Mitigating Front-Running Attacks through Fair and Resilient Transaction Dissemination

Wassim Yahyaoui, Joachim Bruneau-Queyreix, Jérémie Decouchant, Marcus Völp

Mitigating Front-Running Attacks through Fair and Resilient Transaction Dissemination

Wassim Yahyaoui[1], Joachim Bruneau-Queyreix[2], Jérémie Decouchant[3], Marcus Völp[1]
[1] Univ. of Luxembourg, SnT - CritiX group [3] Delft University of Technology
[2] Univ. Bordeaux, CNRS, Bordeaux-INP, LaBRI UMR5800, F-33400 Talence, France
wassim.yahyaoui@uni.lu, joachim.bruneau-queyreix@u-bordeaux.fr, j.decouchant@tudelft.nl, marcus.voelp@uni.lu

***Abstract*—In modern blockchains, efficient, fair, and fault-tolerant information dissemination is critical for performance and security. Several stages of the transaction lifecycle are affected, from the creation and dissemination of transactions to the dissemination of blocks in the consensus layer. Mempool protocols, such as *LØ*, already address some of modern blockchains' security threats. However, others remain unless fairness is embedded most fundamentally in the dissemination layer used by these protocols to share transactions and blocks and to reconcile mempools. This paper introduces HERMES, a novel dissemination protocol for mempools that leverages robust minimal structures to optimize data propagation, balance load, and ensure fairness despite faults and Byzantine actors. Specifically, we address front-running attacks by randomizing the choice of an overlay structure while forcing nodes to prove adherence to the mempools' dissemination policies and the random choice made. Experimental results show significant performance improvements compared to traditional broadcast protocols for both permissioned and permissionless blockchains.**

***Index Terms*—Dissemination, Fault Tolerance, Robust Topologies, Distributed Systems**

I. INTRODUCTION

Efficient, fair, and accountable information dissemination despite accidental and intentionally malicious node and link failures remains a challenging problem. This is particularly true for both permissioned and permissionless blockchains, due to their increasing complexity and scale. Node failures, leading to inefficient bandwidth utilization and repeated retransmission, threaten these properties, and unfair or unbalanced dissemination give rise to security threats, such as miner exploitable value (MEV) attacks. For example, in 2013, Decker et al. [1] proved that propagation delays are the primary cause of blockchain forks, which makes the blockchain vulnerable to double spending and block-withholding attacks. In 2020, Mao et al. [2] highlighted the importance of block propagation delays for the performance of blockchain networks. Currently employed dissemination strategies, including gossip, give rise to front-running attacks leading to unfair competitive advantages when disseminating transactions. The root cause for these attacks is that some nodes may learn about transactions earlier than others. Nodes can exploit this knowledge if they can control how their transactions are disseminated in relation to when they receive this information. Furthermore, nodes can be systematically overwhelmed by a flood of dissemination requests. In such situations, excessive reliance on retransmission further strains network resources, leading to inefficiencies that degrade bandwidth and increase latency. In some situations, flooding can also open the door to further attacks in the worst scenarios.

LØ [3] already addresses a large class of MEV attacks by introducing accountability in the dissemination layer. In a nutshell, *LØ*'s miners generate, exchange with each other, and record cryptographic commitments of the transactions they know (i.e., their mempools) before receiving transactions from each other. Miners then witness each other's behavior, including how their peers disseminate transactions and bundle them into the blocks they create. Because commitments are further exchanged among miners, *LØ* uncovers reordering attacks with high probability. Unfortunately, since the dissemination paths in *LØ* remain to a large degree in the hands of miners (they can establish additional links on top of the singular unidirectional overlay), they may exploit early knowledge by choosing dissemination paths that favor the miner over other nodes. The result is a front-running attack.

Of course, preventing nodes from learning information early is difficult, if not impossible. For example, malicious nodes may well tap into the protocol implementation and access information when received, rather than when delivered. Moreover, they may establish channels outside the overlay, leveraging links in the physical network, or even establish additional networks. We therefore have to tackle the problem by preventing miners from exploiting this advantage. We do so by preventing them from controlling the order in which related messages are delivered.

We present HERMES, a protocol that extends *LØ* to prevent font-running and ensures *dissemination-fair*, accountable, Byzantine-resilient, and efficient transaction dissemination for blockchain systems. To be practical, HERMES also maintains a low network overhead and a low computational overhead. By *dissemination fairness*, we mean ensuring that no correct node is systematically left behind in the delivery of messages and no adversary can exploit timing advantages to reorder messages or front-run their dissemination. Whether by overloading specific nodes to delay message propagation or by acquiring early

This research was in part funded by the French National Research Agency (ANR) under project number ANR-21-CE25-0021-03 (GenBlock) and by the Luxembourg National Research Fund (FNR), through grant reference C22/IS/17232062 (SpaceVote).

knowledge on the message dissemination, our goal is to prevent adversaries' front-running attempts.

To achieve this, HERMES constructs multiple overlay structures optimized for robust and efficient propagation. HERMES simply modifies the neighbor selection procedure of *LØ* by selecting the children of a node in its precomputed dissemination overlays. Having analyzed four dissemination structures, we found robust-tree topologies to yield minimal latency. We have therefore selected robust trees as our overlay topology, but apply simulated annealing to systematically prune links and reposition nodes to optimize for low latency, and balance the dissemination roles, while preserving $f+1$ connectivity. Nodes participate in all overlays. By balancing dissemination roles we ensure that nodes near the root in one overlay, which naturally receive messages sent through that overlay before further away nodes, assume a more distant role in other overlay structures. To prevent malicious nodes from predicting message dissemination times and gain a persistent advantage over others, which would allow them to successfully front-run transactions, HERMES enforces a randomized selection of the overlay a sender must use, leveraging a novel Threshold Random Seed (TRS) scheme.

We further enforce accountability by requiring nodes to prove that they have relayed messages correctly in the designated overlay. Combined with thorough logging to trace node activity, HERMES prevents front-running attempts from remaining undetected, which allows identifying and excluding deviant nodes from the system.

We combine, for the first time, accountability at the mempool layer with randomized dissemination over optimized overlay structures. More specifically, HERMES makes the following contributions:

- **Low-latency balanced overlays:** We present a method for constructing a set of k *robust-tree* overlays and optimize them via simulated annealing. The latter prunes redundant links and balances node roles across overlays to reduce dissemination latency, avoid bottlenecks at frequently used nodes, and provide fair dissemination. Further, each overlay structure is encoded, signed by a $3f+1$ committee, and disseminated to all nodes.
- **Threshold random seed (TRS) construction:** At the core of HERMES 's front-running resistance is our TRS construction. Our approach leverages reliable broadcast among a committee of $3f+1$ nodes to generates a reliable seed for each dissemination round.
- **Randomized and accountable overlay selection:** Via our TRS construction, each message is verifiably bound to a randomly chosen overlay, preventing senders from picking routes that favor them. This ensures dissemination-fairness, enforces adherence to the assigned path, and eliminates systematic front-running attacks.
- **Robust and accountable dissemination:** Our design ensures resilience with $f+1$ connectivity per overlay, tolerating up to f faulty nodes locally to prevent isolation or censorship. Dissemination accountability is enforced by relying on the signed overlay structures and via message sequencing: nodes verify senders as legitimate predecessors in their overlay, and are required to propagate missing messages before relaying their i^{th} message, preventing selective omission.

Section IV explains HERMES in a nutshell. Section V describes our method for overlay construction, while Section VI explains how HERMES achieves dissemination-fairness and accountability. We analyze the performance of our protocol in Section VIII. We start by establishing the above in permissioned settings, where network topologies are known. In Section VII, we relax these constraints to permissionless settings with changing but generally sufficiently stable topologies.

II. RELATED WORK

Propagation Delay. Decker et al. [1] and Mao et al. [2] highlight the importance of propagation delay for the efficiency and security of blockchains. Perigee [2] dynamically adjusts node connections to optimize propagation times. CougaR [4] employs random link selection to balance network latency and protect against eclipse attacks. BlockP2P [5] accelerates broadcast by geographically clustering. BlockP2P-EP [6] builds on this by using overlay networks to further enhance performance. Mercury [7] leverages virtual coordinates and an early outburst strategy to reduce transaction latency. Our protocol creates a set of latency-optimized robust overlays and randomly selects for each sender and message the ones to use in order to mitigate front-running attacks.

Overlay Topologies. Several works leverage overlay structures in Blockchains. For example, Kauri [8], Byzcoin [9], Motor [10], and Omniledger [11] leverage tree structures to minimize latency and balance load. SplitStream [12] addresses link failures by distributing load across multiple trees. Ramanathan et al. [13] and Toshniwal et al. [14] use hypercubes and show that they outperform fat-tree, torus and scale-free technologies. You et al [15] overcomes the power of two limitations of hypercubes by constructing hyperclique overlays. Gossip protocols [16]–[18] avoid some of the flooding-related overheads. Bitcoin and, at least partially, Ethereum [19] leverage gossip to redistribute learned information up to a certain age. Gossip achieves scalability and robustness to node and link failures, albeit at the costs of guaranteeing delivery only with a certain probability. Our protocol constructs robust trees as overlay structures.

K-Connected Topologies. K-connected topologies [20] with $k > f$ tolerate up to f link and node failures by ensuring nodes remain connected despite such failures. Typical examples include k-regular graphs, k-connected random graphs, k-pasted trees, k-diamonds, multipartite wheels, generalized wheels, Barabási-Albert graphs, incomplete hypercubes, and the k vertex-connected minimum spanning subgraph algorithm introduced in Wen et al. [21]. The latter creates one spanning tree per sender and distributes the load across these trees to enhance robustness and efficiency by ensuring multiple disjoint paths for communication. Our approach limits the overhead of having to maintain multiple overlay structures to a constant few, irrespective of the number of senders.

TABLE I: Comparison of transaction dissemination approaches

Metric	Gossip	Reliable Broadcast	Simple Tree	HERMES (Robust Trees)
Dissemination Mechanism	Direct communication	Randomized gossip	Fixed tree overlays	Optimized robust tree overlays
Latency	Moderate	Low	High	Moderate
Msg Complexity	Moderate	High	Low	Moderate
Dissemination-fairness	✓	✖	✖	✓
Accountability	✖	✓	✖	✓
Load Balanced	✓	✖	✖	✓
Robustness	Moderate	High	Moderate	High
Scalability	High	Low	Moderate	High

Reliable Broadcast and Miner Extractable Value. Broadcast protocols that are robust to arbitrary Byzantine and even intentionally malicious behavior of nodes have been widely investigated [22]. Guerraoui et al. [23] guarantees consistent message delivery despite malicious nodes. Narwhal [24] is a mempool for DAG consensus protocols designed to deliver transactions more efficiently. *LØ* [3] addresses Miner Extractable Value (MEV) attacks [25] in which miners selectively broadcast, reorder, or suppress transactions during block creation to gain unfair advantages [26], [27] (e.g., in auctions). In addition to making miners accountable for their actions [3], [27], [28], MEV mitigation almost exclusively focuses on alleviating their effects [29] (e.g., by redistributing MEV profits [30]). Solutions that prevent such manipulations [31]–[34] require significant changes brought to the consensus layer and provide order fairness properties that differ from our dissemination fairness objective.

Our protocol builds upon and extends *LØ* [3] to also address front-running attacks, while introducing dissemination fairness and improving the efficiency of dissemination. Specifically, we force miners to disseminate transactions through randomly selected overlays to randomize when nodes learn about transactions and to prevent them form coordinating their response to such knowledge.

Table I presents a comparison of dissemination approaches based on key metrics that are critical for designing fault-tolerant and scalable networks. The dissemination mechanism describes the underlying communication strategy: Reliable broadcast [24] uses direct communication between nodes, leading to high latency for large networks because of multi-phase all-to-all message exchanges. Gossip-based dissemination [16]–[18] leverages randomized propagation, achieving moderate latency and high scalability by efficiently distributing messages while avoiding the need for complete network knowledge. Simple tree-based dissemination [8], [9] relies on fixed overlays, resulting in increasing latency for larger networks due to deeper tree structures. Its scalability is moderate because the depth of the tree scales with the number of nodes. HERMES, employing optimized robust trees, achieves low latency and high scalability by minimizing redundant paths and maintaining $f{+}1$-robust structured overlays.

Gossip-based dissemination ensures fairness. HERMES achieves the same by randomizing and optimizing dissemination trees. Intuitively, using a larger number of overlays imply a higher bandwidth consumption, but also a lower average latency and higher dissemination fairness. Reliable broadcast and simple tree-based dissemination may suffer from imbalances, potentially leading to inconsistencies. Reliable broadcast supports accountability, which HERMES achieves through mechanisms for detecting and tracing misbehavior, a feature absent in the other approaches. Reliable broadcast guarantees fault tolerance by delivering to all correct nodes using a totality property. In contrast, gossip-based dissemination and simple tree-based dissemination depend on coverage and tree resilience, respectively, providing only moderate fault tolerance. HERMES $f{+}1$-robustness ensures high fault tolerance by guaranteeing delivery even under adversarial conditions. These attributes position HERMES as a robust, efficient, dissemination-fair, accountable, and scalable solution for large-scale, adversarial networks.

III. System and Threat Model

We consider a network with n nodes, modeled as a labeled graph $G = (V, E)$. Each vertex $v_i \in V$ corresponds to a node, and each edge $e_{i,j} \in E$ represents a link between nodes v_i and v_j. Edges $e_{i,j}$ in the graph are labeled as $lat(e_{i,j})$ to capture communication latencies between nodes (here between v_i and v_j). We shall later maintain for each node $v_i \in G.V$ an accumulated rank (as $rank(v_i)$) relative to their position in the robust-tree overlay structures. Initially 0, $rank(v_i)$ estimates the load of that node depending on the role it plays in the already computed overlay structures. We consider $rank(v_i)$ during subsequent assignments when computing the remaining overlay structures. We assume each node is sufficiently connected so that other nodes can reach it through at least t disjoint paths.

Nodes may fail in arbitrary, potentially malicious (Byzantine) ways. In contrast, links are assumed to drop messages stochastically. We consider a fault-density threat model, defined such that within a distance of D hops from any node, at most $f \leq t$ nodes may fail due to accidents or cyberattacks. For simplicity, we assume the network remains sufficiently connected despite f faulty nodes in proximity. This connectivity assumption can be extended with proof-of-connectivity messages (e.g., as proposed in Pistis [35]). At a broader level, fault density adheres to the classical fault tolerance threshold, allowing up to f_{max} faulty nodes out of n, provided $n \geq 3f_{max} + 1$. However, the fault-density model

adds as a key distinction from the classical model that no node is entirely surrounded by faulty neighbors.

IV. Overview of Hermes

In this paper, we are concerned with the efficient, reliable, and fair dissemination of information (transactions and blocks) through blockchain networks, while ensuring accountability to mitigate at the network layer miner-extractable-value attacks, in particular, front-running. More precisely, we first consider the static network of permissioned blockchains and later, in Section VII, the dynamic networks of permissionless blockchains. In these settings, we seek to develop a dissemination protocol that allows clients to interact with the blockchain through source nodes — the *senders* of transactions and blocks — while tolerating locally, in the proximity of every node, up to f arbitrary faulty (i.e., Byzantine) nodes. In addition, we aim to tolerate globally, in the entire network, up to $f_{max} = \lfloor \frac{n}{3} \rfloor$ faulty nodes. This arbitrary behavior may, in particular, include the manipulation of transaction streams based on knowledge obtained ahead of time of other nodes.

To achieve the above, our protocol Hermes performs operations in two phases (see Figure 1), which we detail in the next sections.

Overlay construction and optimization [offline]: In an offline manner, Hermes proceeds through a *robust-tree overlay construction and optimization* phase, that begins by selecting $f+1$ entry points from the physical network G. Using these entry points as roots, it constructs a robust-tree overlay, by identifying up to $2^d(f+1)$ nodes for the new layer d that are connected to all nodes of layer $d-1$. Once no further layer can be constructed in this way, any missing nodes are then added to ensure robustness with $f+1$-connected leaves. The structure is further optimized using simulated annealing and excess links pruned, while preserving $f+1$-connectivity. This process is repeated k times to construct k optimized overlay structures. During optimization, the roles that each node plays in the already constructed overlays is taken into account to avoid positioning the same node at the same location in the different trees. This avoids overburdening frequently used nodes and prevents systematically favoring individual nodes across multiple overlay structures. Intuitively, using a larger k value implies a higher bandwidth consumption, but also a lower average latency and higher dissemination fairness. Once finalized, the constructed overlay structure is communicated to each node in the network. Node memorize from each overlay the entry points as well as direct predecessors and successors.

Dissemination [online]: At runtime, the dissemination of messages proceeds in three steps:

1) ***Randomized structure selection:*** The sender starts by reaching out to a committee of $3f+1$ nodes to ask which overlay structure it must use for transmitting its next message m. To achieve this, the sender computes a hash $H(m)$ and obtains from the committee a $2f+1$-threshold signature for the current transmission round i. Using this signature, the sender selects the overlay structure and forwards m along with the certificate, *i.e.*, the signature, to the $f+1$ entry points of this overlay structure. It does so through $f+1$ disjoint paths, unless of course the sender is connected directly to the overlay's entry points.
2) ***Dissemination and accountability:*** Entry points and intermediate nodes validate the source of the message by certifying which node sent it. They then forward the message to all successors in the selected overlay structure. These nodes also verify whether the received message follows a valid path within the overlay to detect any misbehavior. Nodes found to be faulty are excluded from further participation in the dissemination process.
3) ***Acknowledgment of delivery (optional):*** If higher-level protocols require receipt confirmations, acknowledgment messages are sent back to the sender through the same overlay structure used for dissemination.

In each overlay structure, any node v_i receives messages from at least $f+1$ nodes, ensuring that dissemination cannot be delayed by the up to f faulty nodes near v_i. Furthermore, since the overlay structure is randomly chosen by the committee (and not the sender), there is no opportunity for the sender to manipulate or choose a favorable overlay. This mechanism ensures dissemination-fairness and accountability during the transmission process.

V. Overlay Structures and their Optimization

This section motivates our selection of robust trees as overlay structure and details how we optimize them for latency.

A. Overlay Structures

Introducing overlays over the graph G, we aim to simultaneously achieve efficient message propagation, role-balancing, and resilience to link and node faults. By role-balancing, we mean equally distributing dissemination responsibilities across multiple overlay structures, ensuring that no node is consistently favored against others when receiving a message, and maintaining fair participation in message propagation. To that end, we compared $f+1$-connected chordal rings, hypercube, robust trees and randomly-generated overlay graphs by examining the average latency and load translated as the standard deviation of sent transactions with a constant number of nodes. The detailed setup is shown in Section VIII-A.

Figure 2 illustrates the overlay structures under investigation, along with their respective latencies and the standard deviation of their load distribution. Robust trees, when considered in isolation, exhibit higher load imbalances but achieve significantly lower latency, when compared to the other overlay structures. Our approach involves disseminating messages randomly across k overlay structures, the load and role imbalances in one structure can therefore be compensated by the others. We have therefore selected robust trees as base overlay structure before pruning with simulated annealing.

Robust trees are hierarchical in nature. Nodes at a given layer (depth d) are connected to all nodes in the layer above them (depth $d+1$). However, since not all nodes in G may conform to this strict pattern, the resulting robust-tree overlay, G^{RT}, will generally be incomplete. To address this,

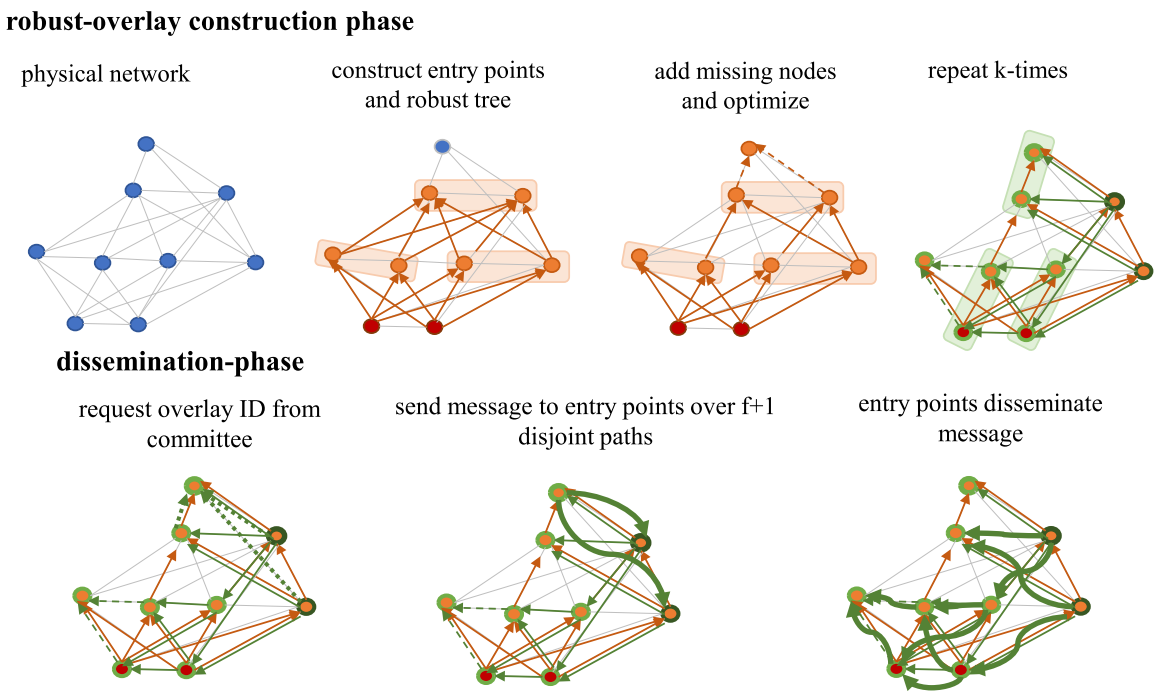


Fig. 1: Overview of HERMES's two phases. Construction phase: k robust-tree overlays are built offline and recomputed after major network changes. This involves selecting entry points, constructing robust-trees, integrating missing nodes, and optimizing for efficiency and robustness. Dissemination phase: A sender queries a committee to determine the overlay for message m based on network state. It then forwards m to entry points via $f+1$ disjoint paths for fault tolerance. The entry points propagate m across the robust-tree overlay, ensuring reliable dissemination.

we augment the overlay structure by adding any remaining nodes from G to G^{RT}, while ensuring that each such node is connected via $f+1$ edges either to existing nodes in G^{RT} or directly within G. This process continues until all nodes in G are integrated into G^{RT}.

While robust-tree overlays are highly connected, their connectivity often exceeds what is strictly necessary for reliable dissemination. Therefore, an additional optimization step is performed to prune redundant links. The result is a low-latency, $f+1$-connected subset of the robust tree. Algorithm 1 outlines the pseudocode for constructing k robust trees, $G_1^{RT}, \ldots, G_k^{RT}$, from the physical network G: Starting with an empty set of graphs, the algorithm selects $f+1$ nodes with the lowest accumulated rank and minimal latency to their neighbors as the entry points and assigns them to rank $d = 0$ (Lines 3–6). The tree, now initiated with $f+1$ roots, is extended iteratively. For each rank, nodes are added that are directly connected to all nodes in the previous rank (depth $d-1$) and have not yet been included in G_l^{RT} (Lines 8–15). Nodes are selected based on their minimal accumulated rank to balance the roles they will obtain and based on their connection latency to neighboring nodes. At each rank, the goal is to double the number of included nodes, such that at depth d there are $2^d(f+1)$ nodes. If doubling is no longer feasible, remaining nodes are added without enforcing the hierarchical tree structure, ensuring they connect to at least $f+1$ edges of existing nodes in G_l^{RT} (Lines 17–21). Before passing the constructed tree to the optimization step (Line 25) and moving to the next tree, the accumulated rank of nodes is updated to reflect their distance from the root in the current tree (Lines 22–24). This ensures a balanced distribution of nodes across the k robust trees. The described methodology ensures the creation of k robust, low-latency overlay structures optimized for scalability, fault tolerance, and efficiency.

B. Overlay Latency Minimization and Role-Balancing

The goal of this step is to remove redundant network links while maintaining $f+1$-connectivity, and to optimize the overlay structures for both latency and role distribution. Indeed, a node positioned near the roots of multiple tree overlays would be advantaged, as it would tend to receive messages earlier on average compared to its neighboring nodes. Additionally, in a tree structure, nodes closer to the root typically bear higher loads than those near the leaves. To address this imbalance, we introduce a penalty system based on the accumulated ranks of nodes across previously constructed overlay structures. The accumulated rank reflects how often a node has been placed near the leaves in past overlays. Nodes with higher accumulated ranks are more likely to have been less loaded, making them preferable candidates for near-root positions in the current overlay structure.

Latency Optimization. In this work, we employ simulated annealing as optimization algorithm, although our approach

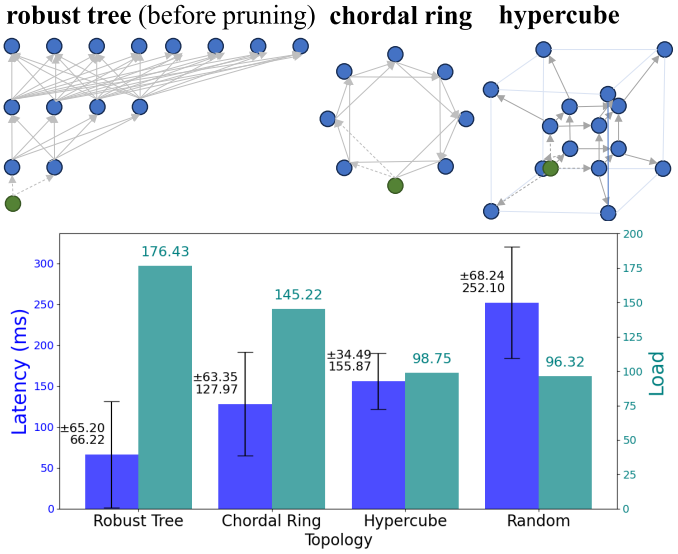


Fig. 2: Dissemination latency and message load variance per node over a single $f{+}1$-connected instance of the considered overlays: robust trees (pre-pruning), chordal rings, hypercubes, and a random overlay ensuring at least $f{+}1$ links per node.

Algorithm 1 CreateRobustTree

// Initialization
initialize empty graphs G_l^{RT} ($l \in \{1, \ldots, k\}$)
for each graph G_l^{RT} **do**
  add $f{+}1$ entry point nodes $v_1^0, \ldots v_{f+1}^0 \in G.V$ (nodes of rank 0)
    such that v_i^0 is among the $f{+}1$ nodes with lowest accumulated
    rank and connecting with lowest latency to its neighbors.
  // Construct robust tree
  for each rank d starting from 1 **do**
    select the up to $2^d(f{+}1)$ nodes v_i^d in $G.V \setminus G_l^{RT}.V$ with
      minimal acummulated rank and connection latency that are
      connected to all nodes v_j^{d-1} of the previous rank
    add selected nodes v_i^d and their edges $e_{i,j} \in G.E$ to nodes
      v_j^{d-1} of the previous rank to the robust tree G_l^{RT}
    repeat until no further nodes fit this pattern
  end for
  // Add missing nodes
  for each node v_i remaining in $G.V \setminus G_l^{RT}.V$ **do**
    add v_i with $f{+}1$ edges $e_{i,j} \in G.E$ to G_l^{RT}, provided
      $v_j^d \in G_l^{RT}.V$ at some rank d.
    repeat until all nodes are added
  end for
  for each node $v_i^d \in G_l^{RT}.V$ **do**
    update rank(v_i^d) = rank(v_i^d) + d
  end for
  Optimize G_l^{RT}
end for
return $G_1^{RT}, \ldots G^{RT_k}$

is agnostic to the specific optimization algorithm employed. Simulated annealing iteratively improves the overlay structure by exploring configurations that minimize latency and complexity while factoring in rank penalties. The algorithm starts with the robust tree G_l^{RT}, which initially relies on a heuristic for node inclusion, and incrementally explores modifications to improve the overlay. These modifications may include adjusting node connections or rerouting specific links. At each

Algorithm 2 Simulated Annealing for Multiple Overlay Optimization

Input: Network graph G and overlay $O_{\text{current}} = G_l^{RT}$, initial temperature T_{initial}, minimum temperature T_{min}, cooling rate α
Output: Optimized overlays O_{best}
// Step 1: Initialize Overlay Optimization
$O_{\text{best}} \leftarrow O_{\text{current}}$
$T \leftarrow T_{\text{initial}}$
// Step 2: Perform Simulated Annealing
while $T > T_{\text{min}}$ **do**
  // Generate neighbor overlay:
  $O_{\text{neighbor}} \leftarrow$ **GenerateNeighbor**(O_{current}, f)
  // Evaluate objective values:
  $O_{\text{neighbor}} \leftarrow$ **ObjectiveFunction**(O_{neighbor})
  $O_{\text{current}} \leftarrow$ **ObjectiveFunction**(O_{current})
  if $O_{\text{neighbor}} < O_{\text{current}}$
      or $e^{-(O_{\text{neighbor}} - O_{\text{current}})/T} >$ rand(0, 1) **then**
    Accept neighbor overlay: $O_{\text{current}} \leftarrow O_{\text{neighbor}}$
    if $o_{\text{neighbor}} <$ **ObjectiveFunction**(O_{best}) **then**
      Update best overlay: $O_{\text{best}} \leftarrow O_{\text{neighbor}}$
    end if
  end if
  Decrease temperature: $T \leftarrow T \times \alpha$
end while
Return O_{best}

iteration, the algorithm evaluates whether a proposed change improves the overall objective function. Changes are accepted probabilistically, based on the current system temperature, *i.e.*, how far it is from the overall objective, which decreases over time. This decreasing temperature enables broad exploration in the early stages and gradual convergence toward an optimal or near-optimal solution. The pseudocode for the simulated annealing algorithm is provided in Algorithm 2.

Balancing roles using rank penalty. In addition to optimizing for latency, we incorporate a rank penalty to discourage the repeated selection of nodes that have already been placed as low-rank nodes (near the root) in other overlay structures. Nodes closer to the root experience earlier message delivery and higher loads. It is therefore preferable to position them closer to the leaves in subsequent overlay structures. To achieve this, we define the objective function as:

$$\begin{aligned} objective\_value = \; & num\_edges + avg\_latency \\ & + connectivity\_penalty \\ & + path\_penalty + rank\_penalty \end{aligned} \tag{1}$$

where $num\_edges = |E_l^{RT}|$ (from $G_l^{RT} = (V, E_l^{RT}) \subseteq G$), $avg\_latency$ the sum of all shorted path latencies in G_l^{RT} divided by $|V| = n$. We penalize nodes with less than $f{+}1$ successors ($connectivity\_penalty$), nodes that are not reachable from the source ($path\_penalty$) and discourage the use of nodes with low accumulated rank ($rank\_penalty$).

Algorithm 3 illustrates the process for generating a neighboring solution from the current robust-tree overlay configuration. The function begins by randomly adding or removing edges between nodes in consecutive layers of the overlay. This random adjustment enables exploration of alternative configurations. To ensure resilience, the algorithm then verifies that each node (except leaves) retains at least $f+1$ outgoing connections. Additionally, the function considers the depth and accumulated rank of nodes, reassigning connections where necessary to balance the role and minimize the rank penalty. By simultaneously optimizing connectivity and latency while reducing the rank penalty, the function generates valid neighbor solutions that preserve robustness and enable a dissemination-fair framework. In theory, highly heterogeneous latency distributions might prevent our optimization procedure from obtaining a high level of dissemination fairness among nodes. However, this effect would be attenuated as the system size increases, and could also be specifically mitigated by modifying our simulated annealing optimization procedure.

Algorithm 3 GenerateNeighbor

Input: Current robust-tree overlay G_l^{RT}, fault tolerance f, accumulated rank $rank(v)$
Output: Neighbor robust-tree overlay $G_{neighbor,l}^{RT}$
function GENERATENEIGHBOR($G_l^{RT}, f, rank(v)$)
    Create a copy of G_l^{RT} named $G_{neighbor,l}^{RT}$
    $layers \leftarrow$ list of node layers in G_l^{RT}
    // Step 1: Randomly Add or Remove an Edge
    if random number < 0.5
        and $G_{neighbor,l}^{RT}$ has more than 0 edges **then**
        Choose a random edge between consecutive layers and remove it
    else
        Choose a random edge between consecutive layers and add it
    end if
    // Step 2: Ensure $f+1$-Connectivity
    for each layer in $layers$ **do**
        if layer $< \max(layer\_keys)$ **then**
            for each node in $layers[layer]$ **do**
                while node has $< f+1$ successors **do**
                    Add an edge from node to a random node in the next layer, minimizing: $latency(node, next\_node)$
                end while
            end for
        end if
    end for
    // Step 3: Adjust for Rank Penalty
    for each node in $G_{neighbor,l}^{RT}$ **do**
        if depth of node i is too close to root and has extra edges **then**
            Attempt to reassign some connections to nodes with higher accumulated rank $rank(v)$ and farther depth
        end if
    end for
    // Step 4: Validate and Return Optimized Neighbor
    if ObjectiveFunction($G_{neighbor,l}^{RT}, rank$) $<$ ObjectiveFunction($G_l^{RT}, rank$) **then**
        return $G_{neighbor,l}^{RT}$
    else
        return G_l^{RT} ▷ Discard if no improvement
    end if
end function

VI. Random, Accountable and Fair Dissemination

To prevent successful front-running attacks, Hermes employs distributed and resilient mechanisms that enable fair, randomized and accountable overlay selection and message dissemination. Together, these mechanisms strive towards dissemination-fairness, ensuring that no correct node is systematically left behind in the delivery of messages and no adversary can exploit timing advantages to reorder messages or front-run their dissemination. Hermes's accountability ensures that any deviations from the prescribed dissemination path can be detected and flagged as protocol violations. In a nutshell, Hermes's dissemination follows three main steps: (i) a committee of $3f+1$ nodes generates a TRS for each message m, (ii) the TRS is used to select a random overlay structure for message dissemination, (iii) nodes verify the legitimacy of the message they receive and the senders of this message according to the known overlay structures before relaying it to their successors.

A. Threshold Random Seed (TRS) Generation

To achieve a fair and unbiased overlay selection, Hermes generates a threshold random seed per message, which is then used to determine uniformly at random the overlay structure to be used for dissemination. This process builds on a reliable broadcast algorithm [36], ensuring consistency and preventing any node from unilaterally affecting the random seed used later in the protocol.

When a node intends to broadcast a message m, it first sends the hash $H(m)$ along with a monotonically increasing sequence number i to a committee of $3f+1$ nodes. The sequence number i ensures that messages are processed in ascending order without duplication. Committee members echo $(i, H(m))$ to one another, and each member dispatches a "Ready" message upon receiving either $2f+1$ "Echo" messages or $f+1$ "Ready" messages. Upon reception, each member creates a partial threshold signature and sends it back to the source node. The source collects $2f+1$ partial signatures, combines them, and obtains a final threshold signature $\varphi(i, H(m))$ that effectively acts as a random seed for the transaction m in round i. Its distributed and f-tolerant generation process guarantees that no single adversarial node (or small group) can skew the seed to benefit front-running or any other manipulation. Moreover, because each message

Algorithm 4 Threshold Random Seed (TRS) Generation via Reliable Broadcast

Input: Committee of $3f+1$ nodes, sequence number i, message hash $H(m)$, source node N_{source}
Output: Threshold Random Seed $\varphi(i, H(m))$
// Step 1: Broadcast by Source Node
N_{source} sends $(i, H(m))$ to all committee members
// Step 2: Reliable Broadcast Among Committee Members
for each committee member n_j **do**
    On receiving $(i, H(m))$:
    - Send an "Echo" message containing $(i, H(m))$ to all other committee members.

    On receiving $2f+1$ "Echo" messages or $f+1$ "Ready" messages:
    - Send a "Ready" message containing $(i, H(m))$ to all other committee members.

    On receiving $2f+1$ "Ready" messages:
    - Deliver the message $(i, H(m))$.

end for
// Step 3: Partial Signature Generation and Collection
for each committee member n_j who delivered $(i, H(m))$ **do**
    Compute partial signature: $\sigma_j = \text{Sign}(n_j, (i, H(m)))$
    Send σ_j to N_{source}
end for
// Step 4: Combine Partial Signatures at Source Node
Source node N_{source} collects all received σ_j into partial_signatures
if $|\text{partial\_signatures}| \geq 2f+1$ **then**
    Combine partial signatures: $\varphi(i, H(m)) = \text{CombineSignatures}(\text{partial\_signatures})$
    Return $\varphi(i, H(m))$
end if

is bound to a unique sequence number, malicious committee members cannot successfully modify it to front run, reorder or skip messages without detection during seed generation.

B. Randomized and Verifiable Overlay Selection

Once the TRS $\varphi(i, H(m))$ is generated, the source node computes *overlay* = *seed mod* k, where $seed = \varphi(i, H(m))$, and k is the total number of available overlays. This step ensures a randomized but verifiable selection of overlay for every message, thereby hindering adversaries from systematically front-running. Consequently, all nodes can independently verify the threshold signature and the selected overlay, preventing dishonest senders from unilaterally choosing favorable routes. Additionally, no node can predict or predetermine which overlay will be used for m's propagation, ensuring an even playing field for all correct nodes.

C. Overlay Encoding and Accountable Dissemination

Alongside randomized overlay selection, HERMES employs *accountable* dissemination to guarantee that messages follow the correct path and that nodes can detect and expose any front-running attempts or deviations.

Before dissemination begins (or when overlays are re-optimized for a permissionless setting), HERMES distributes to every node a signed description of the k *robust-tree* overlays used. Algorithm 5 provides the details of this *Robust Tree Encoding*. Each node learns its predecessors and successors within each overlay, as well as the designated entry points. These entry points enable nodes to forward or receive messages in accordance with the chosen overlay.

A node receiving a message m directly verifies the message's and sender's legitimacy by checking (i) the threshold signature $\varphi(i, H(m))$, (ii) the sequence number i, and (iii) that the sender is a valid predecessor in the overlay. If any check fails, the recipient flags the message as a protocol violation. This auditing step holds each node accountable for adhering to the assigned route.

Through the TRS generation process, the committee binds the message m that a node wishes to send as its i^{th} message to the random seed they generate, ensuring continuity of sequence numbers. This mechanism prevents dissemination omissions by requiring a node to first transmit any missing messages for skipped sequence numbers before sending the current message as its i^{th} message. Nodes cannot omit or reorder messages without detection, since skipping a sequence or forging a threshold signature is easily caught by other nodes. By enforcing this requirement, the protocol ensures compliance and prevents attempts to bypass the sequence ordering and front run. This ensures tamper-proof evidence of each transmission path, enabling the system to isolate and exclude deviant nodes from future communications.

Because decisions about routing and overlay choice depend on threshold signatures rather than a centralized authority, HERMES remains decentralized. Nodes themselves monitor compliance through local checks and cross-verification, ensuring dissemination-fairness and resilience against both targeted node overloads and front-running attempts.

VII. DISCUSSION

A. Fault Density Assumption

The fault-density assumption is fundamental for the use of robust overlays in HERMES. It ensures that within a radius of D hops, no node is surrounded by more than f faulty neighbors. This assumption is particularly practical for permissioned systems, where the network topology can be controlled or monitored, at least partially. However, in permissionless environments, such as public blockchains, this assumption may not always hold due to factors like malicious node clustering or dynamic network churn.

When fault density is violated, HERMES may encounter degraded performance or incomplete message dissemination. Specifically, malicious nodes clustered around a target node may try to isolate and prevent it from reliably receiving or forwarding transactions. Similarly, fault-density violations can lead to disproportionately high workloads at healthy nodes, reducing efficiency and dissemination fairness.

Algorithm 5 Robust Tree Encoding

Input: Committee of $3f+1$ nodes, source node N_{source}
Output: Encoded robust tree $\mathcal{T}_{\text{encoded}}$ and threshold signature $\sigma_{\mathcal{T}}$
// Step 1: Robust Tree Computation
for each committee member n_j **in parallel do**
    Compute robust tree $\mathcal{T}_j$
    Encode robust tree: $\mathcal{T}_{\text{encoded}} = \text{EncodeTree}(\mathcal{T}_j)$
    Generate partial signature: $\sigma_j = \text{Sign}(n_j, \mathcal{T}_{\text{encoded}})$
    Send $\mathcal{T}_{\text{encoded}}$ and σ_j to N_{source}
end for
// Step 2: Threshold Signature Generation at Source Node
Source node N_{source} collects received σ_j into partial_signatures
if $|\text{partial\_signatures}| \geq 2f+1$ **then**
    Combine partial signatures: $\sigma_{\mathcal{T}} = \text{CombineSignatures}(\text{partial\_signatures})$
    Return $\mathcal{T}_{\text{encoded}}, \sigma_{\mathcal{T}}$
end if

To address these scenarios, we incorporate gossip as robust fallback mechanism, activated after a delay T to give HERMES's overlays enough time to disseminate, and leverage the reconciliation mechanism of mempools built on top of HERMES. Gossip operates in the background to ensure eventual consistency and delivery of transactions. If a node fails to receive a certain messages due to faults or adversarial interference, the mempool's reconciliation protocol identifies and retransmits the missing transactions through gossip to ensure reliable dissemination despite initial disruptions, even in scenarios where fault-density assumptions cease to hold.

B. Dynamic Network Topologies

HERMES can be directly integrated into blockchains that rely on epoch-based memberships (e.g., Ethereum [37]): overlays would simply have to be recomputed at the beginning of an epoch. Extending HERMES to other permissionless blockchains with dynamic network topologies is achieved by integrating a Byzantine-fault-tolerant gossip-based peer-sampling mechanism, such as SecureCyclon [38]. This mechanism allows each node to maintain a local, partial view of the network membership, which is periodically refreshed by exchanging information with other nodes. This dynamic update process ensures that even in environments with high churn or malicious nodes attempting to over-represent themselves, every node remains $f+1$ connected and has an up-to-date understanding of the network topology.

When nodes join or leave the network, HERMES adapts by updating the overlay structures. Newly joined nodes are integrated into all overlay structures with at least $f+1$ connectivity, maintaining the system's fault tolerance. Departing nodes are removed from the overlays while ensuring the necessary level of connectivity for the remaining nodes.

These dynamic updates inevitably cause HERMES to diverge from its optimized structure. To address this, HERMES periodically re-executes the steps of the offline overlay construction phase to create new optimized overlays that reflect the current topology during the next epoch. This reconstruction process occurs in parallel and in the background, ensuring that the existing overlay remains operational until the updated structure is disseminated and confirmed. The reconstruction process can either be offloaded to a trusted data center operated by the blockchain provider or executed within the blockchain network itself in a distributed manner. In the latter case, HERMES mitigates attempts by faulty nodes to manipulate the generated structure by validating each step of the construction with $f+1$ neighboring nodes. Additionally, the committee ensures deterministic construction by generating a random seed for use in the pseudo-random optimization steps performed by simulated annealing.

A special case arises when an entry-point node in an overlay structure leaves the network. In this scenario, a new entry point must be elected and disseminated to the network. During this transition, the overlay structure may temporarily become unusable if the remaining f entry-point nodes are faulty. However, the global view maintained through the gossip mechanism allows HERMES to continue efficiently disseminating transactions and blocks across the network. This ensures low latency, fairness, and resilience against adversarial attacks, such as front-running.

VIII. Performance Evaluation

We address the following questions:

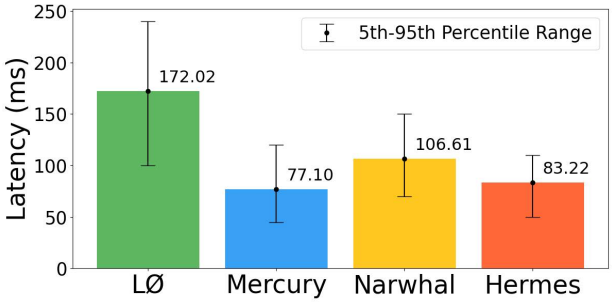
- How does HERMES compare to the state-of-the-art protocols, *LØ*, Mercury, and Narwhal, in terms of latency under the same network conditions?
- What is the bandwidth overhead associated with each protocol, and how does it impact transaction propagation efficiency?
- How does each protocol perform in the presence of Byzantine nodes, and which protocol demonstrates the highest level of robustness and fairness?

We conducted simulations to compare the protocols in controlled environments, evaluating key metrics such as latency, and bandwidth usage. Additionally, we analyzed their performance under various attack scenarios, including front-running and censorship, to assess their resilience and ability to maintain fairness and robustness. This thorough evaluation allows us to identify the strengths and weaknesses of each protocol across diverse network conditions.

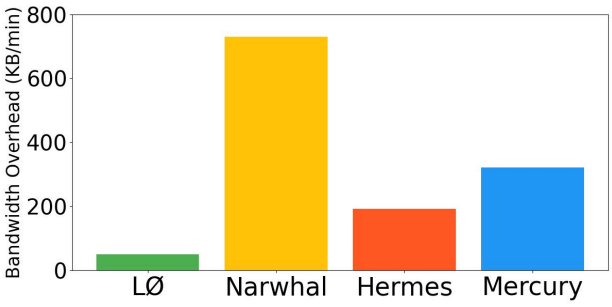
A. Experimental Setup

We evaluated our protocol and the baselines on a server equipped with Intel Xeon Gold 6240 CPUs (36 physical cores, 72 logical CPUs) running at 2.6 GHz (boosting up to 3.9 GHz) and 128 GiB of RAM, interconnected via Gigabit Ethernet. The experiment parameters included $N = 10,000$ nodes, a transaction size of 250 bytes, repeating each experiment 10 times and reporting average results.

All protocols were thoroughly developed entirely in Python, based on a single, common P2P simulation framework. The



(a) Average Latency



(b) Bandwidth overhead

Fig. 3: Achieved performance and comparison to *LØ*, Mercury and Narwhal.

Mercury implementation used was the complete version, including the early outburst and clustering features, with parameters set to: $K = 8$ clusters, $D_{cluster} = 4$ inner-cluster peers, and $D_{max} = 8$ maximum peers. Both Narwhal and *LØ* use connected topology. In HERMES we use optimized robust trees with $f = 1$ and tree depth $D = 14$ following the number of nodes used for the experiments N, and considering 10 different overlays. Under those settings, computing the overlays took less than 15 s.

To simulate realistic network conditions, we used a latency dataset derived from multiple sources, including CAIDA [39], RIPE Atlas [40], and major cloud providers such as AWS [41] and Azure [42], alongside empirical blockchain latency data (Ethereum [43]). Latencies were modeled using inverse gamma distributions for intra-regional communication and normal distributions for inter-regional latencies, covering key regions such as New York, Singapore, Frankfurt, Sydney, Tokyo, Ireland, Ohio, California, and London. Specifically:

- **Intra-regional communication** (e.g., nodes within Frankfurt) followed an inverse gamma distribution with shape $\alpha = 2.5$ and scale $\beta = 14$, resulting in a mean latency of 7 ms and a variance of 2 ms.
- **Inter-regional communication** (e.g., Frankfurt to New York) followed a normal distribution with mean $\mu = 90$ ms and variance $\sigma^2 = 20$ ms.

B. Comparative Analysis

Our comparative analysis examines the performance of four baselines: HERMES (our protocol), Narwhal, *LØ*, and Mercury. Narwhal [24] is a mempool protocol designed for enhancing consensus performance and relies on standard broadcasting. *LØ* [3] is a lightweight mempool reconciliation protocol that optimizes transaction propagation with minimal bandwidth. Mercury [7], a clustering-based propagation protocol, aims to optimize transaction dissemination latency. We evaluated these protocols in terms of their average latency (measuring transaction propagation times), latency distribution (assessing consistency of propagation), scalability (evaluating how well the protocol handles larger networks), and overhead (measuring computational and communication costs). We also conducted a security analysis to provide insights into each protocol's strengths and weaknesses.

C. Transaction Latency

We evaluate the average transaction dissemination latency and its variability, as shown in Figure 3a. Average latency measures the typical delay, while latency variability is represented by the range between the 5th and 95th percentiles that captures the spread of latency values. Together, these metrics provide a comprehensive view of each protocol's efficiency and consistency in transaction propagation. Mercury achieves the lowest average latency of 77.10 ms and a relatively narrow latency range thanks to its clustering-based dissemination approach. However, slight variability arises from inter-cluster communication overhead. HERMES achieves a good latency of 83.22 ms demonstrating slightly higher latency than Mercury. However, HERMES achieves lower variability by generating dissemination paths that are carefully optimized for balanced load and minimized latency variability using simulated annealing. The additional overhead from TRS and tree encoding slightly increases the average latency but does not contribute significantly to variability. Narwhal, with an average latency of 106.61 ms, demonstrates moderate transaction speeds, but its broader latency range indicates less consistency in propagation times due to coordination dependencies between nodes. *LØ* exhibits the highest latency at 172.02 ms, with the widest latency range, reflecting the inefficiency inherent in traditional gossip mechanisms, where propagation relies on repeated message exchanges among nodes.

D. Bandwidth Consumption

Figure 3b shows the differences in bandwidth overhead among *LØ*, Mercury, Narwhal, and HERMES, highlighting how each protocol balances efficiency and performance. For simplicity, we collect results for $N = 200$ nodes. *LØ* demonstrates the lowest bandwidth overhead, with only 50 KB/min, attributed to its highly efficient mempool reconciliation mechanism, making it an excellent choice for resource-constrained environments. HERMES, designed as an enhanced version of *LØ*, introduces mechanisms such as the Threshold Random Seed (TRS) and a compact tree encoding, which prioritize fairness and latency optimization. The measured bandwidth overhead for HERMES is 192 KB/min, assuming frequent dissemination of the encoded robust tree and its

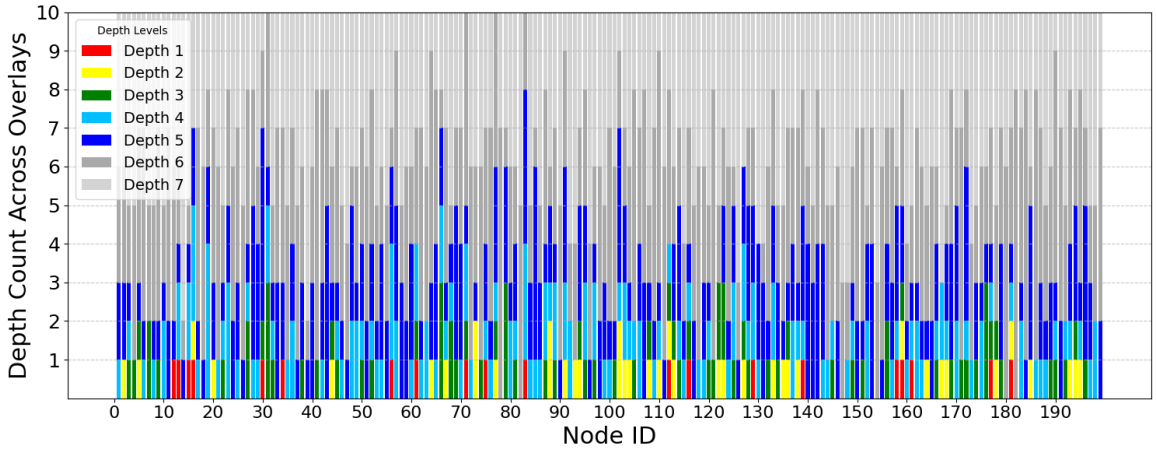
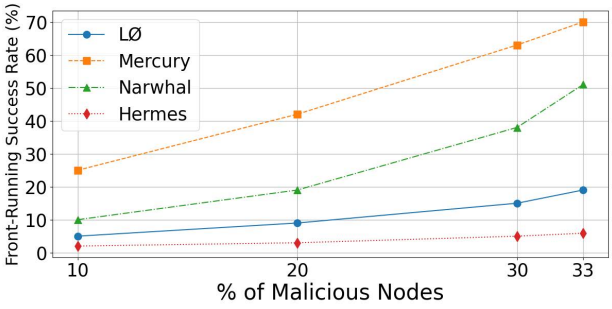
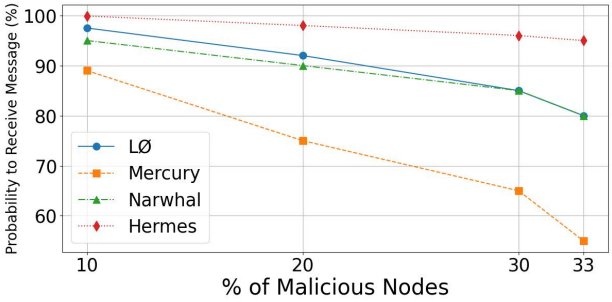


Fig. 4: Distribution of roles for each of the 200 nodes evaluated across all 10 generated overlay structures. Shown are the number of overlay structures in which each node held a given rank. For example, node 30 is the entry point in one overlay, ranked 3 in another, ranked 5 in five more overlays, ranked 6 in two, and ranked 7 in one.



(a) Front-running success rate depending on the proportion (%) of malicious nodes



(b) Probability for a message to be received.

Fig. 5: Front-running resistance and robustness

signature as if a view change is required for every transaction. However, in practice, the encoded tree and signature only need to be transmitted during the initial network setup or upon a view change. Adjusting for this, the actual HERMES overhead reduces by approximately 30 KB/min, resulting in a bandwidth usage closer to 162 KB/min, striking an efficient balance between performance and resource usage.

Mercury, focusing on propagation latency through its clustering-based approach and early outburst strategy, incurs an overhead of 322 KB/min. Mercury adds overhead through its Virtual Coordinate System (VCS) for topology maintenance and clustering-based propagation strategy, which involves metadata exchange and dynamic relaying to optimize latency. Narwhal exhibits the highest bandwidth overhead at 730 KB/min, primarily due to its reliance on an intensive broadcast structure that requires collecting batch approvals from two-thirds of the network.

Several directions exist to further optimize HERMES's performance. First, HERMES could manipulate batches of transactions. Then, an $(k{+}1, f{+}1{+}k)$ erasure coding scheme could divide a message into $f{+}1{+}k$ chunks, each one being disseminated over one of $f{+}1{+}k$ disjoint paths. A node would then receive at least $k{+}1$ chunks and recover the original batch of transactions. Depending on the blockchain, if specific roles are attributed to a subset of the nodes, e.g., validator nodes, then HERMES could be further optimized to minimize the transaction dissemination latency for these nodes.

E. Role Distribution

Figure 4 illustrates the distribution of roles among 200 nodes across 10 overlays. Each bar shows how often a node was assigned a given rank (depth). Rank 1 hereby corresponds to being an entry point. Disseminators start by transmitting through $f{+}1$ such entry points. The lengths of bars along the y-axis indicates how often each node had a specific rank in the 10 overlays. In the robust tree structure, nodes closer to the entry points bear higher dissemination loads, while nodes near the leaves handle fewer messages. HERMES balances this load by varying node roles across overlays. Since each robust tree has $f{+}1$ entry points, $10 \times (f{+}1)$ nodes are marked as such (i.e., in red; depth = 1). Intermediate nodes make up the majority of the network. They are responsible for relaying messages from the entry points towards the leaves. The figure shows that ranks are widely distributed among nodes, ensuring that no specific node is consistently favored on the one side

and also that no node is burdened with heavy forwarding responsibilities. Thus role rotation helps maintain balanced dissemination loads across the network. The dissemination process ends at the leaf nodes. They receive messages but do not forward them. The figure demonstrates that HERMES effectively achieves a balanced role distribution, with nodes taking different responsibilities across overlays, which ensures dissemination fairness and helps prevent front-running.

F. Front-running Resistance.

In the context of front-running resistance, *dissemination fairness* is connected to how often malicious nodes are successful in manipulating the order with which transactions are received (front-running attacks). In this experiment, we evaluate the front-running resistance under different levels of malicious node activity. We simulate a network with varying percentages of malicious nodes (from 10% to 33%) that attempt to front-run transactions by observing and manipulating the mempool. A lower success rate indicates better fairness and resistance to manipulation. The first malicious node to observe a victim transaction attempts to front-run it by generating and disseminating an adversarial transaction immediately in the network. An attack succeeds if the adversarial transaction appears before the victim transaction in the blockchain. Note that under this definition we consider a front-running attack successful even if the adversarial transaction does not appear right before the victim transaction.

Figure 5a shows that HERMES achieves the strongest protection against front-running attacks, with success rates as low as 2% at 10% malicious nodes and only 5.9% at 33%. This exceptional performance stems from HERMES' randomized dissemination mechanism, where the Threshold Random Seed (TRS) ensures unbiased overlay selection, making it exceedingly difficult for malicious nodes to predict and manipulate transaction propagation paths. Furthermore, HERMES' Robust Tree Encoding adds an additional layer of security by encoding dissemination structures in a verifiable manner, significantly mitigating manipulation opportunities. *LØ* provides notable front-running resistance as well, with success rates ranging from 5% at 10% malicious nodes to 19% at 33%. Its mempool reconciliation mechanism ensures consistency across nodes, limiting adversaries' ability to exploit inconsistencies in transaction visibility. However, due to fewer mechanisms for randomized propagation compared to HERMES, *LØ* becomes progressively more vulnerable as the number of malicious nodes increases. Narwhal demonstrates moderate resistance to front-running, with success rates increasing from 10% at 10% malicious nodes to 51% at 33%. It ensures the eventual propagation of transactions but lacks advanced defenses. This makes Narwhal more susceptible to manipulation as the percentage of malicious nodes increases, particularly under high adversarial activity. Mercury exhibits the weakest front-running resistance among the protocols, with success rates escalating from 25% at 10% malicious nodes to 70% at 33%. While its clustering-based dissemination strategy offers efficiency, it makes the protocol vulnerable to manipulation by malicious nodes that can target critical nodes in the clusters to observe and reorder transactions. This centralized reliance on cluster leaders amplifies its susceptibility to front-running attacks under high adversarial conditions.

G. Robustness.

Robustness measures the likelihood that a message is successfully propagated to all honest nodes, even with varying percentages of Byzantine nodes disrupting propagation in a network of 10,000 nodes. Figure 5b shows that HERMES achieves the highest robustness across all scenarios, with almost 99.9% success rate at 10% malicious nodes and maintaining 95% even at 33%. This exceptional performance is attributed to its K-vertex-connected and robust overlay, which provides multiple redundant paths for message dissemination. *LØ* follows with 97.5% robustness at 10% malicious nodes, though its performance drops to 80% as adversarial influence increases. Its mempool reconciliation mechanism ensures consistency and decent robustness; however, its simpler dissemination structure with fewer redundant paths makes it less resilient under higher Byzantine node ratios. Narwhal demonstrates similar robustness to *LØ*, achieving 95% at 10% malicious nodes and decreasing to 79% at 33%. Its reliable broadcast structure facilitates eventual message propagation but can experience delays or disruptions in the presence of adversarial behavior, reducing its overall consistency. Mercury has the lowest robustness, starting at 89% at 10% malicious nodes and declining sharply to 55% at 33%. While its clustering-based approach optimizes for low latency, it introduces vulnerabilities where malicious clusters or failures of key nodes can lead to significant disruptions or network partitioning, highlighting the trade-off between low latency and robustness in its design.

IX. CONCLUSION

This paper presents HERMES, a fair, accountable, and low-latency dissemination protocol for use by the mempool subsystems of modern blockchain systems. Investigating front-running attacks that remain in contemporary mempools, we argued why fairness must be embedded in the lowest dissemination layer. HERMES achieves fairness by randomly selecting offline-optimized robust overlay structures to accelerate message dissemination, while forcing senders to adhere to the selected overlay and dissemination structure. At the costs of a slight bandwidth overhead of 192 KB/min (when compared to the 50 KB/min of the lightest baseline protocol), HERMES successfully mitigates front-running attacks, e.g., with less than 5.9% of success rate in the presence of 33% malicious nodes, at an achieved average communication latency of 83.22 ms (48% of the latency of that protocol). Directions for future work include investigating network hierarchies (e.g., for clustered consensus [44]–[47]) and local overlay transformations that preserve fault-density despite churn.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments and advise.

REFERENCES

[1] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, IEEE, 2013, pp. 1–10.

[2] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," in *Proc. of the 39th Symp. on Principles of Distributed Computing*, 2020, pp. 428–437.

[3] B. Nasrulin, G. Ishmaev, J. Decouchant, and J. Pouwelse, "Lo: An accountable mempool for mev resistance," in *Proc. of the 24th Int. Middleware Conference*, 2023, pp. 98–110.

[4] E. Kolyvas and S. Voulgaris, "Cougar: Fast and eclipse-resilient dissemination for blockchain networks," in *Proc. of the 16th ACM Int. Conf. on Distributed and Event-Based Systems*, 2022, pp. 7–18.

[5] W. Hao, J. Zeng, X. Dai, J. Xiao, Q. Hua, H. Chen, K.-C. Li, and H. Jin, "Blockp2p: Enabling fast blockchain broadcast with scalable peer-to-peer network topology," in *14th Int. Conf. on Green, Pervasive, and Cloud Computing (GPC), Uberlândia, Brazil, May 26–28*, Springer, 2019, pp. 223–237.

[6] W. Hao, J. Zeng, X. Dai, J. Xiao, Q.-S. Hua, H. Chen, K.-C. Li, and H. Jin, "Towards a trust-enhanced blockchain p2p topology for enabling fast and reliable broadcast," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 904–917, 2020.

[7] M. Zhou, L. Zeng, Y. Han, P. Li, F. Long, D. Zhou, I. Beschastnikh, and M. Wu, "Mercury: Fast transaction broadcast in high performance blockchain systems," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, IEEE, 2023, pp. 1–10.

[8] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *28th ACM Symp. on Operating Systems Principles*, 2021, pp. 35–48.

[9] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th Usenix Security Symposium*, 2016, pp. 279–296.

[10] E. Kokoris-Kogias, *Robust and scalable consensus for sharded distributed ledgers, 2019*.

[11] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *Symp. on Security and Privacy (SP)*, IEEE, 2018, pp. 583–598.

[12] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in cooperative environments," in *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2735, Springer, 2003, pp. 292–303.

[13] P. Ramanathan and K. G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. on Computers*, vol. 37, no. 12, pp. 1654–1657, 1988.

[14] B. Toshniwal and K. Kataoka, "Comparative performance analysis of underlying network topologies for blockchain," in *2021 Int. Conference on Information Networking (ICOIN)*, IEEE, 2021, pp. 367–372.

[15] X. You, "Hyperclique: A novel p2p network structure for blockchain systems,"

[16] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *ACM SIGSAC Conf. on Computer and Communications Security, CCS*, 2018, pp. 931–948.

[17] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *1st ACM Conference on Advances in Financial Technologies, AFT, Zurich, Switzerland*, 2019, pp. 199–213.

[18] D. Frey, R. Guerraoui, A. Kermarrec, and M. Monod, "Boosting gossip for live streaming," in *10th Int. Conf. on Peer-to-Peer Computing, P2P, Delft, The Netherlands*, IEEE, 2010, pp. 1–10.

[19] Ethereum, *Ethereum transaction exchange*, https : / / github . com / ethereum/devp2p/blob/master/caps/eth.md.

[20] S. Bonomi, G. Farina, and S. Tixeuil, "Multi-hop byzantine reliable broadcast with honest dealer made practical," *Journal of the Brazilian Computer Society*, vol. 25, pp. 1–23, 2019.

[21] D. Wen, L. Qin, Y. Zhang, L. Chang, and L. Chen, "Enumerating k-vertex connected components in large graphs," in *35th Int. Conf. on data engineering (ICDE)*, IEEE, 2019, pp. 52–63.

[22] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[23] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, and D. Seredinschi, "Scalable byzantine reliable broadcast," in *33rd Int. Symp. on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, ser. LIPIcs, vol. 146, 2019, 22:1–22:16.

[24] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: A dag-based mempool and efficient bft consensus," in *Proc. of the 17th European Conf. on Computer Systems*, 2022, pp. 34–50.

[25] B. Mazorra, M. Reynolds, and V. Daza, "Price of MEV: Towards a game theoretical approach to MEV," in *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, 2022, pp. 15–22.

[26] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," *arXiv preprint arXiv:1904.05234*, 2019.

[27] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" In *Symp. on Security and Privacy (SP)*, IEEE, 2022, pp. 198–214.

[28] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the delivery of blocks and transactions in bitcoin," in *Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security*, 2015, pp. 692–705.

[29] S. Yang, F. Zhang, K. Huang, X. Chen, Y. Yang, and F. Zhu, "Sok: Mev countermeasures: Theory and practice," *arXiv preprint arXiv:2212.05111*, 2022.

[30] *[n. d.]. Flashbots Blockspace Auction*, https : / / docs . flashbots . net / flashbots-auction/overview.

[31] Y. Zhang, S. Setty, Q. Chen, L. Zhou, and L. Alvisi, "Byzantine ordered consensus without byzantine oligarchy," in *14th Symp. on Operating Systems Design and Implementation (OSDI)*, 2020, pp. 633–649.

[32] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, "Themis: Fast, strong order-fairness in byzantine consensus," in *Proc. of the ACM SIGSAC Conf. on Computer and Communications Security*, 2023, pp. 475–489.

[33] P. Zarbafian and V. Gramoli, "Lyra: Fast and scalable resilience to reordering attacks in blockchains," in *IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2023, pp. 929–939.

[34] V. Gramoli, Z. Lu, Q. Tang, and P. Zarbafian, "Aoab: Optimal and fair ordering of financial transactions," in *54th IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2024, pp. 377–388.

[35] D. Kozhaya, J. Decouchant, V. Rahli, and P. Esteves-Verissimo, "Pistis: An event-triggered real-time byzantine-resilient protocol suite," *IEEE Trans. on Parallel & Distributed Systems*, vol. 32, no. 09, pp. 2277–2290, Sep. 2021.

[36] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[37] V. Buterin, "Ethereum white paper," 2013.

[38] A. Antonov and S. Voulgaris, "Securecyclon: Dependable peer sampling," in *43rd Int. Conf. on Distributed Computing Systems (ICDCS)*, IEEE, 2023, pp. 1–12.

[39] C. for Applied Internet Data Analysis (CAIDA), *Macroscopic topology measurements*, Accessed: 2024-07-30, 2023.

[40] R. NCC, *Ripe atlas*, Accessed: 2024-07-30, 2024.

[41] A. W. Services, *Aws region latency monitor*, Accessed: 2024-07-30.

[42] M. Azure, *Latency test tool for azure virtual machines*, Accessed: 2024-07-30, 2024.

[43] Etherscan, *Ethereum*, Accessed: 2024-07-30, 2023.

[44] W. Yahyaoui, J. Bruneau-Queyreix, M. Völp, and J. Decouchant, "Tolerating disasters with hierarchical consensus," in *IEEE Conf. on Computer Communications (INFOCOM)*, IEEE, 2024, pp. 1241–1250.

[45] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "Resilientdb: Global scale resilient blockchain fabric," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 868–883, 2020.

[46] Q. T. Thai, J.-C. Yim, T.-W. Yoo, H.-K. Yoo, J.-Y. Kwak, and S.-M. Kim, "Hierarchical byzantine fault-tolerance protocol for permissioned blockchain systems," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7337–7365, 2019.

[47] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling byzantine fault-tolerant replication to wide area networks," *IEEE Trans. on Dependable and Secure Computing*, vol. 7, no. 1, pp. 80–93, 2008.