Scalable Blockchain Protocols

PIC2 - Master in Computer Science and Engineering
Instituto Superior Técnico, Universidade de Lisboa

Alexandre Ferreira — 103397*
alexandre.bruno.ferreira@tecnico.ulisboa.pt

Advisors: Professors Luís Rodrigues and Miguel Matos

Abstract Permissioned blockchains rely on consensus protocols that are difficult to scale, as they involve one-to-all or all-to-all communication patterns. Recent attempts to increase the scalability of these protocols involve the use of gossip and the use of dissemination and aggregation trees. Unfortunately, the former are very robust but are not resource-efficient and the latter are very resource-efficient but difficult to reconfigure when faults occur. In this work we study techniques that aim at providing new tradeoffs between robustness and resource usage while preserving scalability of blockchain protocols. Inspired by earlier works for the non-Byzantine setting, we propose techniques that combine the use of gossip and tree-based dissemination that are aimed at combining the robustness of gossip with the resource-efficient of dissemination and aggregation trees.

Keywords — Blockchain, Byzantine Fault Tolerant Consensus Gossip Protocols, Scalability, Distributed Systems

*I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa (https://nape.tecnico.ulisboa.pt/en/apoio-ao-estudante/documentos-importantes/regulamentos-da-universidade-de-lisboa/).

Contents

1 Introduction

Blockchain systems can be broadly divided in two main categories: *permissionless*, where the set of participants is not defined *a priori* and *permissioned*, where the set of participants in consensus is known. Permissionless protocols are typically highly scalable but unable to offer finality (informally, the property that ensures that a consensus decision cannot be undone). Permissioned protocols offer finality but are typically hard to scale.

Permissioned blockchains rely on the execution of a byzantine fault-tolerant consensus protocol. Most of these protocols use one-to-all or all-to-all communication patterns at different stages of their execution and are, therefore, difficult to scale because the communication patterns induce bandwidth and/or processing bottlenecks in one or more processes. Two different approaches have emerged to circumvent these scalability limitations. The first consists in using gossip as a dissemination strategy [1, 2]. Gossip has the potential to distribute the load evenly among participating nodes but may induce a non-negligible amount of redundancy in the protocol operation. The second consists in using dissemination and aggregation trees [3]. Trees are also able to distribute the load among a set of inner nodes but are fragile because a faulty inner node may disrupt the communication. As a result, tree-based protocols require the execution of sophisticated reconfiguration procedures, that may cause periods of unavailability.

In this work we study techniques to increase the scalability of blockchain protocols, namely we search for techniques that can combine the advantages of gossip and tree-based approaches. Our work is inspired by previous proposals for the crash-fault model, such as Plumtree [4] and Thicket [5], where gossip is used to quickly repair faults in dissemination trees. We aim at developing mechanisms that will allow the application of similar strategies in the Byzantine setting. We plan to experiment the proposed techniques by creating a variant of the Kauri blockchain protocol and by comparing its performance with the original algorithm. If time allows, we will also study how to integrate our techniques in DAG-based blockchains such as [6, 7].

The rest of the document is structured as follows. Section 1.1 lists our goals and expected results. Section 2 provides the background and Section 3 surveys the related work. Section 4 describes our approach and Section 5 describes how we plan to evaluate our work. Section 6 lays out the expected work schedule for this work. Finally, Section 7 concludes the report.

1.1 Goals and Expected Results

Our work has the following goals:

> *Goals: Our work aims at combining the use of tree-based dissemination tree with gossip mechanisms to build robust and efficient dissemination and aggregation strategies that increase the scalability and efficiency of permissioned blockchain protocols.*

The project is expected to produce the following results:

> *Expected Results: The work will produce i) a specification of the protocol, ii) an implementation of the protocol, and iii) an experimental evaluation of the resulting implementation.*

2 Background

This section introduces the fundamental concepts for understanding the proposed hybrid architecture. We begin by distinguishing between permissionless and permissioned blockchain environments, followed by a formal definition of Byzantine Fault-Tolerant (BFT) consensus and a discussion of the critical path in existing implementations. Finally, we explore the communication topologies—specifically tree-based and gossip-based patterns—that form the basis of our work.

2.1 Blockchain and Finality

A blockchain is a decentralized, immutable ledger maintained by a distributed set of participants. While the concept was popularized by permissionless systems like Bitcoin [8], where any node can participate in block

generation, our work focuses on *permissioned blockchains*. In this setting, the set of validators is known *a priori*, and participation is restricted to authorized entities.

A key distinction between these models is the guarantee of *finality*. Permissionless protocols typically offer only *probabilistic finality*, where the probability of a block being reversed decreases over time but never reaches zero. In contrast, permissioned systems rely on BFT consensus protocols that provide *deterministic finality*: once a block is committed by a quorum of validators, it is permanently part of the ledger and cannot be reverted, a property essential for applications requiring immediate settlement.

2.2 Byzantine Fault-Tolerant Consensus

Byzantine Fault Tolerance (BFT) refers to the general capability of a distributed system to maintain its operational integrity even when a subset of its components fail in arbitrary or malicious ways [9]. These behaviors, known as Byzantine faults, include nodes withholding data, sending conflicting messages, or colluding to disrupt the system. To ensure resilience, such systems typically operate under a model of N total processes where at most f are faulty, requiring $N \geq 3f + 1$ to maintain both safety and liveness under the worst-case fault scenarios [9].

Building upon this fault model, BFT Consensus is the specific distributed computing problem—and the class of protocols used to solve it—concerned with ensuring that all correct processes reach a single, consistent agreement on a common value despite the presence of Byzantine failures . Formally, any protocol solving the BFT consensus problem must satisfy four fundamental properties [10] :

- Termination: Every correct process eventually decides on a value.
- Agreement: No two correct processes decide on different values.
- Integrity: No correct process decides twice.
- Validity: If a correct process decides a value, it must have been proposed by some process.

2.2.1 Network Models and Partial Synchrony

To ensure these properties, protocols must make assumptions about the network. While *safety* can often be maintained in asynchronous networks (where messages can be delayed indefinitely), *liveness* is impossible to guarantee deterministically in a purely asynchronous setting due to the FLP impossibility result [11].

Therefore, most practical BFT protocols, including Kauri [3] and HotStuff [12], assume a *Partially Synchronous* model. In this model, the system may behave asynchronously for an unknown period, but there exists a Global Stabilization Time (GST) after which message transmission delays are bounded by a known constant Δ. This assumption justifies the use of *timeouts* to detect leader failures or stalled paths: if a step exceeds the expected Δ, processes can safely assume a fault and trigger recovery mechanisms.

2.2.2 Critical Path in Consensus Protocols

In many blockchain protocols, multiple steps need to be executed to commit a block before a decision on the next block can start. We call these steps the *critical path*. This path typically includes the dissemination of the full list of transactions and verification of data availability across a quorum of replicas. Optimizing the critical path is the primary objective of scalable consensus research, as any bottleneck here directly limits the system's end-to-end throughput. As we will see in the next sections like [3], some works strive to remove as many of the steps from the critical path as possible in an attempt to better scale the algorithms. The topologies and communication patterns we will discussed can be visualized in Figure 1 and Figure **??**

2.3 Communication Topologies and Patterns

The performance of a distributed system is heavily influenced by its underlying network structure. To analyze this, we must distinguish between the *topology*—the graph of connections between nodes—and the *communication pattern*—the strategy used to propagate messages across that graph.

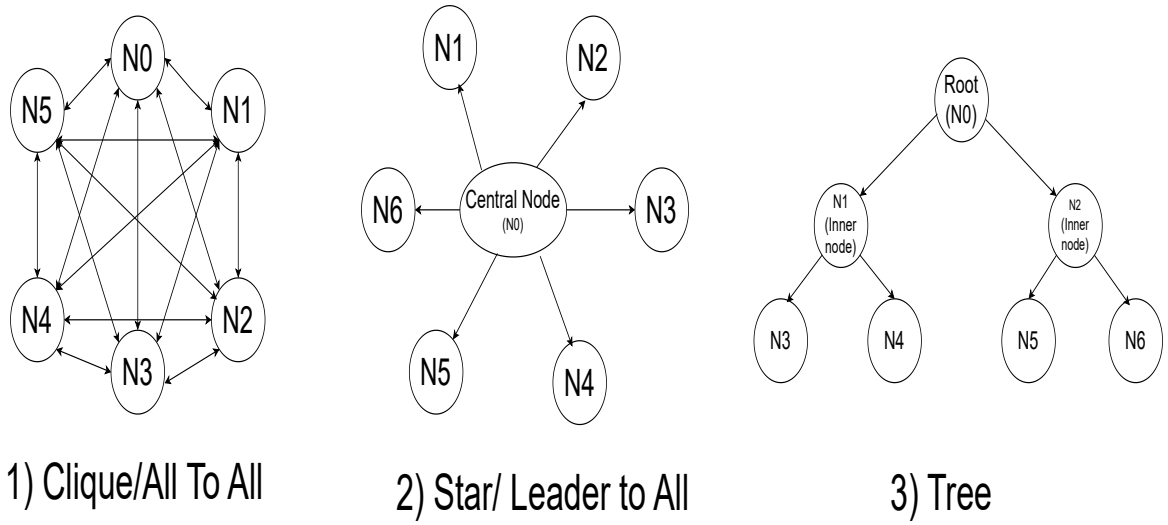

Figure 1: Clique Topology vs Star Topology vs Tree Topology.

2.3.1 Structured Topologies

Structured topologies organize nodes into specific, deterministic graphs. In a *clique* or fully connected topology, every node maintains an active connection with every other node. While conceptually simple to create, this topology incurs a high maintenance cost because the number of links scales quadratically (N^2) as the system grows. This structure typically supports an *All-to-All* communication pattern, as seen in protocols like PBFT [9], which ensures robustness at the cost of bandwidth saturation.

Alternatively, a *star* topology connects all nodes to a single central node, minimizing link complexity to $O(N)$. While easy to construct, it creates a single point of failure and a significant bottleneck at the central node. This structure facilitates *Leader-to-All* and *All-to-Leader* patterns, which protocols like HotStuff [12] utilize to achieve linear message complexity. When using this pattern, limited network bandwidth or CPU resources at the central node may become a source of performance bottlenecks.

Finally, *tree* topologies organize nodes hierarchically. This approach achieves $O(N)$ link complexity but presents significant maintenance challenges. Constructing balanced trees is computationally complex, and repairing the topology is difficult because the failure of an internal node partitions its entire subtree. Trees are designed for *Dissemination* and *Aggregation* patterns, which systems like Kauri [3] leverage to distribute bandwidth across internal nodes. For this pattern to be reliable, the tree must be robust. A tree is considered *robust* if all of its internal nodes are correct, ensuring the leader has a reliable path to disseminate messages to every leaf and aggregate replies back to the root without being blocked by malicious intermediaries.

2.3.2 Unstructured Topologies (Mesh)

In unstructured or mesh topologies, nodes connect to a random subset of peers rather than following a rigid structure. These topologies are highly resilient and low-cost to maintain because nodes can easily recover from link failures by selecting new random peers locally. The dominant communication pattern here is *Gossip*, where nodes periodically exchange information with random peers to create a probabilistic flood of data. This is achieved via *Eager Push*, sending full payloads for low latency, or *Lazy Push*, which sends metadata to save bandwidth [2]. While extremely robust, this pattern introduces higher redundancy and latency compared to structured multicast.

3 Related Work

This section reviews the state-of-the-art in Byzantine Fault-Tolerant (BFT) consensus and network dissemination. We first examine the evolution of consensus protocols from traditional star and clique topologies to modern DAG-based systems, followed by an analysis of tree-based and gossip-based dissemination strategies that inform our hybrid approach.

3.1 BFT Consensus and Mempool Algorithms

3.1.1 PBFT (Practical Byzantine Fault Tolerance)

PBFT [9] was the first protocol to demonstrate that BFT consensus could be achieved with high performance in real-world asynchronous networks. It utilizes a three-phase communication pattern— Pre-prepare, Prepare, and Commit—to ensure agreement. However, PBFT relies on an all-to-all communication pattern, leading to a quadratic $O(N^2)$ message complexity. This limits the protocol's scalability to small validator sets.

3.1.2 HotStuff

HotStuff [12] addressed the scalability limitations of PBFT by introducing a leader-centric star topology. In HotStuff, replicas send votes directly to the leader, who aggregates them into a Quorum Certificate (QC). While this reduces message complexity to $O(N)$, it creates a significant bottleneck at the leader, who must verify N signatures and handle the bandwidth load for block dissemination.

3.1.3 Kauri

Kauri [3] is a Byzantine Fault-Tolerant (BFT) communication algorithm designed to address the scalability bottlenecks found in previous leader-based consensus algorithms. Kauri was originally published in [3] and an extended version was later published in [13]. In this report, we sometimes refer to mechanisms that are only mentioned in the journal version.

Instead of using the all-to-all or leader-to-all communication patterns, it employs a tree topology for both message dissemination and signature aggregation. This helps distribute the computational and bandwidth load across the tree's internal nodes. While leaf nodes contribute less to aggregation, this structure critically removes the processing bottleneck from the leader.

There are two known issues in Byzantine tree topologies:

- *Increased Latency:* Tree communication requires $2 \times h$ steps (where h is the tree's height) for one full round-trip (downstream dissemination and upstream aggregation), potentially hurting throughput.
- *Fault Vulnerability:* Progress requires correct participation from internal nodes. If an internal node is Byzantine, it can disrupt the dissemination and aggregation path for an entire subtree.

To solve the latency problem, Kauri introduces pipelining: messages for round $r + 1$ are sent while votes for round r are still being aggregated. This overlap reduces idle time and, under steady-state conditions, significantly increases throughput. To address fault vulnerability, Kauri constructs multiple aggregation trees with the guarantee that at least one of them is robust. Specifically, all N processes are partitioned into $f + 1$ disjoint bins. Kauri then rotates through these bins, constructing a candidate tree in which all internal nodes are selected exclusively from the current bin, while nodes belonging to other bins are assigned to leaf positions. Since at most f processes may be Byzantine, and the nodes are partitioned into $f + 1$ bins, it is guaranteed that at least one bin contains only correct processes. The tree constructed from this bin therefore contains only correct internal nodes, ensuring robustness.

While the definition of a robust tree assumes all internal nodes are correct, Kauri can still achieve consensus under less restrictive conditions. Specifically, Kauri identifies a *quorum q-robust tree* as a structure where the leader is correct and there is a path composed exclusively of correct processes and "safe edges" between the leader and a Byzantine quorum of nodes. In this model, even if some internal nodes are Byzantine, they only disrupt specific branches; as long as the remaining correct internal nodes can facilitate communication with at

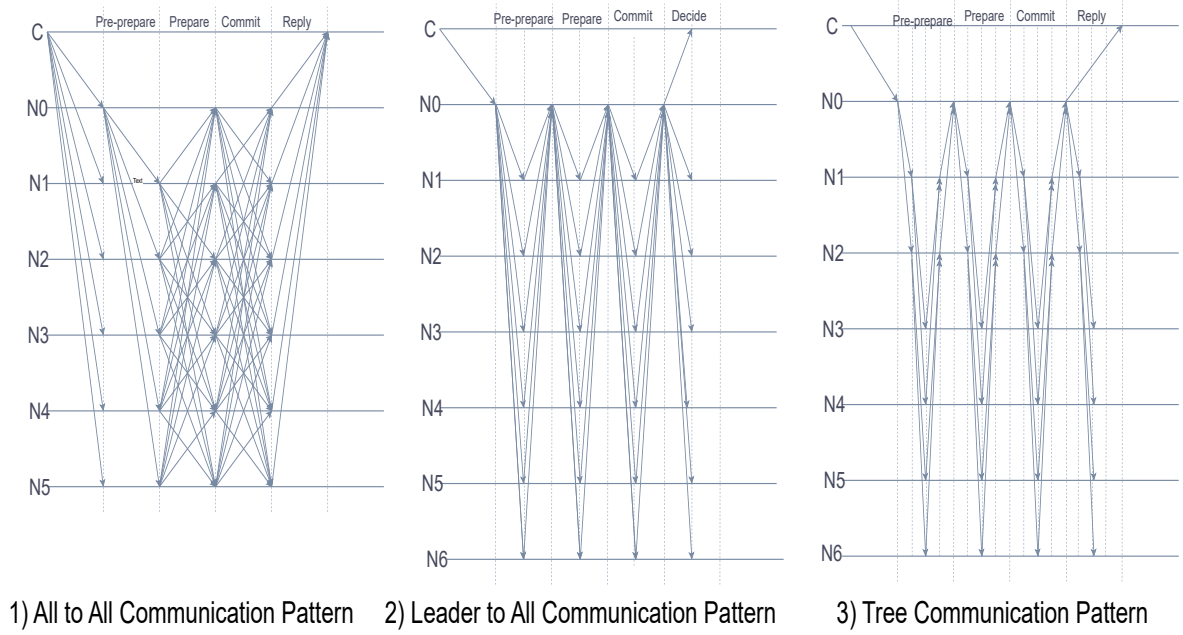

Figure 2: A visualization of how all-to-all and leader-to-all message complexity compares to a tree topology.

least $N - f$ correct processes, the tree is considered q-robust and the system can progress. This distinction is critical because while a perfectly robust tree is the ideal case, q-robustness is the actual necessary and sufficient condition to ensure that the leader can successfully disseminate a block and collect enough signatures to reach an agreement.

When a fault affects the operation of a tree, Kauri reconfigures the system to ensure liveness. It does this by selecting a tree using a different bin, as explained above. As long as the number of faults is smaller or equal to f, at most f reconfigurations are needed to find a robust tree, which is optimal, given that at least one bin includes only non-faulty nodes. During this process, consensus is halted. While systems that utilize a leader to all communication pattern ensure termination as long as the leader is non-faulty, Kauri requires the leader and inner nodes in the tree to be correct. This makes the need for reconfiguration more likely, which is one of Kauri's limitations.

3.1.4 Extentions to Kauri

Some work has already been done in the past with regards to improvements to Kauri's reconfiguration process [14] as well as trying to improve how nodes are selected for the trees [15]. However, for our work, only the former is relevant and for this reason we will briefly describe these improvements below:

Building on the original Kauri architecture, subsequent research [14] has focused on optimizing the reconfiguration process to minimize downtime during faults. A key contribution of this work is the transition from static trees to dynamic path-aware configurations. In the original Kauri, when a node in an aggregation tree is suspected of being faulty, the entire system must halt consensus to search for a new, robust tree.

To address this, the proposed improvements introduce a decentralized monitoring system where each node maintains a local view of its path's performance. The key mechanism, which is particularly relevant to our approach, is the implementation of *hierarchical timeouts*. Instead of a single global timer for the entire tree, different parts of the topology are assigned variable timeouts based on their depth and observed network latency.

This allows the system to distinguish between a localized bottleneck in a specific subtree and a complete failure of the leader, enabling more specific responses. By using these differentiated timers, the protocol can detect faults faster at the lower levels of the tree without triggering a premature global reconfiguration, effectively keeping the consensus pipeline active for a larger portion of the network even during transient faults.

Another extension that stemmed from the latest published version of Kauri [13], introduced a mechanism of early relaying of the responses from an internal node to its parent, without waiting for all the children's replies. While the original protocol [3] followed a strict hierarchical aggregation model—where each internal node had to aggregate a complete quorum of signatures from its entire sub-tree before propagating a single message upward—the newest iteration adopts a streaming aggregation approach [13]. This development is critical for mitigating the impact of stragglers (slow or intermittent nodes) that would otherwise stall an entire branch's contribution to the pipeline [3]. By allowing partial aggregation results to be relayed immediately, the root node can reach the necessary $2f+1$ threshold for a QC significantly faster, effectively decoupling the system's overall throughput from the tail latency of its slowest participants.

3.1.5 Narwhal and Tusk

Many Byzantine Fault-Tolerant (BFT) protocols tightly couple transaction dissemination with transaction ordering, leading to significant scalability bottlenecks. Narwhal and Tusk [6] address this by introducing a clean separation of concerns: Narwhal handles the efficient, reliable dissemination and storage of transactions through a mempool abstraction based on a DAG [16], while Tusk—or other consensus algorithms such as HotStuff [17]—focuses solely on ordering small, fixed-size references, known as certificates, to those blocks. This modular design improves throughput by shifting bulk transaction handling out of the synchronous consensus' critical path (see Section 2) because the consensus layer only waits for small, pre-formed availability certificates.

Narwhal's foundation is a distributed key-value block store that constructs a DAG of blocks structured into rounds. To propose a block, a validator broadcasts it to all others and waits for acknowledgments. Once a block is acknowledged by at least $2f+1$ validators, a collective signature is formed, creating an unforgeable certificate of block availability. This certificate provides two vital properties: *integrity*, ensuring the block content has not been tampered with, and *block-availability*, which guarantees that at least $f+1$ honest validators have stored the block and can provide it to others if needed. To ensure causality, a block in round r must include certificates for at least $2f+1$ blocks from the previous round $r-1$. This strategy ensures that a single certificate implicitly refers to an entire causal history of transactions. To achieve high performance, Narwhal utilizes a scale-out architecture where each validator employs multiple worker machines dedicated to processing transaction batches, while a single primary machine handles the metadata for the DAG. This separation allows throughput to scale almost linearly with added hardware. Furthermore, Narwhal provides censorship resistance; namely it aims at providing a property known as *chain quality*. Chain quality ensures that a fraction of the blocks in the ledger are authored by honest nodes. Because every validator, when creating a block for round r, is required to reference a quorum of $2f+1$ availability certificates from round $r-1$, the DAG is forced to include contributions from a broad set of validators. This structural requirement prevents a subset of Byzantine nodes from selectively ignoring specific transactions, as they cannot advance to the next round without acknowledging the blocks produced by the rest of the network, including honest but potentially slower nodes.

Despite these strengths, the design introduces specific trade-offs. The requirement to collect and verify $2f+1$ signatures to form a certificate is computationally expensive and introduces a baseline latency in the dissemination layer. Furthermore, Narwhal often exhibits lower goodput (the rate of unique, non-redundant transactions) compared to leader-based protocols like Kauri. Because Narwhal is leaderless and lacks centralized coordination, different validators frequently pick up the same client transactions and include them in concurrent blocks. This transaction duplication consumes network bandwidth and storage without increasing the actual number of unique transactions processed.

The system behavior varies depending on the consensus logic used. When combined with HotStuff (a configuration referred to as Narwhal-HS), the system relies on a leader for ordering. Consequently, Narwhal-HS wnds up losing liveness and suffering from increased latency—jumping from 2 seconds to 10 seconds—during periods of network asynchrony or leader failure.

To mitigate this, Tusk was designed as a fully asynchronous companion to Narwhal. While Narwhal ensures data availability, Tusk [6] is a zero-message overhead asynchronous consensus protocol that orders the

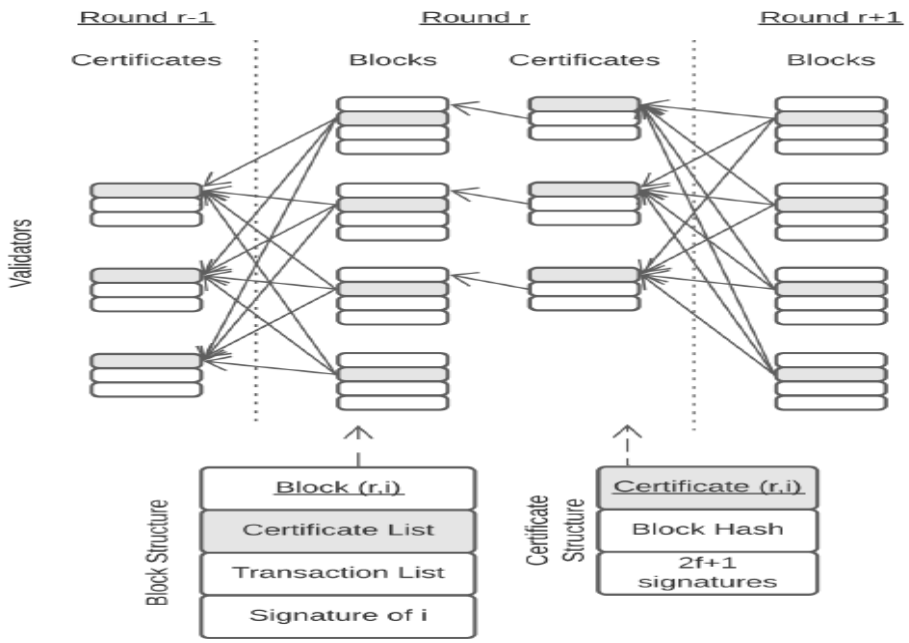

Figure 3: How rounds work in Narwhal [6].

transactions already present in the DAG. It is designed to maintain high performance even during periods of network asynchrony or high fault rates, where partially synchronous protocols typically stall. Tusk operates by interpreting the round-based structure of the Narwhal DAG as a *virtual clock*. In every odd-numbered round r, a validator's block is chosen as the proposer for that round. Unlike previous protocols, Tusk requires no additional communication for this selection because nodes simply examine their local view of the DAG, and all honest validators will apply the same rules to choose the same proposer. If a validator observes that the designated proposer's block for round r has been referenced (and thus "certified") by at least $2f + 1$ blocks in round $r + 1$, it can locally commit that block and its entire causal history. This mechanism allows Tusk to achieve consensus with virtually no additional communication overhead beyond the dissemination already performed by the mempool, while ensuring liveness without relying on timing assumptions or a single, persistent leader.

3.1.6 Bullshark

Bullshark [18] is a high-performance DAG-based BFT consensus protocol designed to address the practical limitations of its predecessor, Tusk. While Tusk provides theoretical liveness under full asynchrony, Bullshark is optimized for the partially synchronous model common in real-world blockchain deployments, significantly simplifying the garbage collection of the DAG and reducing the metadata overhead required for ordering. When integrated with a mempool like Narwhal, Bullshark achieves high throughput by allowing validators to propose blocks concurrently, which are then ordered using a deterministic rule that interprets the DAG's structure.

A key feature of Bullshark is its use of a "fast path" that allows for low-latency commits during periods of network stability, while maintaining a "slow path" to ensure safety and liveness during asynchronous periods or leader failures. However, Bullshark still suffers from the inherent signature verification bottleneck common to DAG-based systems [7]. Because every node must still verify a quorum of signatures for every block in the DAG to ensure data availability, the CPU load scales linearly with the committee size.

3.1.7 Mysticeti

State-of-the-art DAG-based consensus algorithms, such as Narwhal-HotStuff or Bullshark [18], achieve high throughput but often suffer from high latency. The Mysticeti protocol [7] was designed to minimize this latency,

reaching the theoretical lower bound of three message rounds for consensus. It introduces two primary variants: *Mysticeti-C*, a complete BFT consensus protocol, and *Mysticeti-FPC*, which provides a fast-path for transactions that do not require full consensus, such as simple asset transfers.

At the time of Mysticeti's publication, existing protocols could process approximately 100,000 transactions per second but had latencies of roughly two seconds. The authors identified that the primary bottleneck in these Narwhal-based systems was the requirement to explicitly certify every block. In those designs, every block must collect and verify $2f + 1$ signatures to generate an availability certificate before it can even be considered by the consensus layer. This certification step is computationally expensive and adds multiple round-trips to the critical path (as defined in 2), significantly delaying the time it takes for a block to be committed.

Mysticeti reduces this overhead by eliminating explicit availability certificates. Instead of waiting for a quorum of signatures to acknowledge each block, validators broadcast their signed blocks immediately. For a block to be considered valid, it simply needs to include references to at least $2f + 1$ blocks from the previous round. In this model, availability and safety are derived implicitly from the structure of the DAG itself. If a block is later referenced by $2f + 1$ distinct validators in subsequent rounds, it provides the same mathematical assurance as a traditional certificate: at least $f + 1$ honest nodes must have received and validated it. This implicit certification allows the protocol to progress without the latency penalty of a dedicated, synchronous verification step for every individual block.

Consensus in Mysticeti is achieved by having each node interpret its local view of the DAG using a set of deterministic rules. A block is classified as *to-commit* if it is referenced by $2f + 1$ blocks in the following round, a process known as the direct commit rule. Conversely, if a quorum of blocks fails to reference it, it may be marked as *to-skip*. To maintain a total order, an indirect commit rule also applies, where committing a leader block automatically commits its entire causal history of older, linked blocks.

To ensure liveness, Mysticeti designates one block per round as the primary block, with proposers rotating in a round-robin fashion. Validators preparing their own blocks must wait for a predefined delay to receive the current round's primary block before broadcasting. This mechanism ensures that at least one honest primary block is likely to be certified and committed in each round, preventing malicious nodes from indefinitely blocking consensus. Benchmarks show that *Mysticeti-C* delivers a fivefold reduction in latency, dropping from 2 seconds to approximately 400 ms. However, every validator must still verify the signatures of every block it receives from its peers. As the committee size n grows, this $O(n)$ signature verification load can become a bottleneck.

3.1.8 GoSig

The GoSig protocol [1] was developed to address three fundamental limitations identified in earlier BFT systems: the presence of single points of failure often caused by a static or centralized leader, the high latency stemming from block broadcasting or metadata dissemination, and a performance ceiling imposed by the slowest nodes in all-to-all communication protocols. To mitigate these interconnected issues, the authors proposed a decentralized BFT architecture that is deeply integrated with a gossip-based communication network.

The operation of GoSig is structured into rounds, with each round divided into a block proposal stage and a signature collection stage. During the proposal stage, several validators are randomly selected to pack transactions into blocks and broadcast them to the network. By rotating these proposers in every round, GoSig eliminates the leader bottleneck common in single-leader protocols. In the subsequent signature collection stage, nodes vote for a received block by signing it and gossiping their vote to their peers. A key feature of this stage is that validators combine received signatures with their own to form an aggregated signature, which they continue to relay until a Byzantine quorum is reached . This mechanism demonstrates the feasibility of using gossip networks to aggregate messages efficiently.

To minimize idle time and maximize throughput, GoSig utilizes a pipelined design similar to that of Kauri. This is supported by a block structure designed specifically to be pipeline-friendly, consisting of a metadata-rich header, a body containing an ordered list of transaction hashes, and the raw transaction data. The block body is further divided into segments, similar to the chunks used in peer-to-peer swarming protocols [2], allowing nodes to transmit data incrementally without waiting for the entire block to be available. Each segment is individually signed, which allows for immediate integrity checks and prevents malicious actors from flooding the network with invalid data.

The protocol employs a two-stage gossip scheme for data dissemination. In the first stage, only the lightweight

headers are gossiped, providing validators with enough information to decide which block to support without consuming significant bandwidth. In the second stage, once a block is chosen, validators begin a swarming-like exchange to download missing body segments from their peers. This parallel exchange of verified chunks ensures high throughput even for large blocks and prevents the system's performance from being limited by its slowest members.

3.2 Scalable Dissemination in Trees

3.2.1 PlumTree

PlumTree [4] is a broadcast protocol that addresses the inherent trade-off between the efficiency of deterministic tree-based dissemination and the resilience of epidemic gossip protocols. While trees offer low message complexity, they are highly vulnerable to node failures. Conversely, gossip protocols provide extreme robustness through redundancy but have high bandwidth costs due to duplicate message delivery. PlumTree resolves this by constructing a spanning tree embedded within a random gossip overlay, allowing the system to use the tree for primary dissemination while retaining the gossip mesh for rapid recovery.

The protocol manages its communication links by categorizing them into two distinct types: eager push links and lazy push links. Eager links form the active branches of the broadcast tree, where message payloads are forwarded immediately upon receipt to minimize latency. Lazy links, which represent the remaining connections in the underlying gossip overlay, are used only to transmit lightweight message identifiers or metadata. These are known as IHAVE messages, and they serve as periodic advertisements that batch multiple message identifiers into a single lightweight control packet. By only announcing the availability of data rather than transmitting the full payload, these messages allow the protocol to maintain a redundant safety net of potential data sources with minimal bandwidth consumption. This dual approach ensures that in a steady state, the message complexity is equivalent to that of a spanning tree ($n - 1$ messages for n nodes), as only one copy of the full payload is pushed to each participant.

A critical feature of PlumTree is its self-healing capability, which triggers when a node detects a failure in the tree. If a node receives an IHAVE for a message it has not yet received via its eager links, it assumes its tree parent is faulty or that the network is partitioned. The node then initiates a repair process by requesting the missing payload from the peer that sent the metadata and "grafting" that link, transforming it from a lazy link into an eager one. This dynamic reconfiguration allows the protocol to recover from large-scale node failures or network partitions without manual intervention or global restarts.

3.2.2 Thicket

The Thicket protocol [5] is an extension to PlumTree that addresses the fundamental trade-off in tree-based dissemination, where resource efficiency comes at the cost of being vulnerable to failures of processes that serve as inner nodes in the tree and the load unbalance between inner and leaf nodes. It offers a decentralized mechanism for building and maintaining multiple independent spanning trees over a single unstructured peer-to-peer overlay. By distributing the topology across T distinct trees, Thicket ensures that all nodes in the system remain connected while preventing any single process from becoming a critical bottleneck.

A core innovation of Thicket is its decentralized load balancing strategy. The protocol ensures that the majority of participants act as interior nodes in only a single tree while remaining leaf nodes in all others . This structure provides significant fault isolation; because trees are designed to be interior-node-disjoint, the failure of a single node only disrupts the dissemination path in one of the T trees. The remaining $T - 1$ trees continue to function, allowing the system to maintain data availability through redundancy, such as simple replication or erasure coding.

Nodes across different trees periodically exchange lightweight summary messages containing identifiers of recently received data. If a node detects missing data for a specific tree, it uses this metadata to identify a network partition and initiates a localized repair by grafting a new link to a backup peer that possesses the required information. This allows the tree to heal dynamically without the need for global reconfiguration.

3.2.3 SplitStream

SplitStream [19] is a high-bandwidth multicast protocol designed to achieve scalable and balanced data dissemination in large-scale peer-to-peer systems. Its fundamental contribution is the concept of striping, which involves partitioning a single data stream into multiple substreams, each delivered over an independent multicast tree. This multi-tree architecture allows the system to achieve high aggregate throughput while ensuring that the forwarding load is distributed evenly across all participating nodes.

In traditional tree-based systems, a small number of interior nodes carry the entire burden of forwarding messages, which creates significant bottlenecks and increases vulnerability to failures. SplitStream addresses this by ensuring that each node serves as an interior node in only a limited number of trees—ideally just one—while acting as a leaf in the remaining ones. This near-disjoint arrangement of trees ensures that the upload bandwidth requirements are shared among all participants, preventing any single node from being overwhelmed.

The protocol is built upon structured overlay networks, such as Pastry and Scribe [20, 21], which provide the routing substrate necessary to construct and maintain these trees efficiently. Pastry facilitates the routing of messages to specific nodes based on identifiers, while Scribe manages the group communication and tree formation. By leveraging these existing layers, SplitStream can accommodate nodes with varying bandwidth capacities and maintain stability even as nodes frequently join or leave the network. To further enhance reliability, the protocol can be combined with erasure coding, which allows receivers to reconstruct the original data stream even if some stripes are lost due to network congestion or node failure.

3.2.4 Discussion

Analyzing the current landscape of BFT consensus reveals a recurring trade-off between computational efficiency and network scalability. Protocols like Narwhal and Tusk [6] successfully decouple transaction dissemination from ordering to improve throughput, but they introduce a substantial CPU bottleneck. This is primarily due to the requirement of forming and verifying explicit availability certificate which adds significant latency under high load. Furthermore, since Narwhal relies on broadcasting full block payloads, the network costs associated with data propagation remain a limiting factor as the validator committee size grows.

In contrast, systems like Mysticeti [7] identify the per-block certification process as the dominant contributor to end-to-end latency. By deriving implicit certificates directly from the DAG structure, Mysticeti reduces cryptographic overhead and improves latency. However, it does not resolve the underlying network scalability issues; the cost of disseminating large blocks to every participant remains high, leading to network-bound bottlenecks as the committee expands. On the other hand, the $O(n)$ signature verification per round also hinders scalability.

Leader-based protocols like HotStuff represent the traditional approach to consensus, yet they are frequently limited by the leader's bandwidth when disseminating blocks to all replicas [12]. Kauri [3] directly addresses this network bottleneck by introducing the pipelined tree, which distributes the transmission load across internal nodes. While this design improves scalability to hundreds of validators, it creates a new challenge: the system becomes sensitive to internal node faults, where a single Byzantine node can disrupt an entire subtree.

Alternatively, GoSig [1] addresses the core challenges of decentralization and performance by combining proposer rotation, pipelined execution, and a two-phase gossip dissemination strategy. Most importantly for this work, it proves that gossip-based message propagation can be successfully integrated with BFT consensus to aggregate votes and bypass traditional network bottlenecks. This provides a strong theoretical and practical foundation for investigating hybrid architectures that supplement structured consensus trees with gossip-based overlays to improve overall system resilience.

As for Plumtree [4], its' design principles are directly applicable to the diagnostic and recovery mechanisms proposed in this work. By adopting a similar distinction between structured and unstructured communication, our protocol utilizes the primary Kauri tree for dissemination and the redundant gossip channels for auxiliary availability metadata. In this context, the gossip overlay serves a purpose similar to PlumTree's control messages: providing a secondary path for information that allows nodes to detect when a parent in the main tree is withholding data. This strategy enables the system to pinpoint Byzantine faults through discrepancies in these redundant signals, effectively blending Kauri's [3] structured scalability with the robustness of epidemic fault detection.

Table 1: Vulnerabilities and Weaknesses in Tree-Based BFT Topologies and Proposed Robustness Strategies.

Vulnerability/Weakness	**Primary Impact**	**Root Cause**
Cryptographic Load	Increased Latency	CPU Saturation
Parent Node Failure	Total Branch Stalling	Byzantine or Crash Fault
Data Withholding	Block Unavailability	Byzantine Attack
Internal Tree Overload	Slow Path Processing	Hardware Heterogeneity

The design principles introduced by Thicket [5] serve as an important architectural precedent for the redundant channel mechanism proposed in this work. Thicket demonstrates the practical feasibility and scalability of maintaining parallel, independent communication structures over a single unstructured mesh. By adapting these repair and metadata exchange strategies, auxiliary channels can be utilized not just for data recovery, but as diagnostic tools to pinpoint Byzantine behavior.

The principles of SplitStream [19] are particularly relevant to addressing the network bottlenecks found in scalable BFT protocols. For instance, the multi-tree approach demonstrates how block payloads can be distributed across concurrent communication paths to balance network utilization and reduce per-node transmission costs. While SplitStream was originally designed for cooperative environments, its strategy of parallelizing dissemination across independent trees offers a path toward bandwidth-efficient dissemination that remains robust. However, applying these concepts in a Byzantine setting requires additional mechanisms to prevent malicious nodes from omitting or manipulating data within a specific stripe.

A consistent pattern emerges from this literature: when a system reduces CPU work, network dissemination typically becomes the new bottleneck, and vice versa. This observation motivates our research into a hybrid approach that leverages Kauri's structured trees for efficiency while introducing a gossip-based diagnostic layer to address the fragility and fault-vulnerability inherent in tree topologies. To better visualize these fragilities, we compiled them into Table 1.

4 Approach

Building on the analysis of existing BFT bottlenecks, we propose an architectural enhancement to the Kauri protocol designed to address the inherent fragility of tree-based dissemination. While we retain Kauri's hierarchical aggregation to mitigate cryptographic load and prevent the CPU saturation common in large-scale committees, our primary contribution introduces a multi-layered redundancy strategy to secure the dissemination path.

As summarized in Table 1, tree-based topologies are highly sensitive to parent node failures, which typically result in total branch stalling. To counter this, we introduce an *auxiliary gossip and recovery layer* that operates in parallel to the main dissemination tree. This layer leverages IHAVE-based pulling to explicitly detect and neutralize Byzantine data-withholding attacks, ensuring that block unavailability is resolved locally through peer-to-peer exchanges without halting the global consensus pipeline.

Drawing on the design principles of load-balanced parallel trees seen in protocols like SplitStream, our architecture distributes the dissemination burden across multiple redundant paths. By utilizing Kauri's node-partitioning property, we maintain $f + 1$ parallel tree configurations, providing a mathematical guarantee that at least one robust dissemination path is always available to propagate metadata and facilitate recovery.

In systems that use star-based topologies, only a faulty root can prevent the progress of the protocol; in tree-based systems, faulty inner nodes in the tree can also prevent progress. To motivate our hybrid design, Table 1 summarizes the specific vulnerabilities that arise in BFT protocols that use tree-based topologies and contrasts existing mitigations with our proposed gossip-assisted recovery strategies. This comparison shows that our work does not intend to replace Kauri dissemination mechanisms, but rather supplements it by addressing fault recovery, which existing tree-based mechanisms are ill-equipped to handle.

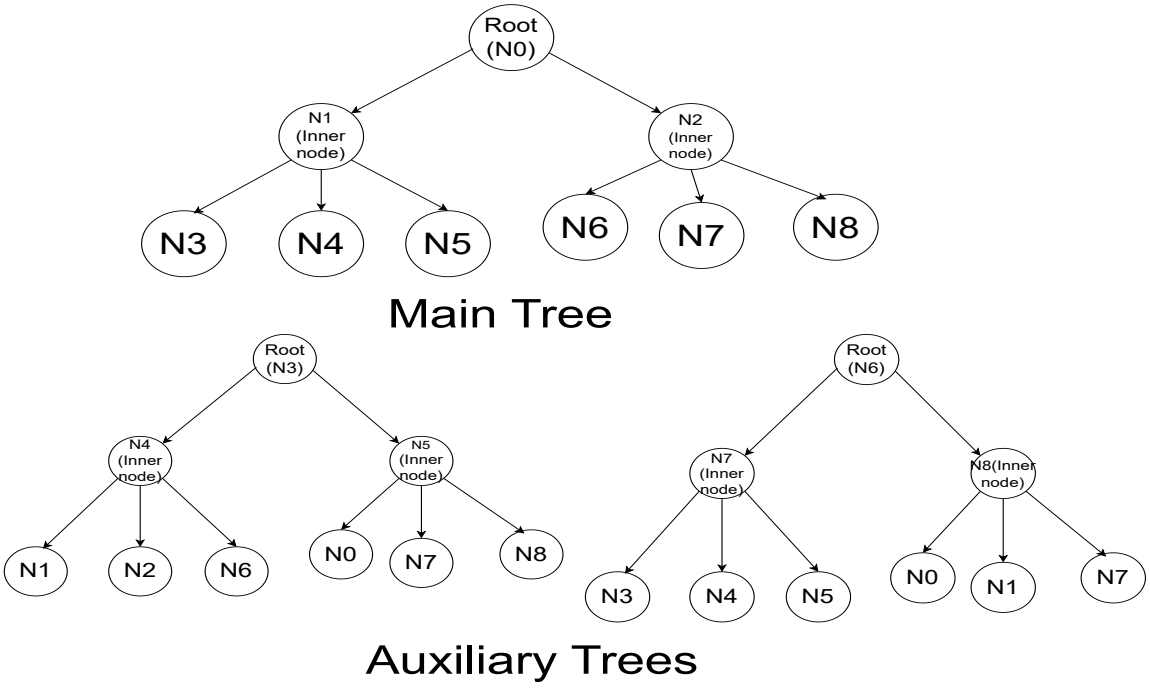

Figure 4: System architecture showing the primary dissemination tree alongside auxiliary gossip channels for metadata propagation.

4.1 Multi-Tree Redundancy and Metadata Propagation

Our architecture leverages Kauri's fundamental property of node partitioning, where the n validators are divided into $f + 1$ disjoint bins. By having the internal nodes of each tree to be drawn exclusively from one of these bins, the protocol constructs $f + 1$ parallel trees with the guarantee that at least one tree is robust—meaning its internal paths consist entirely of honest nodes. We designate one of these trees as the *primary tree*, used for block dissemination and signature aggregation, while the remaining f *secondary trees* serve as redundant metadata channels as seen in Figure 4.

Rather than replicating full block payloads, which would impose significant bandwidth overhead, the secondary trees carry lightweight metadata in the form of IHAVE messages [4]. At the beginning of each consensus round, every validator starts a local diagnostic timeout to monitor the progress on the primary tree. If a node receives a block via the primary path, it immediately broadcasts an IHAVE message to all its peers within the secondary trees. To ensure high availability of this metadata, nodes periodically aggregate the IHAVE signals they receive from others and forward them along with their own. Even nodes that have not yet received the block payload continue to forward metadata received from other nodes. Because the existence of $f + 1$ trees guarantees the existence of one robust tree, metadata or the block will reliably reach every honest node even if the primary dissemination path contains faulty internal nodes.

4.2 Path-Aware Timeouts and Recovery

To manage the transition from structured dissemination to redundant recovery, we utilize hierarchical path-aware timeouts [14]. Every node maintains a local timer adjusted based on its depth in the primary tree to account for natural propagation delays. Nodes higher on the tree haver larger timeouts, and node lower on the tree have smaller timeouts. This prevents lower-level nodes from triggering recovery prematurely due to minor network issues while ensuring they react swiftly to actual parent failures.

If a node's local timeout expires and its aggregated metadata indicates that other peers have received the

block while the node remains stalled, this signals a potential failure in its primary path. The node then performs a lateral pull, requesting the missing block from a peer that issued a verified IHAVE signal. This mechanism allows the node to bypass a faulty or slow parent locally and maintain the consensus pipeline without forcing a reconfiguration (which would halt consensus). Choosing the correct peer to pull from is a critical design decision; for instance, pulling from the node that delivered the first IHAVE notification. This prioritizes latency by favoring the most efficient network paths.

Alternatively, a randomized selection strategy, common in unstructured gossip protocols [2], provides higher security against targeted withholding by preventing an adversary from predicting which node will be selected.

4.2.1 Peer Pull Selection

In this section, we explore potential approaches for choosing a peer to pull from in the case of a fault. Part of the thesis work will involve exploring and deciding on the best heuristic. The selection process represents a significant trade-off between minimizing recovery latency and ensuring resilience against Byzantine "bait-and-switch" [22] attacks, where a malicious node advertises data availability via an IHAVE signal but refuses to fulfill the subsequent data request.

One potential approach is pulling from the first IHAVE received for a given block. This strategy is based on the rationale that the fastest metadata delivery often correlates with the most efficient network path or the least congested node, mimicking the performance-seeking behavior of PlumTree's eager push links. While this minimizes the recovery window in benign settings, it is susceptible to rushing attacks where a Byzantine node sends a metadata signal prematurely to attract PULL requests it has no intention of fulfilling. To mitigate this risk, nodes can implement an aggressive response timeout for the PULL request, quickly falling back to a secondary peer if the first-mover fails to deliver.

Alternatively, a randomized selection strategy offers higher security by picking a peer uniformly at random from the pool of all valid IHAVE signals received during the timeout period. This approach, widely utilized in unstructured gossip protocols to maintain topological robustness, prevents an adversary from deterministically predicting which node will be chosen, thereby reducing the effectiveness of targeted withholding attacks.

4.3 Addressing Faults without Reconfiguration

A significant strength of this approach is its handling of "soft faults" that do not traditionally trigger Kauri's reconfiguration. Kauri is designed to progress as long as a quorum q-robust tree exists [13]—where at least $n - f$ correct nodes are reachable from the root—even if some branches are blocked. However, in a standard tree, nodes in those blocked branches remain stalled until the next reconfiguration round. Our protocol improves this by enabling localized recovery. Instead of waiting for a global view change, a stalled child can actively retrieve missing data from honest peers in other branches identified via the IHAVE messages. This transforms a potential branch-wide failure into a localized recovery event, maintaining high system throughput even under adverse conditions.

4.4 Analysis of Communication Overhead

The feasibility of this redundant gossip layer depends on the ratio of metadata to data. In our topology, each node acts as an internal node in exactly one tree (with a fanout of k) and as a leaf in the remaining f trees. Consequently, each node communicates with k children in the tree where it acts as an internal node and with its parent in all $T = f + 1$ trees (unless it's the root in the tree where it acts as an internal node, where it does not have a parent), resulting in $k + T$ gossip neighbors. Given that an IHAVE message is a lightweight packet (and transaction blocks reach several megabytes), the overhead can be defined as:

$$Overhead \approx \frac{(k+T) \times \text{size}(IHAVE) \times (\frac{Timeout}{T_{period}})}{\text{size}(Block)}$$

Where T_{period} is the frequency of periodic IHAVE broadcasts. By dividing $T_{timeout}$ with T_{period}, we will get the amount of times IHAVE messages are sent within a communication period. By multiplying the amount of

times we send the IHAVE messages, with the amount of nodes we have to sent those to, and the size of the IHAVE message, we will have the total amount of bandwidth spent on sending IHAVE messages.

By keeping the fanout k small, the network maintains metadata resilience across f channels for a fraction of the cost of a single data tree.

4.4.1 Numerical Illustration and Trade-off Analysis

To contextualize this model, consider a typical large-scale Kauri deployment with $N = 400$ validators ($f = 133$) and a block size of 250 KB. With a tree fanout of $k = 20$, each node maintains approximately 154 gossip neighbors ($k + f + 1$). Assuming an IHAVE payload of roughly 64 bytes (combining a 32-byte hash and a compressed BLS signature), the total metadata traffic generated per broadcast is approximately 9.8 KB. This represents less than 4% of the primary block payload. Even if IHAVE messages are broadcast periodically during the timeout window, the cumulative bandwidth consumption remains a small fraction of the system's capacity, as long as the period broadcast of IHAVE messages isn't too small.

This overhead represents a strategic cost to improve robustness. By investing a constant, predictable percentage of bandwidth into the gossip network, our protocol effectively converts the unpredictable, high-cost, consensus halting event of a view change into a manageable, low-cost background process.

4.5 Optimizing Gossip Cryptographic Operations

To prevent IHAVE message signature verification from becoming a CPU bottleneck, we can employ several optimization strategies. First, signature deduplication ensures that a node performs at most one verification per peer per block by caching identifiers. Second, threshold verification capping allows nodes to stop validating signatures once a sufficient number of honest sources ($f + 1$) are identified. Finally, lazy verification triggers cryptographic checks only if the primary tree times out, ensuring the gossip layer remains a silent, low-overhead safety net during steady-state operation. This ensures that the diagnostic layer does not reintroduce the CPU-bound bottlenecks that tree-based aggregation was designed to solve.

4.6 Fault Diagnosis Extension

Beyond simple data recovery, this architecture provides a framework for granular fault diagnosis. By monitoring the arrival of IHAVE messages across the redundant gossip channels, a validator can perform a discrepancy analysis to identify the root cause of a delay. If a node receives metadata from a quorum of gossip peers while its primary parent remains silent, it can conclude with high probability that some parent is either compromised or severely underperforming. This diagnostic capability could eventually be extended into mechanism where signed IHAVE messages serve as cryptographic evidence of data withholding to inform future reconfiguration rounds. With this, the approach would not only help avoid costly reconfigurations, but it would potentially help the system make more informed reconfiguration decisions by identifying potentially problematic tree members and avoiding putting them on the primary tree path.

5 Evaluation

In this section, we discuss how we plan to evaluate our proposal. The evaluation is designed to demonstrate that our hybrid approach maintains system liveness and throughput stability in scenarios where the original Kauri protocol would be forced into costly reconfiguration procedures.

5.1 Deployment Models

Several recent studies note that although blockchains are designed for wide decentralization, real-world node deployment often collapses into a small number of cloud providers or even a single region [23–25]. Empirical analyses of Bitcoin, Ethereum, and permissioned deployments show that validators frequently co-locate in the same availability zone, ASN, or datacenter [23]. Such clustering biases performance measurements: latencies

become unrealistically small, network partitions rarely occur, and the cost of dissemination is systematically underestimated.

Consequently, the evaluation of scalable BFT protocols must consider deployment diversity as a important factor. A BFT protocol that appears scalable in AWS us-east-1 may behave radically differently when validators are geographically dispersed across continents, ISPs, or resource heterogeneity. Based on this, our evaluation plan considers three deployment classes:

- *Multi-region cloud deployment:* validators deployed across at least three distinct geographic regions (e.g., EU, US, Asia), introducing realistic inter-region latencies and heterogeneous routing.
- *Hybrid cloud and edge:* a subset of validators running outside public clouds (e.g., university datacenters, edge devices, commodity hardware) to reflect heterogeneity typical in permissionless and fog-style architectures.
- *Emulated adverse network conditions:* use of network shaping (delay, bandwidth limits, loss) to generate controlled partial partitions and regional failures, following the "Minimum Decentralization Test" guidelines of [24].[1]

This ensures that our evaluation space spans the decentralization dimensions identified in the literature – geographic, provider, and network heterogeneity – and avoids the common pitfall of over-optimistic results obtained from homogeneous single-region cluster deployments.

5.2 Network Emulation

To achieve these evaluation conditions, we will use Kollaps [26], a decentralized network emulator that allows for the precise enforcement of end-to-end network properties within a controlled cluster. Unlike standard simulators, Kollaps enables the creation of a virtual topology where each validator is subjected to specific regional latencies and bandwidth constraints. This facilitates the "Minimum Decentralization Test" by reproducing the performance characteristics of a global network without the overhead of a multi-cloud physical deployment.

Beyond environment setup, we utilize Kollaps to inject dynamic faults into the active consensus tree. Specifically, we will simulate omission faults, where messages are selectively dropped by an internal node, and network partitions, which involve the complete loss of connectivity between a parent and its children. These controlled experiments allow us to measure the recovery time of our gossip pull mechanism in comparison to Kauri's standard global reconfiguration. In Kauri, any failure to reach a quorum via the tree requires a total halt of the consensus pipeline to search for a new robust tree—a process that may require up to $f + 1$ sequential attempts. We refer to this as a "global halt," and our evaluation aims to prove that gossip recovery along can bypass this bottleneck. If the Fault Detection extension is implemented, we also hope to improve the reconfiguration process itself by having the system recognize where the faults happened and having more localized reconfiguration instead of tree-wide.

5.3 Evaluation Metrics

We will extract several metrics to quantify the performance and robustness of the proposed protocol:

- *Throughput Stability:* We will measure the number of transactions processed per second over time. The objective is to demonstrate that our approach maintains a steady throughput when faults in the internal tree (that would cause reconfiguration in Kauri) are injected. Because the gossip layer can bypass faults in the internal tree, dissemination can still occur, and consensus can progress, maintaining throughput in scenarios where Kauri cannot.
- *Recovery Latency:* This is the time elapsed from the moment a node's local timeout expires until it successfully retrieves the missing block via a lateral PULL request.

[1]The "Minimum Decentralization Test" proposed in [24] states that an experimental deployment may only be considered minimally representative if validators are distributed across distinct decentralization strata (e.g., geographic regions, cloud providers, ASNs); i.e., concentrating all nodes within one datacenter or provider does not qualify as a valid evaluation scenario.

- *Reconfiguration Frequency:* We track how many global view-changes occur during a set period. A successful approach should show a significant reduction in reconfigurations compared to standard Kauri under the same fault scenarios. If the Fault Diagnosis extension is implemented, less reconfiguration attempts should occur in the event of a fault, because the system can make more informed decisions.
- *Communication and Bandwidth Overhead:* This metric quantifies the additional network load imposed by the auxiliary gossip layer. We will extract the total volume of metadata traffic—specifically the IHAVE signals and PULL/RESPONSE packets—and compare it against the primary block payload dissemination. This experimental data will be used to validate the theoretical model presented in Section 4, which predicts that the $k+T$ neighbor complexity results in negligible bandwidth consumption due to the high asymmetry between 32–64 byte control messages and multi-megabyte transaction blocks. We will conclude that the architecture is resource-efficient if the cumulative overhead remains below a marginal threshold (e.g., <1% of total network load) while still maintaining the metadata availability necessary for lateral recovery.
- *System Scalability:* We evaluate the protocol's performance as the committee size N increases, following the benchmarks established by Kauri for deployments up to 800 nodes. Specifically, we extract measurements for throughput and CPU utilization at internal nodes as N grows, identifying the point where the cost of maintaining $k + T$ gossip neighbors—particularly the cryptographic verification of IHAVE signatures—begins to impact the consensus pipeline. This allows us to identify the operational limits of our hybrid design. We will conclude that the protocol is scalable if it maintains a stable throughput advantage over star-based topologies at high N, proving that the added diagnostic layer does not reintroduce the centralized processing bottlenecks that structured tree dissemination was designed to mitigate.

The success of our approach will be determined by comparing the behavior of our gossip-assisted protocol against the original Kauri implementation in identical adverse environments. We will draw our conclusions based on the following logic:

1. *Liveness and Throughput Comparison During Faults:* Our approach is expected to outperform Kauri specifically during localized internal node faults or Byzantine withholding attacks affecting subtrees. While Kauri must trigger a global reconfiguration—stalling the progress of consensus for multiple rounds—our protocol allows honest nodes in compromised branches to recover data via the gossip network. We expect to see that as the committee size grows, the time saved by avoiding a reconfiguration increasingly outweighs the cost of the PULL.
2. *Fault Sensitivity:* To test the fault localization extension, we could vary the depth of the injected fault in the internal tree, to assess if our path aware timeouts, along with the IHAVE messages, correctly localize the recovery. If lower-level faults are resolved faster and without affecting higher branches, it proves that our diagnostic layer could be employed for quicker and localized recovery.

By correlating these metrics, we intend to provide empirical evidence that using gossip as a "safety net" for structured trees offers a good trade-off between resource efficiency and Byzantine resilience compared to existing models which only attempt reconfiguration blindly.

6 Work Schedule

The Gantt chart in Figure 5 illustrates the planned schedule for various tasks related to the MSc dissertation.

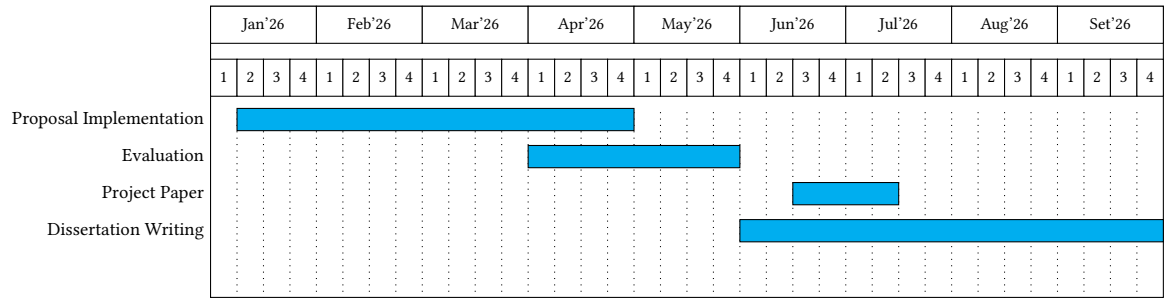

Figure 5: Planned Schedule

To provide context for the schedule presented in Figure 5, the following summary outlines the primary objectives for each project period:

- *Proposal Implementation:* This initial phase involves the practical development of the gossip-assisted recovery layer.
- *Evaluation:* During this stage, the system will be subjected to diverse deployment models and fault-injection scenarios using the Kollaps emulator to quantify performance metrics like throughput stability and recovery latency.
- *Project Paper:* This period is dedicated to documenting the experimental results and architectural findings in a formal technical paper suitable for academic submission.
- *Dissertation Writing:* The final phase encompasses the comprehensive documentation of the research, spanning from the initial drafts to the final revision of the MSc dissertation.

7 Conclusions

As permissioned blockchains are increasingly deployed in critical environments, the demand for protocols that can sustain high throughput at scale has led to the adoption of tree-based dissemination structures. Systems like Kauri demonstrate that organizing validators into dissemination and aggregation trees can successfully eliminate the single-leader bottleneck inherent in star-based protocols like HotStuff. However, this gain in scalability comes at the cost of topological fragility: in current architectures, a single faulty internal node can partition a large segment of the network, forcing the entire system to halt and undergo a costly global reconfiguration process.

In this work, we argue that the resilience of tree-based BFT protocols should not rely solely on the ability to find a new perfect tree, but rather on the ability to survive imperfections in the current one. We propose a hybrid architecture that supplements Kauri's structured efficient paths with a Byzantine-resilient gossip layer. By disseminating lightweight availability metadata (IHAVE messages) through redundant channels, our approach allows nodes to identify and bypass faulty parents locally, performing lateral data pulls without triggering a global view change.

This design effectively decouples localized network faults from system-wide availability. Our proposed evaluation plan, leveraging the Kollaps emulator, aims to empirically demonstrate that this "gossip safety net" can be maintained with negligible bandwidth overhead while significantly reducing the tail latency and consensus halts caused by reconfigurations. Ultimately, this work explores a new design space in scalable consensus, seeking to combine the deterministic performance of structured trees with the probabilistic resilience of epidemic protocols to create a BFT system that is both fast and robust.

Acknowledgments. This work was supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 and UID/PRR/50021/2025 and GLOG (financed by the OE with ref. LISBOA2030-FEDER-00771200).

Bibliography

[1] P. Li, G. Wang, X. Chen, F. Long, and W. Xu, "Gosig: A scalable and high-performance Byzantine consensus for consortium blockchains," in *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC)*. Renton, WA, USA: ACM, 2020, p. 223–237.

[2] J. Leitão, J. Pereira, and L. Rodrigues, "Gossip-based broadcast," in *Handbook of Peer-to-Peer Networking*. New York, NY, USA: Springer, 2010, pp. 83–116.

[3] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the 28th Symposium on Operating Systems Principles (SOSP)*, Koblenz, Germany, 2021, pp. 35–48.

[4] J. Leitao, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS)*. Beijing, China: IEEE Computer Society, 2007, p. 301–310.

[5] M. Ferreira, J. Leitão, and L. Rodrigues, "Thicket: A protocol for building and maintaining multiple trees in a P2P overlay," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems (SRDS)*, New Delhi, India, 2010, pp. 293–302.

[6] G. Danezis, E. K. Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus," 2022.

[7] K. Babel, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, A. Koshy, A. Sonnino, and M. Tian, "Mysticeti: Reaching the limits of latency with uncertified DAGs," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2025.

[8] S. Nakamoto, "Bitcoin: A Peer-to-Peer electronic cash system," Whitepaper, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 99, New Orleans, LA, USA, 1999, pp. 173–186.

[10] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*. Berlin, Heidelberg: Springer Science & Business Media, 2011.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[12] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, Toronto, ON, Canada, 2019, pp. 347–356.

[13] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Bft consensus with pipelined tree-based dissemination and aggregation," *ACM Trans. Comput. Syst.*, Sep. 2025, just Accepted. [Online]. Available: https://doi.org/10.1145/3769423

[14] T. Pereira, "Dynamic trees for Byzantine consensus protocols," Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal, 2024.

[15] H. C. D. Teixeira, "Self-Adapting BFT consensus: Leveraging heterogeneity in dissemination/aggregation trees," Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal, 2023.

[16] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*. Virtual Event: ACM, 2021, pp. 185–195.

[17] D. Malkhi and K. Nayak, "HotStuff-2: Optimal two-phase responsive BFT," *Cryptology ePrint Archive*, 2023.

[18] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, USA, 2022, pp. 2705–2718.

[19] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*. Bolton Landing, NY, USA: ACM, 2003, p. 298–313.

[20] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for Large-Scale Peer-to-Peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. Heidelberg, Germany: Springer, 2001, p. 329–350.

[21] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The design of a Large-Scale event notification infrastructure," in *Proceedings of the 3rd International COST264 Workshop on Networked Group Communication (NGC)*. London, UK: Springer, 2001, p. 30–43.

[22] J. Li, S. Goldberg, C. Borgs, and J. Chayes, "Robustness of Gossip-Based dissemination under Byzantine failures," in *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS)*. Beijing, China: IEEE, 2008.

[23] H. Lee *et al.*, "Taxonomy of centralization in public blockchain systems: A systematic literature review," *Journal of Network and Computer Applications*, 2020.

[24] A. Bia *et al.*, "Sok: A stratified approach to blockchain decentralization," *arXiv preprint arXiv:2211.01291*, 2024.

[25] A. Lebedev and V. Gramoli, "On the relevance of blockchain evaluations on bare metal," *arXiv preprint arXiv:2311.09440*, 2023.

[26] P. Gouveia, J. a. Neves, C. Segarra, L. Liechti, S. Issa, V. Schiavoni, and M. Matos, "Kollaps: Decentralized and dynamic topology emulation," in *Proceedings of the 15th European Conference on Computer Systems (EuroSys)*. Heraklion, Greece: ACM, pp. 1–16.