# Mahi-Mahi: Low-Latency Asynchronous BFT DAG-Based Consensus

Philipp Jovanovic*, Lefteris Kokoris-Kogias†, Bryan Kumara‡, Alberto Sonnino*†,
Pasindu Tennage§, Igor Zablotchi†

*University College London (UCL), †Mysten Labs, ‡The Alan Turing Institute, §EPFL

***Abstract*—We present Mahi-Mahi, the first asynchronous BFT consensus protocol that achieves sub-second latency in a wide-area network setting while processing over 100,000 transactions per second. Mahi-Mahi achieves such high performance by leveraging an uncertified structured Directed Acyclic Graph (DAG) to forgo explicit certification. This reduces the number of messages required to commit and the CPU overhead for certificate verification, significantly. Mahi-Mahi introduces a novel commit rule that enables committing multiple blocks in each asynchronous DAG round. Mahi-Mahi can be parametrized either with a 5 network hops commit delay, maximizing the commit probability under a continuously active asynchronous adversary, or with a 4 network hops commit delay, reducing latency under a more moderate and realistic asynchronous adversary. We demonstrate safety and liveness of Mahi-Mahi in a Byzantine context for all of these parametrizations. Finally, we evaluate Mahi-Mahi in a geo-replicated setting and compare its performance to state-of-the-art asynchronous consensus protocols, showcasing Mahi-Mahi's significantly lower latency.**

***Index Terms*—Asynchronous Consensus, BFT, Blockchain**

## I. Introduction

Applications that require Byzantine Fault Tolerant (BFT) consensus [7], such as blockchains [3], [7], [9], [10], [33], [43], often rely on protocols designed for the partially synchronous network model which aims to approximate mostly benign network conditions. However, protocols designed for partial synchrony lose liveness under asynchronous conditions, which can arise from poor connectivity, an active network adversary, or denial-of-service (DoS) attacks [29]. Asynchronous consensus protocols [13], [15], [40] address this issue by providing as much liveness as the network connectivity allows. To achieve this, these protocols sacrifice performance during periods of network synchrony, resulting in significantly higher latency compared to their partially synchronous counterparts. While state-of-the-art partially synchronous protocols can process over $100,000$ transactions per second with sub-second WAN latency [4], [5], current asynchronous protocols achieve similar throughput with latencies on the order of seconds [20]. This substantial latency drawback has made asynchronous consensus protocols less attractive for practical deployments.

Dual-mode protocols [28], [45] attempt to provide the best of both worlds by operating partially-synchronous consensus by default and reverting to a less performant asynchronous sub-protocol when network conditions become adverse. However, dual-mode protocols introduce complexity and are prone to errors, as they must maintain two separate protocol stacks and implement mechanisms to detect changing network conditions and switch between the two consensus modes. Additionally, they remain vulnerable to targeted attacks that can cause the protocol to switch constantly between the two modes [45]. Due to these drawbacks, no dual-mode protocol has yet been deployed in a production environment, to the best of our knowledge.

We therefore ask if it is possible to design a protocol that can simultaneously: (i) provide liveness under asynchronous network conditions, (ii) achieve performance comparable to state-of-the-art partially-synchronous consensus protocols, and (iii) maintain a simple design that allow for effective security analysis, implementation, and maintenance?

In this paper, we introduce Mahi-Mahi, a novel low-latency and high-throughput asynchronous consensus protocol that achieves these 3 goals. Mahi-Mahi accomplishes this through a combination of the following techniques. (1) While state-of-the-art asynchronous protocols, such as Tusk [20], operate over a certified Directed Acyclic Graph (DAG) and attempt to commit one leader block every 9 message delays, Mahi-Mahi utilizes an *uncertified* DAG as its core data structure. This approach eliminates the overhead associated with the reliable broadcast [13] of DAG vertices and allows Mahi-Mahi to commit most blocks with only five network hops, aligning with the theoretical results of Cordial Miners [32]. (2) Mahi-Mahi introduces a novel commit rule that enables the commitment of multiple leader blocks in each DAG round while ensuring safety and liveness in the presence of an asynchronous adversary. (3) Mahi-Mahi also explores more practical network assumptions and can be parameterized to further enhance average-case performance while maintaining liveness against an asynchronous adversary.

We implement Mahi-Mahi in Rust and show that it can process an impressive $350,000$ transactions per second in geo-distributed environments with $50$ nodes, all while keeping latency below $2$ seconds. Additionally, Mahi-Mahi can process $100,000$ transactions per second with latency below $1$ second. This achievement sets a new record in the realm of asynchronous consensus protocols and was previously only attainable by partially synchronous protocols [4], [5], [44]. We further show that Mahi-Mahi maintains the same throughput while reducing latency over recent state-of-the-art asynchronous BFT protocols, Tusk [20] and Cordial Miners [32] - achieving latency reductions of over 70% and 30%, respectively.

**Contributions.** We make the following contributions:

- We introduce Mahi-Mahi, the first asynchronous consensus protocol capable of committing with sub-second latency while maintaining high throughput. Notably, Mahi-Mahi is the first DAG-based asynchronous BFT consensus protocol capable of committing multiple leader blocks in each round.
- We provide detailed algorithms and formal security proofs for Mahi-Mahi, demonstrating its safety and liveness under an asynchronous network model.
- We conduct a latency analysis of Mahi-Mahi, evaluating its commit probability under various network conditions.
- We present an implementation and evaluation of Mahi-Mahi, comparing it to other state-of-the-art protocols and demonstrating that Mahi-Mahi achieves the lowest commit latency among available asynchronous BFT protocols.

## II. System Overview

### A. Threat model, goals, and assumptions

We consider a message-passing system with $n = 3f + 1$ validators. An adversary can adaptively corrupt up to $f$ validators, referred to as *Byzantine*, who may deviate arbitrarily from the protocol. The remaining honest validators, follow the protocol. The adversary is computationally bounded, ensuring that standard cryptographic properties such as the security of hash functions and digital signatures hold. The communication network is asynchronous and messages can be delayed arbitrarily, but messages among honest validators are eventually delivered. Given these conditions Mahi-Mahi is *live*, meaning honest validators eventually commit transactions.

In addition to the asynchronous model, we analyze Mahi-Mahi under the *random network model* [21], [20], a variant of the asynchronous network model. While the asynchronous model makes the worst-case assumption that the adversary has perpetually full control over the message schedule (*i.e.,* the order in which messages are received by honest validators), the random network model assumes that the message schedule is random (see Section II-C). We analyze Mahi-Mahi with parameters optimized for the random network model, representing an evaluation that characterizes practical asynchronous network settings. Our empirical results show that this parameterization generally outperforms a version of Mahi-Mahi configured for maximum resilience against an active asynchronous adversary, all while maintaining safety and liveness guarantees.

Mahi-Mahi solves Byzantine Atomic Broadcast (BAB) [16], enabling validators to reach consensus on a sequence of messages. According to the FLP impossibility result [38], BAB cannot be solved deterministically in an asynchronous setting. To address this, we employ a global perfect coin to introduce randomization, similar to previous work [11], [14], [31], [35]. This coin can be constructed using an adaptively secure threshold signature scheme [6], [12], with the distributed key setup performed under asynchronous conditions [1], [2], [23]

Each validator $v_k$ broadcasts messages by invoking a_bcast$_k(m, q)$, where $m$ is the message and $q \in \mathbb{N}$ is a sequence number. Every validator $v_i$ has an output a_deliver$_i(m, q, v_k)$, where $m$ is the message, $q$ is the sequence number, and $v_k$ is the identity of the validator that initiated the corresponding a_bcast$_k(m, q)$. Mahi-Mahi implements a BAB protocol and guarantees the following [31]:

- **Validity:** If an honest participant $v_k$ calls a_bcast$_k(m, q)$, then every honest participant $v_i$ eventually outputs a_deliver$_i(m, q, v_k)$, with probability 1.
- **Agreement:** If an honest participant $v_i$ outputs a_deliver$_i(m, q, v_k)$, then every honest participant $v_j$ eventually outputs a_deliver$_j(m, q, v_k)$ with probability 1.
- **Integrity:** For each sequence number $q \in \mathbb{N}$ and participant $v_k$, an honest participant $v_i$ outputs a_deliver$_i(m, q, v_k)$ at most once, regardless of $m$.
- **Total Order:** If an honest participant $v_i$ outputs a_deliver$_i(m, q, v_k)$ and a_deliver$_i(m', q', v'_k)$ where $q < q'$, all honest participants output a_deliver$_j(m, q, v_k)$ before a_deliver$_j(m', q', v'_k)$.

### B. Intuition behind the Mahi-Mahi design

Mahi-Mahi builds upon DAG-based consensus protocols [5], [32], [20], [31] that achieve high throughput by processing $O(n)$ blocks per round. While maintaining the throughput advantages of DAG-based protocols, Mahi-Mahi focuses on reducing the latency in asynchronous state machine replication. It introduces novel techniques to decrease the number of message delays required for block commitment and explores more practical network assumptions to further improve average-case performance.

State-of-the-art asynchronous protocols, such as Tusk [20], operate over a certified DAG and commits one leader block every three certified rounds, requiring three message delays to certify each round. This results in at least nine message delays (nine network hops) per leader block. In contrast, Mahi-Mahi operates over an uncertified DAG by forgoing the reliable broadcast [13] of DAG vertices, committing most blocks with only five network hops which matches the theoretical results of Cordial Miners [32]. Uncertified DAG approach significantly reduces both bandwidth and computation cost.

Uncertified DAGs, however, creates the first challenge (**Challenge 1**): handling equivocations practically. Unlike certified DAG protocols [20], [27], [31], [45], [49], Mahi-Mahi cannot rely on certificates to prevent equivocations, necessitating the design of a novel commit rule that is immune to equivocations. Cordial Miners [32], an existing uncertified DAG based protocol, also faces the same challenge and addresses it by eventually excluding Byzantine validators that provably equivocate. Although theoretically sound, eventually excluding Byzantine validators takes a very long time under asynchrony, hindering the benefits of asynchronous liveness.

While having five rounds between leaders provides a good probability of committing in asynchronous conditions, it also results in relatively high latency. Although it can be shown that we can implement a commit rule that operates in just three message delays (see Section C), this approach would not work under asynchrony as it would introduce significant latency variance and in the worst case lose liveness. Instead, we focus on addressing (**Challenge 2**): developing a commit rule that effectively reduces average-case latency without sacrificing worst-case liveness. We find that it is possible to reduce the number of network hops to four, achieving a balance between average-case latency in random network conditions and worst-case latency in the classic asynchronous model.

Even with this enhancement, committing only once every four message delays still results in significant latency variance for transactions that are not part of a committed leader block's causal history. A primary goal for Mahi-Mahi is to commit multiple blocks in each round, which would ensure that the system's tail latency aligns more closely with the four-message delay. To achieve this, we need to address (**Challenge 3**): commit every block without relying on a sufficient round difference between leader blocks. If Mahi-Mahi were to adopt a traditional recursive commit rule [20], [32], which mandates that each leader block always references all previous leader blocks in their causal history, it would at best be able to commit once every four rounds. However, Mahi-Mahi recognizes that this causal reference is only necessary when there is no sufficient evidence to directly commit a block, which is not the typical case (Section V). This insight indicates that the recursive commit rule used in prior research is overly conservative in its approach to skipping blocks, leading to unnecessary delays, particularly during benign node crashes, which are immediately identifiable. To resolve this issue, we propose a new commit rule capable to promptly determine for each block whether it can be committed or discarded.

Section III-B presents the Mahi-Mahi commit rule that addresses these challenges. Hence, Mahi-Mahi is the first asynchronous BFT consensus protocol capable of committing multiple blocks per round in the average case, while ensuring both safety and liveness in the asynchronous and random network models.

### *C. Structure of the Mahi-Mahi DAG*

We present the structure of the Mahi-Mahi DAG, building an uncertified DAG that offers similar guarantees to a certified DAG, as shown in related work [5], [20], [32].

The Mahi-Mahi protocol operates in a sequence of logical *rounds*. In each round, every honest validator proposes a unique signed *block*, while Byzantine validators may attempt to equivocate by sending multiple blocks or none at all. During a round, validators receive transactions from users and blocks from other validators, which they refer into their proposed blocks. A block includes hash references to blocks from prior rounds, starting with their most recent block, and adds *fresh transactions* not yet included in preceding blocks. Once a block references at least $2f+1$ blocks from the previous round, the validator signs it and broadcasts it. Clients send transactions to a validator, who adds them to their blocks. If a transaction does not finalize quickly enough, the client sends it to a different validator.

**Block creation and validation.** A block must include at least the following elements: (1) the author $A$ of the block and their signature on the block contents; (2) a round number $R$; (3) a list of transactions; (4) at least $2f+1$ distinct hashes of valid blocks from the previous round $R-1$, along with potentially others from prior rounds; and (5) a share of a global perfect coin. As already mentioned the coin can be reconstructed from any $2f+1$ shares.

A block is *valid* if: (1) the signature is valid and the author $A$ is part of the validator set; (2) all hashes point to distinct valid blocks from previous rounds, and the sequence of past blocks includes $2f+1$ blocks from the previous round $R-1$; and (3) the share of the global perfect coin is valid[1]. Honest validators only include valid blocks into their DAG and discard invalid ones. Furthermore, honest validators only include hashes of blocks once they have downloaded their entire causal history, ensuring that they have successfully validated the block's causal history.

**Rounds and waves.** Figure 1 (left) illustrates an example of a Mahi-Mahi DAG with four validators, $(v_0, v_1, v_2, v_3)$ when parametrized to commit in 5 rounds. For the practically efficient 4-round Mahi-Mahi, the second Boost round is omitted.

In its 5-rounds configuration, Mahi-Mahi defines a *wave* of 5 rounds for every block. The first round (Propose) includes the blocks that the wave attempts to commit ($P_0$, $P_1$, $P_2$, $P_3$) and the equivocating block $P_1'$. The second and third rounds (Boost) act as a buffer, helping to propagate these blocks to as many validators as possible. In the fourth round (Vote), every block serves as a *vote* for the first block of the Propose that it encounters when performing a depth- first search following the block hash references. In the example shown in this figure, blocks $V_0$, $V_1$, and $V_2$ are votes for $P_0$, $P_1$, $P_2$ (but not for $P_1'$ and $P_3$), while block $V_3$ is a vote for $P_0$, $P_1$, $P_2$, and $P_3$ (but not for $P_1'$). The procedure $\textsc{IsVote}(\cdot)$ of Algorithm 2 (Section A) formally defines a vote. The fifth round (Certify) reveals which blocks from the Propose round have been implicitly certified. A block from the Propose round is considered *certified* or *has a certificate* if a block from the Certify round contains in its causal history at least $2f+1$ blocks from the Vote round that are a vote for the block. In this example, blocks $C_0$, $C_1$, $C_2$, and $C_3$ serve as certificates for $P_0$, $P_1$, and $P_2$. This round also opens the global perfect coin, which the decision rule (Section III-B)

[1] Each individual share of the coin can be independently verified if the coin is implemented through a threshold signature.

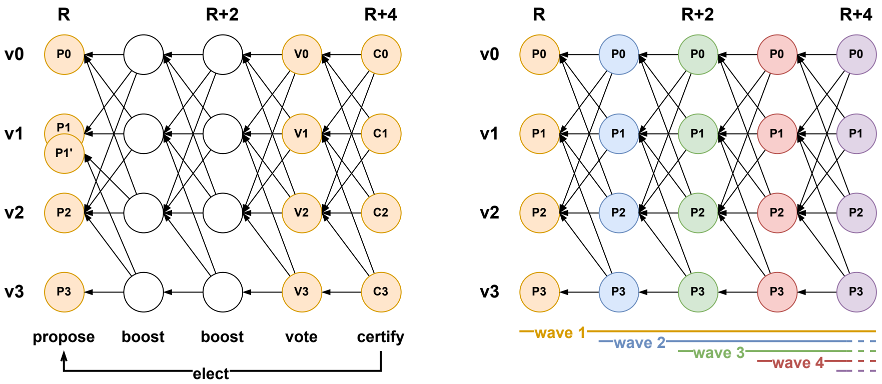

Fig. 1: The structure of the Mahi-Mahi DAG. Left: The structure of a wave, consisting of 5 rounds (Propose, Boost, Boost, Vote, Certify). Right: Waves patterns in the Mahi-Mahi protocol (each round starts a new overlapping wave).

uses to circumvent the FLP result and to commit blocks under asynchrony by electing some of the Propose blocks as *leaders*. Similar to related work [31], [20] this strategy selects leaders "after the fact" to deter a network adversary from strategically delaying leader blocks so that they are not referenced by blocks of the Vote round.

As illustrated in Figure 1 (right), Mahi-Mahi initiates a new wave every round. The rounds of each wave follow a consistent pattern: Propose round: $R$, Boost round: $R+1$, Boost round: $R+2$, Vote round: $R+3$, and Certify round: $R+4$. This pattern repeats continuously, with each new round starting a fresh wave. Algorithm 3 of Section A formally defines a wave.

**Random network model.** We analyze Mahi-Mahi in the standard asynchronous network model, as well as in the random network model [20]. In the asynchronous model, the adversary chooses which blocks are received by each honest validator at each round. In contrast, the random network model assumes that at each round $R+1$, an honest validator receives and references valid round-$R$ blocks from a *uniformly random* subset of $2f+1$ validators. Section C provides further details and analyses the commit probability of Mahi-Mahi under both network models.

## III. The Mahi-Mahi Protocol

In this section, we present Mahi-Mahi configured with a wavelength of 5 rounds. A configuration of Mahi-Mahi with a wavelength of 4 rounds operates similarly, but omits one Boost round, and addresses **challenge 2** of Section II as we empirically show in Section V. Algorithm 1 specifies the Mahi-Mahi main protocol, Algorithm 3 the Mahi-Mahi decider instance, and algorithm 2 contains various DAG helper functions.

### *A. Leader Slot*

Mahi-Mahi leverages a perfect global coin to define several *leader slots* per round. A leader slot is a tuple (validator, round) and can be either empty or contain the validator's proposal for the respective round. If the validator is Byzantine, the slot may also contain more than one (equivocating) block. In line with related work [5], the slot can assume one of three states: `commit`, `skip`, or `undecided`. All slots are initially set to `undecided` and the goal of the protocol is to classify them as `commit` or `skip`. The number of leader slots instantiated per round and the number of boost rounds can be configured (Section V explores different configurations).

### *B. The Mahi-Mahi decision rule*

We present the decision rule of Mahi-Mahi leveraging an example protocol run. Section A provides detailed algorithms and Section B provides a complete step-by-step protocol execution. Figure 2 illustrates an example of a local view of a Mahi-Mahi validator, in a system with four validators, $(v_0, v_1, v_2, v_3)$ and parameterized with two leader slots per round. In this example, we refer to blocks using the notation $B_{(v_i,R)}$, where $v_i$ is the issuing validator and $R$ is the block's round.

All proposer slots are initially in the `undecided` state. The validator holds the portion of the DAG depicted in Figure 2 and attempts to classify as many blocks in the leader slots as possible as either `commit` or `skip`.

**Step 1: Determine the leader slots.** The validator begins by reconstructing the global perfect coin to determine the leader slots for each round. As shown in Figure 1 (Left), the coin shares embedded in round $R+4$[2] (the Certify round) deterministically establish the leader slots for round $R$.

In this example, the validator reconstructs the coin from any set of $2f+1$ blocks from round $R+4$ of a wave, then uses it as a seed to deterministically select two leader slots for round $R$: $L_{1a}$ and $L_{1b}$, as illustrated in Figure 2. The coin also imposes an order between these two slots: by convention, $L_{1a}$ is the *first* leader slot and $L_{1b}$ is the *second* leader slot of round $R$. The validator repeats this process for every subsequent wave, determining leader slots $L_{2a}$ and $L_{2b}$ from the coin shares in round $R+5$, $L_{3a}$ and $L_{3b}$ from those in round $R+6$, and so on. The validator then sorts these leader slots in descending order: $[L_{6b}, L_{6a}, L_{5b}, L_{5a}, L_{4b}, L_{4a}, L_{3b}, L_{3a}, L_{2b}, L_{2a}, L_{1b}, L_{1a}]$.

This mechanism of determining multiple, potentially empty, leader slots from a global perfect coin is the first step towards addressing **challenge 3** (Section II). Even if validators have different views of the DAG, they will still

[2] Or $R+3$ when Mahi-Mahi is configured with a wave length of 4 rounds.

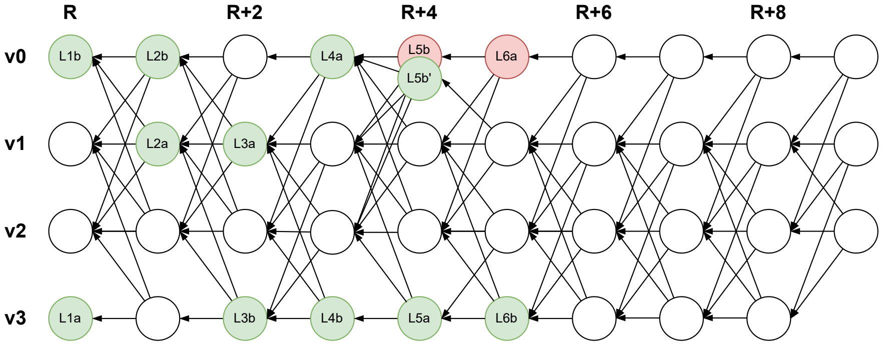

Fig. 2: Example execution with 4 validators, wave length of 5 rounds and 2 leader slots per round.

deterministically decide the same leader slots, in the same order, for a given round—regardless of whether they have a block for that slot in memory. This enables Mahi-Mahi to achieve low latency by electing more than one leader per round and using these slots to order the causally history.

**Step 2: Direct decision rule.** The validator attempts to classify each slot, even those without a block as either `commit` or `skip`. To do so, the validator processes each slot individually, starting with the highest ($L_{6b}$), applying the Mahi-Mahi *direct decision rule*. The validator classifies a block $B$ in a slot as `skip` if it observes $2f+1$ blocks from the subsequent Vote round that do not encounter $B$ when performing a depth-first search following the blocks' references, and as `commit` if it observes $2f+1$ *certificates* over it. As discussed in Section II, a certificate over a block $B$ is a block from the Certify round that references at least $2f+1$ blocks from the Vote round, each of which encounter $B$ when performing a depth-first search starting at the voting block. Otherwise, the validator leaves the slot as `undecided` (for now).

In this example, the validator targets $L_{6b}$ first. It observes that $B_{(v_0,R+9)}$, $B_{(v_1,R+9)}$, and $B_{(v_2,R+9)}$ are certificates for $L_{6b}$. Therefore, it classifies $L_{6b}$ as `commit`. Section V shows that this scenario is the most common (in the absence of an asynchronous adversary) and results in the lowest latency. The validator then targets $L_{6a}$ and observes that $B_{(v_1,R+8)}$, $B_{(v_2,R+8)}$, and $B_{(v_3,R+8)}$ do not vote for it. Therefore, it classifies $L_{6a}$ as `skip`. The presence of $2f+1$ blocks from the Vote round that do not vote for a block ensures that it will never be certified, and will thus never be committed by other validators with a potentially different local view of the DAG. Section V shows that this rule allows Mahi-Mahi to promptly skip (benign) crashed leaders to minimize their impact on the protocol's performance.

Malicious validators may attempt to equivocate by creating multiple blocks for the same slot, such as $L_{5b}$ and $L'_{5b}$ in this example. However, the direct decision rule ensures that at most one of these blocks will be classified as `commit`, while the others will be classified as `skip`. In this example, the block $B_{(v_0,R+7)}$ is a vote for $L_{5b}$ (and not for $L'_{5b}$) as it is the first block of the slot encountered when performing a depth-first search starting at $B_{(v_0,R+7)}$ and recursively following all blocks in the sequence of block hashes. Conversely, $B_{(v_1,R+7)}$, $B_{(v_2,R+7)}$, and $B_{(v_3,R+7)}$ are votes for $L_{5b'}$.

This strategy addresses **challenge 1**. Even though Byzantine validators might equivocate by creating multiple blocks per slot, the causal references defined by the DAG allow the validator to interpret blocks from the Certify round as certificates for blocks from the Propose round. Coupled with the rule that honest validators author at most one block per round, this ensures that at most one block per slot receives a certificate, while all possible other equivocating blocks are skipped. In essence, Mahi-Mahi embeds the execution of a Byzantine consistent broadcast [13] into the DAG.

**Step 3: Indirect decision rule.** In the (rare) case where the direct decision rule cannot classify a slot, the validator uses the Mahi-Mahi *indirect decision rule*. This rule looks at future slots to decide about the current one. First, it finds an *anchor*. This is the first block of the next wave (that is, the earliest slot with a round number $R' > R+4$) that is either still classified as `undecided` or already classified as `commit`. If the anchor is `undecided`, the validator marks the current slot as `undecided`. If the anchor is `commit`, the validator checks if it references at least one certificate over the current slot. If it does, the validator marks the current slot as `commit`. If it does not, the validator marks the current slot as `skip`. Section C shows the direct and indirect decision rules are consistent, namely if one validator direct commits a block no honest validators will indirect skip it (and vice versa).

In this example, the validator fails to classify $L_{1a}$ using the direct decision rule as there is only one certificate for $L_{1a}$ and thus searches for its anchor. Since $L_{6a}$ has been classified as `skip`, it cannot serve as an anchor; therefore, $L_{6b}$ becomes the anchor for $L_{1a}$. Given that block $B_{(v_3,R+4)}$, which serves as a certificate for $L_{1a}$, is referenced in $L_{6b}$'s causal history, the validator classifies $L_{1a}$ as `commit`.

This rule is the last step to solving **challenge 3**. It allows the validator to indirectly decide on a block by leveraging the earliest anchors rather than waiting for the next leader slot which may come much later. This enables Mahi-Mahi to eliminate the need for non-leader blocks between leader slots, achieving low latency by electing leader slots every round.

**Step 4: Leader slots sequence.** After processing all slots, the validator derives an ordered sequence of the blocks contained in the leader slots. It then iterates over this sequence, committing all slots marked as `commit` and skipping all slots marked as `skip`. This process continues until the validator encounters the first `undecided` slot. As shown in Section C, this commit sequence is safe, and eventually, all slots will be classified as either `commit` or `skip`.

In the example shown in Figure 2, the leader sequence output by the validator is $[L_{1a},\ L_{1b},\ L_{2a},\ L_{2b},\ L_{3a},\ L_{3b},\ L_{4a},\ L_{4b},\ L_{5a},\ L'_{5b},\ L_{6b}]$ . Section B provides a detailed walkthrough of the decision rule applied to the example DAG in Figure 2, guiding the reader step-by-step through deriving this commit sequence.

**Step 5: Commit sequence.** Following the approach introduced by DagRider [31], the validator linearizes the blocks within the sub-DAG defined by each leader block by performing a depth-first search. If a block has already been linearized by a previous leader slot, it is not re-linearized. The validator processes leader slots sequentially, ensuring that all blocks are included in the final commit sequence in the correct order, according to their causal dependencies. The procedure LinearizeSubDags$(\cdot)$ of Algorithm 2 (Section A) formally describes this.

In this example, $L_{1a}$ and $L_{1b}$ do not define any sub-DAG (because the example begins at round $R$) and are thus directly added to the commit sequence. Next, $L_{2a}$ defines the sub-DAG $\{L_{1b}, B_{(v_1,R)}, B_{(v_2,R)}, L_{2a}\}$, which is linearized as $[B_{(v_1,R)},\ B_{(v_2,R)},\ L_{2a}]$ since $L_{1b}$ is already part of the commit sequence. The validator continues this process for each leader, linearizing the sub-DAGs defined by $L_{3b}$, then $L_{3a}$ and so forth. The final commit sequence is $[\ L_{1a},\ L_{1b},\ B_{(v_1,R)},\ B_{(v_2,R)},\ L_{2a},\ L_{2b},\ B_{(v_2,R+1)},\ L_{3a},\ B_{(v_3,R+1)},\ L_{3b},\ B_{(v_0,R+2)},\ B_{(v_2,R+2)},\ L_{4a},\ L_{4b},\ B_{(v_1,R+3)},\ B_{(v_2,R+3)},\ L_{5a},\ L'_{5b},\ B_{(v_1,R+4)},\ B_{(v_2,R+4)},\ L_{6b}\ ]$.

## IV. Implementation

We implemented a networked Mahi-Mahi validator in Rust by forking the Mysticeti codebase [34], consisting of about $14,000$ LOC. Our implementation utilizes `tokio` [48] for asynchronous networking and employs raw TCP sockets for communication. We rely on `ed25519-consensus` [24] for asymmetric cryptography and `blake2` [41] for cryptographic hashing. Furthermore, we implemented Cordial Miners [32], a state-of-the-art DAG-based asynchronous consensus protocol, using the same system components. This allowed us to conduct a comparative evaluation with Mahi-Mahi, see Section V. Since the Cordial Miners paper lacks both implementation and evaluation, we believe our implementation and evaluation are additional contributions of our work. We are open-sourcing both our implementations of Mahi-Mahi and Cordial Miners, along with our orchestration tools, to ensure reproducibility of our results[3].

## V. Evaluation

We evaluate the throughput and latency of Mahi-Mahi through experiments conducted on Amazon Web Services (AWS), demonstrating its performance improvements over the state-of-the-art. We evaluate Mahi-Mahi with different parametrizations, with a wave length of $4$ and $5$ and with different numbers of leaders per round.

We compare Mahi-Mahi with Tusk [20], as an example of certified DAG-based consensus protocol, and Cordial Miners [32], as an example of an uncertified DAG-based protocol. We choose these protocols because, to the best of our knowledge, Tusk has shown the highest throughput among all published and implemented asynchronous BFT protocols when evaluated in a geo-distributed environment. Cordial Miners, while lacking an implementation and evaluation, theoretically proves excellent latency bounds and is the protocol most similar to Mahi-Mahi. We also considered a performance comparison with other asynchronous consensus protocols, including Pace [50], Fin [25], ParBFT [18], and SQ [47], but decided against them. The reasons for this decision is that their implementations are either closed-source, only capable of handling a limited number of block proposals (leading to crashes under sustained load), or unable to operate in a WAN environment (resulting in deadlocks after a few seconds).

Our evaluation demonstrates the following 5 claims:

**C1** Mahi-Mahi has similar throughput and lower latency than the baseline state-of-the-art protocols when operating in synchronous network conditions.

**C2** Mahi-Mahi scales well by maintaining high throughput and low latency as the number of validators increases.

**C3** Mahi-Mahi has a similar throughput to, and lower latency than, Cordial Miners, when operating in the presence of (benign) crash faults.

**C4** Mahi-Mahi latency decreases when increasing the number of leader slots per round (up to 3 leaders per round).

**C5** Mahi-Mahi parametrized with a wave length of $4$ rounds has lower latency in our geo-replicated network than when configured with a wave length of 5 rounds.

Note that evaluating the performance of BFT protocols in the presence of Byzantine faults is an open research question [8], and state-of-the-art evidence relies on formal proofs (presented in Section C ). While there is a need to robustly tolerate Byzantine faults, we note that they are rare in observed delegated proof-of-stake blockchains, as compared to crash faults which occur commonly [5].

### A. Experimental Setup

We deploy Mahi-Mahi on AWS, using `m5d.8xlarge` instances across 5 different AWS regions: Ohio (us-east-2), Oregon (us-west-2), Cape Town (af-south-1), Hong Kong (ap-east-1), and Milan (eu-south-1). Validators are distributed across those regions as equally as possible. Each machine provides 10 Gbps of bandwidth, 32 virtual CPUs (16 physical cores) on a 3.1 GHz Intel Xeon Skylake 8175M, 128 GB memory, and runs Linux Ubuntu server 22.04.

In the following, *latency* refers to the time elapsed from the moment a client submits a transaction to when it is committed by the validators, and *throughput* refers to the

[3] https://github.com/PasinduTennage/mahi-mahi-consensus (commit `13a2819800ad3b192ab3c54dfe3ec4d0b34ae361`)

number of transactions committed per second. Each data point is the average latency of 3 runs and the error bars represent one standard deviation (error bars are sometimes too small to be visible on the graph). We instantiate several geo-distributed benchmark clients within each validator submitting transactions in an open loop model, at a fixed rate. We experimentally increase the load of transactions sent to the systems, and record the throughput and latency of commits. As a result, all plots illustrate the steady-state latency of all systems under low load, as well as the maximal throughput they can provide after which latency grows quickly. Transactions in the benchmarks are arbitrary and contain 512 bytes. Unless stated otherwise, we configure Mahi-Mahi with 2 leaders per round. In the following graphs, we refer to Mahi-Mahi with a wave length of 5 as Mahi-Mahi-5 and Mahi-Mahi with a wave length of 4 as Mahi-Mahi-4.

### *B. Benchmark under ideal conditions*

We assess the performance of Mahi-Mahi under normal, failure-free conditions in a wide-area network (WAN). Figure 3 presents the performance results of Mahi-Mahi in a geo-replicated setting, comparing both a small committee of 10 validators and a large committee of 50 validators.

For a committee of 10 nodes, all three systems—Tusk, Cordial Miners, and Mahi-Mahi—reach a peak throughput of approximately 100k-130k transactions per second (tx/s). However, their latencies vary significantly. Tusk and Cordial Miners have average latency of 3.5s and 1.5s, respectively. In contrast, Mahi-Mahi with a wave length 5 has a latency of 1.1s, representing a reduction of 68% compared to Tusk and 27% compared to Cordial Miners. Mahi-Mahi with wave length 4 has a latency of 0.9s, representing a substantial reduction of 74% compared to Tusk and 40% compared to Cordial Miners. Tusk's higher latency stems from its certified DAG architecture, requiring at least 9 network hops to commit a block. Cordial Miners bypasses DAG certification, but can only commit one leader every 5 network hops. By contrast, Mahi-Mahi operating with wave length 5 consistently commits multiple blocks. Mahi-Mahi operating with wave length 4 further reduces latency as it commits blocks after 4 message delays. These results validate claim **C1**.

For a large committee of 50 nodes, Figure 3 shows that the throughput of Cordial Miners and Mahi-Mahi exceeds 350,000 transactions per second (tx/s), while Tusk's throughput remains around 125,000 tx/s. This perhaps surprising increase in throughput occurs because our Mahi-Mahi's validator implementation is optimized for large networks and does not fully utilize all available resources (network, disk, CPU) when deployed with smaller committee sizes. Consequently, adding more validators improves resource multiplexing, boosting Mahi-Mahi's performance. Additionally, as the committee size grows, the number of blocks per round increases, thus a larger number of blocks are included in the causal history of elected leader blocks, without incurring additional network hops. Unlike Tusk, both Cordial Miners and Mahi-Mahi experience no significant CPU overhead as the committee size increases, and bandwidth does not become a bottleneck at these throughput levels. However, we do not expect further throughput gains by increasing the committee size beyond 50 nodes (such experiments would be prohibitively expensive). As expected, Cordial Miners and Mahi-Mahi share nearly identical throughput since both rely on the same DAG implementation, and throughput is determined by the efficiency of the DAG propagation layer.

In terms of latency, Tusk and Cordial Miners achieve average latency of 3.5s and 2.6s, respectively. Mahi-Mahi parametrized with a wave length of 5 has a latency of 2s (at 350,000 tx/s), which is a 42% reduction compared to Tusk and a 23% reduction compared to Cordial Miners. Mahi-Mahi with a wave length 4 has a latency of 1.5s, which is a 57% reduction compared to Tusk and a 42% reduction compared to Cordial miners. These results validate our claim **C2**. Comparing the two versions of Mahi-Mahi in those two experiments also validates our claim **C5**.

### *C. Performance under faults*

Figure 4 depicts the performance of all systems when a committee of 10 validators suffers 3 crash-faults (the maximum that can be tolerated for this committee size).

We observe that all three systems achieve a throughput of approximately $35,000$-$40,000$ tx/s. Tusk and Cordial Miners record a latency of around 7s and 1.7s, respectively. Mahi-Mahi records a latency of 0.95s and 0.85s when running with a wave length 5 and 4, respectively. We observe that despite the presence of faulty validators, the DAG continues to collect and disseminate transactions. The reduction in throughput seen in Figure 4, compared to Figure 3, can be attributed to two primary factors: (1) the loss of capacity due to faulty validators, and (2) the higher frequency of missing elected leader blocks, which leads to increased commit delays. Mahi-Mahi maintains a latency advantage of approximately 50% over Cordial Miners, thanks to its direct skip rule (Section III), which allows Mahi-Mahi to bypass faulty leaders roughly 2 rounds earlier than Cordial Miners. Thus, our claim Item **C3** holds.

### *D. Impact of the number of leader slots per round*

Finally, we assess the impact of multiple leaders on Mahi-Mahi's performance. We evaluate how Mahi-Mahi configured with a wave length of 4 rounds performs with 1, 2, and 3 leaders under both normal conditions and scenarios involving 3 crash faults. Due to space constraints, the graph for this experiment is presented in the extended version of the paper [30] (see Fig. 5).

We observe a notable reduction in average latency as the number of leaders increases. Specifically, when the number of leaders increases from 1 to 3, Mahi-Mahi's average latency decreases by approximately 40ms in the ideal scenario, and by approximately 100ms in the crash failure scenario. This improvement arises because having more leaders per round increases the number of blocks committed directly by leaders,

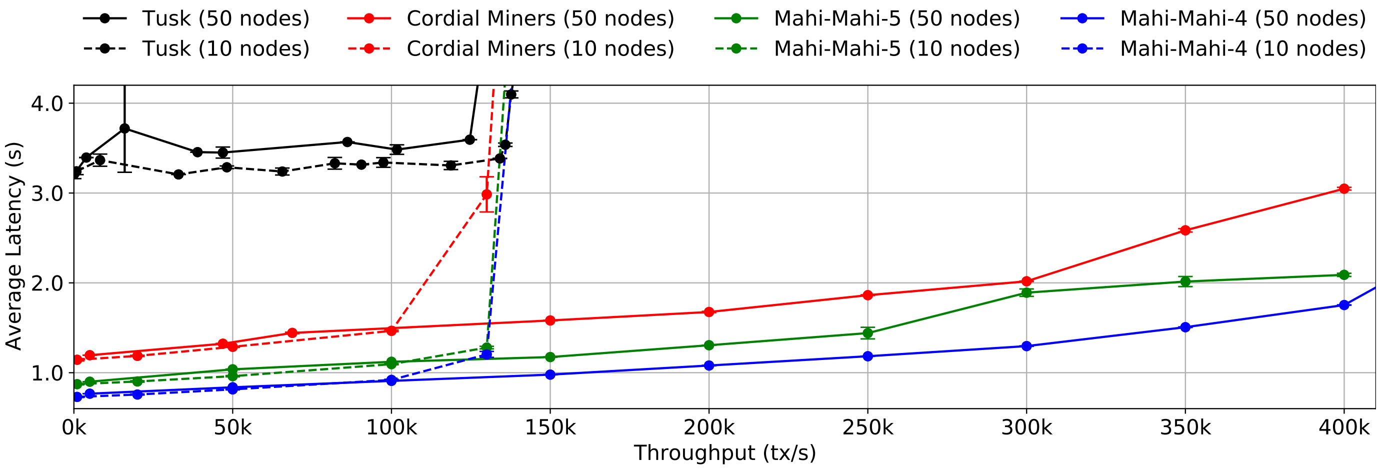

Fig. 3: Comparative throughput-latency performance of Mahi-Mahi, Tusk, and Cordial Miners. WAN measurements with 10 and 50 validators. No validator faults. 512B transaction size.

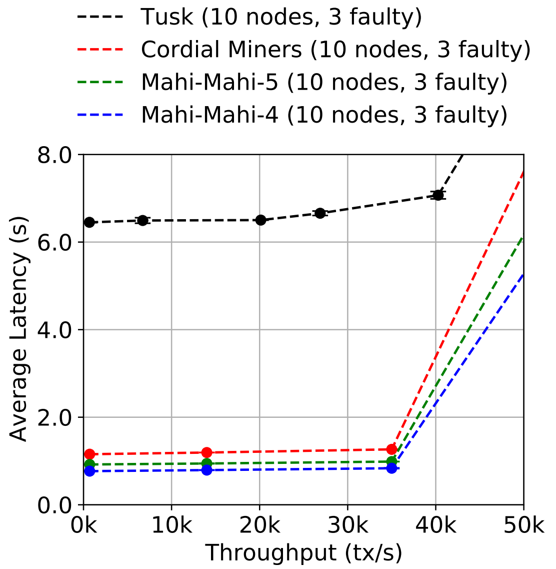

Fig. 4: Comparative throughput-latency of Mahi-Mahi, Tusk, and Cordial Miners. WAN measurements with 10 validators, three faults with 512B transactions size.

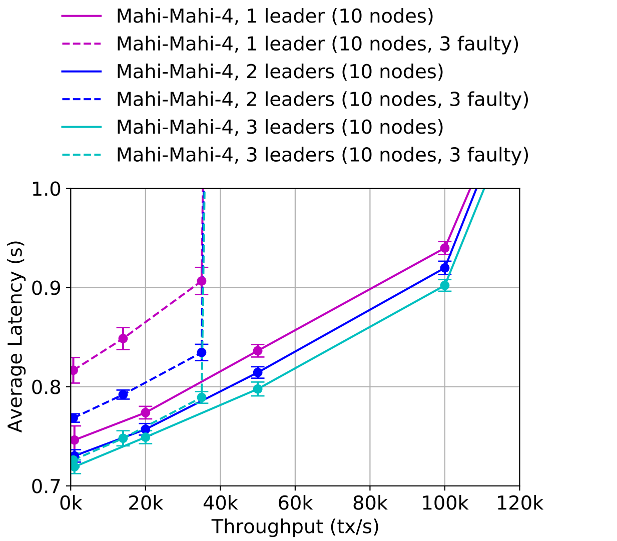

Fig. 5: Impact of the number of leaders per round in Mahi-Mahi. WAN measurements with 10 validators. Zero and three faults. 512B transaction size.

rather than through the causal history of previous leader blocks. These findings validate our claim Item **C4**. Increasing the number of leaders beyond 3 did not further decrease latency. This is due to the lower likelihood of directly committing, which may cause head-of-line blocking and delays the commitment of future leaders.

## VI. Related Work

**Uncertified DAG-based consensus protocols.** The system most similar to Mahi-Mahi is Cordial Miners [32]. Like Mahi-Mahi, Cordial Miners operates over an uncertified DAG, where each vertex represents a block that is disseminated with best-effort to all peers [26]. The primary distinction between the two lies in their commit rules. Cordial Miners can commit at most one leader block every five rounds, which leads to significantly higher latency for transactions not included in that leader block. In contrast, Mahi-Mahi's commit rule allows for a configurable number of blocks to be committed in each round, increasing the number of blocks committed per round and reducing the latency for most transactions. Mahi-Mahi commits more blocks directly through leaders, rather than relying on the causal history of previous leader blocks. Additionally, Cordial Miners does not provide an implementation or evaluation.

Mysticeti [5] is a recent protocol that, like Mahi-Mahi, operates over an uncertified DAG but in a partially synchronous setting. Mysticeti takes advantage of synchronous periods in the network to commit blocks in three rounds, and like Mahi-Mahi, it can commit blocks every round. However, unlike Mahi-Mahi, Mysticeti completely loses liveness when the network is not synchronous. To maintain liveness in asynchronous conditions, Mahi-Mahi interprets the DAG differently from Mysticeti. Specifically, Mahi-Mahi incorporates a global perfect coin into the protocol and modifies the role of several DAG rounds to ensure that an asynchronous adversary cannot indefinitely manipulate message schedules to prevent block certificates from forming—an issue that can easily arise in Mysticeti [29].

**Certified DAG-based BFT consensus protocols.** DAG-Rider [31], Tusk [20], and Dumbo-NG [27] are popular asynchronous certified DAG-based consensus protocols that use reliable or consistent broadcast to explicitly certify every DAG vertex [39]. This approach introduces 3 message delays per DAG round but simplifies the commit rule by ensuring that equivocating DAG vertices never occur. However, this method results in significantly higher latency compared to Mahi-Mahi. For instance, DAG-Rider [31], GradedDAG [19], and LightDAG [17] require at least 12 messages to commit a block, while Tusk and Dumbo-NG require 9 message delays. By contrast, Mahi-Mahi can commit in just 4 or 5 message delays when respectively configured with a wave length of 4 and 5. Also, certified DAGs have higher bandwidth and CPU requirements, as validators must disseminate, receive, and verify the cryptographic certificates generated by consistent broadcast. As shown in Section V, these factors lead to up to 70% higher latency in comparison to Mahi-Mahi.

Sailfish [42], BBCA-Chain [36], Fino [37], Shoal [44], and Shoal++[4] build on the partially synchronous version of Bullshark [46] through various improvements, including the ability to commit more blocks per round and a relaxation of DAG certification requirements. However, these protocols are limited to partially synchronous environments and, unlike Mahi-Mahi, they lose liveness in asynchronous conditions.

**Linear-chain protocols.** Linear-chain asynchronous BFT protocols such as Das *et al.* [22], Pace [50], FIN [25], and SQ [47] do not leverage an underlying DAG structure. They instead rely on explicit Byzantine consistent broadcast [13] and a common coin to elect a leader, whereas Mahi-Mahi incorporates these components implicitly within the DAG. This leader drives the protocol by constructing a linear chain. Consequently, these protocols do not achieve the same level of throughput and robustness as DAG-based systems [20]. Their contributions instead lie primarily in their theoretical foundations. For example, Das *et al.* introduces a protocol that operates without a trusted setup or the need for public-key cryptography; FIN presents the first constant-time asynchronous consensus (ACS) protocol with $O(n^3)$ messages in both information-theoretic and signature-free settings; and SQ reduces this message complexity to $O(n^2)$.

## VII. Conclusion

We introduce Mahi-Mahi, a novel asynchronous BFT consensus protocol achieving a new performance milestone: an impressive 350,000 transactions per second in geo-distributed environments with 50 nodes all while keeping latency below 2 seconds, or 100,000 transactions per second with sub-second latency—an achievement that sets a new record in the realm of asynchronous consensus protocols and that was only thought possible for partially-synchronous protocols. The exceptional performance is made possible through a novel commit rule applied over an uncertified DAG that enables commits of multiple leaders every round. This allows Mahi-Mahi to inherit the robustness and throughput inherent in DAG-based protocols, while establishing a new standard for the latency of asynchronous BFT consensus protocols.

TABLE I: Comparison of asynchronous DAG-based consensus protocols

| | **LB Lat.** | **NLB Lat.** | **Complexity** | **Leaders** |
|---|---|---|---|---|
| DAG-Rider [31] | 12 | 24 | $O(n^3)$ | One |
| Tusk [20] | 9 | 15 | $O(n^3)$ | One |
| Cordial Miners [32] | 5 | 10 | $O(n^2)$ | One |
| GradedDAG [19] | 5 (4)$^\dagger$ | 10 (9)$^\dagger$ | $O(n^3)$ | One |
| LightDAG1 [17] | 6 (5)$^\dagger$ | 10 (9)$^\dagger$ | $O(n^3)$ | One |
| LightDAG2 [17] | 4 | 8 | $O(n^3)$ | One |
| Mahi-Mahi (this work) | 5 (4)$^\ddagger$ | 5 (4)$^\ddagger$ | $O(n^2)$ | Many |

**LB Lat.** and **NLB Lat.** stand for *leader block latency* and *non-leader block latency*, respectively. Latency refers to the number of message delays to commit a block in the best case. Complexity refers to the number of messages sent per DAG round. $^\dagger$GradedDAG and LightDAG can decide early, after the first phase of a two-phase Byzantine consistent broadcast instance; the value in parantheses is the latency when deciding early. $^\ddagger$The value in parantheses shows Mahi-Mahi's latency in the random network model.

*Acknowledgements.*

This work is partially supported by Mysten Labs and the Sui Foundation. This work is also partially supported by the Gates Foundation [INV-057591]; under the grant conditions of the Foundation, a Creative Commons Attribution 4.0 Generic License has already been assigned to the Author's Accepted Manuscript.

## References

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In *CRYPTO*, 2023.

[2] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching Consensus for Asynchronous Distributed Key Generation. *Distributed Computing*, 2023.

[3] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*, 2017.

[4] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast! *CoRR*, abs/2405.20488, 2024.

[5] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. Mysticeti: Low-latency DAG consensus with fast commit path. *CoRR*, abs/2310.14821, 2023.

[6] Renas Bacho and Julian Loss. On the Adaptive Security of the Threshold BLS Signature Scheme. In *ACM CCS*, 2022.

[7] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.

[8] Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, Zekun Li, Avery Ching, and Dahlia Malkhi. Twins: Bft systems made robust. In *ACM PODC*, 2021.

[9] Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *ACM AFT*, 2020.

[10] Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, and et al. Sui lutris: A blockchain combining broadcast and consensus. *CCS*, 2023.

[11] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *ACM TCC*, 2020.

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, 2001.

[13] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. Introduction to reliable and secure distributed programming, 2011.

[14] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *ACM PODC*, 2000.

[15] Junchao Chen, Alberto Sonnino, Lefteris Kokoris-Kogias, and Mohammad Sadoghi. Thunderbolt: Causal concurrent consensus and execution. *CoRR*, abs/2407.09409, 2024.

[16] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. *Information and Computation, Volume 118, Issue 1*, 1995.

[17] Xiaohai Dai, Guanxiong Wang, Jiang Xiao, Zhengxuan Guo, Rui Hao, Xia Xie, and Hai Jin. LightDAG: A Low-latency DAG-based BFT Consensus through Lightweight Broadcast. In *IEEE IPDPS*, 2024.

[18] Xiaohai Dai, Bolin Zhang, Hai Jin, and Ling Ren. ParBFT: Faster Asynchronous BFT Consensus with a Parallel Optimistic Path. In *ACM CCS*, 2023.

[19] Xiaohai Dai, Zhaonan Zhang, Jiang Xiao, Jingtao Yue, Xia Xie, and Hai Jin. GradedDAG: An Asynchronous DAG-based BFT Consensus with Lower Latency. In *SRDS*, 2023.

[20] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *ACM EuroSys*, 2022.

[21] George Danezis, Jovan Komatovic, Lefteris Kokoris-Kogias, Alberto Sonnino, and Igor Zablotchi. Byzantine consensus in the random asynchronous model. *arXiv preprint arXiv:2502.09116*, 2025.

[22] Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. Asynchronous consensus without trusted setup or public-key cryptography. In *ACM CCS*, 2024.

[23] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. In *USENIX Security*, 2023.

[24] Henry de Valence. Ed25519 for consensus-critical contexts. https://crates.io/crates/ed25519-consensus, 2024.

[25] Sisi Duan, Xin Wang, and Haibin Zhang. FIN: Practical Signature-Free Asynchronous Common Subset in Constant Time. In *ACM CCS*, 2023.

[26] Bryan Ford. Threshold logical clocks for asynchronous distributed coordination and consensus. *CoRR*, abs/1907.07010, 2019.

[27] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *ACM CCS*, 2022.

[28] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback. In *FC*, 2022.

[29] Giacomo Giuliari, Alberto Sonnino, Marc Frei, Fabio Streun, Lefteris Kokoris-Kogias, and Adrian Perrig. An Empirical Study of Consensus Protocols' DoS Resilience. In *ACM ASIACCS*, 2024.

[30] Philipp Jovanovic, Lefteris Kokoris Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. Mahi-mahi: Low-latency asynchronous bft dag-based consensus. *arXiv preprint arXiv:2410.08670*, 2024.

[31] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All You Need is DAG. In *ACM PODC*, 2021.

[32] Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial Miners: Fast and Efficient Consensus for Every Eventuality. In *DISC*, 2023.

[33] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *IEEE S&P*, 2018.

[34] Mysten Labs. Mysticeti: Low-latency dag consensus with fast commit path. https://github.com/asonnino/mysticeti, 2024.

[35] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. *IACR ePrint*, 2018.

[36] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. BBCA-CHAIN: One-Message, Low Latency BFT Consensus on a DAG. In *FC*, 2024.

[37] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. In *Tokenomics*, 2022.

[38] Michael S. Paterson Michael J. Fischer, Nancy A. Lynch. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 1985.

[39] Mayank Raikwar, Nikita Polyanskii, and Sebastian Müller. SoK: DAG-based Consensus Protocols. In *IEEE ICBC*, 2024.

[40] Zhijie Ren, Kelong Cong, Johan Pouwelse, and Zekeriya Erkin. Implicit Consensus: Blockchain with Unbounded Throughput, 2017.

[41] RustCrypto. Hashes. https://github.com/RustCrypto/hashes, 2024.

[42] Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving latency of dag-based BFT. *IACR Cryptol. ePrint Arch.*, 2024.

[43] Alberto Sonnino, Shehar Bano, Mustafa Al-Bassam, and George Danezis. Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. In *IEEE EuroS&P*, 2020.

[44] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness. In *FC*, 2024.

[45] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT Protocols Made Practical. In *ACM CCS*, 2022.

[46] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: the partially synchronous version. arXiv preprint arXiv:2209.05633, 2022.

[47] Xiao Sui and Sisi Duan. Signature-free atomic broadcast with optimal O(n$^2$) messages and O(1) expected time. *IACR ePrint*, 2023.

[48] The Tokio Team. Tokio. https://tokio.rs, 2024.

[49] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-throughput byzantine consensus on variable bandwidth networks. In *USENIX NSDI*, 2022.

[50] Haibin Zhang and Sisi Duan. Pace: Fully parallelizable bft from reproposable byzantine agreement. In *ACM CCS*, 2022.

## Appendix A
## Mahi-Mahi Algorithms

This section presents the algorithms used in Mahi-Mahi in pseudocode format. In particular, Algorithm 1 specifies the Mahi-Mahi main algorithms, Algorithm 3 the Mahi-Mahi decider instance, and algorithm 2 contains various DAG helper functions.

As a reminder, Mahi-Mahi operates with a single type of message: a block whose validity is described in Section II. Validators hold these blocks in a data structure called $DAG$. To access the block(s) of round $r$ authored by validator $v$ of the DAG, we write $DAG[r, v]$. If an equivocation happened at a slot $v$, then $DAG[r, v]$ may return multiple blocks. To access all blocks of a given round $r$, we write $DAG[r, *]$.

The entry point is the procedure ExtendCommitSequence$(\cdot)$ (Line Algorithm 1 of Algorithm 1). This is called by the application layer to extend the commit sequence. This procedure is idempotent and is called by our implementation (Section IV) every time the validator receives a new block. This procedure calls TryDecide$(\cdot)$ (Line Algorithm 1 of Algorithm 1) to classify as many blocks as possible as either `commit` or `skip`. The TryDecide$(\cdot)$ procedure iterates over all possible leaders and invokes the decider instance (Line Algorithm 1 of Algorithm 1) to classify each leader slot. The decider instance determines the leader of a given round, certifying blocks, and classifying leader slots. The decider instance uses various helper functions, such as IsVote$(\cdot)$ (Line Algorithm 2 of Algorithm 2), and IsCert$(\cdot)$ (Line Algorithm 2 of Algorithm 2), that are generic utilities for working with the DAG.

## Appendix B
## Example of Mahi-Mahi Execution

This section completes Section III by guiding readers through the protocol execution, step-by-step protocol, using the example illustrated in Figure 2. This figure showcases Mahi-Mahi with four validators, labeled as ($v_0$, $v_1$, $v_2$, $v_3$), operating over a wavelength of five rounds, with two leader slots allocated per round. As mentioned in Section III-B, all proposer slots are initially in the `undecided` state. Validator holds in memory the portion of the DAG depicted in Figure 2

**Algorithm 1** Mahi-Mahi Main Function

```
waveLength                                        ▷ Set to at least 4 (see Section III)
leadersPerRound                                   ▷ See Section V for details

// Called by the application layer to extend the commit sequence (idempotent).
procedure ExtendCommitSequence($r_{committed}, r_{highest}$)
    $L \leftarrow$ TryDecide($r_{committed}, r_{highest}$)      ▷ See Algorithm 1 below
    $L_{commit} \leftarrow [\ ]$                                ▷ Hold decided leader sequence
    for $status \in L$ do
        if $status = \perp$ then break                ▷ Stop at the first undecided leader
        if $status =$ commit($b_{leader}$) then
            $L_{commit} \leftarrow L_{commit} \mid\mid b_{leader}$
    return LinearizeSubDags($L_{commit}$)               ▷ See Line Algorithm 2 of Algorithm 2

// Try to decide proposals, recursively, starting from the latest proposal.
procedure TryDecide($r_{committed}, r_{highest}$)
    $L \leftarrow [\ ]$                                         ▷ Hold decision of each leader
    for $r \in [r_{highest}$ down to $r_{committed} + 1]$ do
        for $l \in$ [leadersPerRound $- 1$ down to 0] do     ▷ All possible leaders
            $i \leftarrow r$ % waveLength
            D $\leftarrow$ Decider(waveLength, $i, l$)               ▷ Algorithm 3
            $w \leftarrow$ D.WaveNumber($r$)
            if D.ProposeRound($w$) $\neq r$ then continue     ▷ Skip if not a leader
            $status \leftarrow$ D.TryDirectDecide($w$)          ▷ Direct decision rule
            if $status = \perp$ then
                $status \leftarrow$ D.TryIndirectDecide($w$)    ▷ Indirect decision rule
            $L \leftarrow status \mid\mid L$
    return $L$                                    ▷ May still contain undecided leaders
```

**Algorithm 2** DAG Helper Functions

```
procedure IsVote($b_{vote}, b_{leader}$)
    function VotedBlock($b, id, r$)
        if $r \geq b.round$ then return $\perp$
        for $b' \in b.parents$ do
            if $(b'.author, b'.round) = (id, r)$ then return $b'$
            $res \leftarrow$ VotedBlock($b', id, r$)
            if $res \neq \perp$ then return $res$
        return $\perp$
    $(id, r) \leftarrow (b_{leader}.author, b_{leader}.round)$
    return VotedBlock($b_{vote}, id, r$) $= b_{leader}$

procedure IsCert($b_{cert}, b_{leader}$)
    $res \leftarrow |\{b \in b_{cert}.parents :$ IsVote($b, b_{leader}$)$\}|$
    return $res \geq 2f + 1$

procedure IsLink($b_{old}, b_{new}$)
    return exists a sequence of $k \in \mathbb{N}$ blocks $b_1, \ldots, b_k$ s.t. $b_1 = b_{old}, b_k = b_{new}$ and $\forall j \in [2, k] : b_j \in \bigcup_{r \geq 1} DAG[r, *] \wedge b_{j-1} \in b_j.parents$

procedure IsCertifiedLink($b_{anchor}, b_{leader}$)
    $w \leftarrow$ WaveNumber($b_{leader}.round$)
    $B \leftarrow$ GetDecisionBlocks($w$)
    return $\exists b \in B$ s.t. IsCert($b, b_{leader}$) $\wedge$ IsLink($b, b_{anchor}$)

procedure LinearizeSubDags($L$)
    $O \leftarrow [\ ]$                                         ▷ Hold output sequence
    for $b_{leader} \in L$ do
        $B \leftarrow \{b \in \bigcup_{r \geq 1} DAG[r, *]$ s.t. IsLink($b, b_{leader}$) $\wedge\ b \notin O\ \wedge$ $b$ not already output $\}$
        for $b \in B$ in any deterministic order do
            $O \leftarrow O \mid\mid b$
    return $O$
```

**Algorithm 3** Mahi-Mahi Decider Instance

```
waveLength                                        ▷ Set to at least 4 (see Section III)
waveOffset                               ▷ Offset creating overlapping waves (Section II)
leaderOffset                              ▷ Each decider operates on a unique leader slot

procedure WaveNumber($r$)
    return $(r -$ waveOffset$)/$waveLength

procedure ProposeRound($w$)
    return $w \cdot$ waveLength $+$ waveOffset                      ▷ See Figure 1

procedure CertifyRound($w$)
    return $w \cdot$ waveLength $+$ waveLength $- 1 +$ waveOffset   ▷ See Figure 1

procedure VoteRound($w$)
    return Self.CertifyRound($w$) $- 1$                             ▷ See Figure 1

procedure LeaderBlock($w$)
    $r_{propose}, r_{certify} \leftarrow$ Self.ProposeRound($w$), Self.CertifyRound($w$)
    $c \leftarrow$ CombineCoinShares($\{b.share$ s.t. $b \in DAG[r_{certify}, *]\}$)
    $l \leftarrow c +$ leaderOffset                              ▷ Modulo committee size
    return $DAG[r_{propose}, l]$     ▷ May return more than one block in case of equivocations

procedure SkippedLeader($w, b_{leader}$)
    $r_{vote} \leftarrow$ Self.VoteRound($w$)
    return $|\{\neg$IsVote($b, b_{leader}$) s.t. $b \in DAG[r_{vote}, *]\}| \geq 2f + 1$

procedure SupportedLeader($w, b_{leader}$)
    $r_{certify} \leftarrow$ Self.CertifyRound($w$)
    return $|\{$IsCert($b, b_{leader}$) s.t. $b \in DAG[r_{certify}, *]\}| \geq 2f + 1$

procedure TryDirectDecide($w$)
    for $b_{leader} \in$ Self.LeaderBlock($w$) do          ▷ Loop over equivocations
        if Self.SkippedLeader($w, b_{leader}$) then return skip($w$)
        if Self.SupportedLeader($w, b_{leader}$) then return commit($b_{leader}$)
    return $\perp$

procedure TryIndirectDecide($w, S$)
    $s_{anchor} \leftarrow$ find first $s \in S$ s.t. $r_{certify} < s.round \wedge s \neq$ skip($w$)
    if $s_{anchor} =$ commit($b_{anchor}$) then
        if $\exists b_{leader} \in$ Self.LeaderBlock($w$) s.t. IsCertifiedLink($b_{anchor}, b_{leader}$) then
            return commit($b_{leader}$)
        else
            return skip($w$)
    return $\perp$                               ▷ The anchor is undecided or not found
```

and attempts to classify as many blocks in the leader slots as possible as either `commit` or `skip`.

The first step for the validator is to identify the leader slots by reconstructing the global perfect coin for each round. As described in Section III-B (step 1), the validator derives the following leader slots:

$$[L_{6b}, L_{6a}, L_{5b}, L_{5a}, L_{4b}, L_{4a}, L_{3b}, L_{3a}, L_{2b}, L_{2a}, L_{1b}, L_{1a}]$$

Next, the validator attempts to classify each leader slot as either `commit` or `skip` using the *direct decision rule* (step 2), starting with the highest slot, $L_{6b}$. As shown in Figure 6, the validator classifies $L_{6b}$ as `commit` since it is certified by $2f + 1$ blocks from round $R + 9$, specifically $B_{(v_0,R+9)}$, $B_{(v_1,R+9)}$, and $B_{(v_2,R+9)}$. The validator proceeds to $L_{6a}$. As illustrated in Figure 7, it classifies this block as `skip` because $2f + 1$ blocks from round $R + 8$ ($B_{(v_1,R+8)}$, $B_{(v_2,R+8)}$, and $B_{(v_3,R+8)}$) do not vote for $L_{6a}$.

Next, the validator examines $L_{5b}$. In its local view of the DAG, it encounters two equivocations for this leader slot: $L_{5b}$ and $L'_{5b}$. The validator then invokes the function IsVote($\cdot$) (shown in Line Algorithm 2 of Algorithm 2, Section A) to determine which of these equivocations, if any, receives votes from the blocks of round $R+7$. For each block of this round, the validator conducts a depth-first search starting from the block, following its hash references to see if it first encounters $L_{5b}$ or $L'_{5b}$.

In this example, the validator first targets $B_{(v_0,R+7)}$ and finds it votes for $L_{5b}$. Upon targeting $B_{(v_1,R+7)}$, it discovers a vote for $L'_{5b}$. Continuing this process with $B_{(v_2,R+7)}$ and $B_{(v_3,R+7)}$ also leads to a vote for $L'_{5b}$. Consequently, since there are $2f + 1$ blocks from round $R + 7$ that do not vote for $L_{5b}$, the validator classifies it as `skip`. And since there

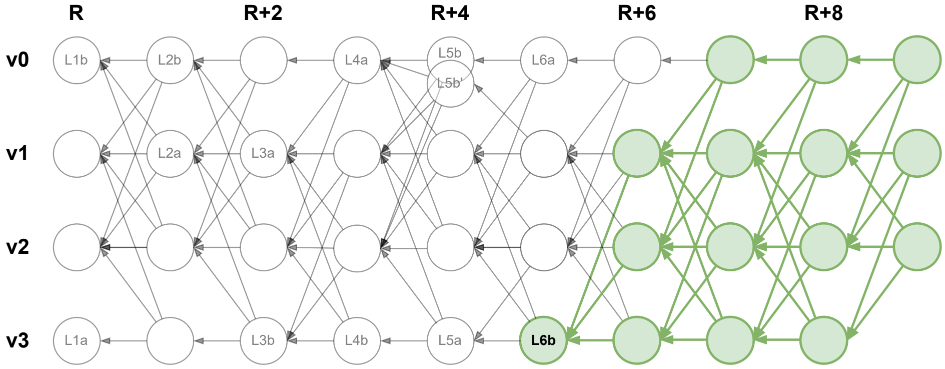

Fig. 6: Example of direct decision rule. $L_{6b}$ is classified as `commit`.

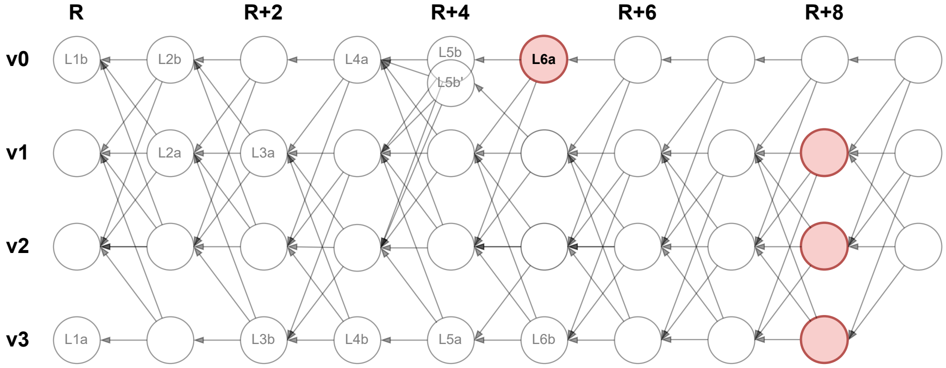

Fig. 7: Example of direct decision rule. $L_{6a}$ is classified as `skip`.

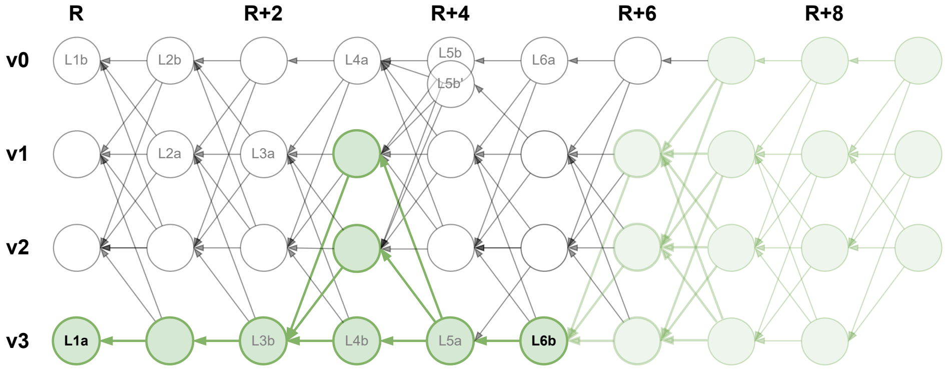

Fig. 8: Example of indirect decision rule. $L_{1b}$ is classified as `commit`.

Fig. 9: Example of Mahi-Mahi execution with 4 validators, configured with a wave length of 5 rounds and 2 leader slots per round.

are at least $2f+1$ blocks from round $R+8$ ($B_{(v_0,R+8)}$, $B_{(v_1,R+8)}$, $B_{(v_2,R+8)}$, and $B_{(v_3,R+8)}$) that certify $L'_{5b}$, the validator classifies it as `commit`.

The validator then moves on to $L_{5a}$, classifying it as `commit` as it receives sufficient certification from blocks of round $R$+8 ($B_{(v_0,R+8)}$, $B_{(v_1,R+8)}$, $B_{(v_2,R+8)}$, and $B_{(v_3,R+8)}$). Similarly, the validator classifies both $L_{4b}$ and $L_{4a}$ as `commit` since they are also certified by $B_{(v_0,R+7)}$, $B_{(v_1,R+7)}$, $B_{(v_2,R+7)}$, and $B_{(v_3,R+7)}$. This reasoning applies to the slots $L_{3b}$ and $L_{3a}$, which are certified by blocks of round $R+6$, as well as to $L_{2b}$ and $L_{2a}$, certified by blocks of round $R+5$. Finally, $L_{1b}$ is certified by $B_{(v_0,R+4)}$ (through both $L_{5b}$ and $L'_{5b}$), $B_{(v_1,R+4)}$, $B_{(v_2,R+4)}$, and $B_{(v_3,R+4)}$.

However, the direct commit rule fails to classify $L_{1a}$. There are neither $2f+1$ blocks from round $R$+3 that do not vote for it (which would classify it as `skip`) nor $2f+1$ blocks from round $R+4$ that certify it (preventing its classification as `commit`). Therefore, the validator turns to the *indirect decision rule* (step 3) to classify $L_{1a}$.

As specified in Section III-B, the validator first seeks the anchor for $L_{1a}$, which is $L_{6b}$. The anchor is defined as the first block with a round number $r' > r+5$ that is classified as either `undecided` or `commit`. Consequently, $L_{6a}$ cannot serve as the anchor for $L_{1b}$, making $L_{6b}$ its anchor. The validator then checks for an existing certificate over $L_{1a}$ that is referenced by the causal history of its anchor, $L_{6b}$. As illustrated in Figure 8, in this case, $L_{1b}$ is certified by $L_{5a}$, which references $2f+1$ votes for $L_{1b}$ ($B_{(v_1,R+3)}$, $B_{(v_2,R+3)}$, and $B_{(v_3,R+3)}$). Thus, $L_{1b}$ is classified as `commit`. Without such a certificate, the validator would have classified $L_{1b}$ as `skip`.

## Appendix C
## Security Proofs

This section proves the correctness of Mahi-Mahi, by showing that Mahi-Mahi satisfies the properties of Byzantine Atomic Broadcast (BAB) from Section II. We prove the correctness of both the 4-round and 5-round versions of Mahi-Mahi. We start with results that hold for both versions (these are mostly safety-related results) in Section C-A and continue with version-specific results in Appendices Section C-B and Section C-C.

### *A. Common Proofs for $w = 4$ and $w = 5$*

We start by proving the Total Order and Integrity properties of BAB. A crucial intermediate result towards these properties is that all honest validators have consistent commit sequences, i.e., the committed sequence of one honest validator is a prefix of another's, or vice-versa. This is shown in Lemmas Theorem C.5 and Theorem C.6, which the following lemmas and observations build up to.

**Lemma C.1.** *If in round $r$, $2f+1$ blocks from distinct validators certify a block $b$, then all blocks at future rounds $r' > r$ will have a path to a certificate for $b$ from round $r$.*

*Proof.* We prove the lemma by induction on $r'$. The base case is $r' = r+1$. Let $b'$ be a block at round $r'$. Since $b'$ points to $2f+1$ blocks at round $r$, by quorum intersection, $b'$ must point to at least one of the certificates for $b$.

For the induction case, assume the lemma holds up to round $r'$ and consider the case of round $r'+1$. Let $b'$ be a block at round $r'+1$. By the induction hypothesis, $2f+1$ blocks at round $r'$ have paths to round-$r$ certificates for $b$. Since $b'$ points to $2f+1$ blocks from round $r'$, by quorum intersection, $b'$ must point to at least one block that has a path to a round-$r$ certificate for $b$. □

**Observation 1.** *A block cannot vote for more than one block proposal from a given validator, in a given round.*

*Proof.* This is by construction. Honest validators interpret support in the DAG through deterministic depth-first traversal. So even if a block $b$ in the vote round has paths to multiple leader round blocks from the same validator $v$ (i.e., equivocating blocks), all honest validators will interpret $b$ to vote for only one of $v$'s blocks (the first block to appear in the depth-first traversal starting from $b$). □

**Lemma C.2.** *At most a single block per round from the same validator can be certified.*

*Proof.* Assume by contradiction that in a given round $r$, there exist two distinct blocks $b$ and $b'$ from the same validator $v$ such that both $b$ and $b'$ are certified. This means that there exist round-$(r+w-1)$ blocks $c_b$ and $c_{b'}$ that certify $b$ and $b'$, respectively. $c_b$ and $c_{b'}$ must point to $2f+1$ votes for $b$ and $b'$, respectively. By quorum intersection, there exists an honest validator that has voted for both $b$ and $b'$ in the vote round. Since honest validators only produce a single block per round, this implies that there exists a block that votes for both $b$ and $b'$, contradicting Observation 1.

□

**Observation 2.** *If an honest validator $v$ directly or indirectly commits a block $b$, then $v$'s local DAG contains a certificate for $b$.*

*Proof.* This follows immediately from our direct and indirect commit rules. □

**Observation 3.** *Honest validators agree on the sequence of leader slots.*

*Proof.* This follows immediately from the properties of the common coin, see Section II-A. □

**Lemma C.3.** *If an honest validator $v$ commits some block $b$ in a slot $s$, then no other honest validator decides to directly skip the slot $s$.*

*Proof.* Assume by contradiction that some honest validator $v'$ decides to directly skip $s$. Then it must be the case that in the local DAG of $v'$, at least $2f+1$ validators did not vote for $b$. However, since $v$ commits $b$ at $s$, by Observation 2, there must exist a certificate for $b$ at $s$. So in $v$'s local DAG there must be $2f+1$ validators that vote for $b$. By quorum intersection, at least one honest validator both voted for $b$ and did not vote for $b$. Since honest validators produce a single block in the vote round, this is a contradiction. □

**Lemma C.4.** *If an honest validator directly commits some block in a slot $s$, then no other honest validator decides to skip the slot $s$.*

*Proof.* Assume by contradiction that an honest validator $v$ directly commits block $b$ in slot $s$ while another honest validator $v'$ decides to skip $s$. By Theorem C.3, $v'$ cannot directly skip $s$; it must be the case therefore that $v'$ skips $s$ using the indirect decision rule. Let $r$ be the round of $s$. Since $v$ directly commits $b$, there exist $2f+1$ certificates for $b$ at $s$. Therefore, by Theorem C.1, all blocks at rounds $r' > r+w-1$, including the anchor of $s$, have a path to a certificate for $b$ at $s$. Thus, $v'$ cannot decide to skip $s$ using the indirect decision rule. We have reached a contradiction. □

**Lemma C.5.** *If a slot $s$ is committed at two honest validators, then $s$ contains the same block at both validators.*

*Proof.* Let $v$ and $u$ be two honest validators and assume that $v$ commits block $b$ at slot $s$. We will show that if $u$ commits slot $s$, then $s$ contains $b$ at $s$. Let $w$ be the validator that produced block $b$. By Observation 2, for $b$ to be committed at slot $s$ at $v$, there must exist at least one certificate for $b$. By Observation 3, $v$ and $u$ agree that $s$ must contain a block by $w$. By Theorem C.2, at most a single block per round from $w$ can be certified. So $u$ cannot have a certificate for any other block than $b$ at slot $s$. □

We say that a slot is *decided* at a validator $v$ if $s$ is committed or skipped, *i.e.,* if it is categorized as `commit` or `skip`. Otherwise, $s$ is *undecided*.

**Lemma C.6.** *If a slot $s$ is decided at two honest validators $v$ and $v'$, then either both validators commit $s$, or both validators skip $s$.*

*Proof.* Assume by contradiction that there exists a slot $s$ such that $v$ and $v'$ decide differently at $s$. We consider a finite execution prefix and assume *wlog* that $s$ is the highest slot at which $v$ and $v'$ decide differently (*). Further assume *wlog* that $v$ commits $s$ and $v'$ skips $s$. By Theorem C.3 and Theorem C.4, neither $v$ nor $v'$ could have used the direct decision rule for $s$; they must both have used the indirect rule. Consider now the anchor of $s$: $v$ and $v'$ must agree on which slot is the anchor of $s$, since by our assumption (*) above, they make the same decisions for all slots higher than $s$, including the anchor of $s$. Let $s'$ be the anchor of $s$; $s'$ must be committed at both $v$ and $v'$. Thus, by Theorem C.5, $v$ and $v'$ commit the same block $b'$ at $s'$. But then $v$ and $v'$ cannot reach different decisions about slot $s$ using the indirect decision rule. We have reached a contradiction. □

We have proven the consistency of honest validators' commit sequences: honest validators commit (or skip) the same leader blocks, in the same order. However, we are not done: we also need to prove that non-leader blocks are delivered in the same order by honest validators. We show this next.

**Causal history & delivery conditions** Consider an honest validator $v$. We call the *causal history* of a block $b$ in $v$'s DAG, the transitive closure of all blocks referenced by $b$ in $v$'s DAG, including $b$ itself. In Mahi-Mahi, a block $b$ is delivered by an honest validator $v$ if (1) there exists a committed leader block $l$ in $v$'s DAG such that $b$ is in $l$'s causal history (2) all slots up to $l$ are decided in $v$'s DAG and (3) $b$ has not been delivered as part of a lower slot's causal history. In this case we say $b$ is *delivered at* slot $s$, or *delivered with* block $l$.

**Lemma C.7.** *If a block $b$ is delivered by two honest validators $v$ and $v'$, then $b$ is delivered at the same slot $s$, and $b$ is delivered with the same leader block $l$, at both $v$ and $v'$.*

*Proof.* Let $s$ be the slot at which $b$ is delivered at validator $v$, and $l$ the corresponding leader block in $s$, also at validator $v$. Consider now the slot $s'$ at which $b$ is delivered at validator $v'$, and $l'$ the corresponding leader block. Assume by contradiction that $s' \neq s$. If $s' < s$, then $v$ would have also delivered $b$ at slot $s'$, since by Theorem C.5 must commit the same leader blocks in the same slots, so $v$ could not have delivered $b$ again at slot $s$; a contradiction. Similarly, if $s < s'$, then $v'$ would have already delivered $b$ at slot $s$, since by Theorem C.5 $v$ and $v'$ must have committed the same block in slot $s$; contradiction. Thus it must be that $s = s'$, and by Theorem C.5, $l = l'$. □

We can now prove the main safety properties of Mahi-Mahi: Total Order and Integrity.

**Theorem C.8** (Total Order)**.** *Mahi-Mahi satisfies the total order property of Byzantine Atomic Broadcast.*

*Proof.* This property follows immediately from Theorem C.7 and the fact that honest validators order the causal histories of committed blocks using the same deterministic function, and deliver blocks in this order. □

**Theorem C.9** (Integrity)**.** *Mahi-Mahi satisfies the integrity property of Byzantine Atomic Broadcast.*

*Proof.* This is by construction: a block $b$ is delivered as part of the causal history of a committed leader block only if $b$ has not been delivered along with an earlier leader block (see "Causal history & delivery conditions" above). So an honest validator cannot deliver the same block twice. □

We now turn to liveness properties. The following two lemmas establish that blocks broadcast by honest validators are eventually included in all honest validators' DAGs.

**Lemma C.10.** *If a block $b$ produced by an honest validator $v$ references some block $b'$, then $b'$ will eventually be included in the local DAG of every honest validator.*

*Proof.* This is ensured by the synchronizer sub-component in each validator: if some validator $w$ receives $b$ from $v$, but does not have $b'$ yet, $w$ will request $b'$ from $v$; since $v$ is honest and the network links are reliable, $v$ will eventually receive $w$'s request, send $b'$ to $w$, and $w$ will eventually receive $b'$. The same is recursively true for any blocks from the causal history of $b'$, so $w$ will eventually receive all blocks from the causal history of $b'$ and thus include $b'$ in its local DAG. □

**Lemma C.11.** *If an honest validator $v$ broadcasts a block $b$, then every correct validator will eventually include $b$ in its local DAG.*

*Proof.* Since network links are reliable, all honest validators will eventually receive $b$ from $v$. By Theorem C.10, all honest validators will eventually receive all of $b$'s causal history, and so will include $b$ in their local DAG. □

The following crucial lemma establishes that in any round $r$, there is at least one block $b$, called a common core, such that all blocks at round $r+2$ have a path to $b$.

**Lemma C.12.** *For any $r$, there is at least one block $b$ from round $r$ such that any valid block from round $r+2$ has a path to $b$.*

*Proof.* Consider a set $B$ of $2f+1$ blocks in round $r+1$ from honest validators. Using $B$, we create a table $T$, as follows: for blocks $b, c \in B$, let $T[b, c] = 1$ if $b$ in $r+1$ references

$c$ in $r$, $T[b, c] = 0$ otherwise. By quorum intersection, any $b$ will reference at least $f + 1$ blocks in round $r$ that are also in $B$, so each row of $T$ has at least $f + 1$ entries equal to 1. Thus, $T$ has at least $(2f + 1)(f + 1)$ entries equal to 1. By a counting argument, there is a block $c^*$ in $B$ that has a 1 entry in at least $f + 1$ rows, i.e., a block from round $r$ which is referenced by $f + 1$ blocks from round $r + 1$. Let $P'$ be the set of blocks from round $r + 1$ which reference $c^*$. Consider now any valid block $b$ in round $r + 2$; $b$ references $2f + 1$ blocks in $r + 1$, so by quorum intersection $b$ references at least one block in $B$. Thus, $b$ has a path to $c^*$. □

### *B. Specific Proofs for $w = 5$*

We continue with proofs that are specific to the liveness of the $w = 5$ version of Mahi-Mahi. We show that each wave has at least $2f + 1$ leader blocks that can be directly committed (Lemmas Theorem C.13 and Theorem C.14), and thus that each wave has a nonzero probability of directly committing at least one block (Theorem C.15). We then show that each slot is eventually decided directly or indirectly (Theorem C.16). Finally, we show that Mahi-Mahi satisfies the Validity and Agreement properties of BAB.

As a consequence of Theorem C.12, we have the following:

**Lemma C.13.** *For any $r$, there exists a set $S$ of at least $2f+1$ blocks from round $r$ such that any valid block from round $r+3$ is a vote for every block in $S$.*

*Proof.* Let $r' = r + 1$. By Theorem C.12, there exists a block $b$ in round $r' = r + 1$ such that any valid block from round $r' + 2 = r + 3$ has a path to $b$. Now let $S$ be the set of blocks referenced by the block $b$. $S$ must contain at least $2f + 1$ blocks from round $r$. Every block from round $r + 3$ has a path to $b$ and thus, through $b$, to every block in $S$. □

From this we can derive the following crucial lemma:

**Lemma C.14.** *For any $r$, there exists a set $S$ of at least $2f+1$ blocks from round $r$ such that every block in $S$ has at least $2f + 1$ certificates in round $r + 4$.*

*Proof.* Take $S$ to be the set from Theorem C.13. There are at least $2f+1$ blocks in $r+4$. Any block $b$ in round $r+4$ must reference $2f + 1$ blocks from round $r + 3$. By Theorem C.13, every block from round $r + 3$ is a vote for every block in $S$, so $b$ must be a certificate for every block in $S$. □

We denote by $\ell \leq 3f + 1$ the number of leader slots per round.

**Lemma C.15.** *Fix a round $r$. If $\ell > f$, then an honest validator directly commits at least one slot corresponding to round $r$. Otherwise, the probability that an honest validator directly commits at least one slot corresponding to round $r$ is at least $p^\star = 1 - \frac{\binom{f}{\ell}}{\binom{3f+1}{\ell}} > 0$.*

*Proof.* By Theorem C.14, at least $2f + 1$ blocks from round $r$ can be directly committed, out of a maximum of $3f + 1$ blocks. When the common coin is released in round $r + 4$, it selects uniformly at random $\ell$ round-$r$ blocks as the $\ell$ slots of round $r$.

In the case $\ell > f$, by quorum intersection, there exists at least one slot selected by the common coin among the $2f+1$ blocks that can be directly committed.

In the case $\ell \leq f$, we can model the number of directly committed slots in round $r$ as a hypergeometric random variable, where a success event corresponds to selecting a slot that can be directly committed. The probability of 0 successes (i.e., not committing any slots directly) is therefore at most $\frac{\binom{f}{\ell}}{\binom{3f+1}{\ell}} < 1$. □

**Lemma C.16.** *Fix a slot $s$. Every honest validator eventually either commits or skips $s$, with probability* 1.

*Proof.* We prove the lemma by showing that the probability of $s$ remaining undecided forever at some honest validator is 0. In order for $s$ to remain undecided forever, $s$ cannot be committed or skipped directly. Furthermore, $s$ cannot be decided using the indirect rule. This means that the anchor $s'$ of $s$ must also remain undecided forever, and therefore the anchor $s''$ of $s'$ must remain undecided forever, and so on. The probability of this occurring is at most equal to the probability of an infinite sequence of rounds with no directly committed slots, equal to $\lim_{t\to\infty}(1-p^\star)^t = 0$, where $p^\star > 0$ is the probability from Theorem C.15. □

**Theorem C.17** (Validity). *Mahi-Mahi satisfies the validity property of Byzantine Atomic Broadcast.*

*Proof.* Let $v$ be an honest validator and $b$ a block broadcast by $v$. We show that, with probability 1, $b$ is eventually delivered by every honest validator. By Theorem C.11, $b$ is eventually included in the local DAG of every honest validator. So every honest validator will eventually include a reference to $b$ in at least one of its blocks. Let $r$ be the highest round at which some honest validator includes a reference to $b$ in one of its blocks. By Theorem C.15, with probability 1, eventually some block $b'$ at a round $r' > r$ will be directly committed. Block $b'$ must reference at least $2f + 1$ blocks, thus at least $f + 1$ blocks from honest validators. Since all validators have $b$ in their causal histories by round $r$, $b'$ must therefore have a path to $b$. Theorem C.16 guarantees that all slots before $b'$ are eventually decided, so $b'$ is eventually delivered. Thus, $b$ will be delivered at all honest validators at the latest when $b'$ is delivered along with its causal history. □

**Theorem C.18** (Agreement). *Mahi-Mahi satisfies the agreement property of Byzantine Atomic Broadcast.*

*Proof.* Let $v$ be an honest validator and $b$ a block delivered by $v$. We show that, with probability 1, $b$ is eventually delivered by every honest validator. Let $l$ be the leader block with which $b$ is delivered, and $s$ the corresponding slot. By Theorem C.16, all blocks up to and including $s$ are eventually decided by all honest validators, with probability 1. By Theorem C.5, all honest validators commit $l$ in $s$. Therefore, all honest validators deliver $b$ eventually. □

### *C. Specific Proofs for $w = 4$*

We now turn to the liveness of the $w = 4$ version of Mahi-Mahi. We first show that under an asynchronous network, liveness is guaranteed, albeit with a smaller probability of direct commit at each wave than in the $w = 5$ version. We later show that under a random network, Mahi-Mahi directly commits all valid leader blocks in each wave with high probability. Recall that in the random network model, a valid block from round $r + 1$ references a set of $2f + 1$ valid blocks from round $r$, sampled uniformly at random among all valid round-$r$ blocks.

**Lemma C.19.** *For any $r$, there exists a block $b$ from round $r$ such that $b$ has at least $2f + 1$ certificates in round $r + 3$.*

*Proof.* By Theorem C.12, there exists $b$ in round $r$ such that any block in round $r + 2$ has a path to $b$, and therefore is a vote for $b$. Since any block in round $r + 3$ must reference $2f + 1$ blocks in round $r + 2$, any block in round $r + 3$ must be a certificate for $b$. Since every honest validator publishes a block in round $r + 3$, there must exist at least $2f + 1$ certificates for $b$ in round $r + 3$. □

We again denote by $\ell \leq 3f + 1$ as the number of leader slots per round.

**Lemma C.20.** *Assume the asynchronous network model and fix a round $r$. If $\ell = 3f + 1$, then an honest validator commits at least one slot corresponding to round $r$. Otherwise, the probability that an honest validator directly commits at least one slot corresponding to round $r$ is at least $p^\star = \frac{\ell}{3f+1}$.*

*Proof.* By Theorem C.19, there exists at least one block $b$ in round $r$ that can be directly committed. When the common coin is released in round $r+3$, it selects uniformly at random $\ell$ round-$r$ blocks as the $\ell$ slots of round $r$.

If $\ell = 3f + 1$, then all possible blocks of round $r$ are included in the slots, and thus $b$ is directly committed.

$\ell < 3f+1$, we can model the number of directly committed slots in round $r$ as a hypergeometric random variable, where a success event corresponds to selecting the slot that can be directly committed. There are $3f + 1$ states in total, out of which only 1 is a success state; and there are $\ell$ draws. The probability of one success is $\frac{\binom{1}{1}\binom{3f}{\ell-1}}{\binom{3f+1}{\ell}} = \frac{\ell}{3f+1}$. □

**Lemma C.21.** *In the random network model, with high probability, every block in round $r + 2$ is a vote for every block in round $r$.*

*Proof.* We prove the lemma by showing, through Markov's inequality, that the probability of any block in round $r$ being unreachable from any block in round $r + 2$ approaches 0 exponentially in $f$.

Take a pair of blocks $b_r$ and $b_{r+2}$ in rounds $r$ and $r + 2$, respectively. We compute the probability that there is no path from $b_{r+2}$ to $b_r$. Block $b_{r+2}$ must reference $2f+1$ blocks from round $r + 1$; let $b_{r+1}$ be one such block. The probability that $b_{r+1}$ references $b_r$ is at least $p = \frac{2f+1}{3f+1}$, since $b_{r+1}$ references $2f+1$ randomly selected blocks from round $r$. The probability that there is no path from $b_{r+2}$ to $b_r$ is therefore at most $q = (1 - p)^{2f+1}$.

We now compute the expected number of pairs of blocks $b_r$ and $b_{r+2}$ from rounds $r$ and $r+2$, respectively, such that $b_r$ is not reachable from $b_{r+2}$. There are at most $(3f+1)^2$ pairs, and each pair is not connected by a path with probability at most $q$, thus the expected number of unreachable pairs is at most $E = q(3f + 1)^2 = (3f + 1)^2(1 - p)^{2f+1}$.

Using Markov's inequality, the probability that there exists at least one unreachable pair is:

$$\Pr(\text{At least one unreachable pair}) \leq E = (3f+1)^2(1-p)^{2f+1}.$$

As $f$ increases, this probability rapidly approaches 0 due to the exponential term. □

**Lemma C.22.** *Assume the random network model and fix a round $r$. With high probability, an honest validator directly commits every leader slot chosen by the common coin in round $r + 3$.*

*Proof.* By Theorem C.21, every block in round $r + 2$ is a vote for every block in round $r$ with high probability. Thus every block in round $r + 3$ is a certificate for every block in round $r$ with high probability. So every honest validator sees $2f + 1$ certificates for every block in round $r$ and thus sees also for the blocks chosen by the common coin at least $2f + 1$ certificates. □

**Lemma C.23.** *Fix a slot $s$. In both the asynchronous network model and the random network model, every honest validator eventually either commits or skips $s$ with probability 1.*

*Proof.* The proof is analogous to the proof of Theorem C.16. By Theorem C.20 and Theorem C.22, the probability of a honest validator directly committing any leader block in a given round is greater than 0, in both the asynchronous and the random network models. Thus, the probability of an infinite sequence of rounds without any directly committed blocks is 0. This implies that every slot that is not directly decided will eventually have a committed anchor and become decided. □

**Theorem C.24** (Validity). *Mahi-Mahi satisfies the validity property of Byzantine Atomic Broadcast.*

*Proof.* The proof is similar to the proof of validity in the $w = 5$ case. Let $v$ be an honest validator and $b$ a block broadcast by $v$. We show that, with probability 1, $b$ is eventually delivered by every honest validator. By Theorem C.11, $b$ is eventually included in the local DAG of every honest validator. So every honest validator will eventually include a reference to $b$ in at least one of its blocks. Let $r$ be the highest round at which some honest validator includes a reference to $b$ in one of its blocks. By Theorem C.20 and Theorem C.22, with probability 1, eventually some block $b'$ at a round $r' > r$ will be directly committed. Block $b'$ must reference at least $2f + 1$ blocks, thus at least $f + 1$ blocks from honest validators. Since all validators have $b$ in their causal histories by round $r$, $b'$ must

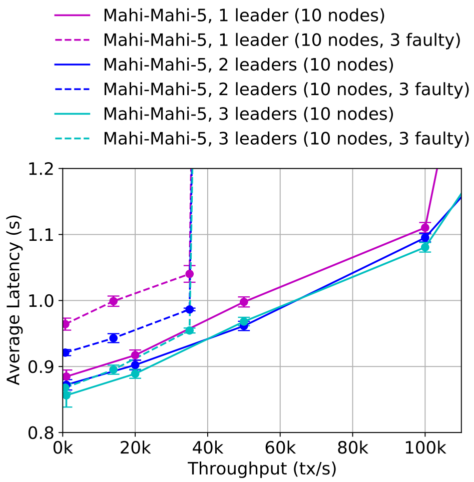

Fig. 10: Impact of the number of leaders per round in Mahi-Mahi. WAN measurements with 10 validators. Zero and three faults. 512B transaction size.

therefore have a path to $b$. Theorem C.23 guarantees that all slots before $b'$ are eventually decided, so $b'$ is eventually delivered. Thus, $b$ will be delivered at all honest validators at the latest when $b'$ is delivered along with its causal history. □

**Theorem C.25** (Agreement)**.** *Mahi-Mahi satisfies the agreement property of Byzantine Atomic Broadcast.*

*Proof.* The proof is similar to the proof of agreement in the $w = 5$ case. Let $v$ be an honest validator and $b$ a block delivered by $v$. We show that, with probability 1, $b$ is eventually delivered by every honest validator. Let $l$ be the leader block with which $b$ is delivered and $s$ the corresponding slot. By Theorem C.23, all blocks up to and including $s$ are eventually decided by all honest validators, with probability 1. By Theorem C.5, all honest validators commit $l$ in $s$. Eventually, all honest validators deliver $b$. □

**Note: Mahi-Mahi with $w = 3$** It is possible to configure Mahi-Mahi with 3-round waves, by removing all Boost rounds, and keeping the Propose round ($r$), Vote round ($r+1$) and Certify round ($r+2$). Such a protocol would still satisfy safety, as all results up to and including Theorem C.9 hold when $w = 3$. However, this 3-round version of Mahi-Mahi would no longer satisfy liveness, because the common core approach in Theorem C.12 can no longer be used to guarantee that at least one leader block can be directly committed in each wave.

## Appendix D
## Mahi-Mahi Impact of Multi-leader

This section completes Section V by presenting the impact of the number of leaders per round in Mahi-Mahi when implemented with 5 rounds per wave. Figure 10 illustrates how Mahi-Mahi configured with a wave length of 5 rounds performs with 1, 2, and 3 leaders per round under both normal conditions and scenarios involving 3 crash faults. We observe a latency reduction as the number of leaders increases similar to Figure 5 (Section V). Specifically, when the number of leaders rises from 1 to 3, Mahi-Mahi's average latency decreases by approximately 40ms in ideal scenario, and by approximately 100ms in the crash failure scenario. Similarly to Figure 5, increasing the number of leaders beyond 3 did not further decrease latency.