misses as omission reduces effective data availability.

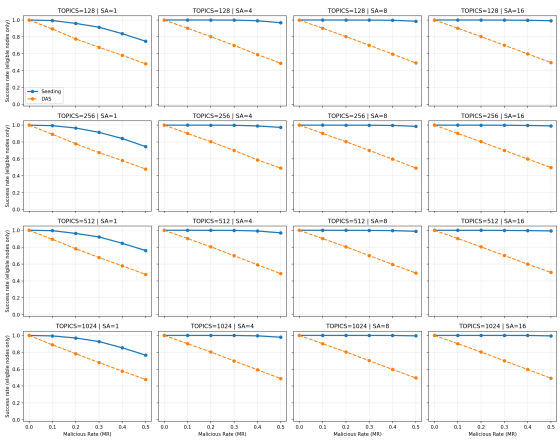
- **Degraded regime ($\alpha = 0.5$):** At the highest tested omission rate, DAS drops to roughly $\sim$0.5 success (e.g., TOPICS = 256), while the seeding success rate remains close to 1.0 for SA ≥ 4. When SA = 1, success rate also degrades, increasing redundancy from $K = 2$ to $K = 4$ substantially mitigates this worst-case behaviour.

The seeding success rate is robust except at very low sampling aggressiveness. Across TOPICS and SA, the seeding success rate stays close to 1.0 for SA ≥ 4 even at $\alpha = 0.5$, suggesting the protocol can route around missing forwarders and still meet the deadline in most settings. The main sensitivity appears at SA = 1. For TOPICS = 256, the seeding success rate decreases from 1.00 at $\alpha = 0.0$ to 0.75 at $\alpha = 0.5$ for $K = 2$, while $K = 4$ improves the worst case to 0.93 at $\alpha = 0.5$.

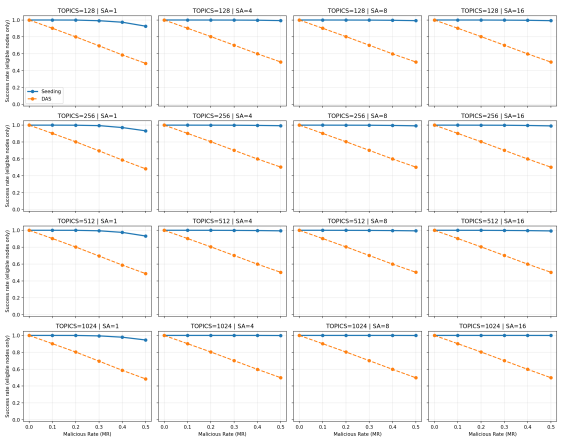
DAS degrades smoothly and approximately linearly with α. In contrast, DAS exhibits a consistent monotonic decline with α, with limited sensitivity to SA, TOPICS, or K within the measured range. For TOPICS = 256, DAS decreases from 1.00 at $\alpha = 0.0$ to roughly 0.50 at $\alpha = 0.5$ for both $K = 2$ and $K = 4$, and similar slopes appear in the line plots across other TOPICS values. This suggests omission primarily reduces data availability in a way that directly translates into missed deadlines, rather than triggering an abrupt collapse.

The performance gap widens with α. The delta heatmaps (DAS$-$Seeding) confirm that the gap between protocols becomes increasingly negative as omission increases. For TOPICS = 256 and SA ≥ 4, the delta reaches approximately -0.49 at $\alpha = 0.5$, i.e., DAS succeeds about 50 percentage points less often than the seeding success rate under heavy omission. For SA = 1, the gap is smaller at $\alpha = 0.5$ (e.g., around -0.27 for $K = 2$), mainly because the success rate also begins to degrade in that regime.

Overall, within $\alpha \leq 0.5$, the results indicate a gradual loss of deadline satisfaction for DAS as omission increases, while the seeding success rate remains near-perfect in most configurations and benefits from higher redundancy (K) when SA is minimal. The per-slot timing CDFs for the final configuration are provided in Appendix A. Figure 15 shows the healthy case, and Figures 20, 25, 30, 35, and 40 show the same timing view as α increases from 0.1 to 0.5. These figures show how the seed-first, seed-last, and sample-arrival curves move relative to the 4000 ms deadline, making the tail-latency effect visible on a per-slot basis.



(a) Mean success rate vs. α for $K = 2$.



(b) Mean success rate vs. α for $K = 4$.

Figure 11: Success rate under omission as the faulty rate (α) increases, showing seeding and DAS success rates across TOPICS and SA for two redundancy settings.

7.4 Impact of Faulty Nodes on Network (Overhead)

We now examine how omission faults affect *network overhead*, focusing on (i) redundant deliveries (*duplication per segment*) and (ii) per-node bandwidth usage. Throughout this section, the faulty rate (α) corresponds to the omission fraction α, and the reported curves aggregate results *across the configuration sweep* (min/median/max over the tested parameter settings at each α).

Duplication decreases monotonically with α. Figure 12 shows a clear and nearly linear decline in duplication as α increases. In the healthy network ($\alpha = 0$), the median duplication is high (about 8.6 duplicate deliveries per segment), reflecting the cost of robustness in GossipSub: eager mesh forwarding plus repair mechanisms create multiple overlapping delivery paths. As omission increases, faulty nodes forward less (or not at all), which reduces the number of redundant paths and therefore lowers duplication. By $\alpha = 0.5$, the median duplication falls to roughly 3.3, with the max across configurations dropping from about 10 ($\alpha = 0$) to about 5 ($\alpha = 0.5$). This reduction should not be interpreted as "improved efficiency" under faults; rather, it reflects that the network is becoming *less connected in terms of effective forwarding*, so fewer copies of the same segment circulate.

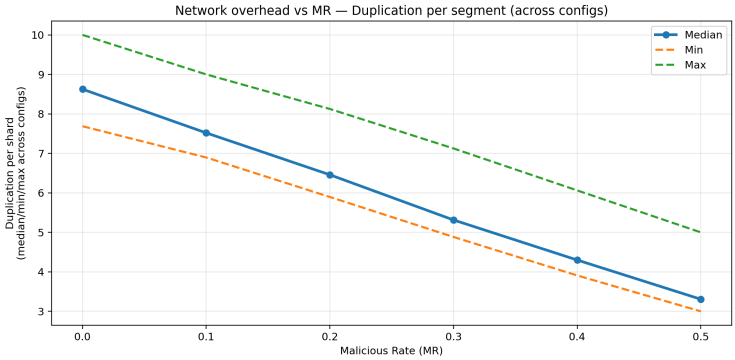
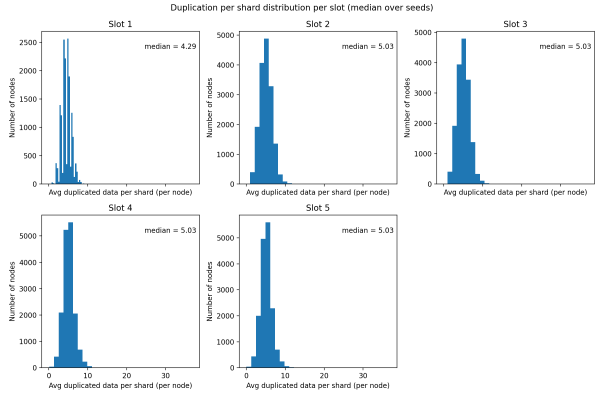
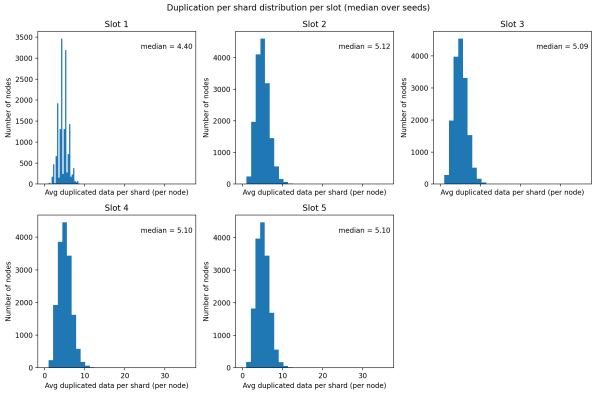


Figure 12: Duplication per segment as a function of omission rate α (MR), aggregated across the configuration sweep (min/median/max over tested parameter settings at each α).

Slot-by-slot duplication stabilises as the mesh recovers. To provide a clearer view of steady-state overhead, we focus on the $K = 4$ setting. Figure 13b shows a representative five-slot run at $\alpha = 0.3$ under the final configuration (SA= 8, 60 Mbit/s per-node cap). Slot 1 exhibits a lower median duplication (4.40), consistent with a "cold-start" period where the topic mesh has not fully converged and omission reduces effective forwarding paths. From Slot 2 onwards, the distribution tightens and the median stabilises around 5.09–5.12 per shard. This slot-to-slot stabilisation is consistent with GossipSub's peer scoring and heartbeat maintenance, which `PRUNE`s low-performing neighbours and `GRAFT`s replacements, bringing meshes back toward the target degree ($D = 8$) while remaining within the $[D_{\text{low}}, D_{\text{high}}]$ bounds. We therefore use $K = 4$ as the default visualisation in the remainder of this section to illustrate typical overhead behaviour under omission.



(a) Per-slot (5-slot) distribution of duplication per shard at $\alpha = 0.3$ (SA= 8, $K = 2$, 60 Mbit/s cap).



(b) Per-slot (5-slot) distribution of duplication per shard at $\alpha = 0.3$ (SA= 8, $K = 4$, 60 Mbit/s cap).

Figure 13: Slot-by-slot duplication per shard under omission faults ($\alpha = 0.3$) for SA= 8 with two redundancy settings ($K = 2$ vs. $K = 4$). Across slots, the distributions stabilise, consistent with GossipSub mesh maintenance (heartbeat `GRAFT`/`PRUNE`) and peer scoring recovering toward the target mesh degree ($D = 8$) within the $[D_{\text{low}}, D_{\text{high}}]$ bounds.

Appendix A provides the complete duplication distributions for the final configuration across all omission rates. The per-slot violin plots are shown in Figures 16, 21, 26, 31, 36, and 41; the corresponding per-slot histograms are shown in Figures 17, 22, 27, 32, 37, and 42. The all-slot duplication summaries are shown in Figures 18, 23, 28, 33, 38, and 43. These supplementary plots show that the decline in duplication under omission is visible across slots and is not caused by a single representative run.

7.5 Summary of Findings

This chapter evaluated GossipSub under a DAS-style workload in three stages: (i) baseline behaviour in a healthy network, (ii) sensitivity to bandwidth limits, and (iii) robustness under omission faults. We summarise the main takeaways and justify the final parameter choices used in the remaining experiments.

Stage 1: Data-model sweep identifies the dominant knobs (SA and K), while TOPICS is secondary. In the healthy network ($\alpha = 0$), Figure 8 shows that duplication is primarily controlled by the *segment amount* (SA) and the replication factor (K). Increasing SA from 1 to 4 yields a large reduction in duplication, while gains beyond SA $\approx$ 4–8 exhibit diminishing returns. Increasing K consistently increases duplication due to additional redundant deliveries. Varying TOPICS (128–1024) changes absolute levels modestly but does not alter the main SA/K trends. Based on this sweep and to match common GossipSub deployments, we adopt TOPICS= 256 as the baseline and focus on the practical region SA$\in \{4, 8\}$ and $K \in \{2, 4\}$.

Stage 2: Healthy-network latency comfortably meets the 4 s DAS budget. Under TOPICS= 256 and the practical segmentation range, seeding completes quickly (median 642 ms; p99 1150 ms; Figure 9a), leaving substantial headroom for sampling. DAS completion remains within the 4 s deadline for (almost) all validators (median 2173 ms; p99 3038 ms; Figure 9b). This establishes a baseline slack of roughly 1 s at the tail, which is important when interpreting later degradation under bandwidth constraints and faults.

Stage 3: Bandwidth mainly impacts the sampling tail; ≥ 30 Mbit/s is sufficient, and 60 Mbit/s provides margin. With SA fixed to 8, Figure 10 shows that seeding is only weakly bandwidth-sensitive and remains well within the deadline even at 10 Mbit/s. In contrast, sampling completion becomes tail-sensitive at low bandwidth: at 10 Mbit/s the sampling distribution approaches (and can slightly exceed) the 4 s line, whereas 20–30 Mbit/s keeps the tail below 4 s and tightens the distribution. For subsequent multi-slot and faulted runs, we therefore adopt a conservative per-node bandwidth cap of 60 Mbit/s to ensure that deadline misses are not dominated by an artificially tight network bottleneck and to isolate the effect of GossipSub parameters under faults.

Stage 4: Omission faults degrade DAS gradually; seeding remains robust for SA≥ 4. The sweeping omission rate α shows two regimes (Section 7.3). For $\alpha \leq 0.4$

seeding, success remains near-perfect for SA≥ 4, while DAS success declines approximately linearly with α. At $\alpha = 0.5$, DAS success drops to around ~ 0.5 (TOPICS= 256), indicating a degraded regime where a significant fraction of validators miss the deadline, even though seeding can still succeed. Increasing redundancy from $K = 2$ to $K = 4$ mainly helps in the most adverse settings (e.g., SA= 1) by improving seeding robustness and provides an additional safety margin against further tail inflation due to retries and mesh churn.

Stage 5: Overhead decreases with α, and per-slot behaviour stabilises after the first slot. Omission reduces effective forwarding and therefore reduces duplication: Figure 12 shows a near-linear drop in median duplication from ≈ 8.6 at $\alpha = 0$ to ≈ 3.3 at $\alpha = 0.5$. To illustrate steady-state dynamics, Figure 13 shows five consecutive slots at $\alpha = 0.3$ under the final configuration (SA= 8, 60 Mbit/s cap). Slot 1 exhibits a lower median duplication (cold-start mesh), while Slots 2–5 stabilise around ≈ 5.1 per shard in the $K = 4$ setting, consistent with peer scoring and heartbeat maintenance (`GRAFT`/`PRUNE`) recovering topic meshes toward the target degree D within $[D_{\text{low}}, D_{\text{high}}]$.

Final parameter choices used in later experiments. We adopt TOPICS= 256, SA= 8, $K = 4$, and a 60 Mbit/s per-node bandwidth cap for the remainder of the thesis. TOPICS= 256 provides the baseline custody/sharding abstraction and exhibited stable behaviour in the healthy network. SA= 8 captures most of the duplication reduction from segmentation while avoiding diminishing returns beyond SA≥ 16. $K = 4$ offers a practical redundancy level that avoids the high overhead of larger K (e.g., $K = 8$) while providing a clearer and more conservative view of steady-state behaviour under faults. Finally, 60 Mbit/s preserves a safety margin for the 4 s DAS deadline so that subsequent results primarily reflect GossipSub dynamics (mesh formation, repair, and omission) rather than bandwidth saturation.

Additional results in Appendix A. For completeness and reproducibility, Appendix A provides the supplementary bandwidth, timing, and duplication plots for the final configuration (TOPICS $= 256, SA = 8, K = 4, 60$ Mbit/s) across omission rates $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. The appendix figures are referenced from the relevant result sections: bandwidth histograms support the bandwidth discussion, per-slot CDFs support the latency discussion, and duplication distributions support the overhead discussion.

8 Discussion

This chapter interprets the empirical findings (Chapter 7) in the context of the system model and evaluation design. We explain why the main trends occur, what they imply for DAS-style dissemination over GossipSub in a FullDAS-like setting, and which modelling assumptions may affect how these results translate to real deployments.

8.1 Interpreting Degradation Under Omission

A central outcome of the experiments is that sampling performance degrades steadily with omission and becomes practically problematic at high fault rates. Up to $\alpha \approx 0.4$, the system maintains near-perfect seeding success in most configurations, and DAS completion remains feasible for the large majority of validators. At $\alpha = 0.5$, however, sampling success drops to roughly ~ 0.5 (for TOPICS= 256 in our measured range), indicating a degraded regime where deadline misses become common even though seeding can still succeed.

This behaviour is consistent with how GossipSub disseminates and repairs missing data. In lower omission regimes, eager mesh forwarding provides fast initial coverage, and lazy repair (metadata advertisements followed by `IWANT` pulls) provides enough path diversity for validators to route around missing forwards. As α increases, two effects compound. First, the effective fanout of eager dissemination drops because a larger fraction of mesh neighbours simply do not forward. Second, repair becomes less reliable because fewer peers both *have* the missing share segment and are *responsive* when contacted, increasing the likelihood that `IWANT` attempts hit unhelpful targets or require multiple retries.

The dominant failure mechanism is therefore *tail latency* rather than average throughput. Sampling success requires collecting a fixed number of distinct shares before a strict $T_{\mathrm{DAS}} = 4\,\mathrm{s}$ deadline. Even when most samples arrive quickly, a validator can miss the deadline if a small remainder is delayed by unlucky neighbour selection, repeated timeouts, or local neighbourhood isolation. This explains why DAS success can remain high over moderate α values but then becomes unacceptable at $\alpha = 0.5$: as the probability of "hard" samples lying behind omission-heavy paths rises, the remaining time budget is consumed by repair attempts and the completion-time tail expands.

8.2 Performance vs. Overhead Trade-offs

The overhead results reflect a core gossip trade-off: robustness comes from redundancy, and redundancy costs bandwidth. In healthy settings, eager forwarding over the mesh naturally creates duplication because multiple paths deliver the same share segment. Lazy repair adds further traffic through metadata exchange and point-to-point requests, but it also reduces the chance that a validator misses a segment due to a single missing forward.

Under omission, measured duplication decreases as α increases. This is expected: when faulty peers stop forwarding, fewer redundant copies circulate. Importantly, this reduction is not an improvement in efficiency; it is a symptom of reduced effective connectivity and weaker coverage. The slot-by-slot distributions provide additional intuition. In the repre-

sentative $\alpha = 0.3$ runs under the final configuration (SA= 8, 60 Mbit/s cap), duplication is lower in Slot 1 and then stabilises across Slots 2–5. This pattern is consistent with a transient "cold-start" period where meshes are still converging, followed by a steadier regime where peer scoring and heartbeat maintenance (`GRAFT`/`PRUNE`) recover the mesh toward the target degree. Using $K = 4$ provides a clearer view of this steady-state behaviour: redundancy remains high enough to observe stable dissemination dynamics, while still avoiding the extreme overhead of larger K values.

These observations also motivate the final parameter choices used in later experiments. SA= 8 captures most of the duplication reduction from segmentation while avoiding diminishing returns beyond SA$\geq$ 16. A per-node bandwidth cap of 60 Mbit/s ensures that the 4 s deadline is not dominated by an artificially tight bottleneck, so that measured degradation under omission primarily reflects overlay dynamics and repair behaviour. Finally, $K = 4$ offers a practical redundancy level that provides robustness headroom under faults without incurring the large duplication overhead associated with higher replication factors.

8.3 Implications of Modelling Assumptions

Several modelling choices likely bias the results toward an optimistic view of real-world performance:

- **Perfect discovery and stable membership:** We assume global membership knowledge and do not model churn. In practice, peer discovery delays, NAT constraints, and disconnects can degrade mesh quality and reduce repair effectiveness.
- **Simplified timing model:** We abstract away RTT variability and treat request/serve as logical events (subject to protocol timers and bandwidth caps). Real networks exhibit latency variation and queueing, which can increase tail latency and therefore worsen deadline misses.
- **Homogeneous node resources:** Identical bandwidth limits and processing capacity remove heterogeneity effects. In real deployments, slower uplinks and overloaded nodes can inflate p95/p99 completion times and concentrate repair load.
- **No cryptographic and decoding costs:** Verification and reconstruction work compete with networking in real clients, reducing the effective time budget available for sampling within a fixed slot deadline.

Consequently, the simulator's success rates should be interpreted as a best-case baseline for dissemination dynamics; real networks may enter the degraded regime at lower α or require more conservative parameter choices to maintain the same tail-latency guarantees.

8.4 Limitations and Future Improvements

Our evaluation focuses on omission faults because they directly stress the missing-share repair loop that dominates DAS tail latency. Other adversarial behaviours are plausible in open P2P

settings but are not quantified here. A natural extension is to add **resource-exhaustion** experiments (e.g., spam/flooding within topics), which would test how bandwidth limits and scoring interact when many messages compete with share dissemination. Additional realism can be achieved by modelling partial discovery and churn, introducing heterogeneous bandwidth distributions, and incorporating RTT variability and queueing to better capture timeouts and tail behaviour.

Finally, the results suggest static parameters may be suboptimal across fault regimes. Future work could explore **adaptive tuning** strategies—for example, temporarily increasing lazy repair fanout or retry parallelism when sampling progress stalls, or adjusting mesh parameters (D, D_{low}, D_{high}) based on observed completion tails. Such adaptations may help contain tail latency under faults while limiting duplication in healthy conditions.

9 Conclusion & Future Work

9.1 Summary of Contributions

This thesis studied how GossipSub-style, topic-based dissemination can support Data-Availability Sampling (DAS)-like workloads under strict timing constraints. The primary contribution is a configurable PeerSim-based simulator that maps a DAS-inspired data model onto GossipSub topics, enabling systematic exploration of how protocol and workload parameters shape dissemination latency, sampling success, and network overhead at scale. Specifically, we implement: (i) a 2D Reed-Solomon erasure-coded (extended) block abstraction partitioned into share segments, (ii) a custody-style topic mapping where topics carry row/column segments and validators subscribe as custodians, (iii) a two-phase slot workflow (seeding then sampling) aligned with a $T_{\mathrm{DAS}} = 4\,\mathrm{s}$ decision window, and (iv) instrumentation for per-node completion times, bandwidth usage, and duplication.

This contribution should be understood as a simulator and measurement contribution: the work integrates existing GossipSub mechanisms with a DAS-specific workload, omission-fault model, and deadline-orientated metrics, rather than proposing a new consensus protocol or a new erasure-code construction.

Using this framework, we established a healthy-network baseline in a practical regime (TOPICS= 256, $SA \in \{4, 8\}$), showing that seeding completes quickly (sub-second median) and that DAS completion remains within the 4 s budget for almost all validators in the absence of faults. We then quantified the effect of bandwidth constraints, showing that seeding is relatively insensitive while sampling completion (especially the tail) depends strongly on available capacity. Finally, we evaluated omission-style faulty behaviour, demonstrating a gradual degradation in sampling success and an increasing dominance of tail latency as α increases, while seeding remains robust for SA≥ 4. Across the explored space, the results indicate that performance and overhead are driven primarily by segmentation and replication choices (SA and K) together with mesh/repair behaviour, while topic scaling has a smaller secondary effect within the tested range.

Code and data availability. The PeerSim simulator source code, experiment `.cfg` files, plotting scripts, and reproduction instructions are publicly available at `https://github.com/calmKyle/Peersim_GossipSub.git`. The repository includes a `README.md` file describing how to build the simulator, run the reported configurations, and regenerate the CSV outputs and figures.

9.2 Answers to Research Questions

RQ1 (Performance): How do parameters such as K-Copies and Segment Amount affect the performance of GossipSub under a DAS workload? Performance – captured by seeding and sampling completion times and by the fraction of validators meeting the 4 s deadline—is primarily shaped by how the workload is partitioned and replicated. Increasing Segment Amount (SA) from very coarse settings substantially improves performance by reducing the dissemination unit size and enabling parallel propagation across

topics; however, the improvement exhibits diminishing returns beyond roughly $SA \in \{4, 8\}$. Replication (K-Copies) increases robustness by spreading each segment to more peers during seeding, which helps reduce tail latency under faults, but in healthy networks higher K mainly increases redundancy rather than providing proportional latency gains. Overall, the practical operating region identified in our baseline is $SA = 8$ with moderate replication, and we adopt $K = 4$ as the default setting for the remainder of the thesis to provide clearer steady-state behaviour and an additional robustness margin in the presence of faults.

RQ2 (Overhead): What is the bandwidth and duplication cost of GossipSub under a DAS-style dissemination task? GossipSub overhead is dominated by redundant deliveries from eager mesh forwarding and by additional repair traffic triggered through metadata advertisements and pull-based recovery. In healthy settings, duplication increases systematically with K because the same segment is intentionally injected to more peers, increasing overlapping delivery paths. Increasing SA reduces duplication from coarse to moderate segmentation, after which the gains diminish. Topic count has a smaller secondary effect: it shifts absolute overhead modestly but does not change the main SA/K trends. Under omission, measured duplication decreases with α because faulty peers stop forwarding; this reduction reflects weakened effective connectivity rather than improved efficiency. Overhead therefore remains controllable primarily through avoiding excessive replication while selecting a moderate SA that prevents large coarse-grained payload duplication.

RQ3 (Resilience and scalability): How do performance and overhead degrade as the faulty fraction α increases? As the omission fraction α increases, sampling becomes increasingly tail-latency limited. Up to moderate α, the mesh plus lazy-pull repair provides enough alternative paths for validators to route around missing forwards, keeping deadline misses relatively rare. At $\alpha = 0.5$, sampling success degrades to roughly ~ 0.5 in the tested TOPICS= 256 settings, indicating a practically degraded regime where many validators miss the 4 s deadline even though seeding can remain near-perfect for SA≥ 4. In parallel, overhead becomes more uneven across nodes: some validators incur substantial repair/retry traffic while others complete quickly or fail due to limited effective neighbourhoods. At larger scales, these tail effects become more visible because the population increases the likelihood of "unlucky" neighbourhoods, making p95/p99 completion times and deadline-miss rates more informative than averages.

9.3 Remaining Limitations

While the simulator enables controlled, large-scale experimentation, several limitations remain. First, we adopt optimistic networking assumptions (perfect discovery, stable membership, and homogeneous resources) that likely underestimate tail latency relative to real deployments with churn, heterogeneous bandwidth, NAT constraints, and variable link delay. Second, cryptographic verification and decoding costs are abstracted away, reducing realism in how clients allocate the slot budget between networking and computation. Third, the adversary model is limited to omission and crash/offline behaviours and does not quantify

other relevant attacks such as topic flooding, Sybil-driven neighbour capture, or targeted eclipse strategies. Finally, the custody/topic mapping and stopping criteria are simplified to support clean measurement; alternative custody assignments and more realistic membership dynamics may change absolute success levels.

9.4 Suggested Follow-ups

Several directions can extend this work toward more realistic and more resilient designs. A first step is to introduce partial discovery and churn, heterogeneous bandwidth distributions, and explicit RTT/queueing models to better capture tail behaviour under real networking conditions. A second direction is to broaden the fault model beyond omission, including controlled spam/flooding within topics and targeted neighbour bias, to stress scoring and mesh adaptation under resource pressure.

On the protocol side, future work can explore adaptive parameter tuning, such as dynamically increasing lazy repair fanout or parallelising requests when sampling progress stalls, or adjusting $(D, D_{\text{low}}, D_{\text{high}})$ in response to observed completion tails to mitigate deadline misses under elevated fault rates while keeping duplication bounded in healthy conditions. Finally, validating the main trends on a small testbed (or in a hybrid hardware-in-the-loop setup) would help confirm that the simulator's conclusions persist under real transport behaviour and implementation constraints.

References

[1] M. Al-Bassam, A. Sonnino, and V. Buterin, "Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities," *CoRR*, vol. abs/1809.09044, 2018. arXiv: 1809.09044. [Online]. Available: http://arxiv.org/abs/1809.09044.

[2] S. Chaliasos, D. Firsov, and B. Livshits, *Towards a formal foundation for blockchain rollups*, 2025. arXiv: 2406.16219 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2406.16219.

[3] A. Chaudhuri, S. Basak, C. Kiraly, D. Ryajov, and L. Bautista-Gomez, *On the design of ethereum data availability sampling: A comprehensive simulation study*, 2024. arXiv: 2407.18085 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18085.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, New York, NY, USA: ACM, Aug. 1987, pp. 1–12. DOI: 10.1145/41840.41841. [Online]. Available: https://dl.acm.org/doi/10.1145/41840.41841.

[5] J. (Douceur, "The sybil attack," in *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Jan. 2002. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-sybil-attack/.

[6] P. T. Eugster, R. Guerraoui, A. Kermarrec, and L. Massoulié, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, May 2004. DOI: 10.1109/MC.2004.1297243. [Online]. Available: https://dl.acm.org/doi/10.1109/MC.2004.1297243.

[7] *FullDAS: Towards massive scalability with 32MB blocks and beyond — ethresear.ch*, https://ethresear.ch/t/fulldas-towards-massive-scalability-with-32mb-blocks-and-beyond/19529, [Accessed 27-02-2026].

[8] P. Gaži, A. Kiayias, and D. Zindros, "Proof-of-stake sidechains," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 139–156. DOI: 10.1109/SP.2019.00040.

[9] L. Heimbach, L. Kiffer, C. Ferreira Torres, and R. Wattenhofer, "Ethereum's proposer-builder separation: Promises and realities," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, ser. IMC '23, Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 406–420, ISBN: 9798400703829. DOI: 10.1145/3618257.3624824. [Online]. Available: https://doi.org/10.1145/3618257.3624824.

[10] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking, "Randomized rumor spreading," in *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, IEEE, 2000, pp. 565–574. DOI: 10.1109/SFCS.2000.892324. [Online]. Available: https://ieeexplore.ieee.org/document/892324.

[11] M. Król, O. Ascigil, S. Rene, E. Rivière, M. Pigaglio, K. Peeroo, V. Stankovic, R. Sadre, and F. Lange, *Data availability sampling in ethereum: Analysis of p2p networking requirements*, 2023. arXiv: 2306.11456 [cs.NI]. [Online]. Available: https://arxiv.org/abs/2306.11456.

[12] A. Kumar, M. von Hippel, P. Manolios, and C. Nita-Rotaru, *Formal model-driven analysis of resilience of gossipsub to attacks from misbehaving peers*, 2023. arXiv: 2212.05197 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2212.05197.

[13] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, pp. 382–401, Jul. 1982. [Online]. Available: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

[14] libp2p contributors, *libp2p/specs: Technical specifications for the libp2p networking stack*, GitHub repository, Default branch state pinned to commit e87cb1c32a666c2229d3b9bb8f9ce1d9 (Dec 11, 2025). Accessed: 2026-02-14. [Online]. Available: https://github.com/libp2p/specs.

[15] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85 727–85 745, 2019. DOI: 10.1109/ACCESS.2019.2925010.

[16] *PeerDAS – a simpler DAS approach using battle-tested p2p components — ethresear.ch*, https://ethresear.ch/t/peerdas-a-simpler-das-approach-using-battle-tested-p2p-components/16541, [Accessed 27-02-2026].

[17] M. Pigaglio, O. Ascigil, M. Król, S. Rene, F. Lange, K. Peeroo, R. Sadre, V. Stankovic, and E. Rivière, *Pandas: Peer-to-peer, adaptive networking for data availability sampling within ethereum consensus timebounds*, 2025. arXiv: 2507.00824 [cs.DC]. [Online]. Available: https://arxiv.org/abs/2507.00824.

[18] P. Soltani and F. Ashtiani, *Delay analysis of eip-4844*, 2024. arXiv: 2409.11043 [cs.DC]. [Online]. Available: https://arxiv.org/abs/2409.11043.

[19] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "Gossipsub: Attack-resilient message propagation in the filecoin and ETH2.0 networks," *CoRR*, vol. abs/2007.02754, 2020. arXiv: 2007.02754. [Online]. Available: https://arxiv.org/abs/2007.02754.

[20] D. Vyzovitis, Y. Psaras, D. Dias, *et al.*, "Gossipsub v1.1 evaluation report," Protocol Labs, Tech. Rep. PL-TechRep-2020-001, 2020. [Online]. Available: https://research.protocol.ai/publications/gossipsub-v1.1-evaluation-report/vyzovitis2020.pdf.

[21] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, Oct. 1999, ISBN: 0780353919. [Online]. Available: https://ieeexplore.ieee.org/book/5264570.

A Supplementary Bandwidth, Latency, and Duplication Results by Omission Rate

A.1 Supplementary results for $\alpha = 0.0$

Figure 14 shows the per-node bandwidth distribution in the healthy network under the final configuration. This plot is used to check that the 60 Mbit/s cap leaves sufficient headroom and that the baseline run is not dominated by bandwidth saturation.

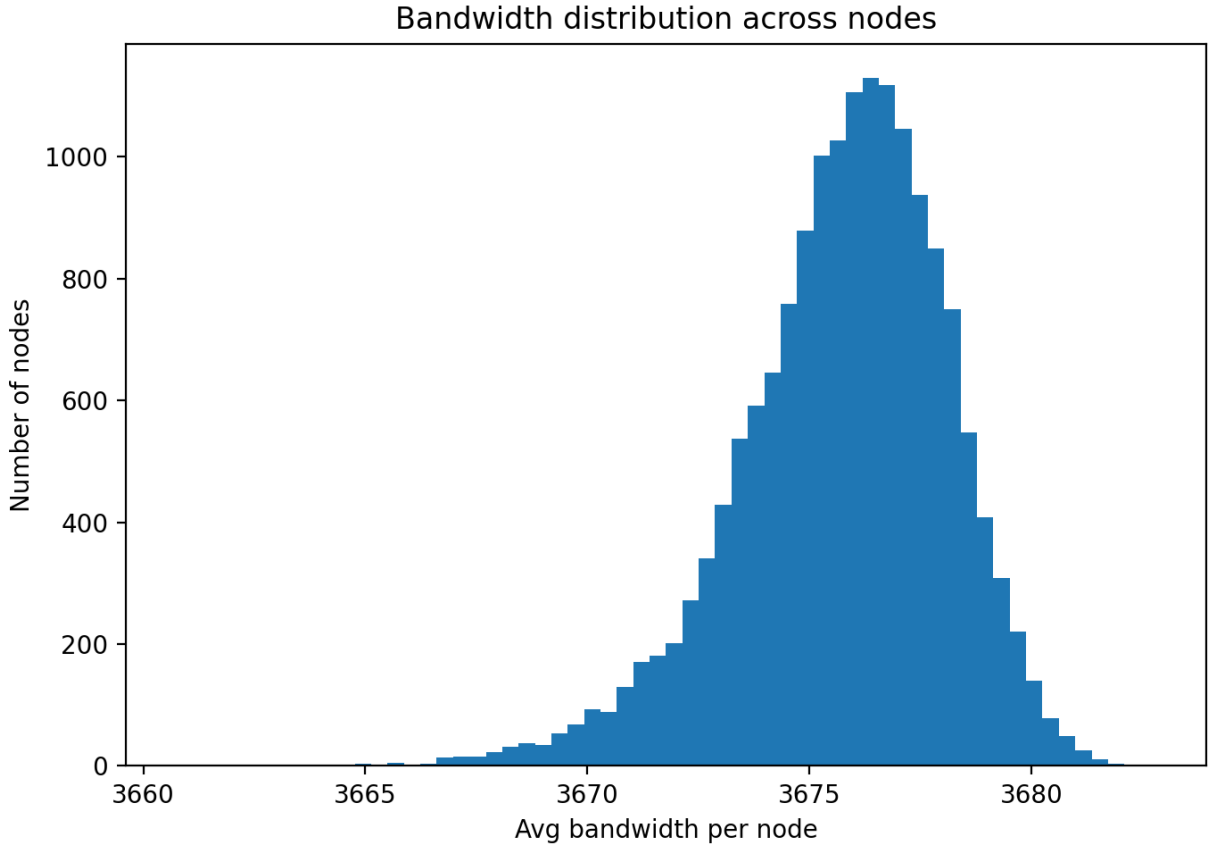


Figure 14: Bandwidth distribution across nodes: histogram of the average bandwidth per node ((MR= 0.0, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 15 shows the per-slot timing CDFs for the same healthy configuration. The key takeaway is that both seeding and sampling remain comfortably below the 4000 ms deadline in all five slots.

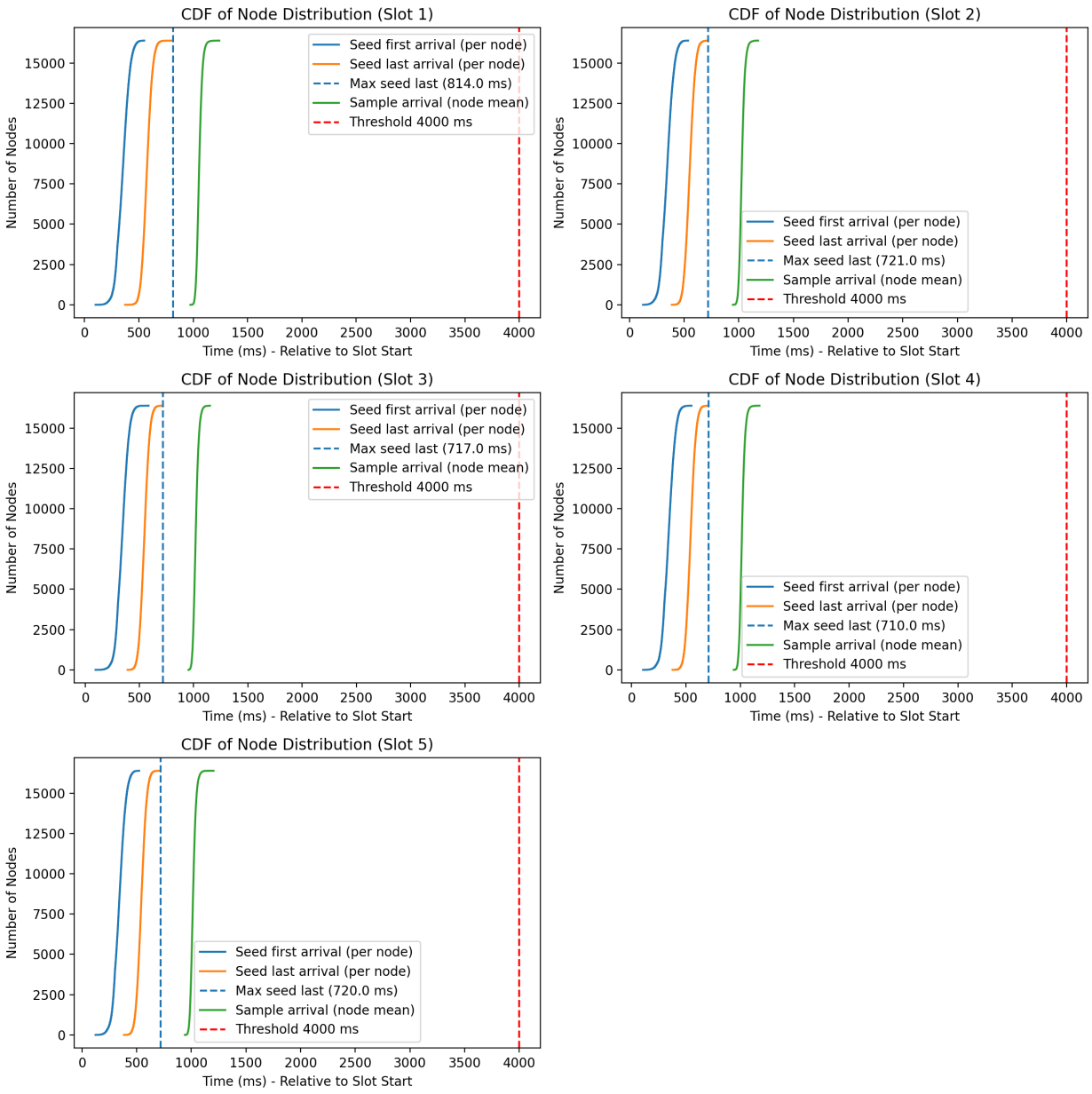


Figure 15: CDF of node arrival times per slot (Slots 1-5), measured relative to slot start. Curves show seed first/last arrival (per node) and sample arrival (node mean); dashed lines mark the maximum seed-last arrival and the 4000 ms threshold ((MR= 0.0, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 16 shows the per-slot duplication distribution as violin plots. The figure highlights the spread of duplicate deliveries across nodes and slots in the absence of omission faults.

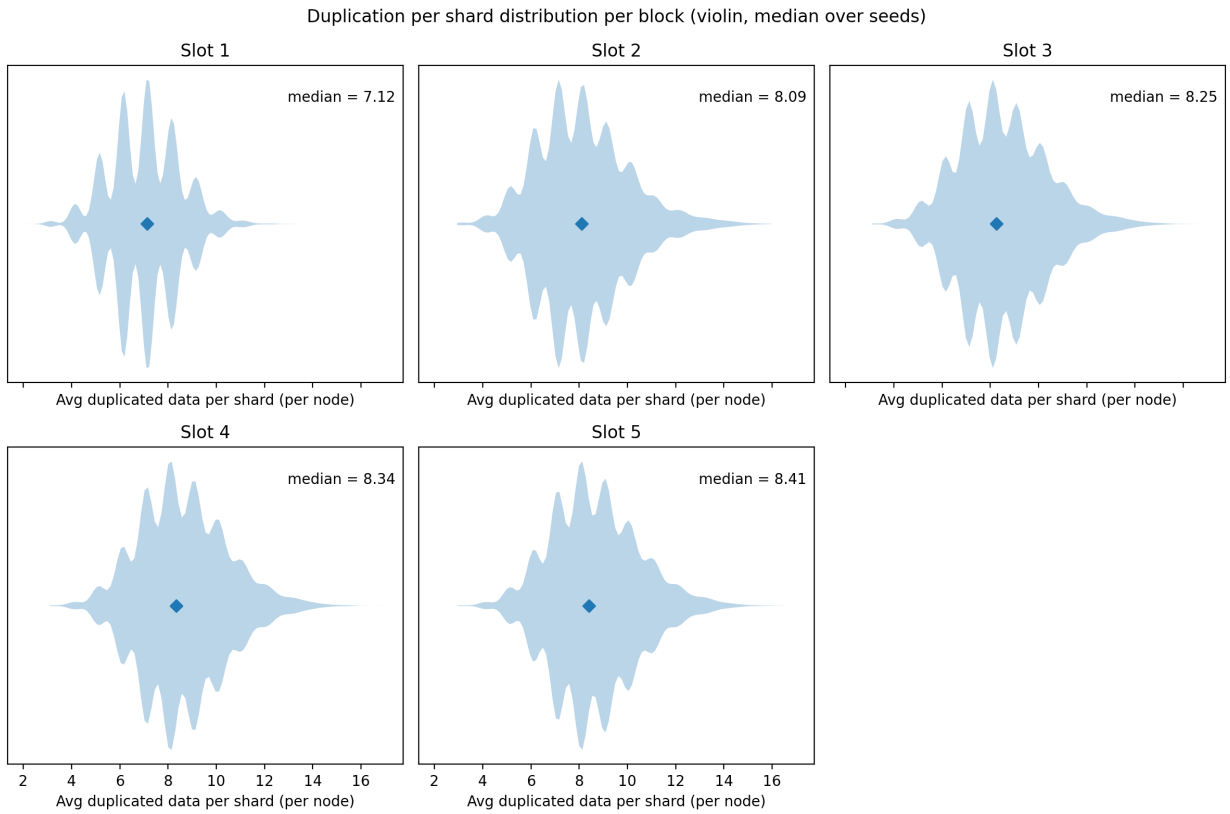


Figure 16: Per-slot duplication per shard per block: violin plots (Slots 1–5) of average duplicated data per shard (per node), aggregated as the median over seeds; diamonds indicate the median ((MR= 0.0, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 17 shows the same per-slot duplication behaviour as histograms, making the modal duplication level in each slot easier to inspect.

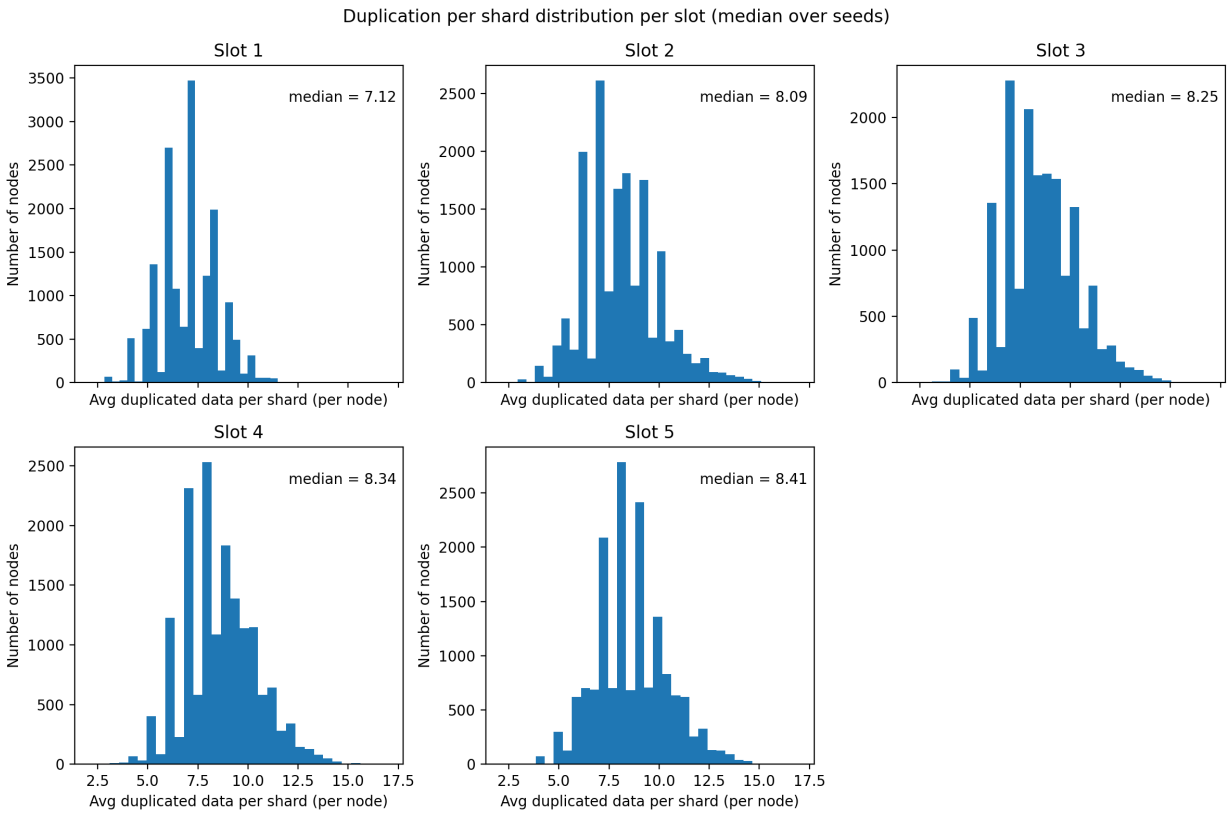


Figure 17: Per-slot duplication per shard distribution (Slots 1–5): histograms of average duplicated data per shard (per node), aggregated as the median over seeds ((MR= 0.0, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 18 aggregates duplication across all slots. This provides the overall healthy-network duplication baseline against which the omission-rate cases are compared.

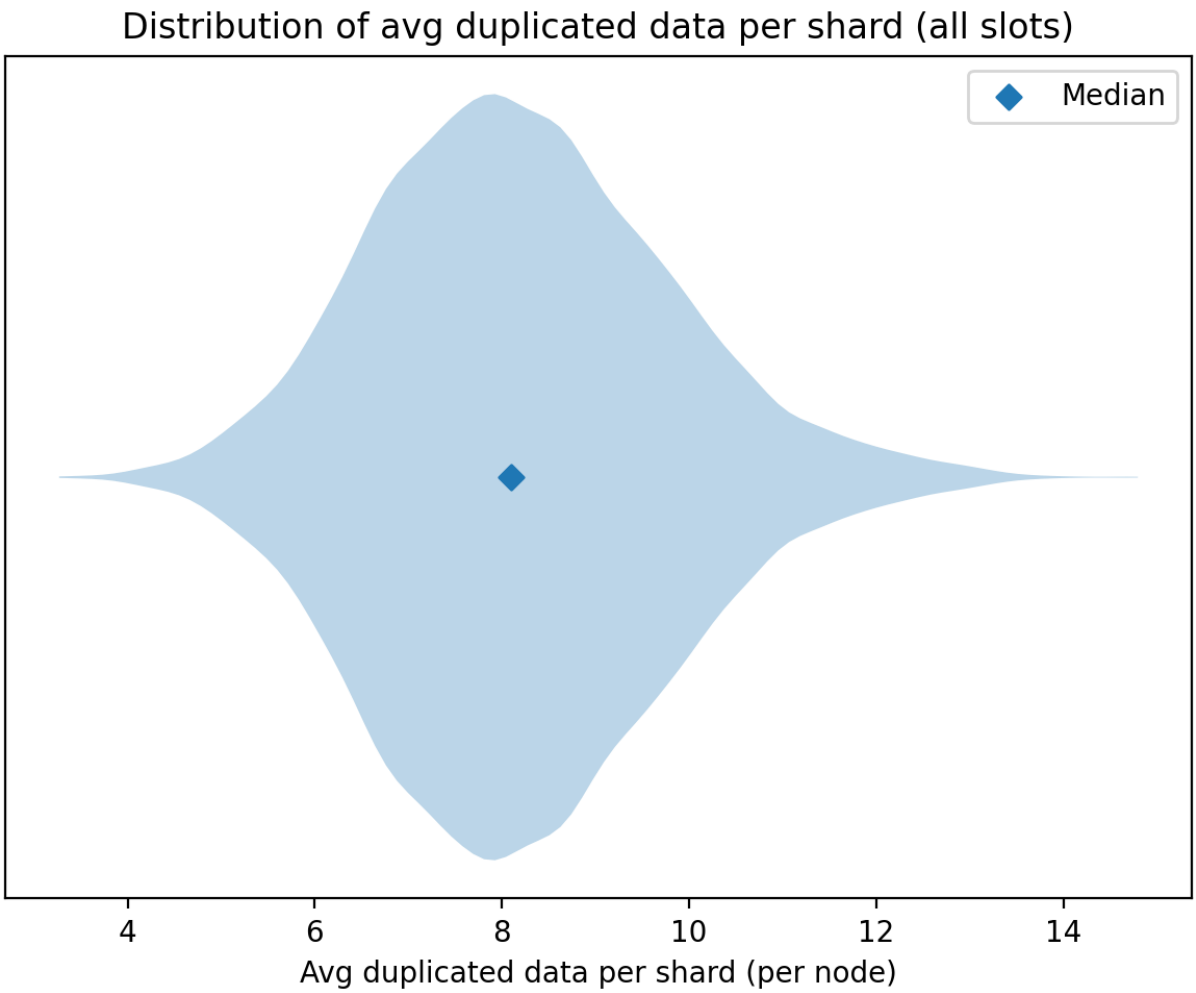


Figure 18: Overall duplication per shard (all slots combined): violin plot of average duplicated data per shard (per node); the diamond marks the median ((MR= 0.0, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

A.2 Supplementary results for $\alpha = 0.1$

Figure 19 shows the per-node bandwidth distribution at $\alpha = 0.1$. This plot indicates how the first omission setting changes transmission load relative to the healthy baseline.

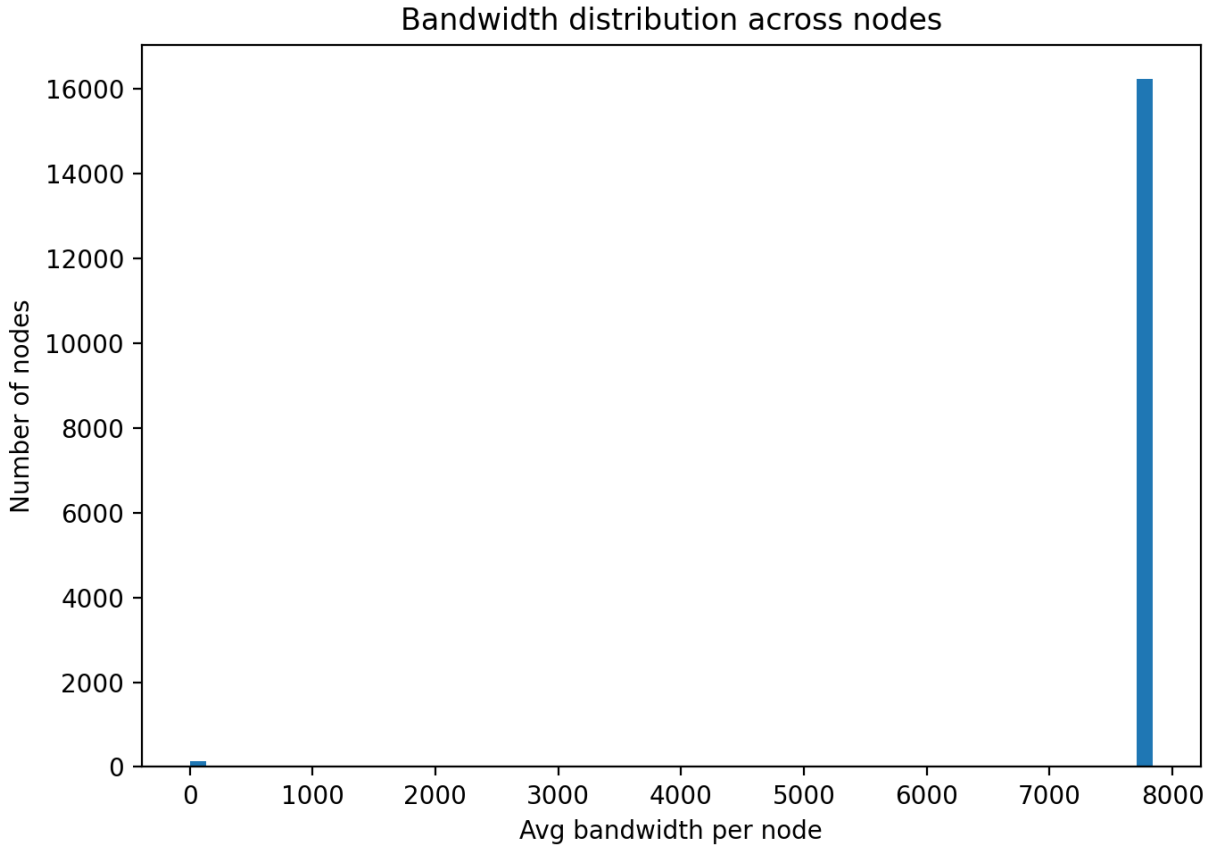


Figure 19: Bandwidth distribution across nodes: histogram of the average bandwidth per node ((MR= 0.1, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 20 shows the corresponding per-slot timing CDFs. The figure is used to check whether the seed and sample arrival distributions remain below the 4000 ms deadline.

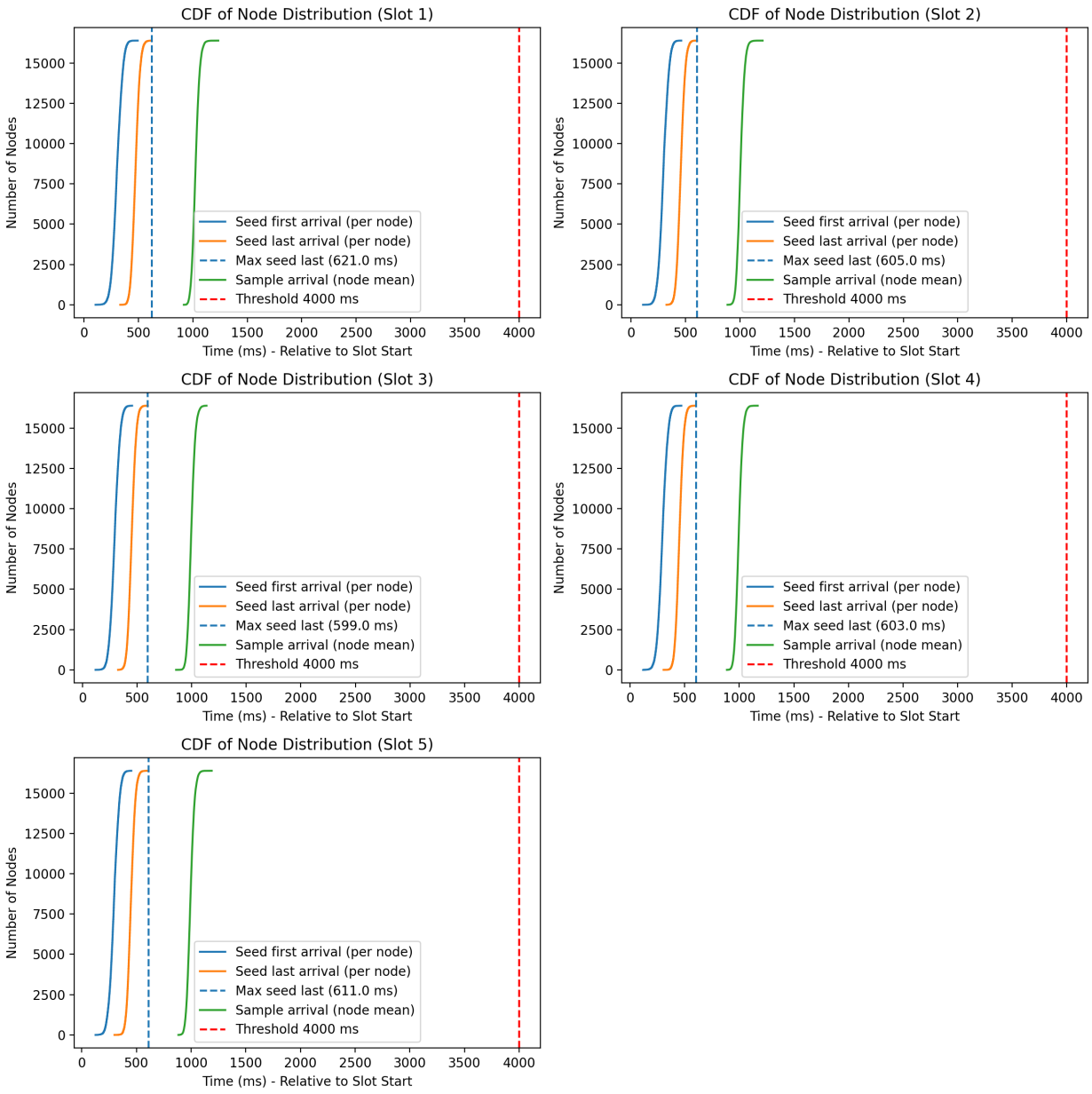


Figure 20: CDF of node arrival times per slot (Slots 1-5), measured relative to slot start. Curves show seed first/last arrival (per node) and sample arrival (node mean); dashed lines mark the maximum seed-last arrival and the 4000 ms threshold ((MR= 0.1, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 21 shows the per-slot duplication distribution as violin plots at $\alpha = 0.1$, while Figure 22 shows the same distributions as histograms. Together, these plots show how small omission rates begin to reduce redundant forwarding.

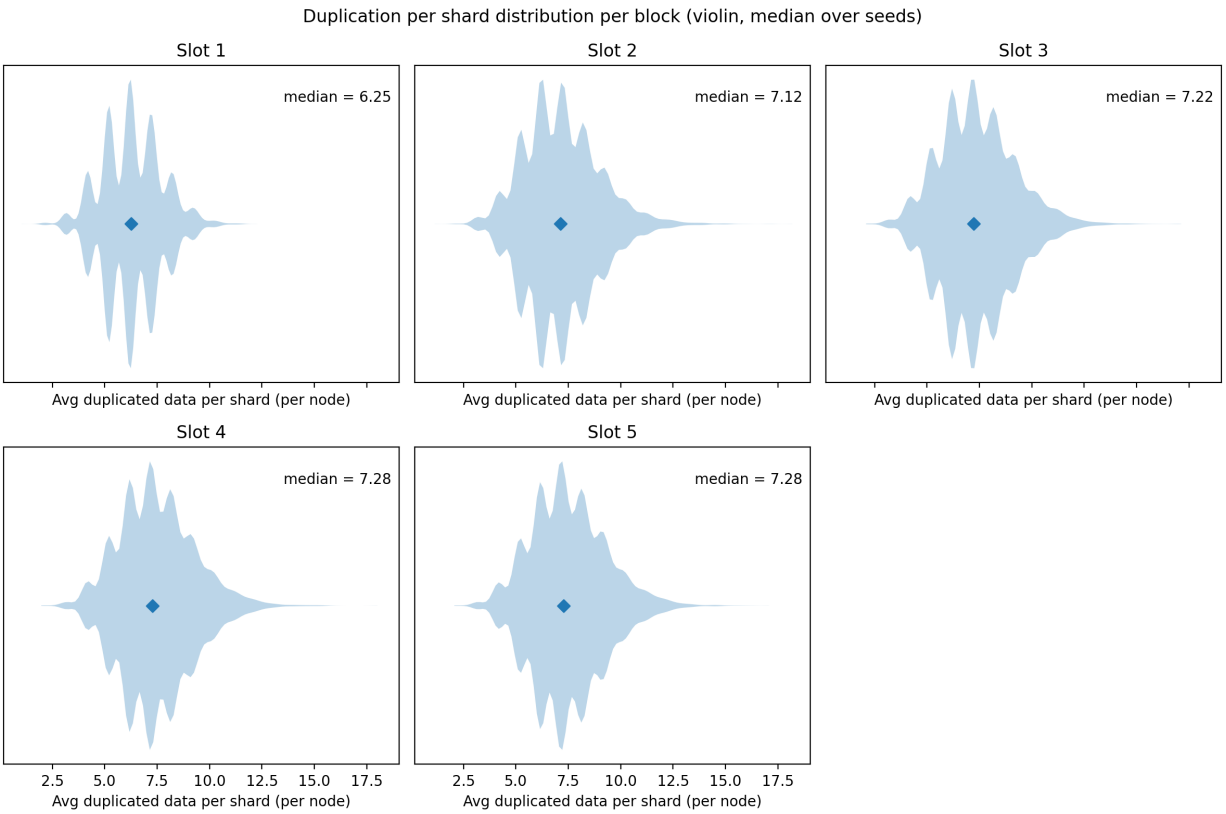


Figure 21: Per-slot duplication per shard per block: violin plots (Slots 1–5) of average duplicated data per shard (per node), aggregated as the median over seeds; diamonds indicate the median ((MR= 0.1, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 22 shows the same per-slot duplication behaviour as histograms. This view makes the modal duplication level in each slot easier to inspect.

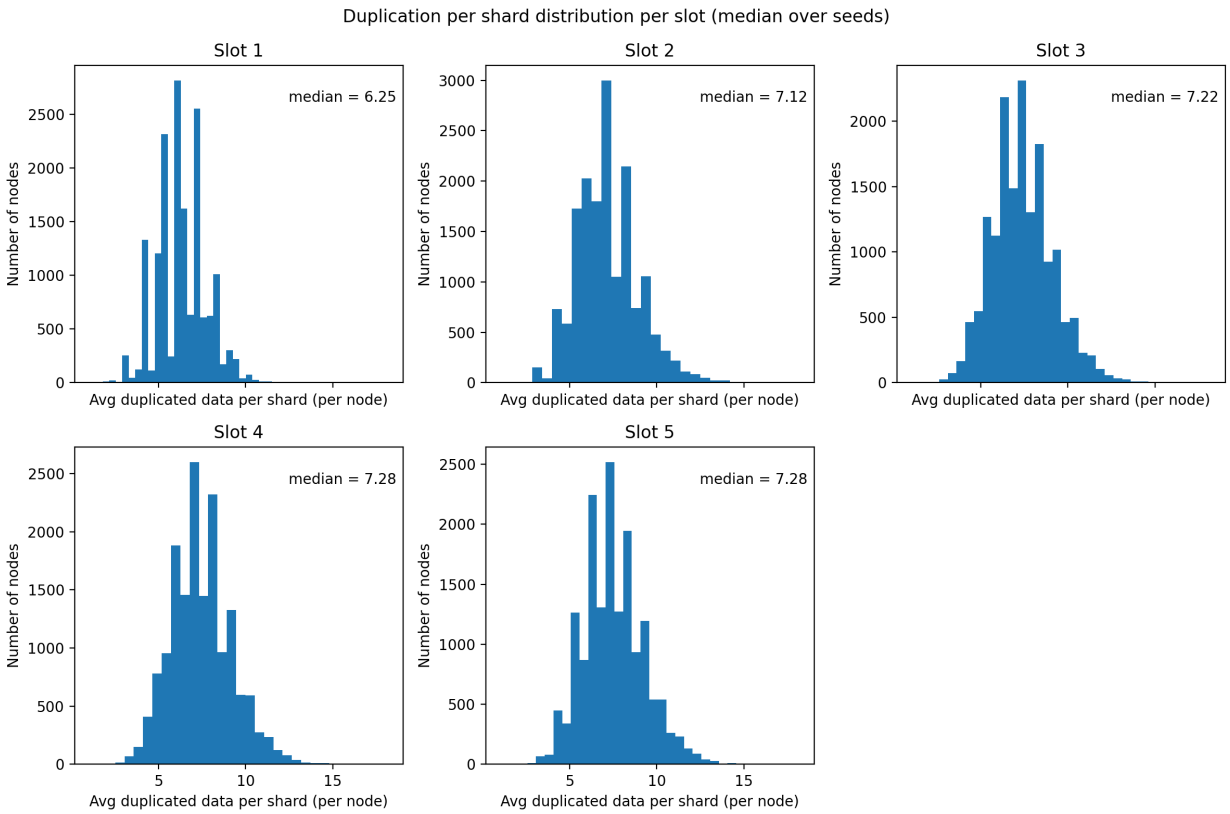


Figure 22: Per-slot duplication per shard distribution (Slots 1–5): histograms of average duplicated data per shard (per node), aggregated as the median over seeds ((MR= 0.1, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 23 aggregates the duplication distribution across all slots for $\alpha = 0.1$. This provides the overall duplication level for comparison with the healthy baseline and higher omission rates.

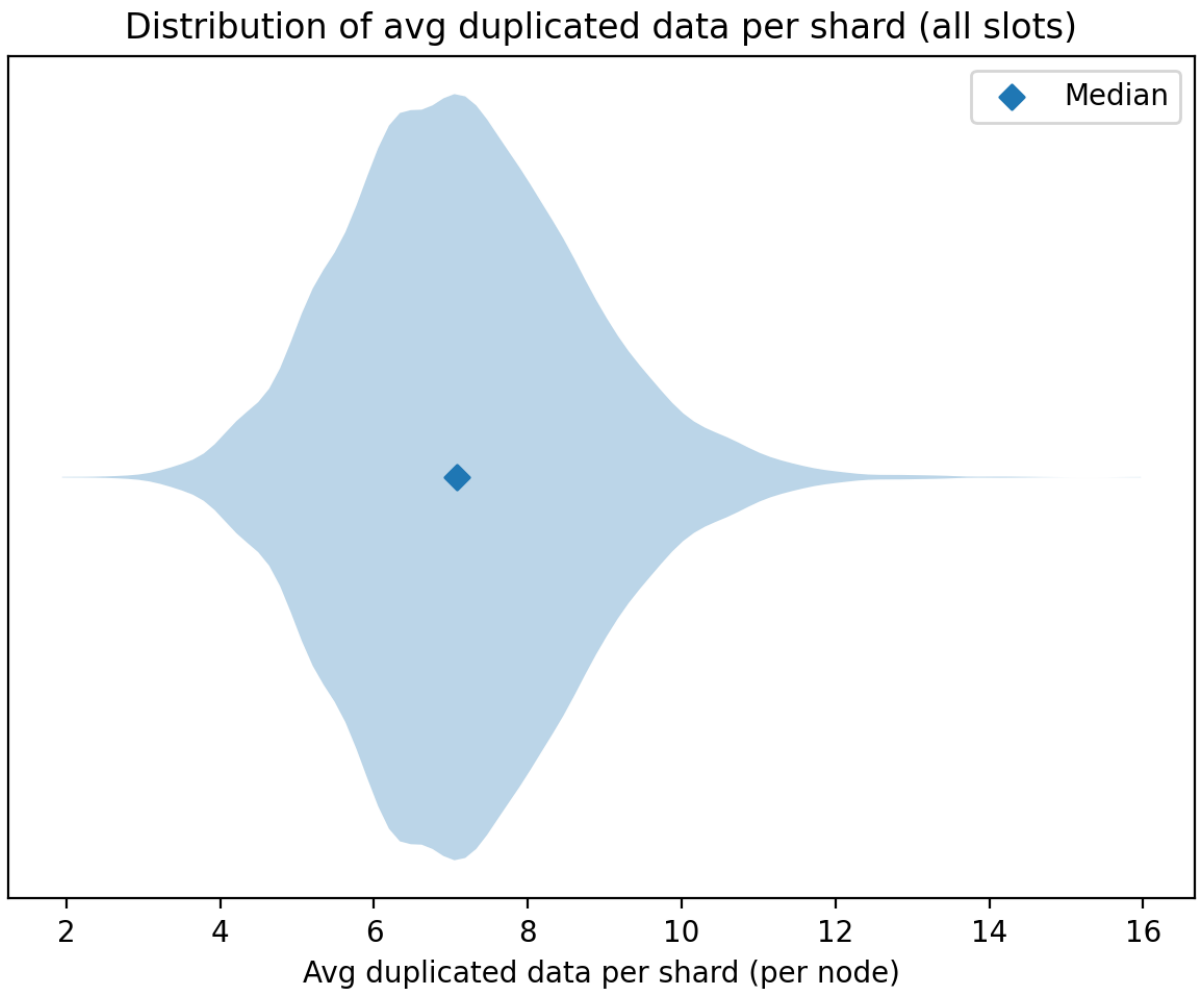


Figure 23: Overall duplication per shard (all slots combined): violin plot of average duplicated data per shard (per node); the diamond marks the median ((MR= 0.1, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

A.3 Supplementary results for $\alpha = 0.2$

Figure 24 shows the per-node bandwidth distribution at $\alpha = 0.2$. The plot helps distinguish bandwidth effects from omission-driven repair effects.

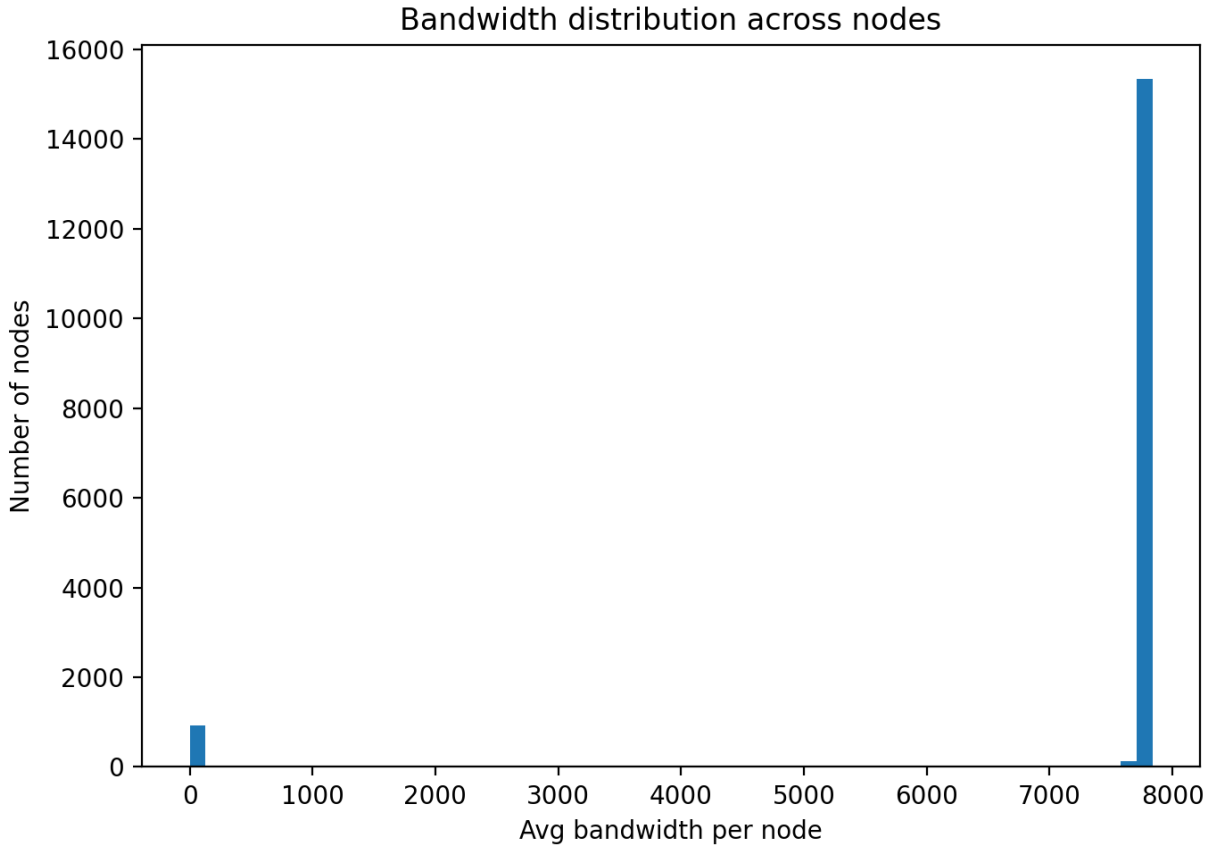


Figure 24: Bandwidth distribution across nodes: histogram of the average bandwidth per node ((MR= 0.2, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 25 shows the per-slot seeding and sampling timing CDFs at $\alpha = 0.2$. The main point is to compare the sample-arrival curve against the 4000 ms deadline.

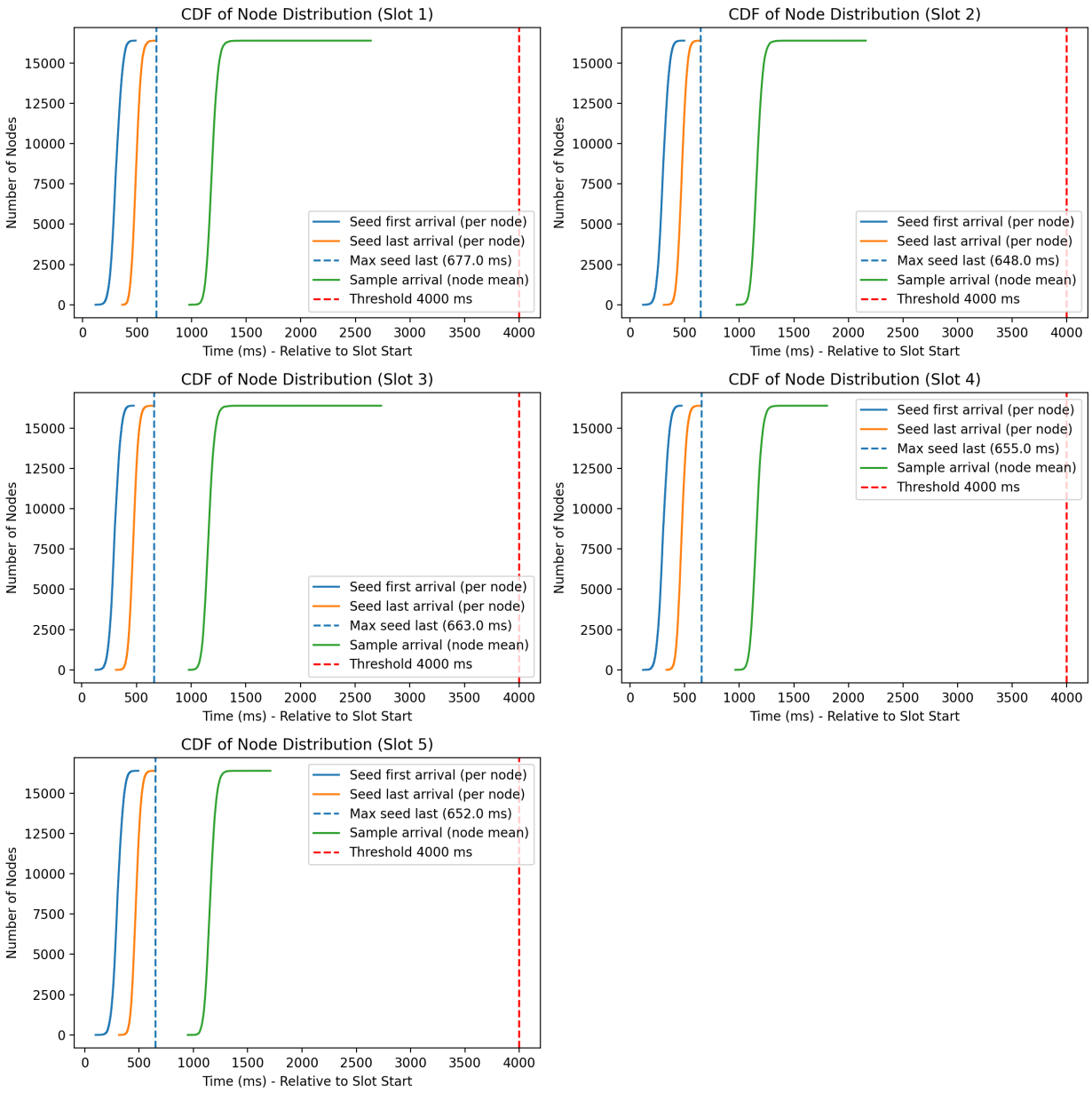


Figure 25: CDF of node arrival times per slot (Slots 1-5), measured relative to slot start. Curves show seed first/last arrival (per node) and sample arrival (node mean); dashed lines mark the maximum seed-last arrival and the 4000 ms threshold ((MR= 0.2, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 26 shows the per-slot duplication distribution as violin plots at $\alpha = 0.2$. This figure supports the main-text observation that duplication decreases as omission reduces effective forwarding.

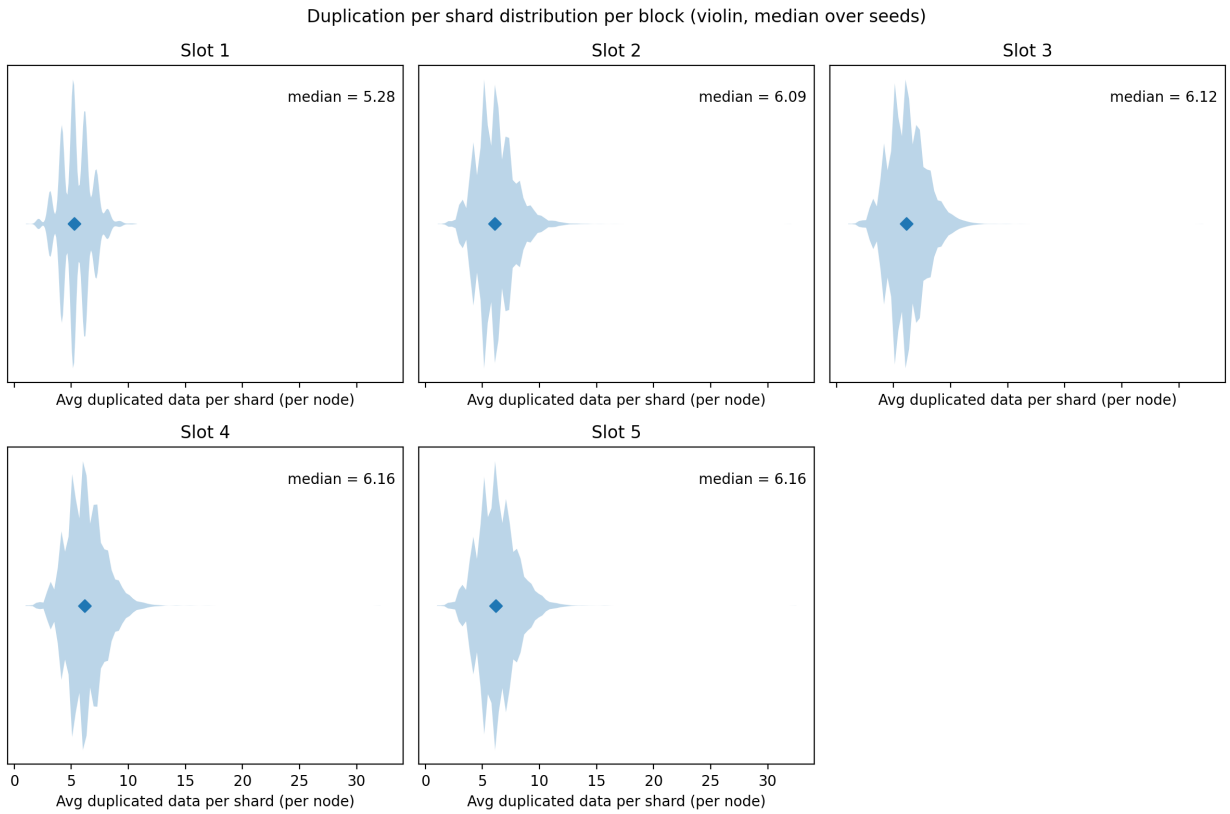


Figure 26: Per-slot duplication per shard per block: violin plots (Slots 1–5) of average duplicated data per shard (per node), aggregated as the median over seeds; diamonds indicate the median ((MR= 0.2, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 27 shows the corresponding per-slot duplication histograms at $\alpha = 0.2$. This view makes it easier to see how the distribution shifts within each slot.

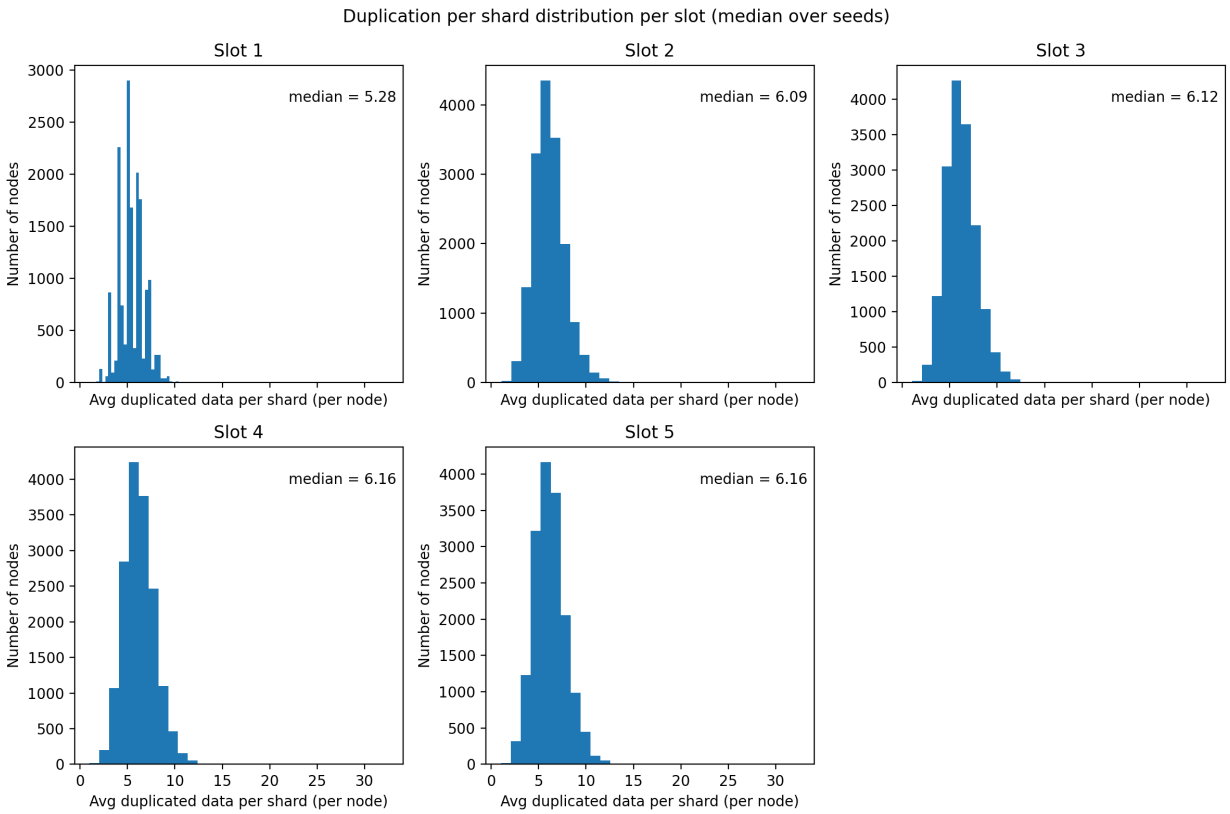


Figure 27: Per-slot duplication per shard distribution (Slots 1–5): histograms of average duplicated data per shard (per node), aggregated as the median over seeds ((MR= 0.2, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 28 shows the all-slot duplication distribution for $\alpha = 0.2$. This figure provides the aggregate duplication level used to compare omission rates.

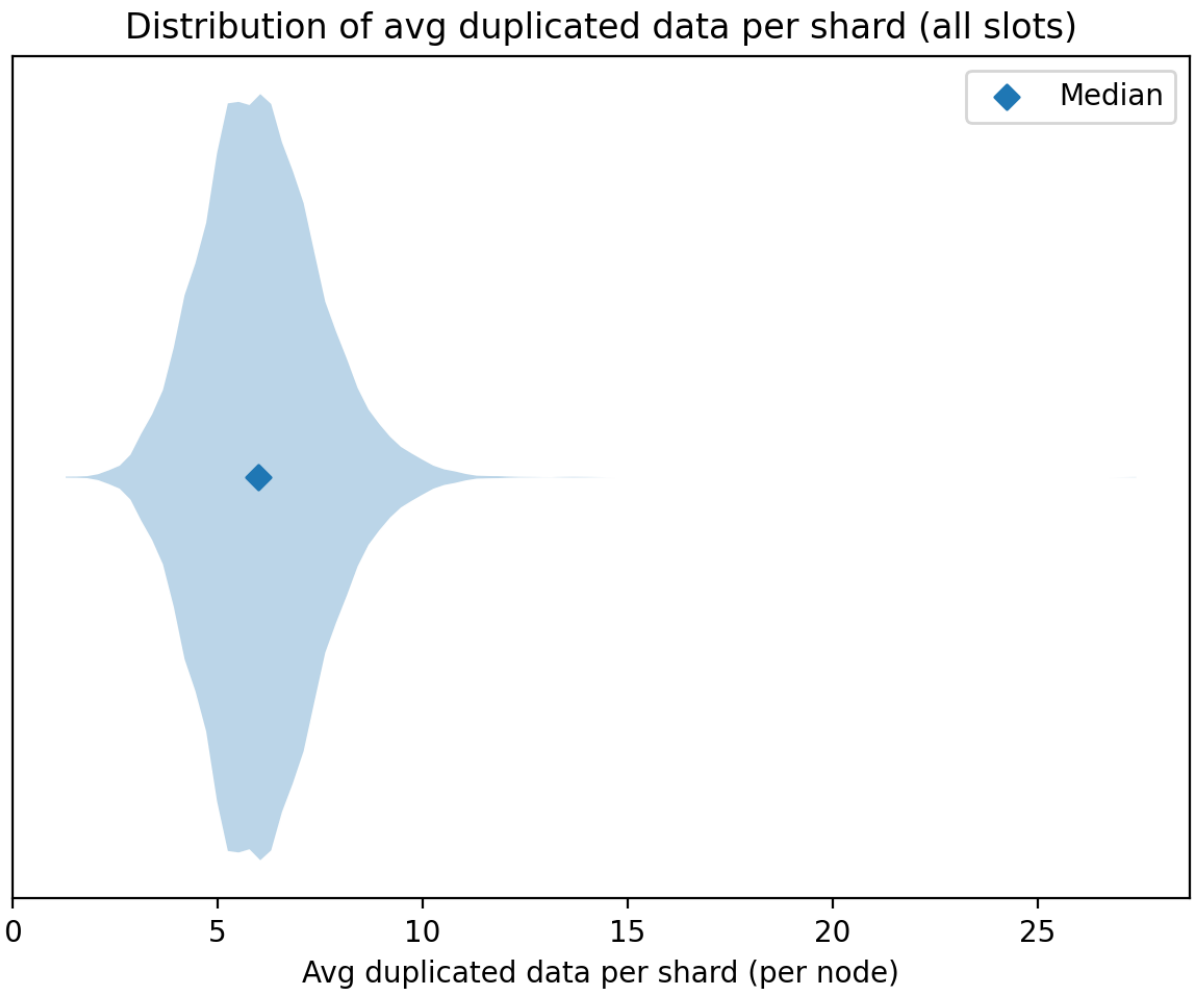


Figure 28: Overall duplication per shard (all slots combined): violin plot of average duplicated data per shard (per node); the diamond marks the median ((MR= 0.2, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

A.4 Supplementary results for $\alpha = 0.3$

Figure 29 shows the per-node bandwidth distribution at $\alpha = 0.3$, the representative omission setting used in the main overhead discussion.

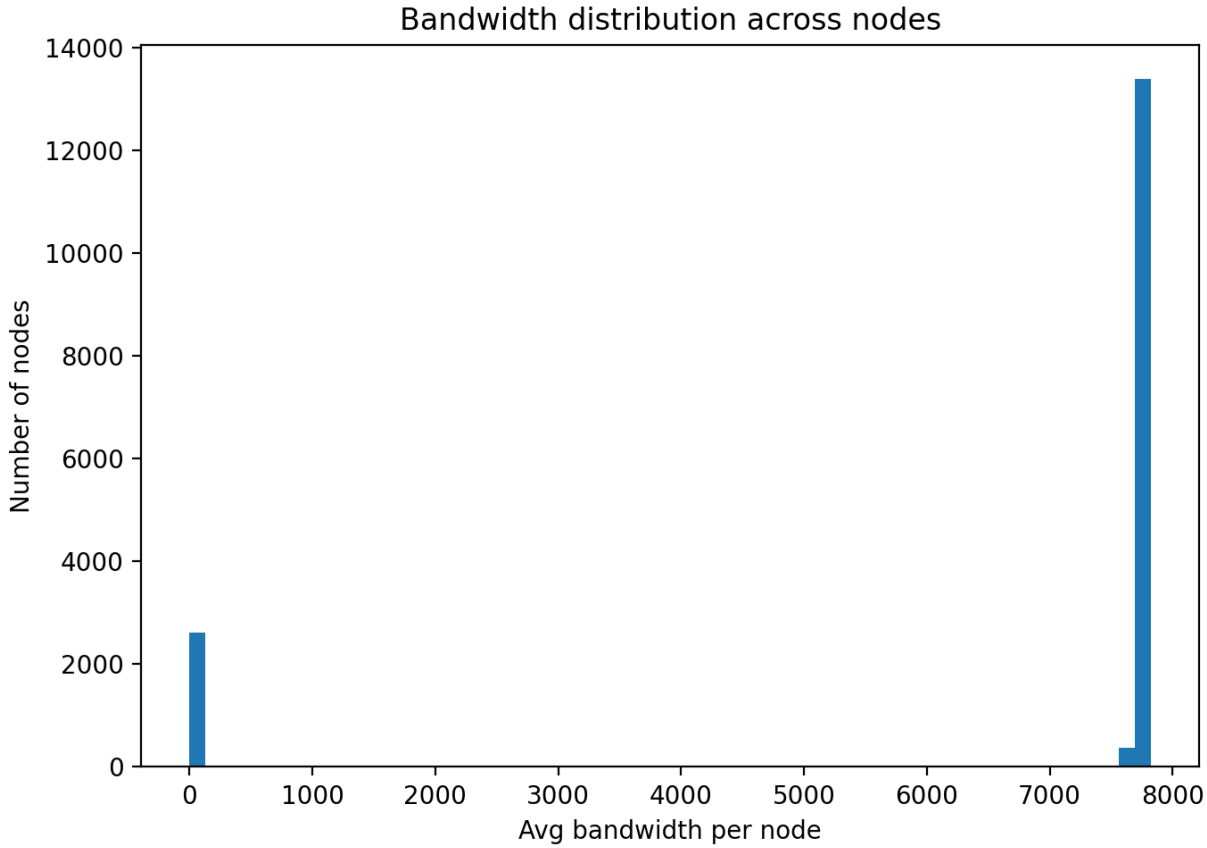


Figure 29: Bandwidth distribution across nodes: histogram of the average bandwidth per node ((MR= 0.3, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 30 shows the per-slot timing CDFs at $\alpha = 0.3$. This figure helps show whether the sampling curve remains within the DAS deadline after omission begins to meaningfully reduce available forwarding paths.

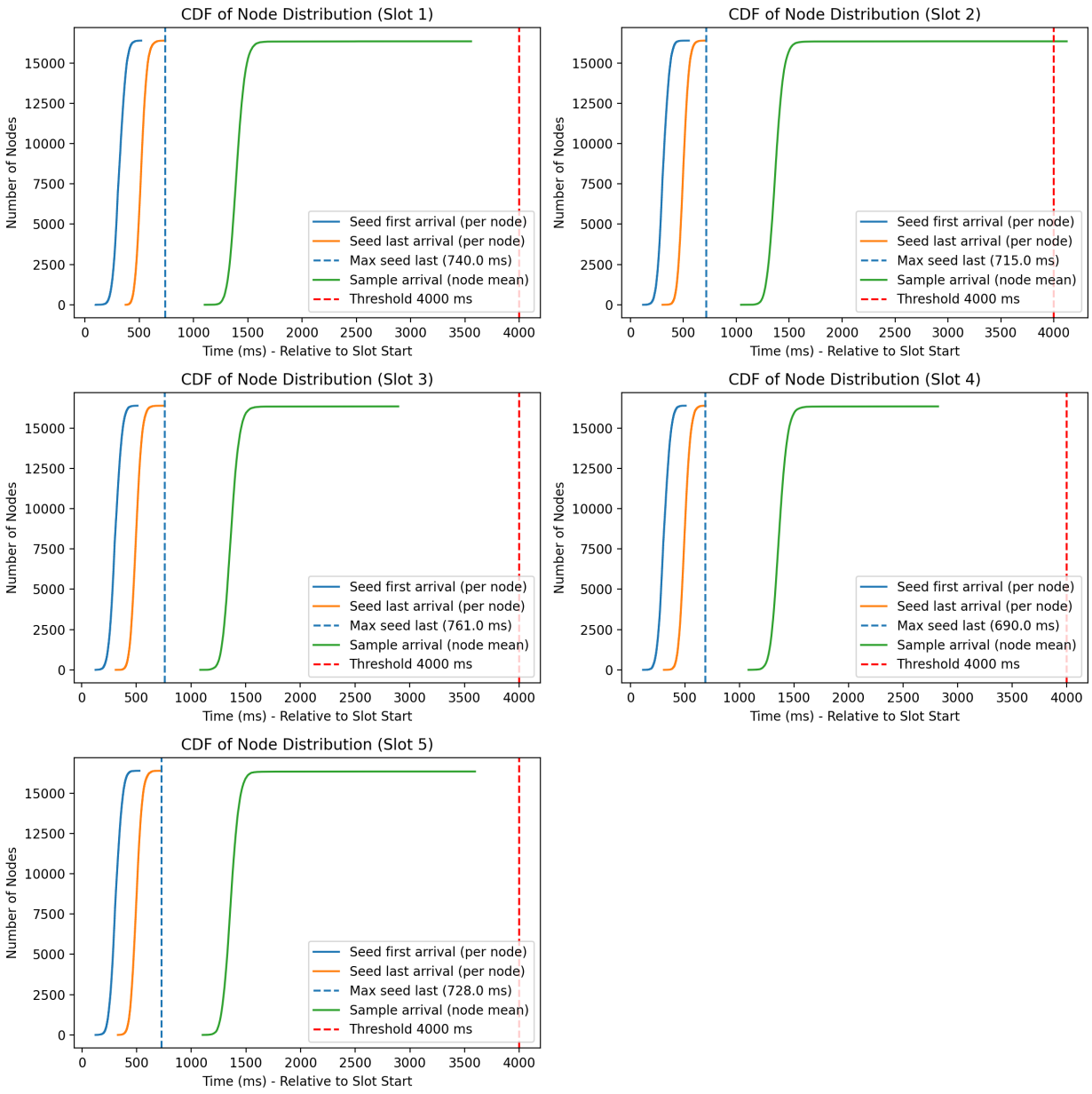


Figure 30: CDF of node arrival times per slot (Slots 1-5), measured relative to slot start. Curves show seed first/last arrival (per node) and sample arrival (node mean); dashed lines mark the maximum seed-last arrival and the 4000 ms threshold ((MR= 0.3, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 31 shows the per-slot duplication distribution as violin plots at $\alpha = 0.3$. This is the appendix counterpart to the main-text steady-state discussion: it shows the slot-by-slot duplication behaviour under the final configuration.

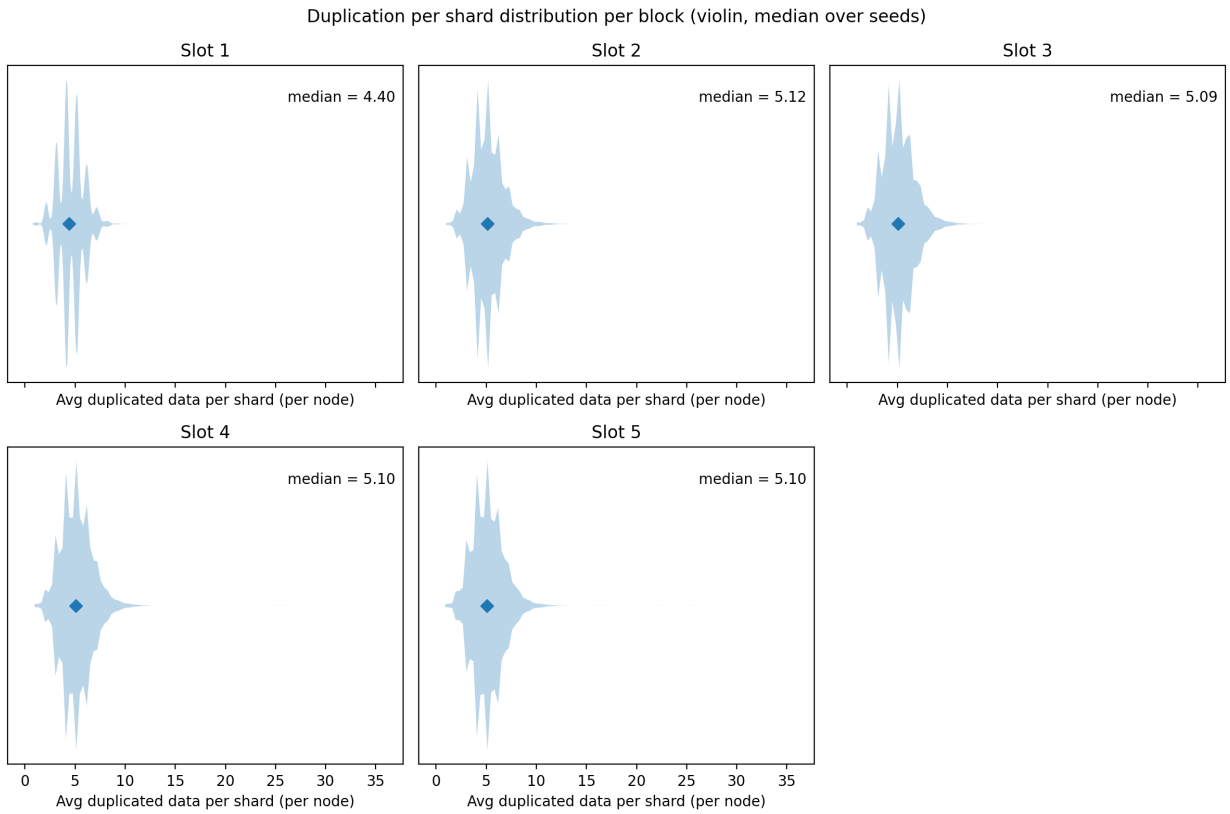


Figure 31: Per-slot duplication per shard per block: violin plots (Slots 1–5) of average duplicated data per shard (per node), aggregated as the median over seeds; diamonds indicate the median ((MR= 0.3, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 32 shows the same per-slot duplication data as histograms. This view helps identify the most common duplication-per-shard range in each slot.

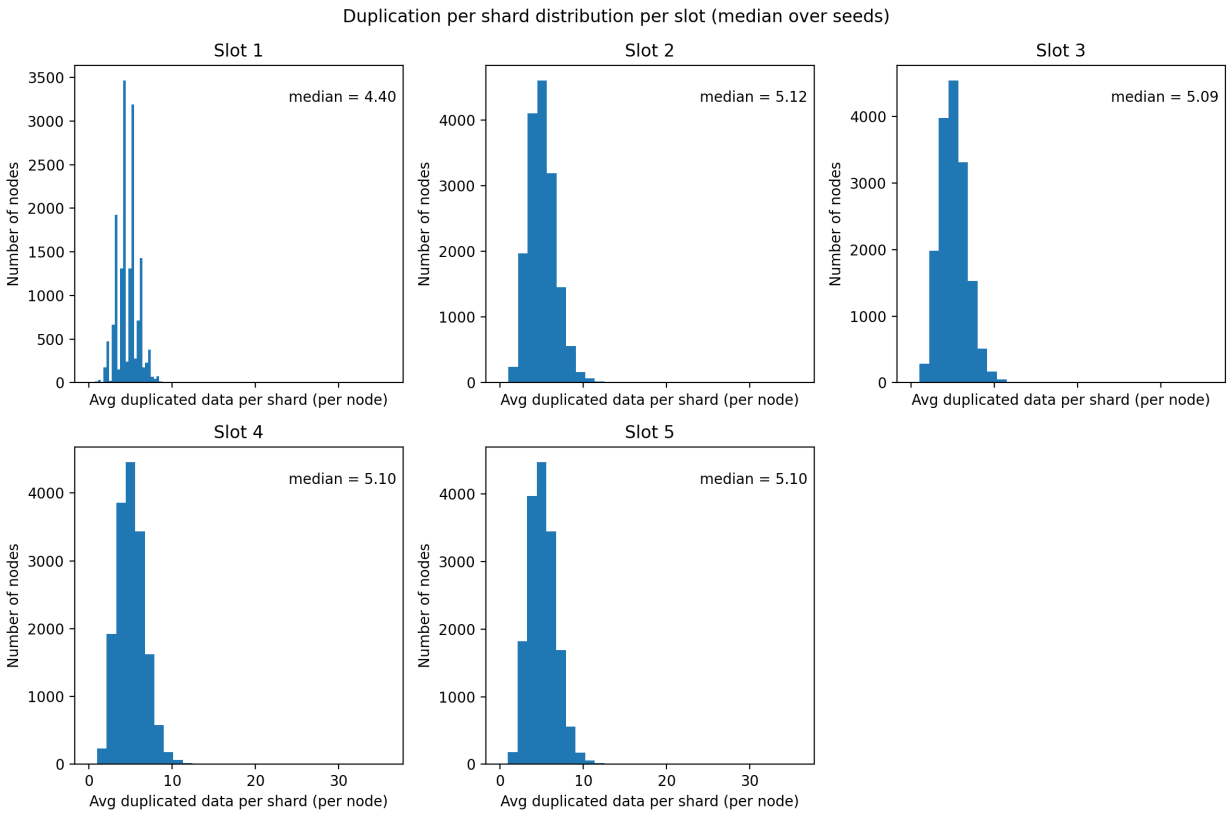


Figure 32: Per-slot duplication per shard distribution (Slots 1–5): histograms of average duplicated data per shard (per node), aggregated as the median over seeds ((MR= 0.3, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 33 aggregates duplication across all slots at $\alpha = 0.3$. This figure provides the overall duplication distribution for the representative omission setting.

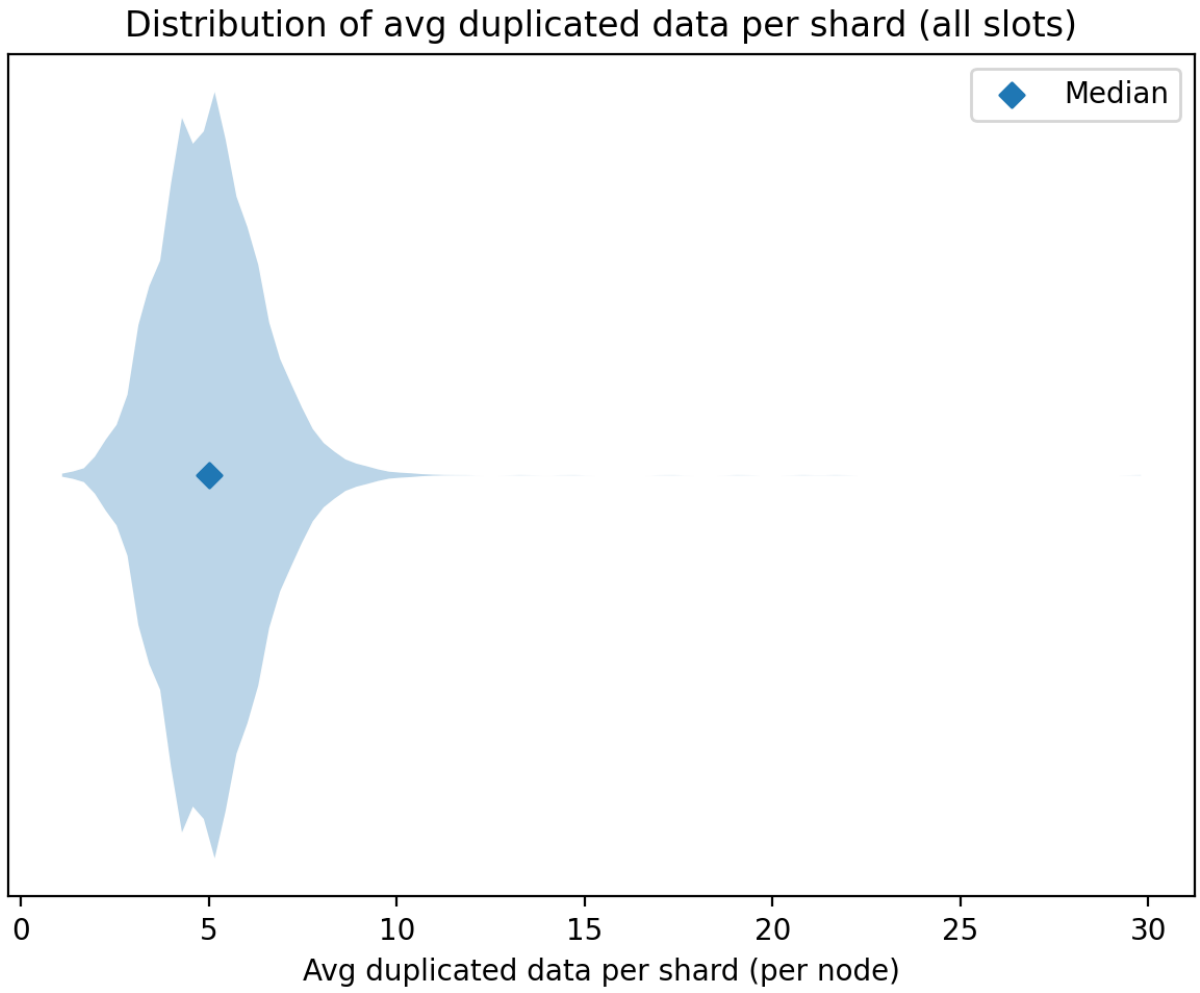


Figure 33: Overall duplication per shard (all slots combined): violin plot of average duplicated data per shard (per node); the diamond marks the median ((MR= 0.3, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

A.5 Supplementary results for $\alpha = 0.4$

Figure 34 shows the per-node bandwidth distribution at $\alpha = 0.4$, close to the upper end of the stable regime identified in the main results.

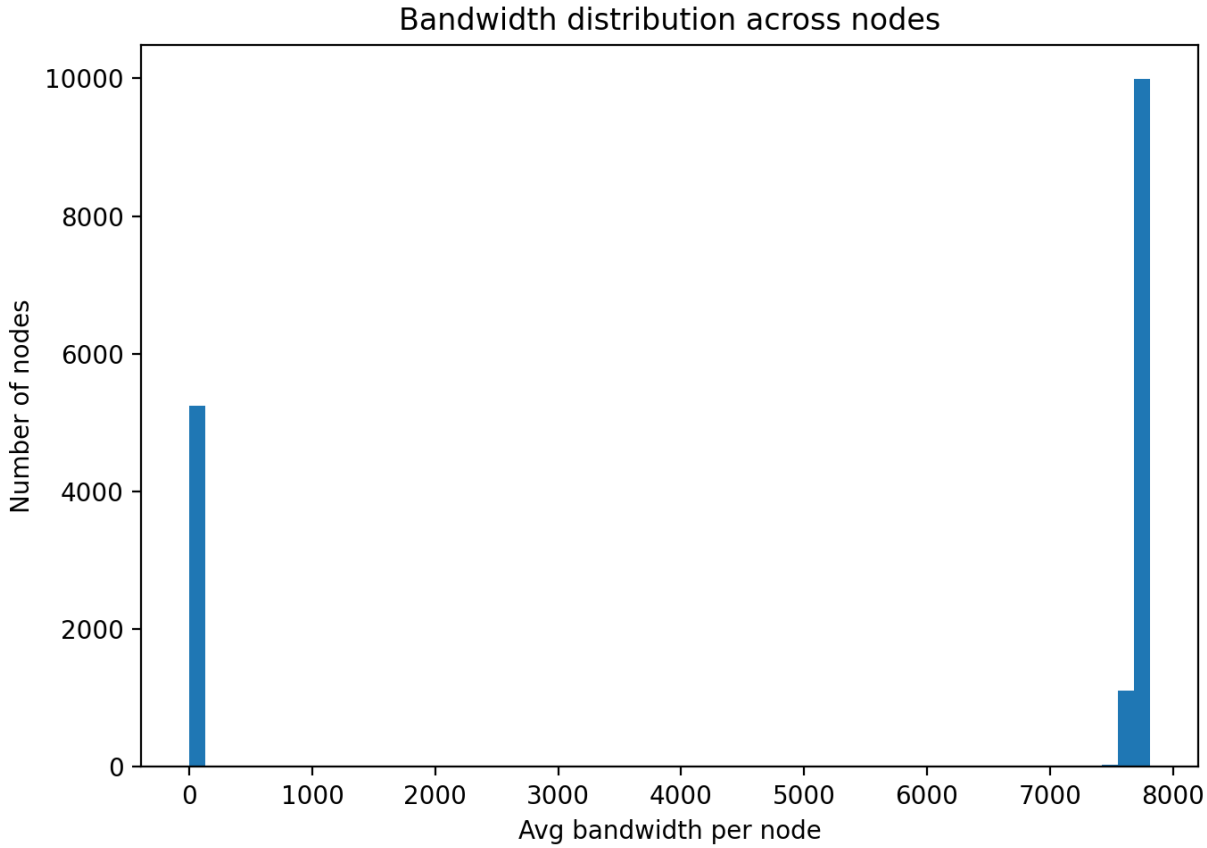


Figure 34: Bandwidth distribution across nodes: histogram of the average bandwidth per node ((MR= 0.4, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 35 shows the corresponding per-slot timing CDFs. The figure is included to show how close the sampling distribution moves toward the 4000 ms deadline under high omission.

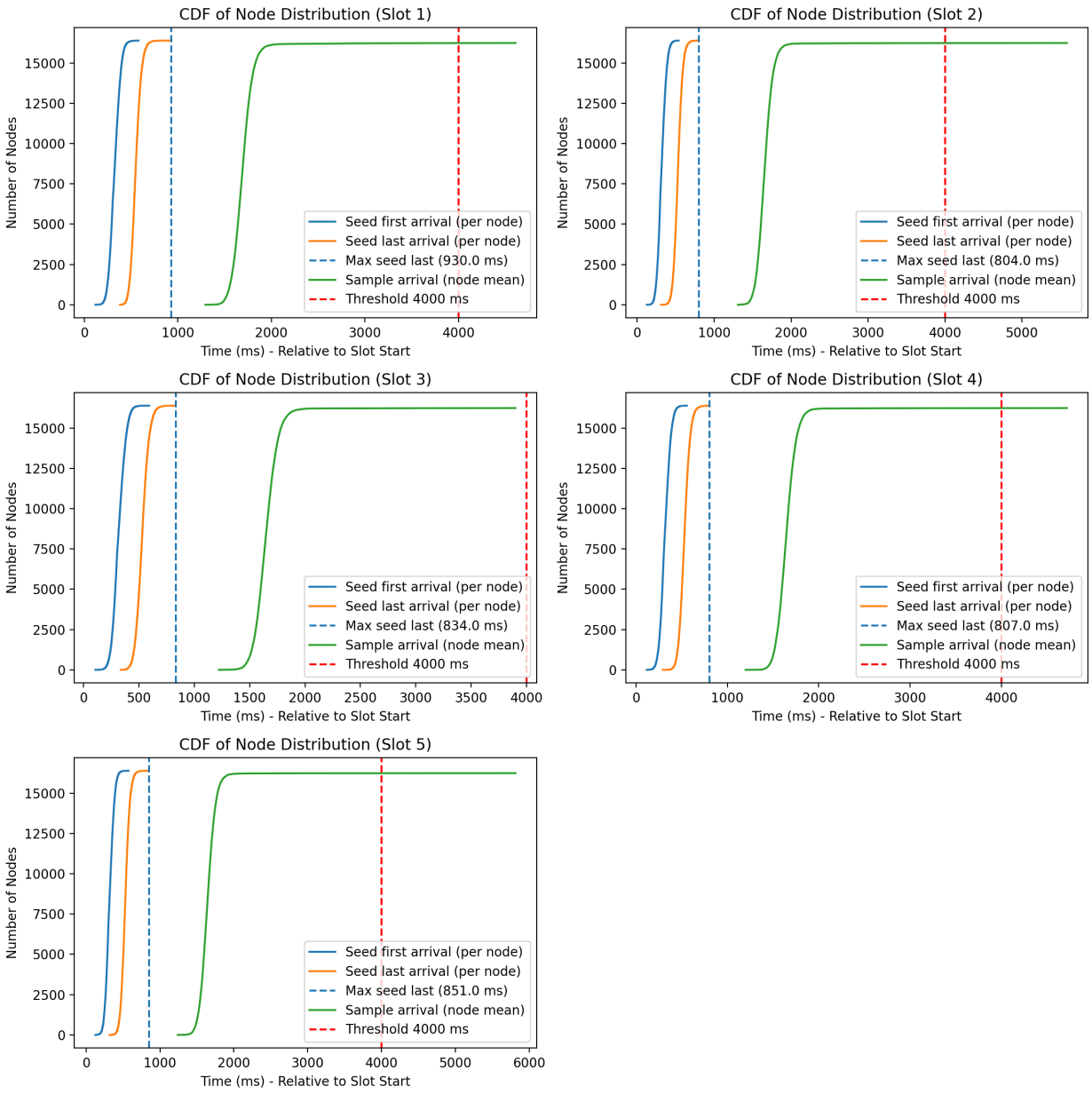


Figure 35: CDF of node arrival times per slot (Slots 1-5), measured relative to slot start. Curves show seed first/last arrival (per node) and sample arrival (node mean); dashed lines mark the maximum seed-last arrival and the 4000 ms threshold ((MR= 0.4, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 36 shows the per-slot duplication distributions as violin plots at $\alpha = 0.4$. This plot supports the observation that lower duplication under omission reflects weaker effective forwarding rather than improved efficiency.

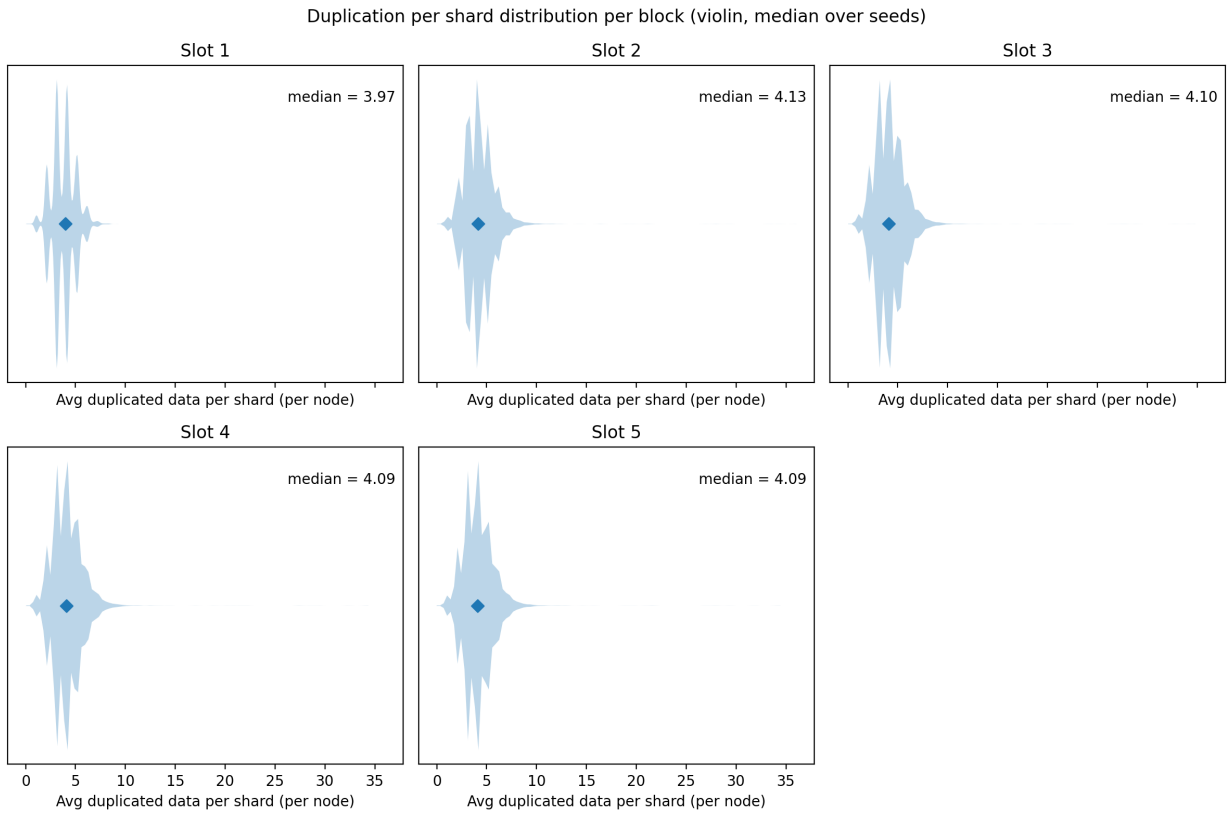


Figure 36: Per-slot duplication per shard per block: violin plots (Slots 1–5) of average duplicated data per shard (per node), aggregated as the median over seeds; diamonds indicate the median ((MR= 0.4, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 37 shows the corresponding per-slot duplication histograms at $\alpha = 0.4$. This histogram view makes the slot-level concentration of duplication values easier to inspect.

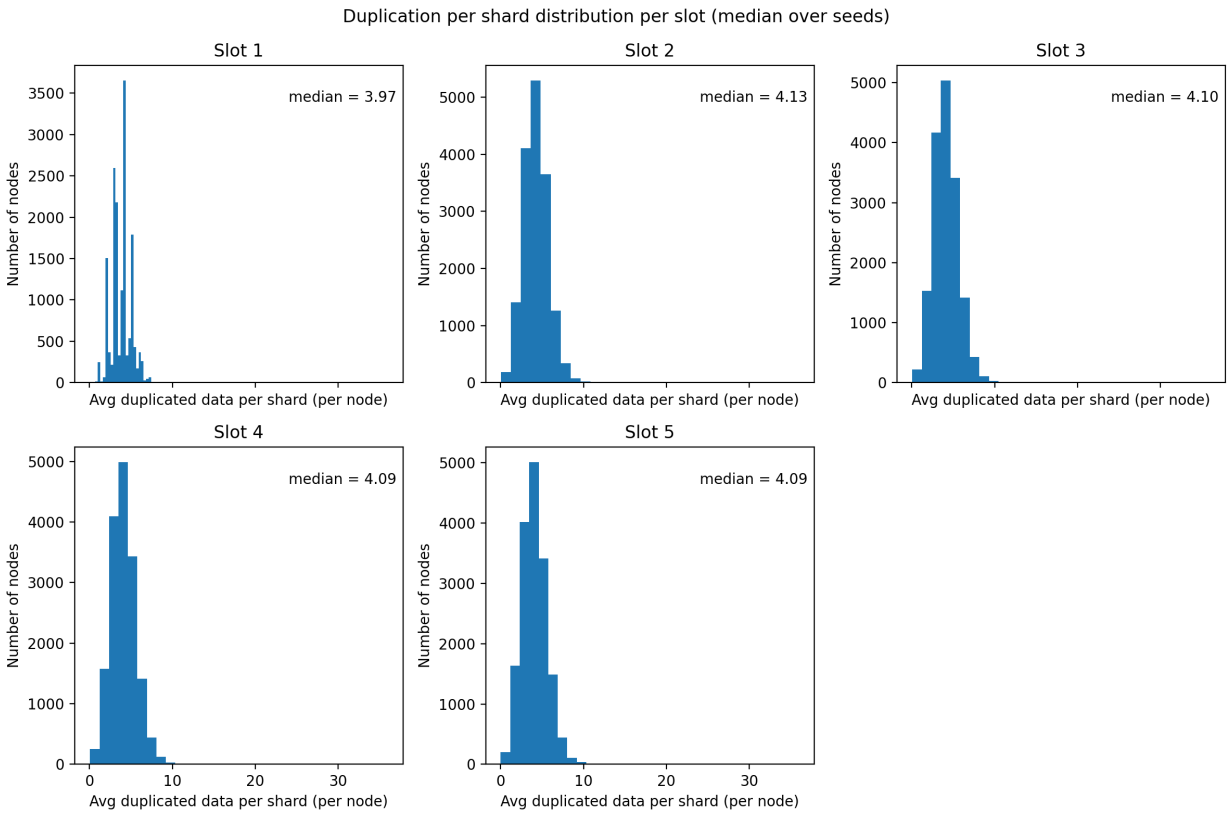


Figure 37: Per-slot duplication per shard distribution (Slots 1–5): histograms of average duplicated data per shard (per node), aggregated as the median over seeds ((MR= 0.4, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 38 shows the all-slot duplication distribution for $\alpha = 0.4$. This figure summarises the aggregate duplication behaviour close to the highest tested omission rates.

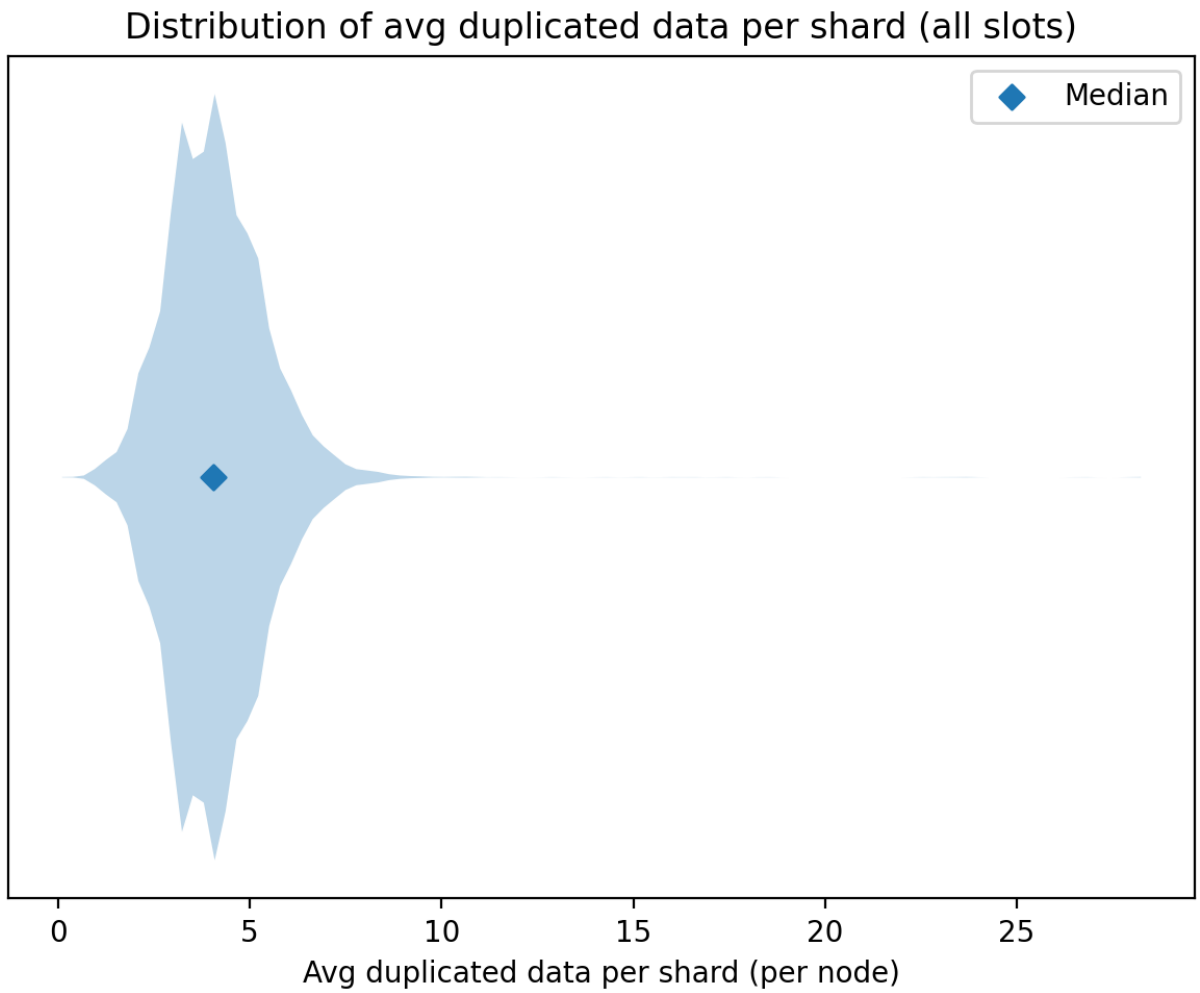


Figure 38: Overall duplication per shard (all slots combined): violin plot of average duplicated data per shard (per node); the diamond marks the median ((MR= 0.4, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

A.6 Supplementary results for $\alpha = 0.5$

Figure 39 shows the per-node bandwidth distribution at $\alpha = 0.5$, the highest omission rate evaluated in the thesis.

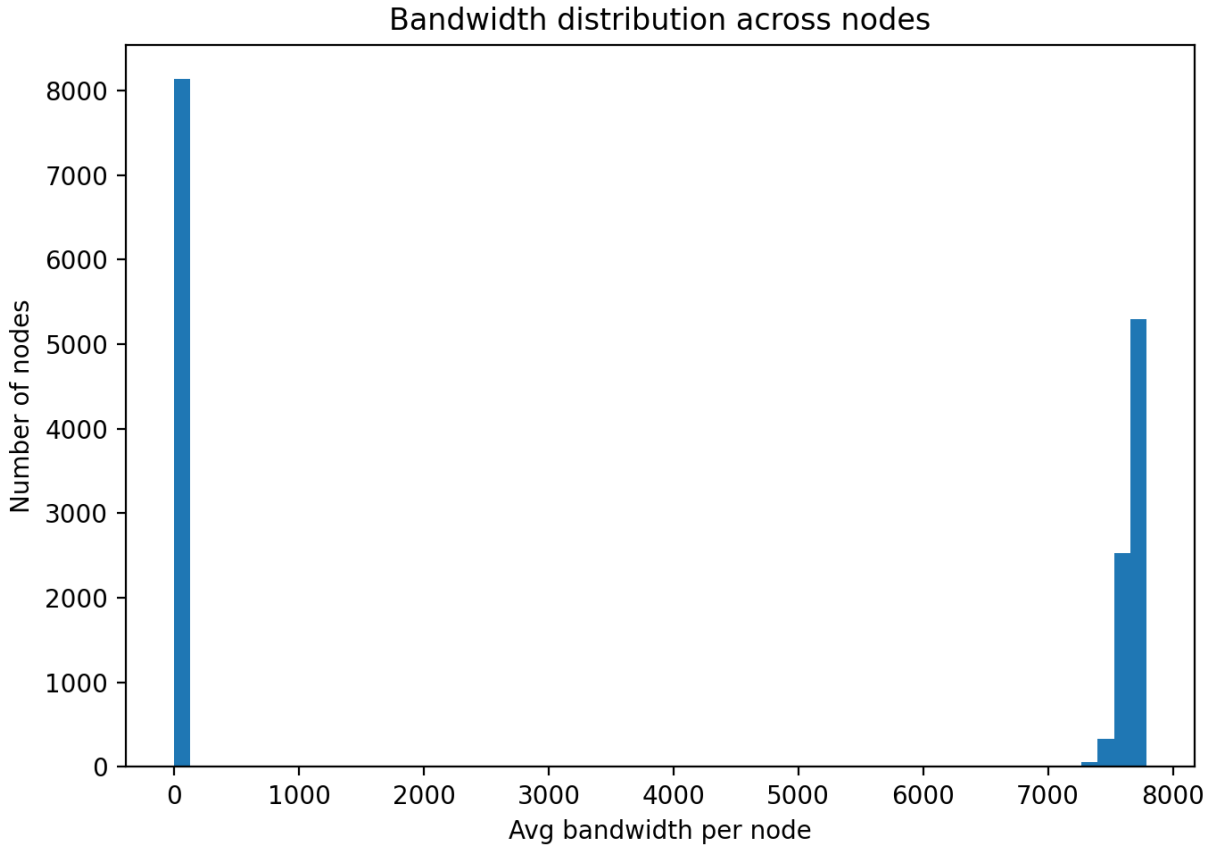


Figure 39: Bandwidth distribution across nodes: histogram of the average bandwidth per node ((MR= 0.5, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 40 shows the per-slot timing CDFs at $\alpha = 0.5$. This figure supports the main-text conclusion that the system enters a degraded regime where sampling deadline misses become common.

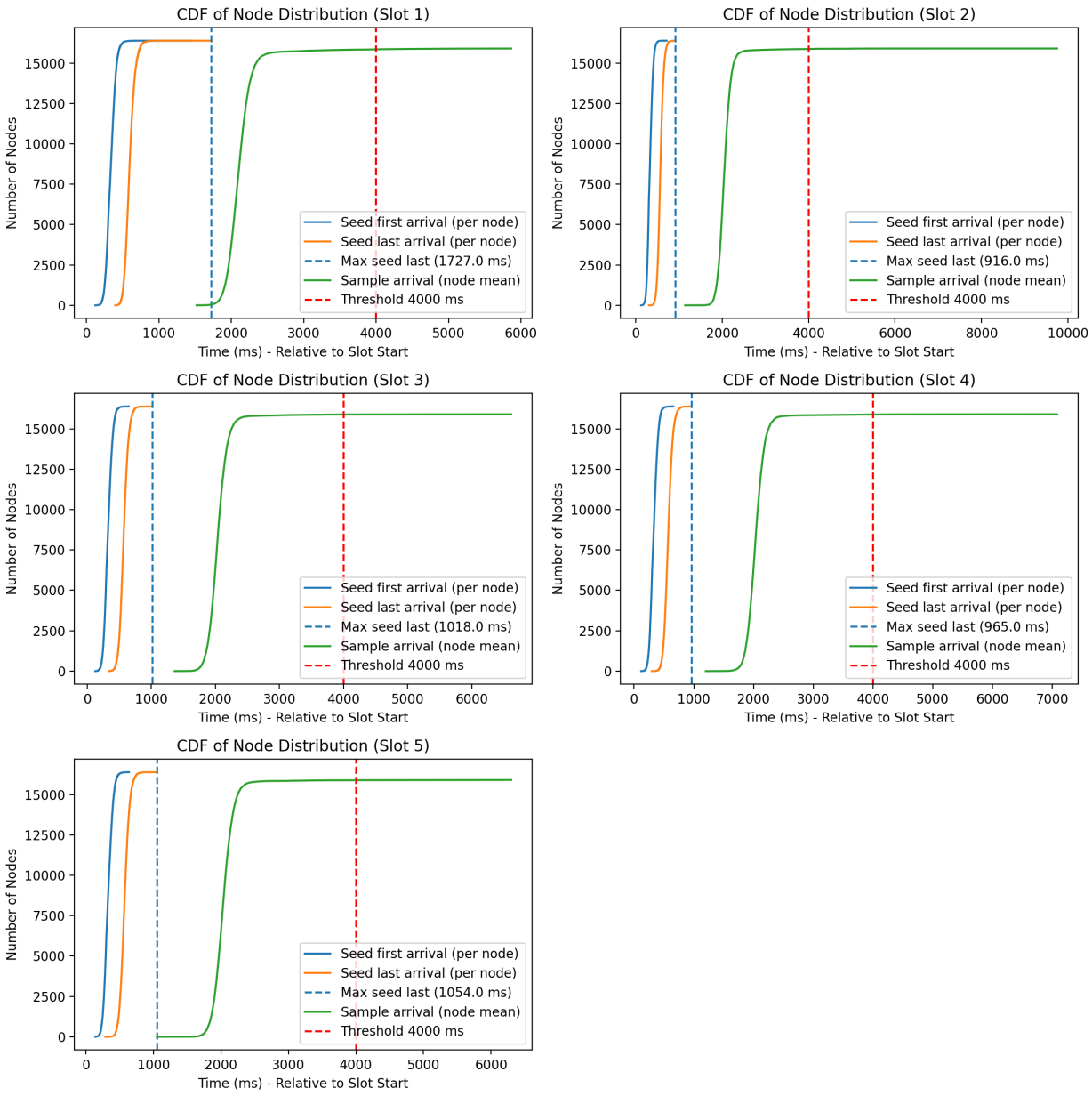


Figure 40: CDF of node arrival times per slot (Slots 1-5), measured relative to slot start. Curves show seed first/last arrival (per node) and sample arrival (node mean); dashed lines mark the maximum seed-last arrival and the 4000 ms threshold ((MR= 0.5, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 41 shows the per-slot duplication distributions as violin plots at $\alpha = 0.5$. These plots illustrate that duplication continues to fall as faulty nodes reduce effective forwarding.

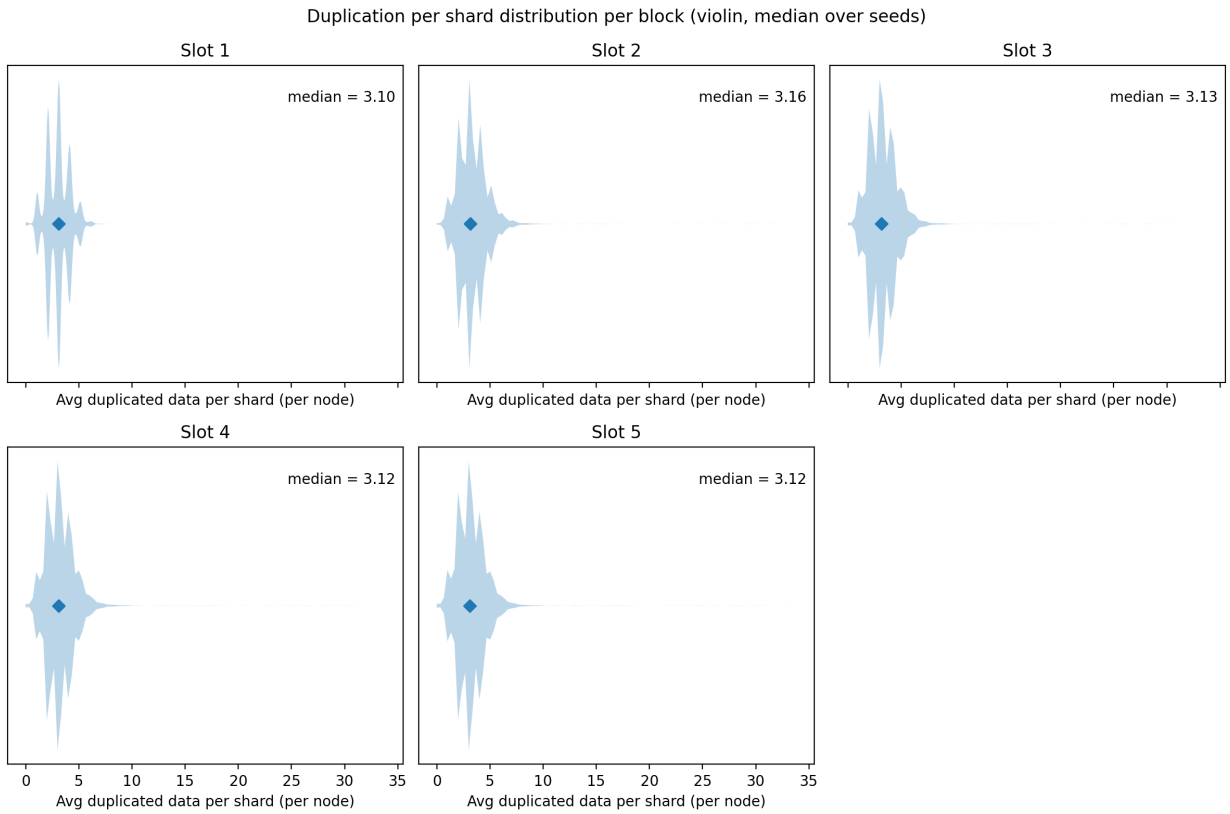


Figure 41: Per-slot duplication per shard per block: violin plots (Slots 1–5) of average duplicated data per shard (per node), aggregated as the median over seeds; diamonds indicate the median ((MR= 0.5, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 42 shows the same per-slot duplication behaviour as histograms. This view helps confirm that the reduction in duplication is visible across individual slots, not only in the aggregate.

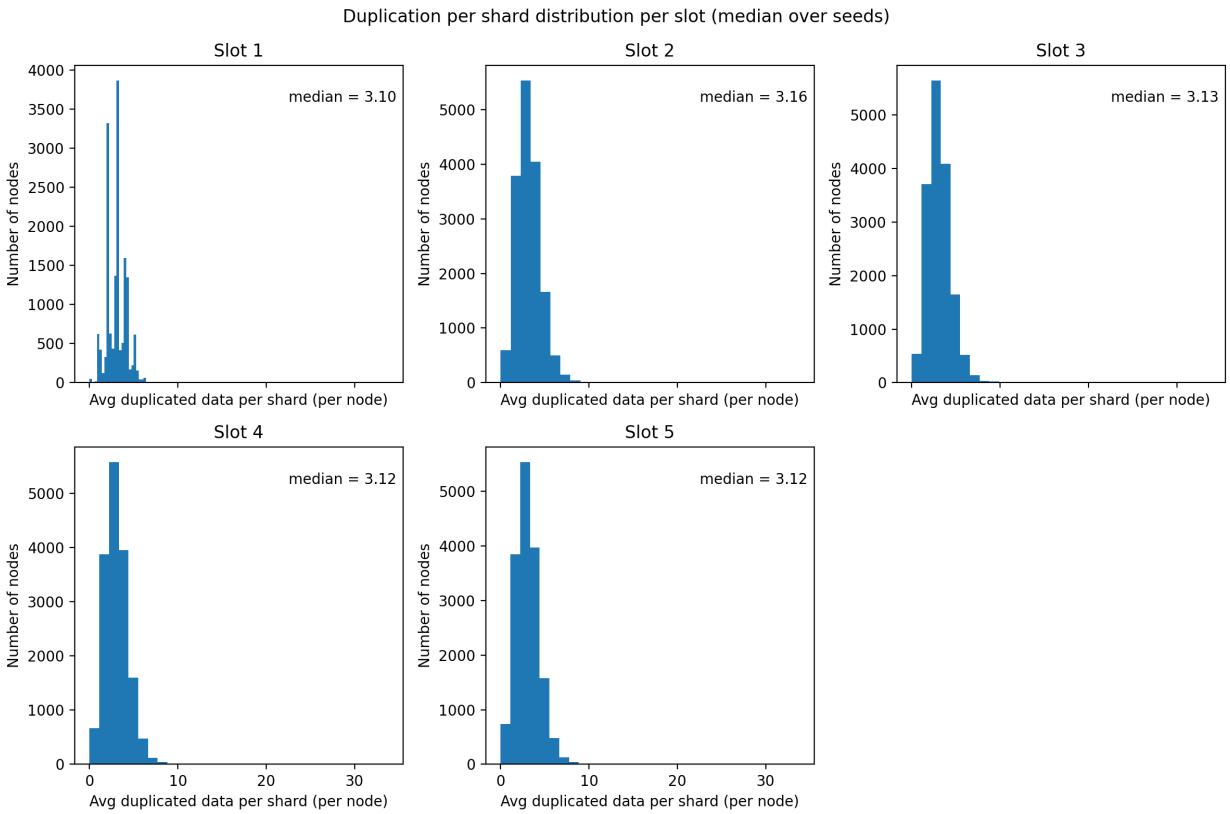


Figure 42: Per-slot duplication per shard distribution (Slots 1–5): histograms of average duplicated data per shard (per node), aggregated as the median over seeds ((MR= 0.5, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).

Figure 43 aggregates duplication across all slots for $\alpha = 0.5$. This provides the final comparison point for the omission sweep.

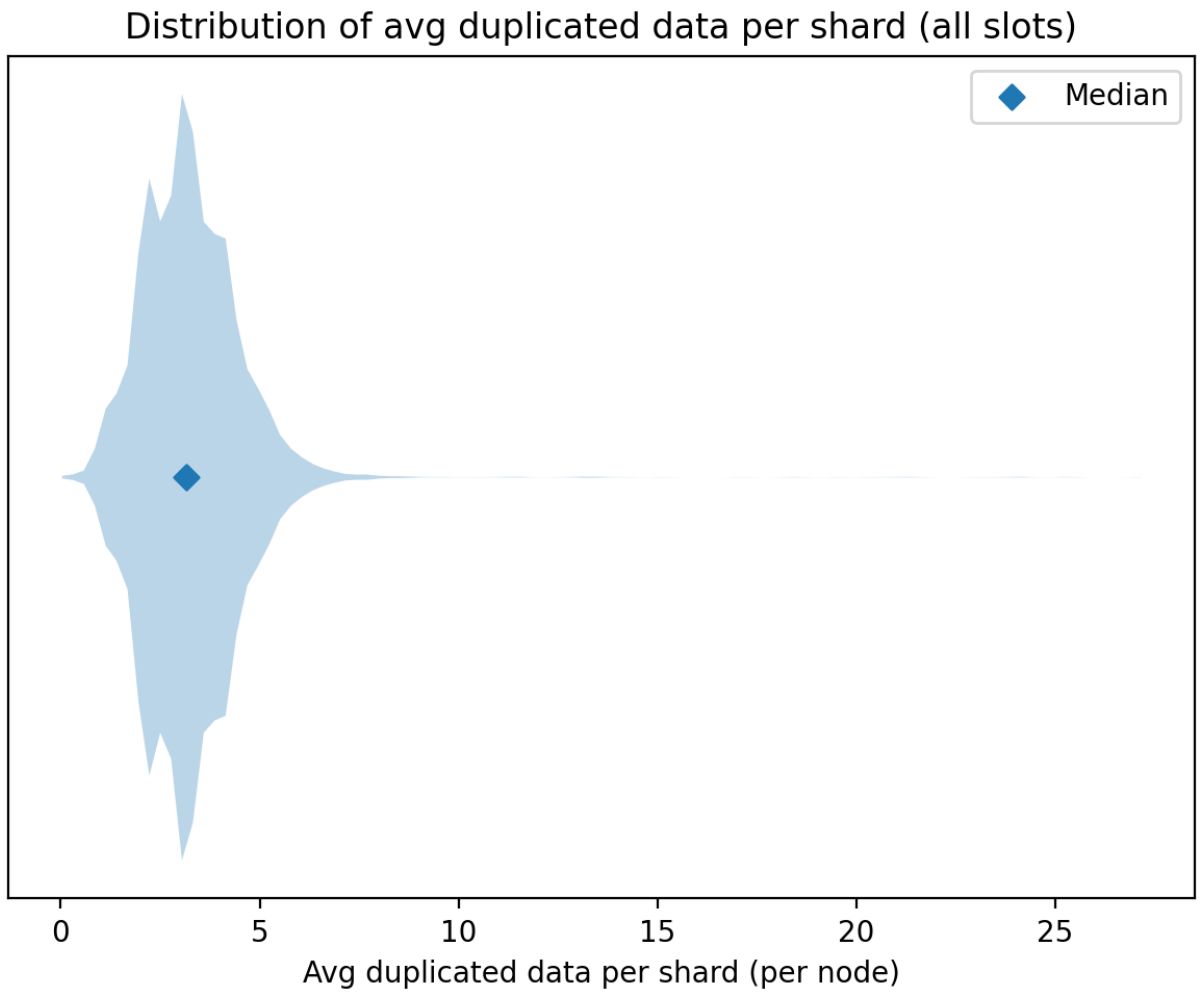


Figure 43: Overall duplication per shard (all slots combined): violin plot of average duplicated data per shard (per node); the diamond marks the median ((MR= 0.5, TOPICS= 256, $K = 4$, SA= 8, bandwidth cap = 60 Mbit/s, validators = 16,384).).