Lean Gossip: Big Gains From Small Talk

Ankit Kumar
Northeastern University
Boston, United States of America
kumar.anki@northeastern.edu

Panagiotis Manolios
Northeastern University
Boston, United States of America
p.manolios@northeastern.edu

Abstract

Gossipsub is the primary peer-to-peer dissemination protocol used by large-scale Web3 systems such as Ethereum, Filecoin, and IPFS. Despite its widespread deployment, the choice of its key parameters—the eager mesh degree D (number of peers that receive messages eagerly) and the gossip degree D_{lazy} (number of peers periodically notified via gossip)—has largely relied on heuristics, with little quantitative guidance. Consequently, production networks lack a principled understanding of the delivery rate, bandwidth cost, and latency tradeoffs induced by these parameters.

We present the first systematic, simulation-based study of Gossipsub parameterization under churn. We perform a comprehensive parameter sweep across four churn regimes (0–30%) in a 1000-node network and validate our baseline against published Protocol Labs measurements. Using Pareto dominance over delivery rate and bandwidth cost, we characterize the tradeoffs between eager redundancy and gossip-based recovery. Our results yield three main findings: (1) Simple flooding (Floodsub) is dominated by Gossipsub in the delivery rate–bandwidth cost plane under non-zero churn (10–30%). Floodsub's only consistent advantage is sub-second latency. (2) Lean gossip configurations ($D_{lazy} \leq 3$) strictly dominate Ethereum's deployed configuration across all performance dimensions: (D=8, D_{lazy}=3) achieves higher delivery, equal latency, and less than a third of Ethereum's bandwidth cost. The minimal configuration (D=8, D_{lazy}=1) reduces cost further to under a tenth of Ethereum's, with slightly better latency and within 0.5 percentage points of its delivery. (3) We show that gossip-based recovery—not eager-push redundancy—is the dominant mechanism for reliable dissemination under churn.

Our simulation framework and analysis methodology are released under the MIT license, enabling principled, multi-objective evaluation of gossip-based dissemination protocols.

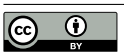
CCS Concepts

• **Networks** → **Network protocols**; **Peer-to-peer protocols**; *Network performance evaluation.*

Keywords

Peer-to-peer networks, Gossipsub, Parameter optimization

ACM Reference Format:
Ankit Kumar and Panagiotis Manolios. 2026. Lean Gossip: Big Gains From Small Talk. In *The 20th ACM International Conference on Distributed and Event-based Systems (DEBS '26), June 23–26, 2026, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3809481.3812610



DEBS '26, Lisbon, Portugal

ACM ISBN 979-8-4007-2693-4/2026/06
https://doi.org/10.1145/3809481.3812610

1 Introduction

Gossipsub [23, 24] is a widely deployed peer-to-peer network protocol used in practice by popular distributed systems such as Ethereum, Filecoin, and the InterPlanetary File System (IPFS) network. It was designed to disseminate messages quickly and efficiently by allowing peers to forward the full content of messages only to a select subset of their neighboring peers (eager-push). To guarantee that a message will reach all targets, Gossipsub employs lazy pull, which allows nodes to communicate with other nodes outside their mesh by gossiping about messages they have seen. Both of these features are configurable via the following parameters: (1) D: Number of mesh neighbors each peer tries to maintain, and (2) D_{lazy}: Number of non-mesh neighbors to gossip. Since peers in a Gossipsub network are free to leave or join, the protocol depends on regular interval heartbeat maintenance to maintain the mesh degree D at each peer. The choice of these parameters plays a critical role in how the network performs with respect to message delivery rate, bandwidth usage and message delivery latency.

Lean Gossip. This paper introduces and evaluates what we call *Lean Gossip*: a configuration regime for gossip-based dissemination protocols in which recovery via gossip is deliberately kept minimal. Concretely, Lean Gossip refers to Gossipsub configurations with a minimal gossip degree ($D_{lazy} \leq 3$), relying on gossip primarily as a recovery mechanism. As we show, this minimal use of gossip is sufficient to compensate for churn-induced message loss while avoiding the substantial bandwidth overhead associated with aggressive gossiping.

We investigate how to maximize dissemination efficiency through *lean gossip* (minimal gossip degree D_{lazy}) and *lean dissemination* (low mesh degree D). More broadly, we propose a methodology for tuning gossip-based dissemination protocols: first characterize the target deployment using empirical measurements (e.g., churn, topology, latency, and peer connectivity), then construct and validate a simulation model grounded in those measurements, and finally perform systematic parameter exploration to identify configurations that minimize redundancy while preserving dissemination performance. Following this methodology, we use network characteristics reported in prior studies, including Kiffer [11] and Protocol Labs' evaluation of Gossipsub [21], to construct a realistic simulation model of Gossipsub. We then validate the simulator against published measurements to ensure fidelity. Using this empirically grounded model, we perform an extensive parameter sweep over mesh degree D and gossip degree D_{lazy} across churn regimes.

Our results show that a single gossip peer (D_{lazy}=1) is sufficient to enable effective recovery. The key observation is that once a peer

learns a message identifier through gossip, it can retrieve the corresponding payload from the gossiping peer independently of mesh connectivity. Consequently, a single gossip peer is often enough to propagate recovery throughout the network, while additional gossip peers contribute little beyond marginal redundancy at a disproportionately higher bandwidth cost.

1.1 Motivation

The Gossipsub parameters D and D_{lazy} were originally selected heuristically to ensure attack resilience [20, 21] and maintain "balance" between bandwidth and speed [20, 23], with limited quantitative justification. The official documentation states that "D_{lazy} is considered optional... By default, we simply use D for both" [14].

Previous work by Cristo *et al* [4] presents a systematic evaluation to characterize and explain Gossipsub's behavior across several axes when integrated with XRP's consensus layer. The study examines nine key dimensions—including message propagation delay, mesh stability, scoring dynamics, resilience under churn, fault tolerance, and adversarial conditions to understand performance and delivery trade-offs.

Orthogonally, Vyzovitis *et al* [20, 21, 24] presented a detailed analysis of the resilience of Gossipsub implementations against Sybil and Eclipse attacks. On the contrary, Kumar *et al* [12, 13] used formal model driven analysis to show that there exist Ethereum Gossipsub configurations prone to sybil attacks that can lead to eclipse or even network partitions.

There has been no work on finding optimal Gossipsub parameters. Production deployments use fixed parameters (*e.g.*, D=8 in Ethereum [7]) regardless of network conditions, leading to concerns about excessive bandwidth consumption. As explicitly acknowledged by the Ethereum community, "I don't know of anyone that can/has modelled our current eth2 network and can definitively say what parameters would be optimal" [7]. While these parameters are known to influence typical performance, our results show that Lean Gossip configurations ($D_{lazy} \leq 3$) strictly dominate Ethereum's deployed parameters across all performance dimensions at 20% churn. Even the most minimal configuration (D_{lazy}=1) achieves comparable delivery to Floodsub at 4–30× lower bandwidth cost under non-zero churn (10–30%). This provides the first principled guidance for parameter selection: sparse mesh connectivity with gossip-based recovery outperforms redundant eager-push connections, and Ethereum's deployed gossip degree is substantially over-provisioned.

1.2 Contributions

This paper makes the following contributions:

First systematic parameter study of Gossipsub under churn. We present the first comprehensive simulation-based evaluation of Gossipsub parameterization across mesh degree D and gossip degree D_{lazy} under four churn regimes (0–30%) in a 1000-node network. Our simulator implements Gossipsub v1.0 semantics and is validated against published Protocol Labs measurements. We release our simulation framework and analysis scripts for reproducibility [1].

A methodology for lean dissemination design. We propose a general methodology for tuning gossip-based dissemination protocols under churn. The approach consists of: (1) characterizing the target deployment using empirical measurements of network properties such as churn, latency, and topology; (2) constructing and validating a simulation model grounded in those measurements; and (3) performing systematic parameter exploration to identify configurations that minimize redundancy while preserving dissemination performance. We instantiate this methodology for Gossipsub and show that aggressively reducing gossip redundancy can preserve delivery while substantially reducing bandwidth overhead.

Floodsub lies off the delivery–bandwidth Pareto frontier under churn. We show that simple flooding (Floodsub) is dominated by Gossipsub in the delivery–bandwidth plane under non-zero churn (10–30%). For every such churn level, there exist Gossipsub configurations that achieve both higher delivery and lower bandwidth cost simultaneously. At 0% churn, Floodsub achieves 100% delivery, which sparse Gossipsub meshes do not match, so the dominance relationship holds only under churn. Floodsub's only consistent advantage across all churn levels is lower latency.

Lean Gossip ($D_{lazy} \leq 3$) strictly dominates Ethereum's deployed configuration across all performance dimensions. We identify *Lean Gossip* configurations ($D_{lazy} \leq 3$) that strictly dominate Ethereum's deployed Gossipsub parameters across all three objectives simultaneously. At 20% churn, (D=8, D_{lazy}=3) achieves 99.9% delivery, 4.9 s p99 latency, and 34× bandwidth cost—beating Ethereum's 99.8% delivery and 106× cost while matching its latency, at 3.2× lower cost. The leaner (D=8, D_{lazy}=1) reduces cost to 9.8× (nearly 11× cheaper than Ethereum) while maintaining lower p99 latency and sacrificing only 0.5% points of delivery.

Recovery dominates redundancy. We demonstrate that gossip-based recovery—not eager-push redundancy—is the dominant mechanism for reliable dissemination under churn. Increasing D beyond approximately 8 yields diminishing delivery returns, while increasing D_{lazy} beyond 1 sharply increases bandwidth cost for negligible delivery improvement (0.3 percentage points across $D_{lazy} \in [1, 21]$ at 20% churn).

Pareto dominance framework for protocol evaluation. We introduce a two-dimensional Pareto dominance analysis over worst-case delivery and normalized bandwidth cost, enabling principled, multi-objective comparison of dissemination strategies.

2 Related Work

Early work on reliable broadcast studied redundancy as a mechanism for survivability in unreliable networks. Baran [2] proposed flooding-based communication for resilient distributed systems, an approach later shown to suffer from redundancy-induced congestion, formalized as the broadcast storm problem [17]. Subsequent work explored controlled rebroadcast and timing-based suppression schemes to balance delivery and efficiency [9, 22]. These studies primarily consider deterministic or MANET-style environments rather than large-scale churned overlays.

In parallel, epidemic dissemination protocols demonstrated that probabilistic gossip with bounded fanout achieves high coverage

with logarithmic time complexity [3, 5, 8]. These protocols trade deterministic guarantees for scalability and robustness, forming the conceptual foundation for modern peer-to-peer publish-subscribe systems.

Gossipsub [23, 24] combines eager meshes with lazy gossip repair. Protocol Labs evaluated its resilience against adversarial behavior, including Sybil and Eclipse attacks [20, 21]. Cristo *et al* [4] studied performance behavior across multiple operational dimensions in XRP's deployment. Kumar *et al* [12, 13] used formal model-driven analysis to identify attack vulnerabilities in Ethereum configurations.

No prior work has performed a systematic multi-parameter sweep to identify Pareto-optimal Gossipsub configurations. Production deployments use fixed parameters (e.g., D=8 in Ethereum [7]) without quantitative guidance on delivery–bandwidth tradeoffs. Our work fills this gap by introducing a worst-case Pareto dominance framework for evaluating parameter configurations under churn.

3 Methodology

This section describes the simulation framework used to evaluate Gossipsub dissemination as a function of mesh degree D and gossip degree D_{lazy} under churn. We isolate protocol-level delivery–bandwidth–latency tradeoffs by modeling a single-topic overlay and excluding scoring and multi-topic interactions.

3.1 Gossipsub Overview

Gossipsub combines eager push via a maintained mesh overlay with lazy pull via metadata gossip. Each peer maintains a neighbor set $p.nbrs$ and, for a subscribed topic t, an eager mesh $p.\mathcal{M}(t)$ of size approximately D to which full payloads are forwarded. Periodically, peers gossip message identifiers to up to D_{lazy} non-mesh neighbors, enabling recovery via IHAVE/IWANT exchanges. Heartbeat maintenance enforces mesh degree bounds and emits gossip.

For a subscribed topic $s \in p.\mathcal{S}$, peer p maintains a *mesh* $p.\mathcal{M}(s) \subseteq \{q \in p.nbrs : s \in q.\mathcal{S}\}$, a subset of s-subscribing neighbors to which p eagerly forwards full message payloads upon first receipt. The mesh degree is bounded: $D_{lo} \le |p.\mathcal{M}(s)| \le D_{hi}$, where D is the target degree and by default $D_{lo} = D - 2$ and $D_{hi} = D + 2$. Likewise, for an unsubscribed topic $u \in p.\mathcal{U}$, peer p maintains a *fanout* $p.\mathcal{F}(u) \subseteq \{q \in p.nbrs : u \in q.\mathcal{S}\}$, enabling message origination on topics without subscription. Meshes, fanouts, and subscriptions are all mutable: a peer may unsubscribe from a topic, delete its corresponding mesh, and build a fanout; or vice versa.

Metadata dissemination. Each peer maintains a bounded message cache $p.mcache$ mapping recent message identifiers to payloads. Metadata about recently received messages are periodically broadcast to up to D_{lazy} randomly selected neighbors outside the mesh, called *gossip peers*. These IHAVE announcements contain message identifiers from $p.mcache$, allowing metadata to disseminate quickly with low overhead. Upon receiving IHAVE(M) from peer q, a peer p identifies missing messages $M' = M \setminus p.seen$ and responds with IWANT(M'), after which q transmits the requested payloads. This lazy pull mechanism enables recovery of messages missed due to mesh disconnection.

Heartbeat maintenance. Gossipsub executes a periodic heartbeat procedure at configurable intervals (default 1 s) to maintain mesh structure and disseminate metadata. For each topic $t \in p.\mathcal{S}$, the heartbeat performs the following operations in sequence:

Mesh maintenance. Enforce degree bounds by grafts and prunes.

$$|p.\mathcal{M}(t)| < D_{lo} \Rightarrow \text{select } k = D - |p.\mathcal{M}(t)| \text{ peers from } p.nbrs \setminus p.\mathcal{M}(t), \text{ send GRAFT to each, add to } p.\mathcal{M}(t)$$

$$|p.\mathcal{M}(t)| > D_{hi} \Rightarrow \text{select } k = |p.\mathcal{M}(t)| - D \text{ peers from } p.\mathcal{M}(t), \text{ send PRUNE to each, remove from } p.\mathcal{M}(t)$$

Fanout maintenance. For each topic $u \in p.\mathcal{U}$ with non-empty fanout, we remove peers that have unsubscribed and replenish if $|p.\mathcal{F}(u)| < D$. Expire fanouts inactive beyond a configurable TTL.

Gossip emission. Select up to D_{lazy} peers from $p.nbrs \setminus p.\mathcal{M}(t)$ and emit IHAVE announcements containing message identifiers from $p.mcache$.

Cache shift. Advance the message cache window, expiring entries older than the gossip history length (default 3 heartbeat intervals). This bounds memory usage while retaining sufficient history for IWANT responses.

3.2 Model Simplifications

To isolate dissemination dynamics, we apply the following simplifications:

Single topic. All peers subscribe to one topic ($|\mathcal{T}| = 1$), eliminating fanout interactions and cross-topic effects.

No peer scoring. We omit v1.1 scoring, as it addresses adversarial behavior rather than benign churn.

Static neighbor sets. Peer tables are initialized once and change only due to churn.

No Floodsub compatibility. We model pure Gossipsub semantics.

Under these assumptions, peer state reduces to mesh neighbors, gossip peers, pending queue, seen set, bounded message cache, availability state, and delivery record. The parameters D and D_{lazy} fully determine redundancy.

Simplified peer state. Under these simplifications, peer state reduces to a septuple. For the single topic $t \in \mathcal{T}$, we write: $p.\mathcal{M} = p.\mathcal{M}(t)$: the eager mesh neighbors, $p.\mathcal{L} \subseteq p.nbrs \setminus p.\mathcal{M}$: the gossip peers selected for IHAVE emission, $p.pending$: the queue of inbound messages yet to be processed, $p.seen$: the set of received message identifiers (cleared on rejoin), $p.mcache$: the bounded message cache, $p.\omega \in \{\text{online}, \text{offline}\}$: the availability state, $p.delivered$: a persistent record of all messages received by peer p, which survives peer rejoin events.

The parameters D (target mesh degree) and D_{lazy} (maximum gossip degree) fully determine the protocol's redundancy–efficiency tradeoff in this simplified model. Note that $p.seen$ is used for duplicate detection during protocol execution and is cleared when a peer rejoins, while $p.delivered$ is used solely for delivery measurement and persists across churn events to ensure accurate delivery rate statistics.

These simplifications preserve the core dissemination dynamics: eager mesh forwarding provides primary delivery, while lazy pull serves as a recovery mechanism for messages missed due to mesh disconnection. Our single-topic model essentially captures

the behavior of either a mesh overlay (when peers subscribe) or a fanout overlay (when peers originate messages without subscription) in a full multi-topic Gossipsub network. Since both structures share identical eager-push semantics and gossip recovery mechanisms, our model isolates the fundamental dissemination dynamics common to all Gossipsub overlays.

We perform single-topic simulation because Gossipsub defines mesh maintenance (via GRAFT/PRUNE actions), gossip exchange (via IHAVE/IWANT actions), and forwarding per topic. Each peer maintains its own topic mesh of D peers; our simulation models exactly these dynamics. The only cross-topic interaction is peer scoring, which affects adversarial scenarios, not benign churn. Furthermore, shared connections affect mesh formation, not dissemination: Ethereum nodes maintain 40–80 connections across ~70 topics. This affects mesh formation, but once formed, each topic's dissemination operates as we model. Connection scarcity may increase reliance on gossip recovery, but our results show D_{lazy}=3 provides sufficient recovery (>99% delivery under 20% churn)—and the question is whether D_{lazy}=8 helps more than D_{lazy}=3; our data shows it does not, while costing 3.2× more bandwidth. Finally, multi-topic amplifies our savings: D_{lazy}=8 across 70 topics incurs 70× the gossip traffic of our single-topic model. Reducing to D_{lazy}=3 cuts this by 62%. If bandwidth or CPU is constrained (realistic for home validators), high-D_{lazy} configurations hit limits first.

We implement Gossipsub v1.0 semantics, omitting the peer scoring mechanism introduced in v1.1 [15]. Peer scoring is primarily for building attack resilience by penalizing misbehaving peers and is orthogonal to dissemination dynamics. This simplification isolates the core delivery–efficiency tradeoffs arising from mesh structure and gossip parameters.

4 Protocol Implementation

Having formally described the protocol in the previous section, we now explain key design decisions, evaluation metrics, and protocol parameter values used in our simulation.

4.1 Aggregate Message Workload

Table 1: Ethereum message sizes

Category	Size (Bytes)
Attestation	512
Aggregated attestation	1536
Beacon block / Blob	131072
Message identifier	32
IHAVE / IWANT	32 / 16
GRAFT / PRUNE	100

Ethereum's Gossipsub layer disseminates a heterogeneous mix of beacon blocks, attestations, aggregated attestations, and protocol control messages (Table 1). Beacon blocks (blobs) are large (approximately 128 KiB) but rare, with only one block produced per 12-second slot [6]. In contrast, attestation-related messages dominate by count. Empirical measurements of the Ethereum P2P network show that attestation-derived messages dominate gossip traffic by count, while beacon blocks are comparatively rare [10]. Individual attestations are aggregated prior to gossip dissemination, making aggregated attestations the primary message type processed by the Gossipsub layer. Accordingly, we model message dissemination using a single representative aggregated message size of 1.5 KiB which dominates Ethereum gossip traffic by volume (Table 1). Note that our bandwidth recommendations are conservative with respect to message size: larger payloads (e.g., 128 KiB beacon blocks) amplify the absolute bandwidth gap between high-D_{lazy} and low-D_{lazy} configurations proportionally, so our cost ratios and dominance relationships remain unchanged.

According to Ethereum's attestation workload, each peer publishes messages according to a Poisson process with rate $\lambda_{\text{pub}} = v/384$, where v denotes the number of validators colocated with the peer. We primarily evaluate the case $v = 1$, yielding roughly one message every 384 seconds per peer. We use a Poisson process to smooth slot-synchronous effects while preserving Ethereum's long-run attestation rate.

4.2 Network Model

Topology. We simulate a network of 1000 peers. We verified that qualitative trends remain unchanged for larger network sizes; we use 1000 peers to enable exhaustive parameter sweeps. Each peer maintains a neighboring peer table of 40–80 connections drawn from this population. The peer table size is consistent with measurements of Ethereum consensus nodes showing approximately 60 average connections per node [11]. Eager and lazy neighbors are sampled uniformly from this table.

Network Delay Models. We evaluate our simulator under two network delay models:

Constant delay model. All per-hop latencies are fixed at 25 ms, matching Protocol Labs' 50 ms RTT assumption [21]. We use this model for validation (Section 4.8).

Geographic delay model. Nodes are partitioned into three regions reflecting Ethereum's geographic distribution [19]: 30% Europe, 40% North America, and 30% Asia. Per-hop latencies are drawn from region-pair-specific normal distributions (Table 2). **We use this model for all parameter sweep experiments (Section 5)**, as it better reflects production Ethereum network conditions.

	Europe	N. America	Asia
Europe	$\mathcal{N}(30, 10^2)$	$\mathcal{N}(90, 20^2)$	$\mathcal{N}(180, 30^2)$
N. America	—	$\mathcal{N}(40, 10^2)$	$\mathcal{N}(150, 25^2)$
Asia	—	—	$\mathcal{N}(35, 10^2)$

Table 2: Per-hop latency distributions (ms) by region pair

The effective per-hop mean under this model is:

$$\mu_h^{\text{eff}} = \sum_{r,s} p_{rs} \cdot \mu_{rs} \tag{1}$$

where p_{rs} is the proportion of edges between regions r and s. With random peer selection, the effective per-hop mean latency is $\mu_h^{\text{eff}} \approx 105$ ms.

Churn. Based on real-world measurements by Kiffer *et al* [11], we model churn using a bimodal peer classification. A fraction

c of peers are designated as "churny" and cycle between online and offline states with transition probabilities $\lambda_{\text{leave}} = 0.01$ and $\lambda_{\text{rejoin}} = 0.05$ per tick, resulting in rapid cycling with mean connection duration of 10 seconds, consistent with the short-lived connections observed in Ethereum's P2P network [11]. The remaining $1-c$ "stable" peers remain online throughout the experiment ($\lambda_{\text{leave}} = 0$). Note that when an offline peer comes back online, it loses all its state except its churn type and peer table (ambient peer discovery). At steady state, churny peers maintain approximately 83% availability, resulting in an overall network availability of $(1-c) + 0.83c$. We evaluate configurations with $c \in \{0.0, 0.1, 0.2, 0.3\}$.

4.3 Simulation Timing

The simulator advances in discrete ticks ($\Delta t = 0.1$ s), executing heartbeats every 10 ticks (1 s). At each tick, the simulator: refills token buckets, produces messages via a Poisson process ($\lambda_{\text{pub}} = 1/384$ per peer), processes scheduled deliveries, applies churn transitions, enforces per-peer bandwidth limits, and runs heartbeat routines. We discard the first 2000 ticks as warmup. Algorithm 1 formalizes this loop.

4.4 Bandwidth Model

We model upload bandwidth via a token-bucket abstraction of TCP (25 Mbps capacity; bucket $B/8$ bytes; refill $B \cdot \Delta t/8$ per tick). Bandwidth exhaustion did not cause delivery loss in any experiment. This setting is deliberately generous: 25 Mbps sustains Floodsub's 60-peer eager push without drops. Under tighter constraints typical of home validators, Floodsub would saturate first, so our results are *conservative* regarding Floodsub's disadvantage.

4.5 Evaluation Metrics

We evaluate along three axes. Let V denote the set of all peers, and $V_t \subseteq V$ the online subset at time t.

Definition 4.1 (Delivery Set) *The* delivery set $R_{m,t}$ *is the set of peers that received message m by time t during the simulation:*

$$R_{m,t} = \{p \in V : m \in p.\text{delivered by time } t\} \tag{2}$$

where delivered is a persistent record of all messages received by peer p, which survives peer rejoin events.

Definition 4.2 (Delivery rate) *We define the time-dependent delivery rate $\delta_{m,t}$ for $t \geq t_m$ as:*

$$\delta_{m,t} = \frac{|R_{m,t}|}{|V|} \tag{3}$$

This captures the evolution of message dissemination through the network. At publication time, $\delta_{m,t_m} = 0$ since no peer has yet received the message—not even the publisher v_m, which inserts m into its *pending* queue before m is processed, forwarded to mesh peers, and added to $v_m.seen$. As the protocol propagates m through eager-push and gossip-based recovery, $\delta_{m,t}$ increases monotonically, *i.e.*, $\langle \forall t_i, t_j :: t_i \leq t_j \Rightarrow \delta_{m,t_i} \leq \delta_{m,t_j} \rangle$. Let f_m be the first time after t_m when m is no longer in the *pending* queue of any peer and all gossip-based recovery attempts have completed. After f_m, m will not be forwarded again. In a stable network without churn, we expect $\delta_{m,f_m} = 1$. However, under churn, some peers in V may go offline before receiving m, causing δ_{m,f_m} to stabilize below 1. We define the *final delivery rate* $\delta_m = \delta_{m,f_m}$.

4.5.1 Aggregate Metrics.

Definition 4.3 (Mean Delivery Rate) *For a simulation producing message set M, the mean delivery rate is:*

$$\bar{\delta} = \frac{1}{|M|} \sum_{m \in M} \delta_m \tag{4}$$

Our simulation computes δ_m at the end of the measurement period T rather than waiting for stabilization at f_m. Messages published near the end of the measurement period may still have m in some peer's *pending* queue when statistics are computed, yielding $\delta_{m,T} \leq \delta_m$. This slightly underestimates delivery rates, particularly for configurations with high tail latency. However, since this effect applies uniformly across all configurations, relative comparisons remain valid.

Definition 4.4 (Total Bandwidth) *Total bytes transmitted during the measurement period:*

$$B_{total} = \sum_{p \in V} \left(B_p^{\texttt{Payload}} + B_p^{\texttt{Ctrl}} \right) \tag{5}$$

where $B_p^{\texttt{Payload}}$ counts message payload bytes and $B_p^{\texttt{Ctrl}}$ counts control message (`IHAVE`, `IWANT`, `GRAFT`, `PRUNE`) bytes sent from peer p.

This metric captures protocol overhead from multiple sources: redundant payload transmissions (e.g., eager push to multiple mesh neighbors), control messages for lazy push or mesh maintenance, and wasted bandwidth from transmissions to peers that subsequently depart. Lower values indicate better resource utilization, with the theoretical minimum corresponding to perfect amortization of a single payload transmission across all recipients.

Definition 4.5 (Bytes Per Delivery) *Quantifies bandwidth efficiency by measuring the average bytes transmitted per successful delivery:*

$$\beta = \frac{B_{total}}{\sum_{m \in M} |R_m|} \tag{6}$$

Definition 4.6 (Normalized Cost) *We normalize by payload size $P = 1536$ bytes:*

$$\beta / P \tag{7}$$

This normalization yields a dimensionless measure capturing the average multiple of a payload's worth of traffic required per successful delivery, enabling comparisons across parameter settings independent of payload size. A normalized cost of 10 indicates that, on average, 10 payload-equivalents of total traffic are transmitted per successful delivery.

Definition 4.7 (Per-Delivery Latency) *For each successful delivery (m, p) where $p \in R_m$, the latency is:*

$$\ell_{m,p} = (t_{m,p} - t_m) \cdot \Delta t \tag{8}$$

where $t_{m,p}$ is the tick at which peer p first received message m in its pending queue.

Definition 4.8 (P99 Latency) *The 99th percentile latency is computed over all successful deliveries:*

$$p99 = Percentile_{99}\left(\{\ell_{m,p} : m \in M, p \in R_m\}\right) \tag{9}$$

High $p99$ may indicate suboptimal mesh connectivity, high churn, or protocol mechanisms (such as lazy pull) that delay message propagation under certain conditions. Latency is used only as a relative ordering metric for Pareto dominance; we do not claim that absolute $p99$ values predict production behavior.

Together, these metrics characterize the delivery-efficiency-latency tradeoff space that distinguishes gossip protocol designs. An effective protocol must balance high delivery rates against bandwidth costs while maintaining acceptable latency bounds. All results are averaged across three independent random seeds to ensure statistical confidence.

4.6 Simulation Algorithm

Algorithm 1 describes the discrete-event simulation loop. Each simulation tick executes the protocol components in a fixed order, yielding deterministic execution for a given random seed.

Algorithm 1 Gossipsub Simulation Main Loop

Require: Network parameters (n, D, D_{lazy}, c), simulation length T, warmup T_w
Ensure: Metrics $(\bar{\delta}, \beta_{norm}, p99)$

```
Initialize n peers with peer tables of size [40, 80]
Initialize eager mesh M_p and gossip peers L_p for each peer p
t ← 0
while t < T_w + T do
    RefillTokenBuckets                          ▷ Bandwidth accounting
    ProduceMessages(t)                          ▷ Poisson arrivals
    ProcessDeliveries(t)                        ▷ Scheduled network arrivals
    ProcessChurn(t)                             ▷ Bimodal leave/rejoin
    EnforceBandwidth(t)                         ▷ Drop excess messages
    if t mod 10 = 0 then                        ▷ Heartbeat every 1 second
        EmitGossip(t)                           ▷ IHAVE announcements
        MaintainMesh(t)                         ▷ GRAFT/PRUNE
    end if
    t ← t + 1
    if t = T_w then
        Reset all counters                      ▷ End of warmup
    end if
end while
return ComputeMetrics
```

The simulation loop is deliberately structured as a modular discrete-event driver. Each major protocol or environment mechanism is factored into a separate subroutine: message generation, network delivery, churn, bandwidth enforcement, gossip emission, and mesh maintenance. This separation keeps the Gossipsub-specific mechanisms isolated from workload and network assumptions, making it straightforward to vary churn models, message arrival processes, or bandwidth policies without changing the rest of the simulator. The full algorithm, along with description of each of the functions it depends on are available as auxiliary material [1].

Table 3: Simulation parameter values

Parameter	Symbol	Value
Network		
Number of peers	n	1000
Peer table size		$[40, 80]$
Validators per peer	v	1
Timing		
Tick duration	Δt	0.1 s
Heartbeat interval		1.0 s
Warmup period	T_w	2000 ticks
Measurement period	T	10000 ticks
Protocol		
Mesh degree	D	$\{2, 4, \ldots, 14\}$
Gossip degree	D_{lazy}	$\{1, 3, \ldots, 33\}$
Gossip history		3 heartbeats
Messages		
Payload size	P	1536 bytes
Message ID size		32 bytes
IHAVE overhead		32 bytes
IWANT overhead		16 bytes
GRAFT/PRUNE size		100 bytes
Bandwidth		
Global upload cap	B	25 Mbps
Token bucket size		$B/8$ bytes
Refill rate		$B \cdot \Delta t/8$
Churn		
Churny fraction	c	$\{0, 0.1, \ldots, 0.3\}$
Churny leave rate	λ^c_{leave}	0.01 / tick
Churny rejoin rate	λ^c_{rejoin}	0.05 / tick
Stable leave rate	λ^s_{leave}	0
Replication		
Seeds per config		3

4.7 Pareto Dominance Analysis

Definition 4.9 (Pareto Dominance) *Configuration A dominates configuration B on delivery rate and bandwidth cost if $\delta_A \geq \delta_B$ and $(\beta/P)_A \leq (\beta/P)_B$, with at least one strict inequality.*

We exclude latency from the dominance computation because Floodsub achieves the lowest latency (0.4 s p99). Including latency would prevent any configuration from strictly dominating Floodsub, obscuring the clear delivery-cost advantage of Gossipsub. Latency is instead encoded as marker shape in Figure 5.3 for visual inspection. The algorithm description for pareto analysis is available as auxiliary material [1]. Table 3 lists the simulation parameters used in our experiments.

4.8 Validation Against Published Measurements

To validate our simulator, we compare against Protocol Labs' Gossipsub v1.1 evaluation [21], which tested a 1000-node network with $D = 8$, $D_{lazy} = 8$, and no churn. We use a fixed 25 ms per-hop delay

Table 4: Validation against Protocol Labs Gossipsub evaluation (1000 nodes, $D = 8$, $D_{lazy} = 8$, 0% churn, 10 ms ticks)

Metric	Ours	Protocol Labs	Notes
Delivery rate	99.7%	100%	matches
p50 latency	0.240 s	0.100 s	2.4× (tick overhead)
p99 latency	0.390 s	0.165 s	2.4× (tick overhead)
Maximum latency	0.540 s	0.350 s	1.5×
ETH2 req. (<3 s)	✓	✓	Both satisfy

to match Protocol Labs' 50 ms RTT assumption. Our validation aims to establish the correctness of Gossipsub's dissemination dynamics and the relative ordering of configurations, rather than to predict exact absolute performance in production deployments. We therefore validate against published testbed results under controlled conditions and emphasize qualitative agreement and dominance relationships, which are robust to simulation artifacts.

4.8.1 Theoretical Latency Analysis. We first derive expected latency bounds from graph-theoretic properties of the mesh overlay.

Average shortest-path length. The eager mesh forms a random regular graph with degree D. For such graphs, the average shortest-path length (ASPL) scales logarithmically:

$$\text{ASPL} \approx \frac{\ln n}{\ln D} = \frac{\ln 1000}{\ln 8} \approx 3.32 \text{ hops} \tag{10}$$

Empirical measurements from our simulation refine this to ASPL ≈ 3.0 hops, consistent with ProbeLab's DEthna measurements on the Goerli testnet (ASPL ≈ 2.7 for ∼1–2k nodes) [25].

Expected end-to-end latency. Given one way latency per-hop with mean μ_h and standard deviation σ_h, the expected end-to-end latency is:

$$\mu_{\text{e2e}} \approx \text{ASPL} \times \mu_h \tag{11}$$

The variance combines within-path variability and between-path hop-count variability:

$$\sigma^2_{\text{e2e}} \approx \text{ASPL} \times \sigma_h^2 + \text{Var(hops)} \times \mu_h^2 \tag{12}$$

Assuming Gaussian approximation, the 99th percentile is:

$$\text{p99} \approx \mu_{\text{e2e}} + 2.326 \times \sigma_{\text{e2e}} \tag{13}$$

For Protocol Labs' parameters ($\mu_h \approx 25$ ms from 50 ms RTT, negligible variance), predicted p99 is $3.0 \times 25\text{ms} \approx 75$ ms. Protocol Labs reports 165 ms, with the ≈2× difference attributable to protocol processing overhead (deserialization, validation, forwarding decisions).

4.8.2 Validation Results. Table 4 compares our simulator against Protocol Labs measurements (25 ms per-hop delay, 10 ms tick resolution).

Delivery rate matches (99.7% vs. 100%, with the small difference due to end-of-period measurement truncation as discussed in §4.5). Latency is 2.4× higher due to tick quantization ($\lceil \text{delay}/\Delta t \rceil + 1$ ticks per hop); Protocol Labs' event-driven model avoids this overhead. Crucially, both simulators satisfy Ethereum's 3 s propagation requirement, and relative comparisons across configurations remain valid since all 320 configurations use the same simulation methodology.

Table 5: Delivery rate under regional outages on 20% background churn. All configurations maintain >98.7% delivery even under 50% regional outages. GS(8,3) is within 0.4 percentage points of GS(8,8) at 3.2× lower cost.

Scenario	GS(8,1)	GS(8,3)	GS(8,8)
Baseline (no outage)	99.31%	99.80%	99.44%
US 30% offline 10s	99.16%	99.82%	99.80%
US 50% offline 10s	99.11%	99.64%	99.30%
Asia 30% offline 10s	98.99%	99.74%	99.84%
EU 30% offline 10s	98.74%	99.61%	99.98%
EU 50% offline 10s	99.21%	99.47%	99.88%
EU 30% offline 30s	99.18%	99.43%	99.81%

4.8.3 Correlated-Churn Validation. To address concerns that correlated failures might simultaneously damage both the eager mesh and the gossip recovery chain, we simulated regional outages on top of 20% background churn. Table 5 shows delivery rates for GS(8,1), GS(8,3), and GS(8,8) under various regional outage scenarios.

The data shows that even under 50% regional outages, all configurations maintain >98.7% delivery. Gossip peers in unaffected regions provide recovery via IHAVE/IWANT. GS(8,3) remains within 0.4 percentage points of GS(8,8) at 3.2× lower cost. This confirms that correlated failures do not invalidate Lean Gossip recommendations.

4.8.4 Limitations of Validation. We acknowledge several limitations in our validation approach:

Churn validation: No published measurements characterize Gossipsub behavior under controlled churn conditions comparable to our simulation. Our churn model is based on connection duration measurements from Kiffer *et al* [11], but delivery rate degradation under churn has not been validated against production data.

Scale gap: Our 1000-node simulation is substantially smaller than production Ethereum networks. While we verified that qualitative tradeoffs and the relative ordering of parameter configurations persist at larger scales, absolute metric values may differ.

Simplified protocol model: We omit peer scoring, multi-topic overlays, and Floodsub compatibility. These simplifications may affect behavior under adversarial conditions but should minimally impact benign churn scenarios. Despite these limitations, the consistency between our simulation and Protocol Labs' testbed results, particularly for baseline latency, and the qualitative structure of delivery-bandwidth tradeoffs, provides confidence that our methodology captures the essential dissemination dynamics of Gossipsub.

5 Simulation Results

We now evaluate how Gossipsub parameters influence the delivery–efficiency–latency tradeoff defined in Section 4.5. Our analysis focuses on two protocol parameters: the mesh degree D, which controls eager-push connectivity, and the gossip degree D_{lazy}, which controls lazy dissemination. We simulate Floodsub (D=60, D_{lazy}=0) as a baseline representing pure eager-push flooding without gossip-based recovery. Since Floodsub is modeled via Gossipsub parameters, it inherits mesh maintenance behavior; rejoining peers establish mesh connections within one heartbeat interval (∼1 s). This approximates realistic Floodsub behavior, as establishing 60 TCP connections after rejoin would incur comparable latency. For each

parameter configuration, we report results across all four churn regimes (0–30%) with error bars indicating the range across the other parameter dimension. This visualization reveals both typical behavior and the envelope of outcomes under varying network conditions. The separation is critical in churned networks, where churn-induced message loss can significantly impact protocol performance.

5.1 Effect of Mesh Degree

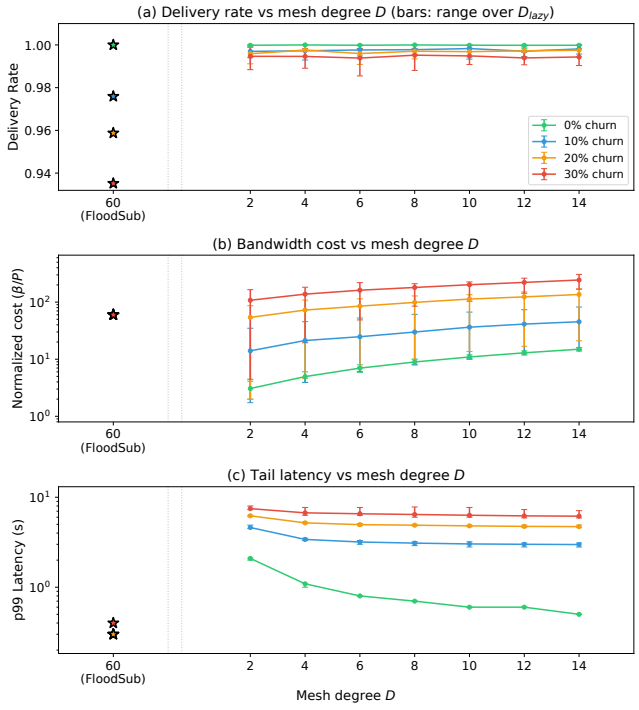


Figure 5.1: Effect of mesh degree D under varying churns. Each line represents a churn level (0–30%), plotting the mean across all D_{lazy} values; error bars show the range over $D_{lazy} \in \{1, 3, \ldots, 21\}$. Floodsub ($D$=60, D_{lazy}=0) shown as stars. (a) Delivery rate vs. D. (b) Normalized cost (β/P) vs. D. (c) Tail latency vs. D.

Figure 5.1 shows the impact of mesh degree D on delivery rate, bandwidth cost, and tail latency. Each line corresponds to a churn regime, with vertical bars indicating the range across gossip degrees D_{lazy}.

Delivery. Delivery rate improves with increasing D but exhibits strong diminishing returns, plateauing beyond $D \approx 8$. At 0% churn, all Gossipsub configurations achieve ~100% delivery. At 10–20% churn, delivery remains above 99% for $D \geq 4$. At 30% churn, configurations with $D \geq 4$ maintain ~99% delivery (98.8–99.2% depending on D_{lazy}), while Floodsub (D=60) degrades to 93.5%. The narrow error bars indicate that D_{lazy} has modest impact on delivery compared to D and churn level.

Efficiency. Normalized cost (β/P) grows monotonically with D. Floodsub incurs ~60× overhead, while Gossipsub ranges from 3× at low D and churn to ~200× at high D, D_{lazy}, and churn. The wide vertical bars reveal that D_{lazy} substantially impacts cost—higher gossip degrees increase bandwidth consumption significantly. Low-D configurations with minimal gossip achieve comparable delivery at substantially lower cost.

Latency. Tail latency decreases with increasing D, reflecting faster propagation in well-connected meshes. Floodsub achieves the lowest latency (~0.3 s) due to pure eager-push with no gossip delays. Gossipsub latency ranges from ~0.5–2 s at 0% churn to 5–8 s under high churn, with higher D reducing latency at all churn levels.

5.2 Effect of Gossip Degree

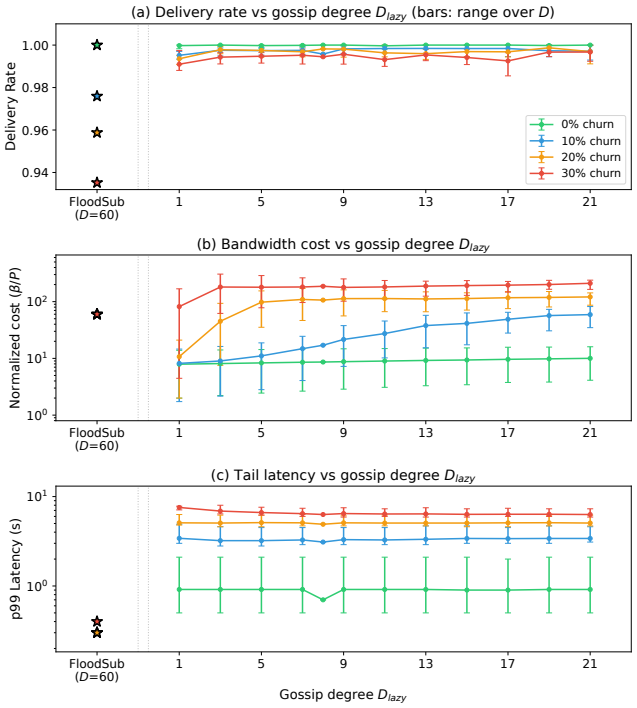


Figure 5.2: Effect of gossip degree D_{lazy} under varying churns. Each line represents a churn level (0–30%), plotting the mean across all D values; error bars show the range over $D \in \{2, 4, \ldots, 14\}$. Floodsub ($D$=60, D_{lazy}=0) shown as stars. (a) Delivery rate vs. D_{lazy}. (b) Normalized cost (β/P) vs. D_{lazy}. (c) Tail latency vs. D_{lazy}.

Figure 5.2 evaluates the impact of gossip degree D_{lazy}. Each line corresponds to a churn regime, with vertical bars indicating the range across mesh degrees $D \in \{2, 4, \ldots, 14\}$.

Delivery. Minimal gossip (D_{lazy}=1) achieves ~100% delivery at 0% churn and >99% delivery at 20% churn, exceeding Floodsub's 95.9% at the same churn level. Increasing D_{lazy} provides negligible improvement: at 20% churn, delivery rises by only 0.3 percentage points from D_{lazy}=1 to D_{lazy}=21 (99.4% to 99.7%). The wide vertical bars indicate that D has substantial impact on delivery, particularly

Table 6: Gossipsub configurations dominating Floodsub at each churn level. Minimal gossip (D_{lazy}=1) with sparse mesh exceeds Floodsub delivery at every churn level while achieving substantial cost savings.

Churn	Floodsub	Gossipsub	GS Delivery	Cost Savings
0%	100.0%	(2, 1)	100.0%	30×
10%	97.6%	(2, 1)	99.4%	35×
20%	95.9%	(2, 1)	99.2%	30×
30%	93.5%	(2, 1)	98.8%	13×

under high churn, while D_{lazy} has almost no effect beyond the minimal value.

Efficiency. Normalized cost increases sharply with D_{lazy}, from 8–82× at D_{lazy}=1 to 10–210× at D_{lazy}=21 depending on churn. This reflects IHAVE/IWANT control message overhead. High-D_{lazy} configurations under churn exceed Floodsub's ∼60× cost while providing negligible delivery improvement.

Latency. Tail latency shows high variability under churn, with D_{lazy} having modest direct effect. At 0% churn, latency remains below 2 s. Under 10–30% churn, latency ranges from ∼3 s to ∼8 s depending primarily on D (shown by wide vertical bars). Floodsub achieves the lowest latency (∼0.3 s) due to pure eager-push.

5.3 Delivery–Cost Tradeoffs and Dominance

Figure 5.3 plots delivery against normalized cost at 20% churn (the empirically grounded churn level from Kiffer *et al* [11]), with marker shape encoding p99 latency. Floodsub achieves the lowest latency (0.4 s), so it is not Pareto-dominated in the full three-dimensional objective space. However, on the delivery–cost plane at this churn level, it lies below the frontier: multiple Gossipsub configurations achieve both higher delivery and lower cost simultaneously.

Gossipsub dominates Floodsub on delivery and cost under churn. Floodsub (D=60, D_{lazy}=0) achieves 95.9% delivery at 60× cost at 20% churn. Every D_{lazy}=1 configuration (blue markers) exceeds this delivery at lower cost. Table 6 shows representative configurations: at 10% churn, (2, 1) achieves 99.4% delivery at 35× lower cost; at 20% churn, (2, 1) achieves 99.2% at 30× lower cost.

Floodsub's only advantage is sub-second latency. D_{lazy}=1 configurations (blue markers) fall in the 1–5 s p99 latency tier. (D=8, D_{lazy}=3) matches Ethereum's 4.9 s latency exactly while beating it on both delivery and cost; (D=8, D_{lazy}=1) achieves 4.8 s— slightly faster than Ethereum. Both are well within Ethereum's 12 s slot time. Floodsub's 0.3 s latency matters only for applications requiring sub-second propagation, at 60× the bandwidth cost.

5.4 Tradeoff Against Ethereum's Deployed Configuration

Ethereum deploys Gossipsub with mesh degree D=8 and gossip degree D_{lazy}=8. We compare this configuration against the Pareto frontier at 20% churn—the empirically calibrated churn level from Kiffer *et al* [11]. At 20% churn, Ethereum's configuration (D=8, D_{lazy}=8) achieves 99.8% delivery and 4.9 s p99 latency at 106× normalized bandwidth cost. Figure 5.3 shows that Lean Gossip configurations with $D_{lazy} \leq 3$ strictly dominate Ethereum on all three objectives simultaneously—higher delivery, lower or equal latency, and lower cost. (D=8, D_{lazy}=3) is the clearest example: it achieves 99.9% delivery and 4.9 s p99 latency at 34× cost—strictly better than Ethereum on delivery and cost, at identical latency, for a 3.2× bandwidth saving. (D=14, D_{lazy}=3) further improves latency to 4.6 s at 99.9% delivery, though with a more modest 1.1× cost saving. The leaner (D=8, D_{lazy}=1) pushes cost down to 9.8×—nearly 11× cheaper than Ethereum—while achieving slightly better latency (4.8 s) at the cost of 0.5 percentage points of delivery.

Table 7: Comparison of GS(8,3) vs GS(8,8) across churn levels. GS(8,3) dominates or matches GS(8,8) on delivery at every churn level while achieving 1.1–3.2× lower bandwidth cost.

Churn	GS(8,3) Delivery	GS(8,3) Cost	GS(8,8) Delivery	GS(8,8) Cost	Cost Ratio
0%	100.0%	8.1×	100.0%	8.6×	1.1×
10%	99.8%	9.0×	99.6%	17.1×	1.9×
20%	99.9%	33.8×	99.8%	106.5×	3.2×
30%	99.5%	179.9×	99.5%	186.5×	1.0×

This establishes that Ethereum's deployed gossip degree is overprovisioned for benign churn scenarios: delivery scales primarily with mesh degree D, while increasing D_{lazy} beyond 1 incurs steep bandwidth cost for only 0.3 percentage points of additional delivery across the full D_{lazy} range. Table 7 provides a detailed breakdown across all churn levels. At 20% churn, GS(8,3) achieves 3.2× lower cost than GS(8,8) while matching or exceeding delivery. At low churn (0–10%), both configurations achieve comparable delivery, but GS(8,3) still saves bandwidth (1.1–1.9×). At 30% churn, both converge to similar costs, but GS(8,3) achieves equivalent delivery.

Thus, Lean Gossip occupies a more favorable position on the delivery–bandwidth Pareto frontier than Ethereum's deployed configuration at empirically grounded churn levels.

6 Discussion

Our results reveal three structural insights about gossip-based dissemination under churn.

6.1 Why Floodsub Lies Off the Pareto Frontier Under Churn

Floodsub attempts delivery through network-layer redundancy: maintaining 60 eager-push connections to maximize delivery probability. However, under churn, if a message fails to reach a node because its mesh peers are offline, there is no recovery mechanism. Gossipsub employs protocol-layer repair via IHAVE/IWANT exchanges, enabling missed messages to be recovered even with sparse meshes. As a result, under non-zero churn Floodsub delivers fewer messages at dramatically higher bandwidth cost. At 0% churn, Floodsub achieves 100% delivery that sparse Gossipsub meshes cannot match. Its only consistent advantage across all churn levels is sub-second latency.

6.2 Why Lean Gossip Strictly Dominates Ethereum's Configuration

Ethereum deploys D=8, D_{lazy}=8, reflecting a conservative balance between redundancy and repair. Our Pareto analysis reveals that

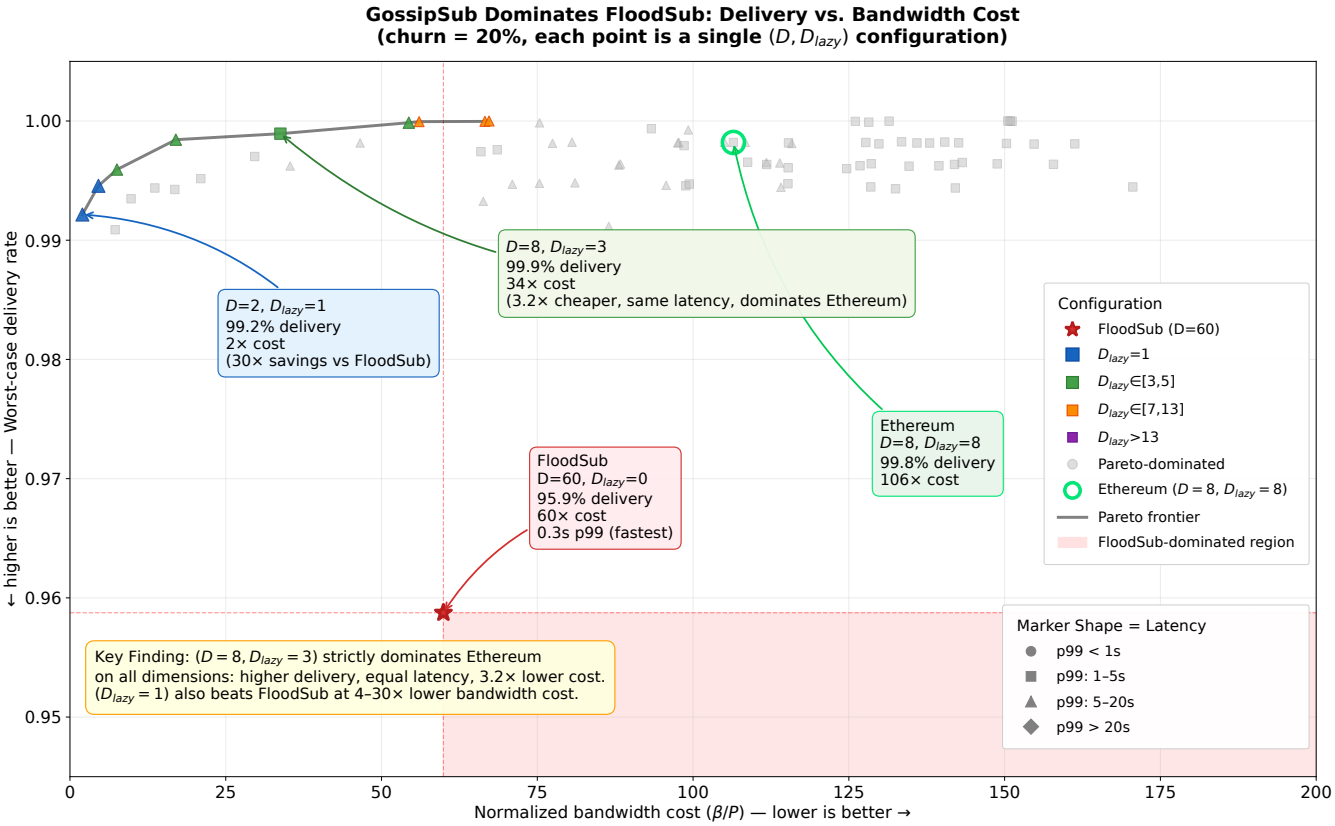


Figure 5.3: Pareto dominance: delivery vs. normalized cost at 20% churn (empirically grounded churn level from Kiffer *et al* [11]). Each point is a (D, D_{lazy}) configuration. Marker color indicates D_{lazy}; shape indicates $p99$ latency tier. Floodsub (D=60, red star) achieves 95.9% delivery at 60× cost. D_{lazy}=1 configurations (blue) achieve higher delivery at 2–21× cost. Red region: configurations inferior to Floodsub on both delivery and cost.

this balance is unnecessary: $D_{lazy} \leq 3$ configurations strictly dominate Ethereum on all three performance objectives at 20% churn. (D=8, D_{lazy}=3) achieves higher delivery (99.9% vs. 99.8%), equal latency (4.9 s), and 3.2× lower cost. The leaner (D=8, D_{lazy}=1) reduces cost to nearly 11× below Ethereum with slightly better latency, at the cost of only 0.5 pp of delivery. Delivery scales primarily with mesh connectivity (D), not gossip fanout (D_{lazy}); reducing D_{lazy} from 8 to 3 loses nothing and saves 3× in bandwidth.

6.3 Recovery Dominates Redundancy

A central conceptual contribution of this work is identifying that gossip-based recovery—not eager-push redundancy—is the dominant mechanism for delivery under churn.

Increasing D beyond approximately 8 yields diminishing delivery returns. Increasing D_{lazy} beyond 1 sharply increases bandwidth cost for negligible delivery benefit: at 20% churn, delivery improves by only 0.3 percentage points across the full range $D_{lazy} \in [1, 21]$, while cost increases by more than 10×. The true tradeoff is therefore latency versus cost—not delivery versus cost. Sparse meshes combined with minimal gossip recovery dominate brute-force flooding in the delivery–bandwidth plane under realistic churn.

6.4 Security Implications of D_{lazy}

Throughout this paper, we recommend D_{lazy}=3 as the primary configuration and D_{lazy}=1 as the extreme for bandwidth-constrained environments. However, these choices have security implications. D_{lazy}=3 provides 3× message-identifier diversity against a single malicious gossip peer compared to D_{lazy}=1: if a peer's only gossip contact is adversarial and suppresses IHAVE announcements, the peer may miss recovery opportunities. With D_{lazy}=3, the probability that all three gossip peers are adversarial drops exponentially. D_{lazy}=8 (Ethereum's value) provides 2.7× more diversity than D_{lazy}=3, but at 3.2× the bandwidth cost. Practitioners must weigh this security margin against bandwidth savings.

We note that Gossipsub v1.1 peer scoring is designed to penalize misbehaving peers detected through behavioral metrics (e.g., graft storms, message validation failures). However, we lack empirical data on scoring-induced pruning rates in benign Gossipsub deployments, making it difficult to calibrate how much security D_{lazy} diversity provides relative to scoring-based defenses. For this, we recommend production deployments instrument peer prune rates per topic.

Table 8: Representative configurations emphasizing different points in the reliability–latency–efficiency tradeoff space (worst-case across churn regimes). Gossipsub configurations with D_{lazy}=1 achieve higher delivery than Floodsub at lower cost; Floodsub's only advantage is latency.

D	D_{lazy}	Delivery	p99 Latency	Cost (β/P)
2	1	98.8%	7.6 s	4×
4	1	98.9%	7.7 s	19×
8	1	98.8%	7.8 s	85×
12	1	99.2%	7.3 s	143×
14	1	99.0%	7.1 s	168×
Floodsub reference (D=60, D_{lazy}=0)				
60	0	93.5%	0.4 s	60×

Table 9: Delivery rate by churn level for Gossipsub (D=8, D_{lazy}=1) compared to Floodsub (D=60, D_{lazy}=0). Gossipsub achieves higher delivery at every churn level >0%.

Churn	GS(8,1) Delivery	GS(8,1) p99	Floodsub Delivery	Floodsub p99
0%	100.0%	0.7 s	100.0%	0.3 s
10%	99.6%	3.2 s	97.6%	0.3 s
20%	99.3%	4.8 s	95.9%	0.3 s
30%	98.8%	7.8 s	93.5%	0.4 s

6.5 Practical Recommendations

Based on our simulations, we offer guidance for Gossipsub deployments.

For delivery-critical applications. Use $D \geq 8$ with D_{lazy}=1. These achieve approximately 98.8% delivery even at 30% churn at lower cost than Floodsub (Table 8).

For bandwidth-constrained environments. Configurations with $D_{lazy} \leq 3$ strictly dominate Ethereum on all performance dimensions and achieve delivery exceeding Floodsub at 3–30× lower cost. We recommend D=8, D_{lazy}=3 as the primary configuration: it strictly beats Ethereum on all three metrics at 3.2× lower cost. For the most bandwidth-constrained deployments, D=8, D_{lazy}=1 reduces cost to nearly 11× below Ethereum with only 0.5 percentage points of delivery reduction. Note that at 0% churn, Floodsub's 100% delivery cannot be matched by sparse configurations.

For latency-critical applications. If sub-second propagation is required, Floodsub (D=60, D_{lazy}=0) achieves 0.3 s p99 latency—but at 60× cost and 4% lower delivery at 20% churn.

Ethereum's parameters. Ethereum's deployed configuration (D=8, D_{lazy}=8) achieves 100% delivery at 0% churn and ∼99.8% at 20% churn; our results show that (D=8, D_{lazy}=3) strictly dominates these parameters—matching or exceeding all three metrics at 3.2× lower cost (Table 9).

Note that Table 8 reports the worst-case delivery rate and cost across all churn regimes (0–30%), not at a single churn level. This conservative presentation ensures that recommendations are robust to varying network conditions. Configuration (8, 1), for example, achieves 98.8% worst-case delivery, but its per-churn performance ranges from 100% (0% churn) to 98.8% (30% churn) as shown in Table 9.

Table 10: Diminishing returns of gossip degree (D=8, 30% churn). Increasing D_{lazy} from 1 to 21 improves delivery by only 0.8% points while increasing traffic 3×.

D_{lazy}	Delivery	Δ Delivery	Total Traffic	Traffic Mult.
1	98.8%	—	67.9 GB	1×
5	99.3%	+0.5 pp	144.4 GB	2×
9	99.5%	+0.7 pp	145.2 GB	2×
13	99.6%	+0.8 pp	150.8 GB	2×
17	99.6%	+0.8 pp	160.0 GB	2×
21	99.7%	+0.8 pp	170.5 GB	3×

Gossip explosion under churn. At D=8 with D_{lazy}=21 and 30% churn, total traffic reaches 170.5 GB—3× higher than D_{lazy}=1 (67.9 GB)—while improving delivery by only 0.8 percentage points (98.8% to 99.7%) relative to D_{lazy}=1 (Table 10). This occurs because churning peers repeatedly request messages via IHAVE/IWANT that never complete. Protocol designers should consider gossip backoff under detected churn.

7 Limitations and Future Work

Our analysis has several limitations. First, our churn model assumes uniform leave/join probabilities, whereas real networks may exhibit correlated churn (e.g., geographic outages affecting multiple peers simultaneously). No published measurement study currently characterizes correlated churn in Gossipsub deployments; our modular simulator is designed to accommodate alternative churn models as such data becomes available. Second, we do not model adversarial behavior; our parameter recommendations optimize for benign churn, and attack resilience under these parameters remains an open question. Protocol Labs evaluated Gossipsub v1.1's resilience against Sybil and Eclipse attacks [20, 21], and a Least Authority security audit identified further concerns with the peer scoring mechanism [16]. We note that empirical data on scoring-induced pruning rates in benign networks is not yet available—a go-libp2p issue explicitly requests metrics such as "peer prune rate per topic"—making it difficult to calibrate scoring models for non-adversarial conditions. More recently, Kumar *et al* [12, 13] used formal model-driven analysis to identify correctness problems with Gossipsub's adversarial safeguards in Ethereum configurations, showing that certain parameter settings can be exploited to cause Eclipse attacks or network partitions. Whether our recommended Lean Gossip parameters are equally or more vulnerable to such attacks is ongoing research.

Future work should extend this analysis to multi-topic scenarios, incorporate peer scoring mechanisms from Gossipsub v1.1 [15], and validate recommendations through testbed experiments or instrumentation of production networks. Additionally, our Pareto optimization framework could be applied to emerging protocol variants such as Gossipsub v2.0 [18], which introduces probabilistic lazy forwarding within the mesh itself. Finally, integrating our parameter selection methodology into adaptive protocols that dynamically adjust D and D_{lazy} based on observed network conditions represents a promising direction.

8 Conclusions

We presented a systematic, simulation-based study of Gossipsub parameterization under churn and identified *Lean Gossip* as a dominant dissemination regime. By sweeping mesh degree D and gossip degree D_{lazy} across four churn levels (0–30%) and applying Pareto dominance analysis over delivery and normalized bandwidth cost, we established three principal results.

Floodsub lies off the Pareto frontier in the delivery–bandwidth tradeoff under non-zero churn. Across churn levels of 10–30%, there exist Gossipsub configurations that achieve higher delivery at substantially lower bandwidth cost. At 0% churn, Floodsub achieves 100% delivery, that sparse Gossipsub meshes cannot match. Floodsub's only consistent advantage is sub-second latency; it does not provide superior delivery under realistic churn.

Second, Lean Gossip ($D_{lazy} \leq 3$) strictly dominates Ethereum's deployed Gossipsub configuration across all three performance dimensions simultaneously. (D=8, D_{lazy}=3) achieves higher delivery (99.9% vs. 99.8%), equal latency (4.9 s), and 3.2× lower cost. The leaner (D=8, D_{lazy}=1) extends the cost advantage to nearly 11× with slightly better latency, sacrificing only 0.5% points of delivery.

Third, we demonstrate that gossip-based recovery—not eager-push redundancy—is the dominant mechanism for reliable dissemination under churn. Increasing mesh degree beyond approximately D=8 yields diminishing delivery gains, while increasing D_{lazy} beyond 1 incurs steep bandwidth costs for 0.3 percentage points of improvement. Delivery scales with recovery rather than brute-force forwarding.

Collectively, these results overturn the intuition that redundant eager flooding ensures robustness. Instead, sparse meshes combined with minimal but effective gossip repair provide superior delivery–cost tradeoffs. We release our simulation framework and analysis methodology to enable principled, multi-objective evaluation and parameter optimization of gossip-based dissemination protocols.

Acknowledgements

We used generative AI tools (Claude Opus 4.5 and ChatGPT 5.2) to assist with writing simulation code and editing parts of this paper. All AI-generated content was critically reviewed, revised, and validated by the authors, who take full responsibility for the final work.

References

[1] 2025. *Lean Gossip: Gossipsub Parameter Optimization Simulator.* https://github.com/ankitku/lean-gossip Simulation framework accompanying "Lean Gossip: Big Gains From Small Talk".

[2] Paul Baran. 1964. *On Distributed Communications: I. Introduction to Distributed Communications Networks.* Technical Report RM-3420-PR. RAND Corporation. https://www.rand.org/pubs/research_memoranda/RM3420.html Accessed: 2025-06-08.

[3] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. 1999. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS).* doi:10.1145/317785.317786

[4] Flaviene Scheidt de Cristo, Jorge Augusto Meira, Jean-Philippe Eisenbarth, and Radu State. 2024. A 9-dimensional Analysis of GossipSub over the XRP Ledger Consensus Protocol. In *NOMS 2024 IEEE Network Operations and Management Symposium.* doi:10.1109/NOMS59830.2024.10575688

[5] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. 1987. Epidemic Algorithms for Replicated Database Maintenance. *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC).*

[6] Ethereum Foundation. 2020. *Ethereum Consensus Layer Networking Specification: Phase 0 P2P Interface.* Technical Report. https://github.com/ethereum/consensus-specs/blob/master/specs/phase0/p2p-interface.md

[7] Ethereum Foundation. 2021. Gossipsub parameter tuning. https://github.com/ethereum/consensus-specs/issues/2692. GitHub Issue #2692, ethereum/consensus-specs.

[8] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. 2001. Peer-to-peer membership management for gossip-based protocols. *Proceedings of the 2001 International Conference on Middleware.* doi:10.1007/3-540-45518-3_18

[9] Ajei S. Gopal, Inder S. Gopal, and Shay Kutten. 1999. Fast broadcast in high-speed networks. *IEEE/ACM Transactions on Networking* (1999). doi:10.1109/90.769773

[10] Lioba Heimbach, Yann Vonlanthen, Juan Villacis, Lucianna Kiffer, and Roger Wattenhofer. 2025. Deanonymizing Ethereum Validators: The P2P Network Has a Privacy Issue. https://arxiv.org/abs/2409.04366

[11] Lucianna Kiffer, Asad Salman, Dave Levin, Alan Mislove, and Cristina Nita-Rotaru. 2021. Under the Hood of the Ethereum Gossip Protocol. In *Financial Cryptography and Data Security.* Springer.

[12] Ankit Kumar, Max von Hippel, Panagiotis Manolios, and Cristina Nita-Rotaru. 2023. Verification of GossipSub in ACL2s. In *International Workshop on the ACL2 Theorem Prover and Its Applications.* doi:10.4204/EPTCS.393.10

[13] Ankit Kumar, Max von Hippel, Pete Manolios, and Cristina Nita-Rotaru. 2023. Formal Model-Driven Analysis of Resilience of GossipSub to Attacks from Misbehaving Peers. arXiv:2212.05197 [cs.CR] https://arxiv.org/abs/2212.05197

[14] libp2p. 2020. gossipsub v1.0: An extensible baseline pubsub protocol. https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md. GitHub Repository - libp2p specifications.

[15] libp2p. 2020. gossipsub v1.1: Extensions for attack resistance. https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md. GitHub Repository - libp2p specifications.

[16] Dylan Lott. 2020. Audit of Gossipsub v1.1 Protocol Design + Implementation for Protocol Labs. https://leastauthority.com/blog/audit-of-gossipsub-v1-1-protocol-design-implementation-for-protocol-labs/. Accessed 3 March 2022.

[17] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. 1999. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99).* doi:10.1145/313451.313525

[18] ppopth and libp2p contributors. 2024. Gossipsub v2.0 spec: Lower or zero duplicates by lazy mesh propagation. https://github.com/libp2p/specs/pull/653. GitHub Pull Request #653, libp2p/specs.

[19] ProbeLab. 2024. Ethereum Network Topology. https://probelab.io/ethereum/topology/. Accessed: 2025-01-15.

[20] Protocol Labs Research. 2020. GossipSub: An Attack-Resilient Messaging-Layer Protocol for Public Blockchains. https://research.protocol.ai/blog/2020/gossipsub-an-attack-resilient-messaging-layer-protocol-for-public-blockchains/. Protocol Labs Research Blog.

[21] Protocol Labs Research. 2020. *Gossipsub-V1.1 Evaluation Report.* Technical Report. Protocol Labs. https://research.protocol.ai/publications/gossipsub-v1.1-evaluation-report/ Protocol Labs Research Publication.

[22] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. 2002. The Broadcast Storm Problem in a Mobile Ad Hoc Network. *Wireless Networks* (2002). doi:10.1023/A:1013763825347

[23] Dimitris Vyzovitis, Yusef Nasirifard, Yiannis Psaras, and David Dias. 2019. *GossipSub: A Secure PubSub Protocol for Unstructured, Decentralized P2P Overlays.* Technical Report. Protocol Labs. https://research.protocol.ai/blog/2019/a-new-lab-for-resilient-networks-research/PL-TechRep-gossipsub-v0.1-Dec30.pdf Protocol Labs Technical Report.

[24] Dimitris Vyzovitis, Yusef Nasirifard, Yiannis Psaras, and David Dias. 2020. GossipSub: Attack-Resilient Message Propagation in the Filecoin and ETH2.0 Networks. *arXiv preprint arXiv:2007.02754* (July 2020). https://arxiv.org/abs/2007.02754 arXiv preprint.

[25] Chonghe Zhao, Yipeng Zhou, Shengli Zhang, Taotao Wang, Quan Z. Sheng, and Song Guo. 2024. DEthna: Accurate Ethereum Network Topology Discovery with Marked Transactions. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications.* IEEE, 1711–1720. doi:10.1109/INFOCOM52122.2024.10621281