Message Latency in Waku Relay with Rate Limiting Nullifiers

Alvaro Revuelta[†], Sergei Tikhomirov[†], Aaryamann Challani[‡], Hanno Cornelius[†] and Simon Pierre Vivier[†]

Abstract

Waku is a privacy-preserving, generalized, and decentralized messaging protocol suite. Waku uses GossipSub for message routing and Rate Limiting Nullifiers (RLN) for spam protection. GossipSub ensures fast and reliable peer-to-peer message delivery in a permissionless environment, while RLN enforces a common publishing rate limit using zero-knowledge proofs.

This paper presents a practical evaluation of message propagation latency in Waku. First, we estimate latencies analytically, building a simple mathematical model for latency under varying conditions. Second, we run a large-scale single-host simulation with 1000 nodes. Third, we set up a multi-host Waku deployment using five nodes in different locations across the world. Finally, we compare our analytical estimations to the results of the simulation and the real-world measurement.

The experimental results are in line with our theoretical model. Under realistic assumptions, medium-sized messages (25 KB) are delivered within 1 second. We conclude that Waku can achieve satisfactory latency for typical use cases, such as decentralized messengers, while providing scalability and anonymity.

Keywords

GossipSub, Waku, zkSNARKS, RLN, anonymity, latency, rate-limiting

1. Introduction

Peer-to-peer (P2P) protocols allow for fast and resilient data exchange. Their applications include data dissemination in blockchains and decentralized social networks. The goal of P2P protocol design is to navigate the trade-offs between speed, reliability, efficiency, security, and privacy.

Flooding and gossip are the two main approaches to message propagation in P2P networks. In flooding, nodes push messages to (a subset of) their neighbors eagerly, which ensures fast and reliable propagation at the cost of redundant bandwidth usage (*bandwidth amplification*). In gossip, nodes announce messages to their neighbors, and the full message is relayed only if the neighbor expresses interest in it. Gossip is more complex than flooding but more efficient in terms of bandwidth.

Security and privacy are especially important considerations for P2P protocols. In a permissionless network, an adversary can set up many nodes and launch a coordinated *Sybil attack*, such as isolating a victim from the rest of the network (*eclipse attack*), or overwhelming the

DLT2024: 6th Distributed Ledger Technologies Workshop, May 14–15, 2024, Turin, Italy

[†] Waku Research

[‡] Vac Research and Development

alrevuelta@status.im (A. Revuelta); sergei@status.im (S. Tikhomirov); aaryamann@vac.dev (A. Challani); hanno@vac.dev (H. Cornelius); simvivier@status.im (S. P. Vivier)

network with unwanted (spam) messages causing denial of service (DoS). Centralized mitigation methods, e.g. linking users' accounts to personally identifiable information such as a phone numbers, are harmful for privacy. Although proof-of-work was originally proposed as a spam countermeasure [1], it has since proved impractical due to computational requirements for end users' devices.

Publish-subscribe (PubSub) is a popular design pattern for P2P networks. Messages in a PubSub network are classified by topic. Nodes only receive messages from topics they are subscribed to.

Waku is a protocol suite for generalized, permissionless, and privacy-preserving P2P messaging. Waku Relay is the main Waku routing protocol based on GossipSub, a P2P PubSub protocol that combines flooding and gossip. Additionally, Waku Relay offers zero-knowledge-based spam protection, peer discovery, and sharding. Waku light protocols (Store, Filter, and Lightpush) allow resource-restricted devices to interact with Waku Relay nodes.

Rate Limiting Nullifiers (RLN) is a novel approach to spam protection for Waku Relay [2]. Let us refer to Waku nodes that wish to publish messages as *publishers*. RLN enforces a common rate limit upon every publisher using smart contracts and zero-knowledge proofs. The publisher registers on-chain, and provides a proof of their valid membership alongside each message they broadcast. Relaying nodes verify the proof before forwarding the message. Publishers that exceed the rate limit get disconnected from the network.

Low latency is important for user-facing applications, such as messengers. However, RLN proof verification at every node increases the latency in Waku.

Our contributions In this work, we aim to quantify the latency of Waku in realistic settings. We estimate the expected latency based on the assumptions regarding network delays, available bandwidth, and benchmarks for cryptographic operations that routing nodes perform locally. To validate our estimations, we run a single-host simulation and a multi-host measurement. In a single-host simulation with 1000 nodes, we obtain the distribution of message latencies. We then compare simulation results to real latencies measured using five Waku nodes deployed in different geographic regions.

Our results indicate that the overall latency is satisfactory for a user-facing application. Under realistic assumptions about node count and connectivity, messages of 25 KB or less get delivered within 1 second. We conclude that RLN is a practical spam countermeasure for a scalable, permissionless, decentralized messaging network.

The rest of this paper is structured as follows. In Section 2, we provide the necessary background on P2P networks, GossipSub, Waku, and RLN. In Section 3, we introduce the analytical model for latency in Waku. We describe our experimental methodology in Section 4. We present and discuss our results in Section 5, review related work in Section 6, outline avenues for future work in Section 7, and summarize the key findings in Section 8.

2. Background

2.1. GossipSub

GossipSub [3] is a P2P protocol that combines PubSub and Gossip. It was initially developed with the blockchain use case in mind (namely, for Ethereum 2.0 and Filecoin). GossipSub nodes may be connected in one of two ways.

- A **gossip connection** is only used to announce the messages that a node has recently seen to its neighbor, who may then selectively request full messages (the "lazy pull" approach).
- A **mesh connection** is used to relay full messages (the "eager push" approach). Nodes may also receive and send gossip announcements on mesh connections. To avoid excessive bandwidth consumption, nodes only maintain a handful of mesh connections (typically between $D_{low} = 4$ and $D_{high} = 12$).

A node can *graft* a connection, upgrading it from gossip to mesh, or *prune* a connection, downgrading it from mesh to gossip-only.

GossipSub nodes assign dynamic reputation scores to their neighbors, incentivizing good behavior. Multiple parameters are considered in score calculation [4], such as the time in the mesh and the number of invalid messages relayed. Connections to low-score peers are eventually dropped, whereas connections to high-score peers are kept.

GossipSub has high guarantees of delivery in adversarial environments, while having a reasonable amplification factor and latency [3].

2.2. TCP

We assume TCP as the underlying transport protocol. TCP flow control may affect the latency of message transmission. The TCP window size is negotiated in the early stages of a TCP transfer, and limits the amount of bytes a sender can send before waiting for an acknowledgement from the receiver. This implies that the round-trip time (RTT) between the nodes affects the latency. The maximum TCP window size is 65 KB, although RFC 1323 [5] introduces window scaling to well beyond this limit. We assume that all nodes implement this extension, as is the standard in all modern operating systems. Furthermore, data is transferred in individual segments each with a maximum size derived from the underlying data link layer maximum transmission unit (MTU) size. MTU size varies, but is typically 1.5 KB on Ethernet interfaces. Since each segment adds a small header, which increases the bandwidth overhead, MTU size may also affect latency. However, for small messages this effect is negligible. For large messages, TCP configuration and features such as MTU size, could become significant. These secondary effects are nontrivial to model and considered out of scope.

2.3. Zero-knowledge proofs

Zero-knowledge proofs (ZKPs) are cryptographic protocols enabling one party (the prover) to convince another (the verifier) of a statement's truth without revealing additional information [6, 7]. zkSNARKs, a subset of ZKPs[1], are additionally characterized by succinctness (the proofs are small) and non-interactivity (besides the prover sending the proof to the verifier, no communication between the parties is needed). The Ethereum Virtual Machine facilitates zkSNARK verification within smart contract execution [8].

2.4. Spam protection in P2P networks

Without countermeasures, attacker can abuse the resources of a permissionless P2P network. Early P2P file-sharing networks relied on reputation for rate limiting. Peers would keep score of their neighbor's behavior and allocate their resources (such as bandwidth) accordingly.

Proof-of-work (PoW), later used in Bitcoin and other blockchains, was initially invented as a spam countermeasure [1]. A notable advantage of PoW-based solutions, at least in theory, is stronger privacy protection, as peers do not need to be identified. However, PoW as a rate limiting tool in general-purpose messaging networks has not gained popularity (a notable attempt was Whisper [9]). The key challenge turned out to be setting the PoW puzzle difficulty high enough to deter attackers, but low enough to keep the network usable on resource-restricted devices.

P2P-protocols that underlie blockchains, such as `libp2p` in Ethereum, protect from spam with a combination of peer scoring and a degree of monetary protection stemming from the financial nature of the messages being broadcast. In particular, relaying nodes ensure that transactions pay a minimum fee required by the protocol, which burdens a potential spammer.

2.5. Waku

Waku is a suite of generalized messaging protocols that follows the GossipSub model, offers ZK-based privacy-preserving spam protection, and supports resource-restricted devices via light protocols. Waku is built according to the following design principles:

- **privacy-preserving**: avoid linking Waku users' identities to any sensitive information such as on-chain identity or IP address, thus providing transport privacy;
- **decentralized**: remove any central point of failure by using P2P architecture and encouraging a mesh topology;
- **permissionless**: allow anyone to join the network using open-source software, possibly using a resource-restricted device;
- **generalized**: support multiple use cases with unicast or multicast communication patterns.

Waku is built on top of `libp2p` [10], a modular networking stack for P2P protocols. In particular, Waku Relay, the backbone P2P protocol in the Waku suite, is built on top of `libp2p`'s GossipSub implementation. The Waku reference implementation, called `nwaku` [11], is written in Nim, a compiled high-level programming language. Other implementations include go-waku [12] in Go and js-waku [13] in Javascript. Waku is used as the backend for Status [14], a messaging app. The Waku Network (TWN) is a deployment of the Waku suite of protocols that launched in late 2023 [15].

[1] Strictly speaking, zkSNARKs function as arguments rather than proofs of knowledge [6].

2.5.1. Rate Limiting Nullifiers

Rate Limiting Nullifier (RLN) is a zero-knowledge gadget used in Waku [2] for privacy-preserving spam protection. RLN operates as follows:

Publisher registration The publisher generates the private key s_k and derives the corresponding commitment: $c = H(s_k)$. H is a cryptographic hash function (namely, Poseidon [16]). The publisher can obtain an RLN membership by registering its commitment c with an on-chain smart contract. The registration is permissionless. The contract stores a list of all valid commitments. Holding a valid membership entails knowing the secret key s_k such that its commitment c is part of the list of valid commitments.

Each registration incurs on-chain transaction fee. The registration requirement thus deters attackers from easily creating multiple (Sybil) identities. In the original RLN proposal [2], registration also involves putting down a deposit that is slashed in case of publisher's misbehavior.

Prerequisites for relaying nodes All Waku nodes are supposed to relay messages. Each node must be in sync with the current list of valid commitments. Synchronization is done through RPC calls to either an Ethereum RPC Provider (Infura, Alchemy, etc.) or a node's own Ethereum Execution Client. Each node locally constructs a Merkle tree T of all currently valid commitments. The tree must be updated when necessary according to on-chain events.

Message propagation An RLN identifier $rln\_identifier$ uniquely identifies an application and prevents cross-application replay attacks. The *epoch* is a value derived from or equal to the current UNIX timestamp divided by the length of time over which we want to rate limit. We define[2] an external nullifier $\emptyset$ as follows:

$$\emptyset = H(epoch, rln\_identifier)$$

We define an internal nullifier ϕ as follows:

$$\phi = H(s_k, \emptyset)$$

To publish a message, the publisher generates a proof π of holding a valid membership for the current epoch. The zero-knowledge property of zkSNARKs ensures that π conceals the publisher's identity beyond confirming their valid membership. The publisher attaches π, internal nullifier ϕ, $epoch$, $rln\_identifier$, and the Merkle root τ of tree T to the message.

Enforcing the rate limit We define the maximum epoch gap g as the maximum allowed gap between the relaying node's internal clock and the epoch for which a proof for an incoming message was generated. Currently, g is set to 20 s.

Each node maintains a set of nullifiers (ϕ) of messages relayed within the last g seconds. A message is considered valid if:

[2] Our definition of $\emptyset$ differs from the original definition [2] in that we add the RLN identifier to support multiple applications running on the same Waku network deployment.

- its proof π proves that the publisher holds a valid membership;
- no other messages within the last g seconds had the same nullifier ϕ;
- the gap between the epoch used for proof generation and the node's internal clock does not exceed g.

Only valid messages are forwarded. Nodes that relay invalid messages have their GossipSub scores lowered. In particular, unsuccessful validation affects the P_4 score component ("Invalid Messages for a topic" in GossipSub terms [4]). Eventually, the violating node is isolated from the network. Note that the violating node is not necessarily the original publisher. This mechanism discourages forwarding invalid messages, as a node that does so will itself be punished.

In summary, RLN prevents malicious actors from overwhelming the network with messages. At the same time, RLN respects the privacy of publishers, as they only have to prove that they hold a valid membership without specifying any further details.

Note that the current nwaku implementation does not yet support economic punishment of malicious publishers, which was described in the original RLN paper [2].

3. Analytical model for propagation latency

We define latency as the time between the publisher's expressed intention[3] to publish a message and the receiver node receiving it.

For a given message propagation, the total latency is the sum of individual delays along the shortest path from the publisher to the receiver. Therefore, the total path latency depends on individual hop-level delays and on the number of hops. From a network-wide perspective, the distribution of latencies depends on the number of nodes in the network, their degrees, and the network topology.

3.1. Latency components on a given path

Let us denote the number of the nodes in the network as N, and the degree of each node as D (we assume that the network is a regular graph). The overall latency L for a propagation path can be divided into three components: proof generation time at the publishing node, transmission latency, and proof validation at each relaying node (see Figure 1 as an illustration for $D = 2$ and $N = 8$). Let us consider the three latency components in turn.

3.1.1. Proof generation

The publisher generates one proof π of valid RLN membership per message sent. L_g^i denotes the time it takes the publisher i to generate π. Proof generation time depends on the hardware platform (see Table 1 for benchmarks).

After generating π, the publisher may optionally verify the proof locally before broadcasting the message. This might be useful, for instance, if the publisher outsources proof generation to a third party and wants to independently check that the proof is valid. In our calculations, we do not account for this optional step.

[3] In other words, latency includes preparatory steps before the message is relayed, such as generating the RLN proof.

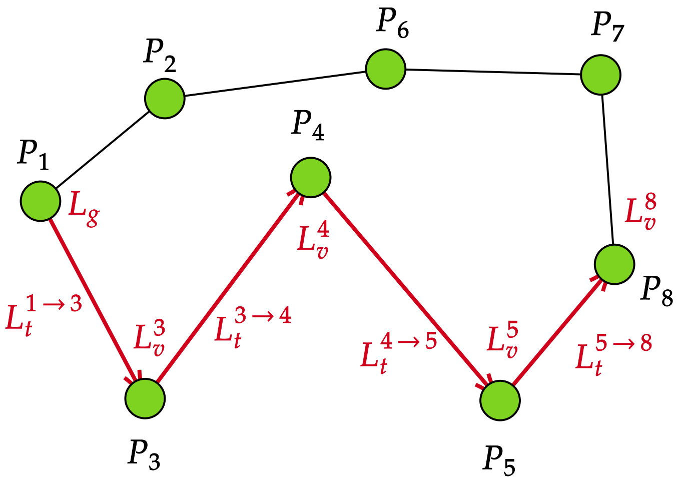

Figure 1: An example of a message propagation in a network with $N = 8$ and $D = 2$, with latency components indicated. The message is relayed from P_1 to P_8 via four hops.

Platform	Generation (ms)	Verification (ms)
MacBook M1 Pro 16GB	85.7	2.7
Bare Metal 256GB AMD EPYC 7502P 32-Core	171.6	7.9
Digital Ocean 8GB/4CPU	**276.3**	**4.5**
Digital Ocean 2GB/2CPU	329.5	4.4
Raspberry Pi 4B 4GB	766.8	18.7

Table 1
Benchmarks for RLN proof generation and verification times for nwaku [17] on various platforms (the platform we used in the multi-host measurement described in Section 4.2 is indicated in bold).

3.1.2. Message transmission

$L_t^{i\rightarrow j}$ denotes the message transmission time from node i to its neighboring node j. This time depends on the message size, the available bandwidth between the nodes, and the underlying transport protocol (TCP assumed).

We discard the effect of TCP on latency (see Section 2.2), under the assumption that messages are small, no retransmissions occur, the connection handshake was already established, and the flow control is already adapted between the neighboring nodes. Therefore, once the neighboring nodes have adapted their TCP window size and window scaling, the transmission time L_t^h for small messages (such as under 100 KB) should be close to $RTT/2$ in theory.

3.1.3. Message validation

L_v^i denotes the validation time at a relaying node P_i. Remember that every node validates every incoming message before forwarding it. Validation involves multiple steps, such as decoding and RLN proof verification, which is the most resource-consuming step. Messages are cached to avoid duplicate validation. We have performed proof verification benchmarks on different platforms (see Table 1 for results and [17] for methodology).

3.2. Analytical formula for latency

With all the latency contributions from Sections 3.1.1, 3.1.2 and 3.1.3, we can model the total latency with Equation 1. L_g and L_v can be taken from Table 1 depending on the platform. Note that message decoding and other non-significant time contributions are not considered. i denotes the index of a node in the path ($i = 0$ is the publisher, $i = h$ is the receiver). h can take any value in $[h_{min}, h_{max}]$, where $h_{min} = 1$ can be used for the best-case latency for neighboring nodes.

$$L = L_g^0 + \sum_{i=1}^{h} (L_t^{(i-1)\to i} + L_v^i) \tag{1}$$

3.3. Bandwidth amplification

The bandwidth amplification factor is the ratio of total bandwidth utilized to relay a message at each node to the size of that message. Since in Waku Relay each message is routed exactly once on each mesh connection, the amplification factor is roughly equal[4] to the mesh degree D at each node. Increasing D increases bandwidth amplification (independent of N) and decreases latency. The effect of D on latency lasts only up to a certain threshold for any given N (Figure 2). We observe that while higher values of D allow for faster message dissemination, it only makes sense to increase D up to a certain threshold to avoid useless bandwidth amplification. For that reason, in real Waku networks, D is limited to $D_{high} = 12$.

3.4. Estimating the number of hops

Waku Relay is based on GossipSub, which strongly enforces same-degree topology. Peer discovery is randomized, and nodes try to maintain the degree of $D \in [D_{min}, D_{max}]$. These protocol properties allow us to make a reasonable assumption that the network topology tends towards a mesh topology rather than hub-and-spoke[5].

As an simplified model, consider a network with a constant node degree D and N nodes in total. Let us denote as N_h the number of nodes that have received the message after h propagation steps. We can derive h_{max}, which is a lower bound on h for given values of N and D that is sufficient for all nodes to receive a message.

Let us consider two cases: $D = 2$ and $D > 2$. The edge case where $D = 2$ implies a circle topology, where a message is only relayed to two new nodes on each propagation step. In other words, propagation is described by an arithmetic rather than geometric sequence. After h propagation steps, $N_h = 1 + 2h$ nodes know the message. Therefore, $h_{max} = \lceil \frac{N-1}{2} \rceil$. Now, let's consider the general case of $D > 2$. Before the first step, only one node (the publisher) knows the message. After the first step, all D neighbors of the publisher receive the message. For all subsequent steps, the process can be characterized by a geometric sequence:

[4] Not exactly equal due to additional increase in control message frequency.

[5] In hub-and-spoke, most nodes are only connected to a handful of well-connected hubs, which is undesirable for decentralization and censorship resistance.

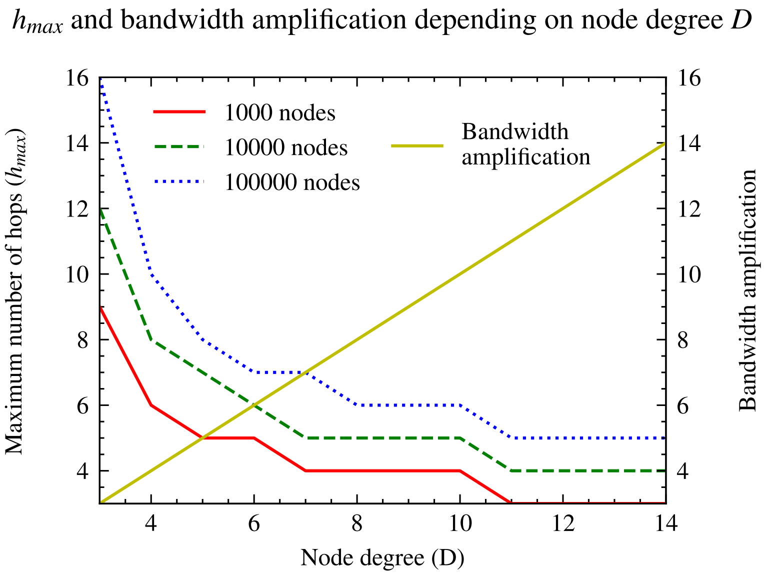

Figure 2: The best-topology maximal path length h_{max} and bandwidth amplification for various values of of node degree D, calculated using Equation 2.

$$N_h = \begin{cases} D+1 & \text{if } h=1 \\ D \times (D-1)^{h-1}+1 & \text{if } h>1 \end{cases}$$

Applying the formula for the partial sum of a geometric sequence, we derive Equation 2:

$$h_{max} = \left\lceil \frac{\log\left(\frac{(N-1)(D-2)}{D}+1\right)}{\log(D-1)} \right\rceil \tag{2}$$

The best-case maximum number of hops h_{max} is inversely related to bandwidth amplification (Figure 2). We ignore the case of $D = 2$ in Figure 2 because it is unrealistic for Waku in practice. Intuitively, higher node degrees lead to faster message propagation but at the same time increase bandwidth consumption. While this observation holds in practice and in our subsequent experiments, the exact formula is not necessarily applicable for Waku because of two reasons. First, Equation 2 concerns the theoretically optimal topology in terms of propagation efficiency. Second, nodes in Waku have a variable degree, which may lead to different connectivity properties compared to the simplified model of Equation 2. Keeping these limitations in mind, we can plug $N = 1000$ and $D = 6$ into Equation 2 and derive $h_{max} = 5$.

4. Methodology

We estimate latency in Waku using a single-host simulation (Section 4.1) and a multi-host node deployment in different geographic locations (Section 4.2). The two approaches complement each other. The single-host simulation allows us to model a large network, while nodes in different locations allow us to measure the impact of real network conditions. The multi-host measurements are intended to assess the performance of Waku as a message traverses an individual path of geographically distributed nodes.

We use nwaku implementation in all experiments and consider message sizes of 2, 25, 100, and 500 KB. The depth of the Merkle tree T is 20 in all experiments, as per the protocol specification [18]. Deeper Merkle trees support more users and provide stronger anonymity, but require more computational resources for proof generation and verification. The tree depth of 20 was chosen as a reasonable trade-off.

4.1. Single-host simulation

We simulate a network of 1000 nodes using Shadow [19], a simulation framework. Each node tries to maintain $D = 6$ (mesh) connections and is additionally connected to 25 gossip peers. $D = 6$ is the desired node degree in a typical `libp2p` network[6].

The upstream and downstream bandwidth is set to 100 Mbps. The latency is set to 150 ms. See [21] for the full configuration file.

Ten nodes are designated as publishers. Each of these publishes one message, which is propagated on all mesh connections. This leads to a total of nearly 10000 message received events (9990, accounting for the fact that the publishers do not need to receive the messages they send). Effects from the first two messages are excluded from our measurements, to prevent latency bias due to TCP window size negotiation.

The Shadow simulation framework does not model CPU time. To account for processing delays, we manually introduce delays for RLN proof generation and verification, according to our benchmarks (see Table 2).

In all simulations, four hops were sufficient to deliver a message to all nodes. This result is lower to what Equation 2 suggests (namely, five hops for $N = 1000$ and $D = 6$). Due to variable node degrees in our simulations, Equation 2 is not directly applicable, although we can use it as a sanity check. Simulation results inform our choice of a four-hop path in the subsequent multi-host measurements (Section 4.2).

4.2. Multi-host measurements

To estimate the effects of real-world communication delays, we deploy $N = 5$ nodes in different geographic locations. We use Digital Ocean [22] machines with 8 GB RAM and 4 vCPU (virtual CPUs). The five nodes are statically connected (without peer discovery) in a linear fashion, in the following order by location: Singapore, Bangalore, San Francisco, New York, and Frankfurt. Hence, node degrees are $D = 1$ for the sender (Singapore) and receiver (Frankfurt) nodes, and $D = 2$ for the intermediary nodes. Each node can only receive messages on a mesh connection from the previous node on the path, and relay the message on a mesh connection to the following node. Messages are therefore forced to travel along this route, eagerly pushed along the path (no messages are pulled using gossip dissemination).

[6] See `libp2p` documentation [20]: "In `libp2p`'s default implementation the ideal network peering degree is 6 with anywhere from 4–12 being acceptable."

To From	Bangalore	Frankfurt	Singapore	San Francisco	New York
Bangalore	N/A	136	60	220	213
Frankfurt	132	N/A	159	150	87
Singapore	59	160	N/A	176	236
San Francisco	218	146	176	N/A	68
New York	214	90	235	68	N/A

Table 2
Round-time trip (RTT) ping latency (ms) between deployed nodes in various locations.

The goal of this experiment is to measure the latency of a message as it travels through a path of geographically distributed nodes. The setup is chosen to simulate a path that is sufficiently long to get to most nodes. We choose the path length of $h = 4$ for the multi-host measurement. Based on the single-host simulation results (Section 4.1), we observe that the majority of messages take no more than four hops to propagate in a mesh network with $N = 1000$ and $D = 6$ (see interpretation of Figure 3). The exact distribution of path lengths depends not only on the network size and the node degree, but also on topology. We argue that measuring a path of length four is justified to validate the single-host simulation results in realistic network conditions. The measurement results can also be extrapolated to longer paths using the analytical formula for latency (Equation 1).

We measure latency as the difference in the arrival times of a message, according to the local clock of the relevant nodes. We use NTP [23] to synchronize the clocks on the nodes, reducing possible errors to a few milliseconds.

We use ping times between the relevant city pairs as the baseline for message latency (see Table 2). We use the ping protocol implemented in `libp2p` [10] and the wakucanary tool [24]. Ping is measured as an average of the round-trip time (RTT) of 5 requests done within a 15-minute time span.

5. Results and Discussion

Table 3 shows the multi-host measurements of the proof generation time L_g, transmission time $L_t^{i \to j}$, and validation time L_v, as discussed in Section 3.

As expected, larger messages cause higher latency. Latencies between geographically closer nodes are smaller. The transmission time L_t of small messages is close to half the RTT as measured in Table 2. For example, the RTT between Bangalore and San Francisco is 220 ms, and $L_t^{Si \to Ba} = 105$ ms.

Proof generation time is much larger than validation time (Table 1). Proof generation does not depend on the message size: proofs are generated on message hashes, and we do not count hashing as part of proof generation. According to our benchmarks (Table 4), the hashing time is under 1.2 ms in the worst case, which is insignificant compared to the overall latency.

Validation time L_v, on the other hand, generally increases with message size. This is explained by the fact that validation includes not only proof verification but also decoding the message. The decoding time is linear in the size of the message.

	Singapore	Bangalore		San Francisco		New York		Frankfurt		
Msg size (KB)	L_g	$L_t^{Si \to Ba}$	L_v^{Ba}	$L_t^{Ba \to SF}$	L_v^{SF}	$L_t^{SF \to NY}$	L_v^{NY}	$L_t^{NY \to Fr}$	L_v^{Fr}	L_{total}^{avg}
2	236	38	6	105	7	29	5	42	6	477
25	258	81	10	344	11	100	6	127	7	945
100	223	119	8	755	12	261	9	289	11	1689
500	247	275	16	1017	19	391	21	462	17	2468

Table 3

Multi-host network measurements of five `nwaku` [11] nodes deployed at different locations for various message sizes. The message travels from Singapore to Frankfurt via the other nodes in the order listed. Latencies (one-way, or $RTT/2$) are in ms. L_g, L_t, and L_v are averages across five runs. L_{total}^{avg} is an average of the total latencies across five runs. Note: L_{total}^{avg} (the average of sums of sub-latencies for each run) is not equal to the sum of averages (i.e., the value in the last column is not necessarily equal to the sum of other values in that row).

Message size (KB)	Hashing time (ms)
2	0.005516
25	0.061144
100	0.243716
500	1.222206

Table 4

Benchmarks for `Keccak256` hashing and conversion to a `BN254` field element using `go-zerokit-rln` library [25] (`goos`: `darwin`, `goarch`: `arm64`).

Now consider the distribution of latencies obtained in the single-host simulation compared to the latencies measured in a multi-host deployment (Figure 3).

Simulated latency distributions for small messages tend to be discrete, reflecting separate hops. As we increase the message size, the latency distribution starts resembling a normal distribution. We explain this as follows: for larger messages, a larger share of the total latency is spent on message transmission. For small messages, the transmission is nearly instant compared to validation time at each node, which explains the peaks in the upper charts.

Simulation latencies largely fall between the minimal and maximal latencies as measured in the multi-host deployment (dashed lines in Figure 3). Simulations seem to underestimate the latency for 100 KB and 500 KB messages, most likely due to the fact that our simulation framework does not accurately model CPU time (we do account for proof verification time as benchmarked in Table 1, but do not account for other related tasks, such as message decoding). In a similar vein, simulations overestimate latency for small messages, which is especially visible in the plot for 2 KB messages. Messages of 25 KB or smaller are always delivered in under 1 s, both in simulations and in the multi-host measurements. 95% of large messages (500 KB) are delivered in under 1.7 seconds according to simulation results (Table 5).

For a four-hop multi-host path, proof generation represents between around 10% to 50% of the total latency (for 500 KB and 2 KB messages, respectively). We consider this tolerable, as the proof is generated once per message. Proof verification (which is only a part of message validation) does not contribute significantly to the total latency.

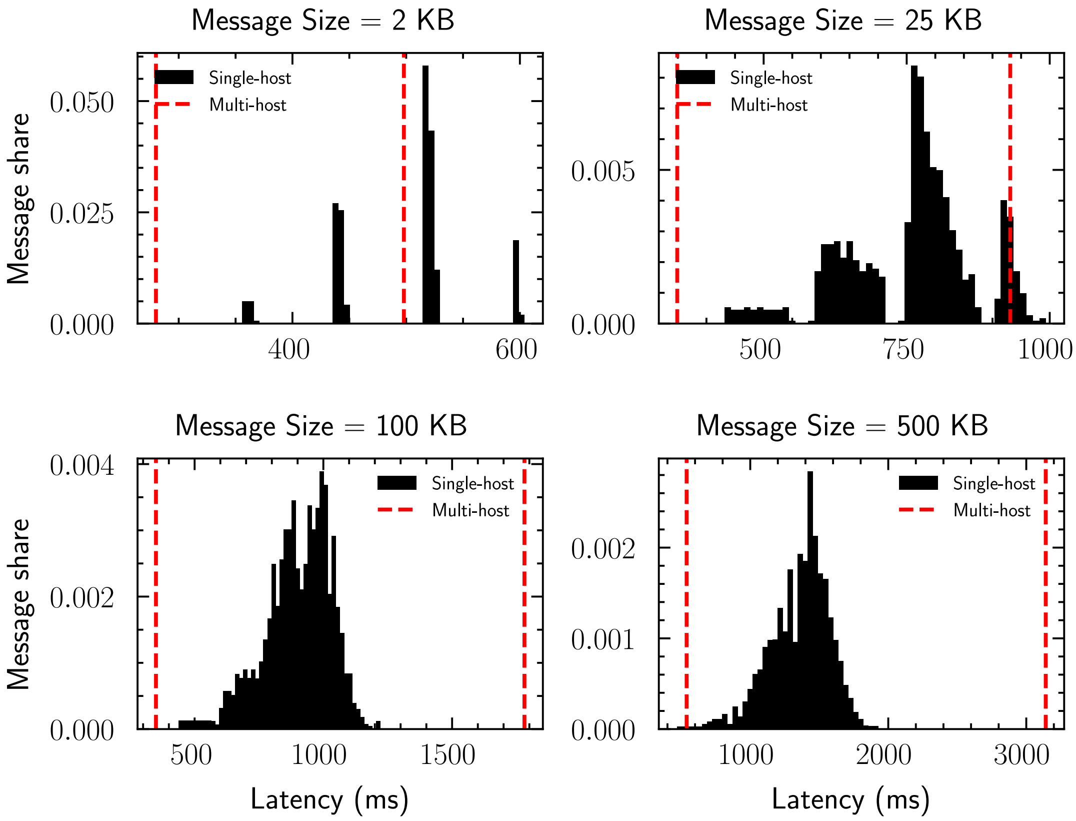

Figure 3: Single-host simulation results of 1000 nwaku [11] nodes with $D = 6$ in Shadow [19] simulator. Message propagation latency distribution is shown for different message sizes. The average, 95% percentile, min and max values are shown, values in ms. In red worst and best case propagation times from multi-host simulation extracted from Table 3.

Msg size	Min (m-h)	Min (s-h)	Avg (s-h)	95-perc. (s-h)	Max (s-h)	Max (m-h)
2	280	356	497	597	604	498
25	349	432	756	930	993	931
100	350	439	900	1076	1221	1781
500	539	471	1358	1665	2468	3141

Table 5
The minimum, average, 95-percentile, and maximum total latencies (ms) in the single-host distributions (marked "s-h") and multi-host measurements (marked "m-h") for different message sizes (KB). We do not provide the averages and the 95-percentile data for the multi-host measurements due to a small number of data points.

6. Related Work

P2P protocols for message dissemination in blockchains have been studied extensively, including works on latency measurements in Bitcoin [26, 27] and Ethereum [28]. GossipSub [3] has been designed for information propagation in Ethereum and Filecoin. Waku [29], based on GossipSub, uses RLN [2], a ZKP-based protocol, for rate limiting.

Decentralized messaging is a related but separate line of work with a long history [30]. Decentralized messaging protocols normally define the rules of communication between a client and a server, and between two servers. Clients connect to the servers of their choosing, while servers forward users' messages to one another, forming a federation. Notable examples include: ActivityPub [31] (which e.g. Mastodon is based on [32]), Matrix [33], Nostr [34], XMPP [35], Diaspora [36], AT Protocol [37], and Farcaster [38].

Waku distinguishes itself from federated protocols: it aims to be generalized (not only useful for chat-like applications) and provides transport privacy. Waku also embeds scalability and security mechanisms into its transport and routing layer. Specifically, it leverages RLN to prevent abuse of the open infrastructure and provides various scalability avenues, including sharding and light protocols (see Section 1).

7. Future Work

Dynamic topologies. Our analysis assumes a stable full mesh topology (see Section 3). However, real-world networks experience significant churn: nodes join and leave dynamically. Churn may lead to temporary neighborhoods of denser or sparser connectivity. One future avenue of research is investigating the impact of node churn and different topologies on propagation latency.

Higher message rates. We have analyzed network performance at low message rates. Increasing message rates leads to more control messages, higher processing overhead, and other transient effects impacting latency. Future work should explore these effects systematically.

Testing under more complex scenarios. Future research might evaluate the performance of Waku with RLN under more challenging conditions, which may involve node faults, network failures, clock de-synchronization, and other adversarial conditions.

On-chain RLN membership tree. We currently require all RLN publishers and verifiers to construct and maintain a copy of the membership Merkle tree T locally (see Section 2.5.1), which may be impractical for resource-restricted nodes. We are investigating models where the entire tree T resides on-chain, allowing for delegated proof generation and simplified verification.

Security and privacy analysis. Further research should focus on identifying vulnerabilities and attack vectors. Enhancing security mechanisms to prevent spam, Sybil attacks, and adversarial behavior will contribute to the robustness of the network.

Comparison with traditional P2P protocols without RLN. Another research direction may consider comparable P2P protocols with other rate-limiting methods (for instance, reputation-based) and compare their properties with those of Waku.

8. Conclusion

In this work, we studied message propagation latency in Waku — a GossipSub-based P2P messaging protocol that uses RLN [2] for privacy-preserving rate limiting. We simulated a network of 1000 nodes under realistic assumptions (Section 4.1), and used a geographically distributed cloud deployment (Section 4.2). The results show that the delays that RLN imposes are not overwhelming compared to the overall message latency. Proof generation, which the publisher does only once per message, requires under 300 ms per proof and does not depend on message size, assuming that the message has been hashed. Proof verification, performed by every routing node, is roughly an order of magnitude faster than proof generation. Overall, RLN-related tasks account for between around 10% to 50% of the total latency, depending on message size. This percentage is higher for small messages, as proof generation is independent of message size, and transmission time is lower for small messages. In absolute numbers, all messages of 25 KB and smaller are delivered in under 1 second. We conclude that RLN can be a practical rate limiting tool in real-world protocols underpinning user-facing applications.

Acknowledgments

We would like to acknowledge Mikel's contributions to benchmarking Waku and RLN on Raspberry Pi.

References

[1] A. Back, et al., Hashcash-a denial of service counter-measure (2002).

[2] S. Taheri-Boshrooyeh, O. Thorén, B. Whitehat, W. J. Koh, O. Kilic, K. Gurkan, WAKU-RLN-RELAY: Privacy-preserving peer-to-peer economic spam protection, in: 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW), 2022, pp. 73–78. doi:10.1109/ICDCSW56584.2022.00022.

[3] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, Y. Psaras, Gossipsub: Attack-resilient message propagation in the filecoin and ETH2.0 networks, CoRR abs/2007.02754 (2020). URL: https://arxiv.org/abs/2007.02754. arXiv:2007.02754.

[4] P. Labs, Gossipsub v1.1. score function formal specification, https://github.com/libp2p/specs/blob/b9efe152c29f93f7a87931c14d78ae11e7924d5a/pubsub/gossipsub/gossipsub-v1.1.md#the-score-function, 2024. Accessed: 2024-02-13.

[5] D. A. Borman, R. T. Braden, V. Jacobson, TCP Extensions for High Performance, RFC 1323, 1992. URL: https://www.rfc-editor.org/info/rfc1323. doi:10.17487/RFC1323.

[6] A. Nitulescu, zk-SNARKs: A Gentle Introduction (2020).

[7] A. Ballesteros Rodríguez, zk-SNARKs analysis and implementation on Ethereum (2020).

[8] What are zero-knowledge proofs?, https://ethereum.org/en/zero-knowledge-proofs/, 2024. Accessed: 2024-03-11.

[9] V. Gluhovsky, Eip-627: Whisper specification, https://eips.ethereum.org/EIPS/eip-627, 2017. Accessed: 2024-04-23.

[10] libp2p official website, https://libp2p.io/, 2024. Accessed: 2024-02-28.

[11] nwaku repo, https://github.com/waku-org/nwaku/, 2024. Accessed: 2024-02-13.

[12] go-waku repo, https://github.com/waku-org/go-waku, 2024. Accessed: 2024-03-11.

[13] js-waku repo, https://github.com/waku-org/js-waku, 2024. Accessed: 2024-03-11.

[14] Status app official website, https://status.app/, 2024. Accessed: 2024-03-05.

[15] Waku Network launch press-release, https://cointelegraph.com/press-releases/waku-launches-first-decentralised-privacy-preserving-dos-protections-for-p2p-messaging, 2023. Accessed: 2024-02-29.

[16] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, M. Schofnegger, Poseidon: A new hash function for zero-knowledge proof systems, in: USENIX Security Symposium, USENIX Association, 2021, pp. 519–535.

[17] A. Revuelta, nwaku benchmark, https://github.com/waku-org/nwaku/blob/097cb36279897801a5963343c7298220b6290228/apps/benchmarks/benchmarks.nim, 2024. Accessed: 2024-02-13.

[18] B. Whitehat, S. Taheri, O. Thorén, O. Kilic, B. Dimovski, Rate Limit Nullifier, https://github.com/vacp2p/rfc-index/blob/main/vac/32/rln-v1.md, 2024. Accessed: 2024-04-23.

[19] The Shadow simulator. a discrete-event network simulator that directly executes real application code, https://shadow.github.io/, 2024. Accessed: 2024-02-13.

[20] libp2p documentation. what is publish/subscribe, https://docs.libp2p.io/concepts/pubsub/overview/, 2024. Accessed: 2024-03-11.

[21] nwaku shadow simulation config, https://github.com/waku-org/research/blob/3a2da04e548b034dfaf63aada54d63197aa8ef8e/rln-delay-simulations/shadow.yaml, 2024. Accessed: 2024-02-13.

[22] Digital ocean, https://www.digitalocean.com/, 2024. Accessed: 2024-02-13.

[23] Network Time Protocol version 4: Protocol and algorithms specification, https://datatracker.ietf.org/doc/html/rfc5905, 2024. Accessed: 2024-03-04.

[24] waku canary tool, https://github.com/waku-org/nwaku/tree/097cb36279897801a5963343c7298220b6290228/apps/wakucanary, 2024. Accessed: 2024-02-13.

[25] Hashing benchmark code, https://github.com/waku-org/go-zerokit-rln/pull/20, 2024. Accessed: 2024-04-22.

[26] C. Decker, R. Wattenhofer, Information propagation in the bitcoin network, in: P2P, IEEE, 2013, pp. 1–10.

[27] T. Neudecker, P. Andelfinger, H. Hartenstein, Timing analysis for inferring the topology of the bitcoin peer-to-peer network, in: Proceedings of the 13th IEEE International Conference on Advanced and Trusted Computing (ATC), 2016.

[28] L. Zhang, B. Lee, Y. Ye, Y. Qiao, Evaluation of ethereum end-to-end transaction latency, in: NTMS, IEEE, 2021, pp. 1–5.

[29] Waku official website, https://waku.org/, 2024. Accessed: 2024-02-27.

[30] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, M. Smith, Sok: Secure messaging, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, 2015, pp. 232–249.

[31] Activitypub official website, https://www.w3.org/TR/activitypub/, 2024. Accessed: 2024-02-27.

[32] A. Raman, S. Joglekar, E. D. Cristofaro, N. Sastry, G. Tyson, Challenges in the decentralised

web: The mastodon case, in: Internet Measurement Conference, ACM, 2019, pp. 217–229.

[33] Matrix official website, https://spec.matrix.org/latest/, 2024. Accessed: 2024-02-27.

[34] Nostr official website, https://nostr.how/en/what-is-nostr, 2024. Accessed: 2024-02-27.

[35] Xmpp official website, https://xmpp.org/, 2024. Accessed: 2024-02-27.

[36] Diaspora official website, https://diasporafoundation.org/, 2024. Accessed: 2024-02-27.

[37] M. Kleppmann, P. Frazee, J. Gold, J. Graber, D. Holmgren, D. Ivy, J. Johnson, B. Newbold, J. Volpert, Bluesky and the AT protocol: Usable decentralized social media, CoRR abs/2402.03239 (2024).

[38] Farcaster official website, https://docs.farcaster.xyz, 2024. Accessed: 2024-02-27.