FinWhale: An Optimally Resilient Two-Round Terminating DAG Protocol

Razya Ladelsky ✉
Technion, Israel

Roy Friedman ✉
Technion, Israel

Abstract

DAG-based Byzantine Fault Tolerant (BFT) protocols provide high-throughput consensus under partial synchrony, but existing DAG protocols still require at least three message delays to commit decisions. In contrast, fast-path BFT protocols can achieve optimal two-message-delay termination under favorable conditions, though they do not naturally extend to DAGs.

We present FinWhale, the first DAG-based BFT protocol with a two-message-delay fast path. FinWhale extends Mysticeti with a novel fast-path commit mechanism that safely coexists with the protocol's original slow-path rules. To preserve safety across different local DAG views, we introduce new commit structures based on FP-evidence blocks, enabling validators to combine fast-path and slow-path reasoning consistently.

FinWhale operates in the partially synchronous model with $n = 3f + 2p - 1$ validators, matching the known lower bound for fast Byzantine consensus. The protocol tolerates up to f Byzantine faults and achieves fast termination whenever at most $1 \leq p \leq f$ validators fail during the fast path. Our results show that optimal-latency fast paths can be integrated into uncertified DAG consensus protocols.

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases DAG protocols, Byzantine Atomic Broadcast, Consensus, Fast Termination

1 Introduction

Directed Acyclic Graph (DAG)-based protocols aim to provide a Byzantine Fault Tolerance (BFT) replicated ledger, or blockchain, abstraction, i.e., to maintain a consistent total order of transactions across distributed validators, even in the presence of malicious and faulty nodes [2, 3, 4, 6, 9, 10, 11, 12, 17, 22, 24, 25]. DAG-protocols are claimed to offer higher throughput even under asynchrony and failures compared to classical leader-based protocols. However, many DAG protocols suffer from high latency. Mysticeti is a recent DAG-based BFT protocol that can achieve the latency lower bound of three message delays and maintains low latency even in the presence of certain crash failures. As an indication of its practical relevance, Mysticeti has been recently adopted by both the Sui and IOTA blockchains.

Interestingly, it was shown that BFT consensus can be accelerated with a *fast path*, which allows validators to commit values in just two message delays under favorable conditions. This matches the optimal latency achievable in crash fault-tolerant systems. Early work by Martin and Alvisi, who introduced the FaB protocol [16], presented a two-round termination algorithm built on PBFT with $n \geq 5f + 1$ validators, along with a parameterized variant requiring $n = 3f + 2p + 1$, where f denotes the total number of faulty validators tolerated, and $p \leq f$ represents the number of faulty validators that need not participate for the fast path to succeed.

Subsequent works by Kuznetsov et al. [14] and Abraham et al. [1] further reduced these thresholds to $n = 5f - 1$ and $n = max(3f + 2p - 1, 3f + 1)$ (for $p \leq f$), which have been shown to constitute lower bounds for fast-path BFT partially synchronous protocols.

Banyan [26] introduced a fast path that achieves fast termination in just two message delays, while departing from traditional PBFT-style designs: it is built on the Internet Computer Consensus (ICC) protocol [5] and operates with $n = 3f + 2p - 1$ validators, where $p \geq 1$.

In this work, we address the question of whether decisions within two message delays can be achieved in DAG-based protocols. In DAG protocols, the graph continuously grows as rounds advance, and the challenge lies in identifying commit patterns within an already constructed DAG. Our approach integrates fast-path commit patterns with those used for slow-path commits, enabling both paths to coexist consistently within the DAG[1].

Our Contribution: In this work, we introduce *FinWhale*, a fast-path mechanism for the Mysticeti protocol, making it the first DAG-based protocol to support termination in just two message delays. We prove both safety and liveness under partial synchrony with $n = 3f + 2p - 1$ validators, matching the lower bound. The protocol tolerates up to f Byzantine failures and achieves fast termination when the number of actual faults does not exceed p ($1 \leq p \leq f$), the round leader is honest, and the system has passed GST. This is while preserving the core guarantees and benefits of Mysticeti.

The rest of the paper is organized as follows. Section 2 describes the system model and problem definition. Section 3 reviews the Mysticeti protocol. Section 4 presents the FinWhale protocol. An execution example of FinWhale is illustrated in Section 5. Formal safety and liveness proofs are provided in Section 6 and Section 7. Section 8 discusses related work, and Section 9 concludes the paper.

2 Model

As is common in many BFT works, we assume a message-passing network among a set of n validators, connected via reliable and authenticated point-to-point links. Messages sent by honest validators are eventually delivered and cannot be forged, duplicated, or altered by the adversary. The adversary is computationally bounded and cannot break standard cryptographic primitives such as digital signatures or hash functions. In particular, validators cannot impersonate each other.

The network operates under the *partially synchronous* model [7]. That is, in each run of the system, there exists a Global Stabilization Time (GST), which is unknown to the validators. Before GST, message delays are arbitrary and unbounded. After GST, the network becomes synchronous: all messages between honest validators are guaranteed to be delivered within a known bounded delay Δ.

We assume that the system can have up to f *Byzantine validators*, i.e., validators that may behave arbitrarily, including maliciously, by sending conflicting or incorrect messages to disrupt the protocol. The remaining validators are considered *honest*, meaning they follow the protocol exactly as specified.

The total number of validators participating in the protocol is $n \geq 3f + 2p - 1$, where p is a parameter satisfying $1 \leq p \leq f$. Intuitively, p serves as a flexible threshold for enabling a *fast-path decision*: if the number of faults in the system is at most p, the protocol can potentially reach consensus quickly using the fast path. However, even when the number of Byzantine faults exceeds this threshold, the protocol still tolerates up to f Byzantine faults overall by relying on a slower, more conservative decision process to ensure safety and

[1] The Mysticeti paper [3] introduces Mysticeti-FPC, which provides a fast path for transactions that do not require consensus. In contrast, our work adds a fast path to the consensus protocol itself, covering all types of transactions.

liveness. Setting $p = 1$ enables the protocol to operate with $3f + 1$ validators, achieving the optimal resilience bound [7]. In this setting, the protocol takes the fast path after GST under an honest leader provided that at most one validator fails. In contrast, setting $p = f$ guarantees that, under an honest leader after GST, the fast path is always taken.

As in prior DAG works [3, 11, 24], the protocol's goal is to solve the *Byzantine Atomic Broadcast* (BAB) problem. Specifically, a validator v_k can invoke $\mathsf{a\_bcast}_k(m, id)$ to broadcast a message m with a unique sequence number $id \in \mathbb{N}$. Each validator v_i outputs delivered messages via $\mathsf{a\_deliver}_i(m, id, v_k)$, indicating that v_i has delivered the message m associated with id that was broadcast by v_k. A protocol solving the BAB problem must satisfy the following properties:

1. **Agreement:** If an honest validator v_i outputs $\mathsf{a\_deliver}_i(m, id, v_k)$, then every other honest validator v_j eventually outputs $\mathsf{a\_deliver}_j(m, id, v_k)$.
2. **Integrity:** For each sequence number $id \in \mathbb{N}$ and validator v_k, there exists at most one message m such that an honest validator v_i outputs $\text{deliver}_i(m, id, v_k)$.
3. **Validity:** If an honest validator v_k calls $\mathsf{a\_bcast}_k(m, id)$, then every honest validator v_i eventually outputs $\mathsf{a\_deliver}_i(m, id, v_k)$.
4. **Total Order:** If an honest validator v_i outputs $\mathsf{a\_deliver}_i(m, id, v_k)$ before $\mathsf{a\_deliver}_i(m', id', v'_k)$, then no honest validator v_j outputs $\mathsf{a\_deliver}_j(m', id', v'_k)$ before $\mathsf{a\_deliver}_j(m, id, v_k)$.

3 Mysticeti Overview

The Mysticeti paper [3] refers to the core DAG-based atomic broadcast protocol as Mysticeti-C. For brevity, in the rest of this work, we simply refer to both the system and the protocol as Mysticeti.

Below, we present the basic building blocks of Mysticeti, which also constitute the slow path of our protocol. The original construction as described in [3] was shown to suffer from liveness issues [18, 22], which were subsequently resolved in [17]. We therefore adopt the version of Mysticeti as presented in [17] , which augments the protocol with a push-based pacemaker and provides provable liveness guarantees.

DAG structure: The protocol proceeds in consecutive rounds. In each round r, every honest validator broadcasts a single signed block containing its identifier (author), the round number r, payload (users' transactions), and a set of n edges to the latest blocks created by distinct validators in rounds up to $r - 1$. Among these edges, at least $n - f$ refer to blocks created in round $r - 1$.

Each edge includes the hash of a previously received block, and the referenced block is called a *parent* of the new block. In addition, every block includes an edge that references the previous block created by the same validator.

A block is said to be *valid* if it conforms to the above block structure and its signature is correctly verified under the public key of its author. Each honest validator maintains a local DAG consisting of the blocks it has created and the valid blocks it has received. A block is inserted only when all its parent blocks are present; otherwise, it is buffered until the missing parents are received. The conditions under which a validator creates a block and advances to the next round are defined below when we discuss the pacemaker.

When an $\text{a\_bcast}_i(m, id)$ is invoked at validator v_i, the tuple (m, id) is pushed to the DAG layer. Validator v_i will include (m, id) in the payload field of its next created block.

Leader slots: The protocol deterministically designates a *leader* for each round, according to a round-robin schedule over the validators. Each leader together with its round defines a

leader slot. Note that if the leader equivocates, multiple blocks may be associated with the leader slot.

Each leader slot is initially marked as `undecided`. Each validator analyzes its local DAG and decides, for every leader slot, whether its state should be changed to `to-commit` or `to-skip`. Once taken, such decisions are non-reversible.

A slot marked `to-commit` designates its associated leader block as `committed`, and the decision rules ensure that at most one block is committed per slot. The committed leader blocks determine the ordering of all blocks in the DAG.

We now define the notion of voting, a key concept in identifying DAG-based patterns underlying the decision rules.

Voting: A block b' in round $r+1$ *votes for* a leader block b in round r if b is one of its parents. Clearly, a block created by a Byzantine validator may reference more than one leader block; in this case, it votes for the first leader block in its parent set.

Since Mysticeti serves as the slow path of our protocol, we retain its certificate-based decision mechanism, and refer to it as the *slow-path certificate (SP-certificate)*, and refer to its skip condition as the *slow-path skip pattern (SP-skip)*.

SP-certificate: Given a leader block b at round r, a block b' at round $r+2$ is called an *SP-certificate* for b if it references a quorum of $\lceil \frac{n+f+1}{2} \rceil$ blocks from distinct validators that vote for b.

SP-skip: An *SP-skip pattern* occurs when, for each block b of the leader slot (there may be multiple such blocks if the leader equivocates), there exists a quorum of blocks from distinct validators in round $r+1$ that do not vote for b. Particularly, if a block b of the leader slot is not in the local DAG, then all observed blocks in round $r+1$ trivially do not vote for b.

Note that in case of conflicting leader blocks for the same slot, at most one of them can have an SP-certificate, since by quorum intersection, at most one can garner the required quorum of blocks from distinct validators that vote for it. This property is crucial for the correctness of the decision rules.

Decision rules: Each validator attempts to decide each leader slot by first applying the direct decision rule, and then applying the indirect commit rule to earlier undecided slots. This process proceeds from the highest decided round to the lowest undecided round.

The *direct decision rule* marks a leader slot at round r as `to-commit` if there exists a quorum of SP-certificates for its leader block at round $r+2$, in which case that block is *committed*(see Fig. 2a). The slot is marked `to-skip` if the DAG contains an SP-skip pattern for the slot.

The direct commit rule enables committing leader blocks within three message delays when there are no failures and the system is fully synchronized. The direct skip rule allows slots to be decided quickly when the leader has crashed or failed to produce a valid block, thereby reducing the number of undecided slots.

If the direct decision rule does not decide the leader slot at round r, the validator applies the *indirect decision rule*: The validator first searches for an *anchor*. The anchor is the earliest leader slot with round number greater than $r+2$ that is marked as either `to-commit` or `undecided`. If the anchor is `undecided`, then the slot remains `undecided`. If the anchor is marked `to-commit`, the validator checks whether there exists a path from the anchor to an SP-certificate for a block from the leader slot. If such a path exists, the slot is marked `to-commit` and the corresponding block is *committed* (see Fig. 2d); otherwise, it is marked `to-skip`.

Commit sequence: After applying the decision rules to all leader slots, the validator extends its *commit sequence* in ascending order of round numbers. Leader slots marked as

`to-skip` are omitted, while committed blocks from slots marked as `to-commit` are included in the sequence. The sequence terminates at the first `undecided` slot. The safety and liveness proofs of Mysticeti establish that all honest validators commit a consistent sequence of leader blocks and that, eventually, every slot is decided.

Total ordering: Validators obtain a total order over all blocks in the DAG by deterministically sorting the causal histories of leader blocks in the commit sequence. Specifically, for each leader block in the commit sequence, the latter part of its causal history that has not been already ordered by previous leader blocks is deterministically sorted and added to the total ordering sequence. For each ordered block b, validators deliver $(m, id, b.author)$ tuples derived from the (m, id) tuples in the payload of b. Any $(m, id, b.author)$ tuple that has already been delivered is filtered out, ensuring that each tuple is delivered exactly once.

Safety intuition: If a validator commits a block b for a given slot, then no other validator can skip that slot or commit a conflicting block for the same slot. To that end, a key property proved in [3, 17] and by our Lemma 3 states that: if b is committed directly by some validator, then a path from the anchor to an SP-certificate for b exists in every validator's DAG. This property rules out indirect skipping by definition. Moreover, the presence of an SP-certificate, whether b is committed directly or indirectly, excludes both direct skipping and committing a conflicting block, by quorum intersection. The remaining case is when b is committed indirectly by different validators. In this case, all validators agree on the same committed anchor block, as shown in [3, 17] and by our Lemma 12. Consequently, they observe the same causal history and reach the same decision for that slot.

Push Pacemaker: The Push Pacemaker is responsible for ensuring liveness, by specifying the conditions for advancing rounds, creating blocks, and broadcasting unknown parts of the history. It consists of the following aspects:

Advancing rounds: A validator advances from round $r-1$ to round r when the following two conditions are satisfied:

A1: There are $n-f$ round $r-1$ blocks from distinct validators in the local DAG.

A2: The validator has created its own block in round $r-1$.

Creating blocks: After advancing to round r, a validator records its local time. We set $\delta_{LT} = 2\Delta$ as the timeout duration. The validator then creates its block when one of the following three conditions holds:

C1: The following two requirements are satisfied:

- **L1:** The local DAG contains a leader block from round $r-1$.
- **L2:** The local DAG contains either (i) a quorum of voters from distinct validators for the leader of $r-2$, or (ii) an SP-skip pattern for the leader of $r-2$.

C2: The δ_{LT} timeout has expired.

C3: The local DAG contains $n-f$ blocks from distinct validators in round r.

When **C1** triggers block creation, the validator should choose to include in its parents the set of blocks that were used to satisfy requirement **L2** (it is important in cases where there are multiple versions of $r-1$ blocks belonging to Byzantine validators). Waiting for **C1** to be satisfied ensures that after GST, honest leaders get committed. **C2** requires that upon expiration of a timeout, the validator creates a block, even if condition **C1** is not satisfied. **C3** ensures that slower validators can catch up: if **C1** is not satisfied but $n-f$ blocks have already been produced in round r, a validator may proceed without waiting for the timeout to expire.

Broadcasting unknown history: When a validator receives a block, it attempts to add it to its local DAG only if all its parents (and, transitively, its entire causal history) are

already present in the DAG. Since Byzantine validators may send blocks to only a subset of validators or send conflicting versions to different validators, when an honest validator creates and sends its block, its parents may include blocks created by Byzantine validators. To ensure that such blocks can be added to other validators' DAGs, validators follow the principle of cordial dissemination [19], by which a validator v_i sends to another validator v_j the blocks in its local DAG that it believes v_j did not yet receive. Formally, let DAG_i denote the set of blocks in the local DAG of v_i. From the perspective of v_i, the set of blocks known to v_j is defined as $\text{known}_i(j) = H_{i,j} \cup S_{i,j}$, where $H_{i,j}$ is the union of the causal histories of all blocks in DAG_i created by v_j, and $S_{i,j}$ is the set of all blocks sent from v_i to v_j. Accordingly, the set of blocks unknown to v_j is $\text{unknown}_i(j) = \text{DAG}_i \setminus \text{known}_i(j)$.

Upon any of the following pacemaker triggers, a validator v_i sends the unknown portion of its history to its peers:

B1: creating a block.

B2: advancing to a new round.

The Push pacemaker ensures synchronicity after GST: when a fast honest validator advances to a new round, its history is broadcast to all validators via **B2**. As a result, they can create blocks in all rounds up to that round via **C3** and advance to the new round according to **A1** and **A2**. Moreover, **B1** ensures that if a fast honest validator creates a block under **C1**, then all honest validators observe the same conditions and can therefore also create their blocks under **C1**.

Liveness intuition: With the Push pacemaker, [17] establishes synchronicity after GST: validators enter each new round within Δ and create blocks within Δ of one another. This property was assumed without proof in [3]. Building on this, [3, 17] show that, after GST, any honest leader slot is eventually marked `to-commit` by the direct decision rule, and that all leader slots are eventually decided. Consequently, every honest leader block is included in the commit sequence. Moreover, any block created by an honest validator eventually becomes part of the causal history of some committed honest leader block, and is therefore ordered and delivered.

4 *FinWhale*

4.1 Protocol Description

In addition to the Byzantine Atomic Broadcast properties, FinWhale satisfies an additional *Fast Termination* property. We adapt its definition from Banyan [26] to our setting.

▶ **Definition 1.** *If the network is momentarily synchronous, the leader is honest, and at least $n - p$ validators behave momentarily honest, then the leader block is committed (its slot is marked `to-commit`) within two message delays.*

To support the Fast Termination property, we introduce an additional direct commit rule: A leader block b of round r is *fast-directly committed* if there exist $n - p$ blocks from distinct validators in round $r + 1$ that reference b, i.e., if there are $n - p$ voters for b. With this rule, a leader block can be committed within two message delays when network conditions permit.

In Mysticeti, the presence of a direct commit pattern within a given DAG guarantees that any other DAG will contain a corresponding pattern serving as evidence that neither a skip can be determined for the slot nor a conflicting block can be committed. Similarly, we require that if a fast direct commit of the leader block occurs in the DAG of some validator, then every other validator must observe in its DAG a corresponding pattern that rules out both conflicting commits and skip decisions for the same slot.

We introduce a new notion, where blocks in round $r+2$ serve as an *FP-evidence* for a leader block b of round r. Intuitively, when a block is committed directly via the fast path in a DAG of validator v_i during round $r+1$, it leaves evidence in round $r+2$ in all other DAGs in the form of FP-evidence blocks.

We note that a single FP-evidence block represents a weaker form of voting evidence than an SP-certificate, and the DAG may simultaneously contain FP-evidence for conflicting leader blocks. However, the presence of a quorum of such FP-evidence blocks suffices to guarantee consistent behavior across all honest validators' DAGs.

Block construction: In Mysticeti, advancing from the current round to the next round according to condition **A1** requires observing $n-f$ blocks from different validators. To enable the fast path under the tighter bound $n = 3f+2p-1$, an honest validator must take into account not only the blocks from the current round, but also the behavior of the leader of the previous round[2].

Specifically, a Byzantine leader in the previous round may equivocate by broadcasting multiple conflicting versions of its block. However, if an honest validator receives blocks that reference more than one version of the leader block, then this is an evidence that the previous round leader is Byzantine. In this case, the validator can safely wait for $n-f$ blocks not including the block authored by the Byzantine leader without blocking. Consequently, the validator observes and references at least $n-f$ blocks, among which at most $f-1$ are from Byzantine validators, because the block authored by the detected equivocating leader is not included as a parent when the validator constructs its next block.

Throughout this section, we denote L_r the leader of round r. In particular, L_{r-1} and L_{r-2} denote the leaders of rounds $r-1$ and $r-2$, respectively.

Leader-consistent. A set of round-r blocks B_r is said to be *leader-consistent* with respect to L_{r-1} if all blocks in B_r vote for at most one block proposed by L_{r-1}.

Block validity As before, blocks are added to the DAG of an honest validator only if it obeys the block validity rules. FinWhale's definition of a valid block extends Mysticeti's validity rules by adding the following requirements:

1. Each block has at most a single parent per validator. We do not allow malformed blocks (e.g., those created by Byzantine validators) that contain multiple edges to blocks of the same validator.
2. A block in round r must be valid with respect to round $r-2$. That is, it links to at least $n-f$ blocks from round $r-1$, each from a distinct validator, and satisfies one of the following conditions:
   - **(a)** the set of parent blocks is leader-consistent with respect to L_{r-2}, or
   - **(b)** the round-$r-1$ block created by L_{r-2} is not included among the parents.

FP-evidence, Non-FP-evidence: We say that a block b' of round $r+2$ *exposes equivocation by* L_r if its parent set is not leader-consistent, i.e., its causal history contains multiple conflicting versions of L_r's block.

A block b' of round $r+2$ is an *FP-evidence* for a leader block b of round r if one of the following conditions holds:

- **Equivocating case:** b' exposes equivocation of L_r. It references at least $f+p$ parent blocks that vote for b, while fewer than $f+p$ of its parent blocks vote for any block conflicting with b.

[2] This additional requirement does not arise under the looser bound $n = 3f+2p+1$, as discussed later.

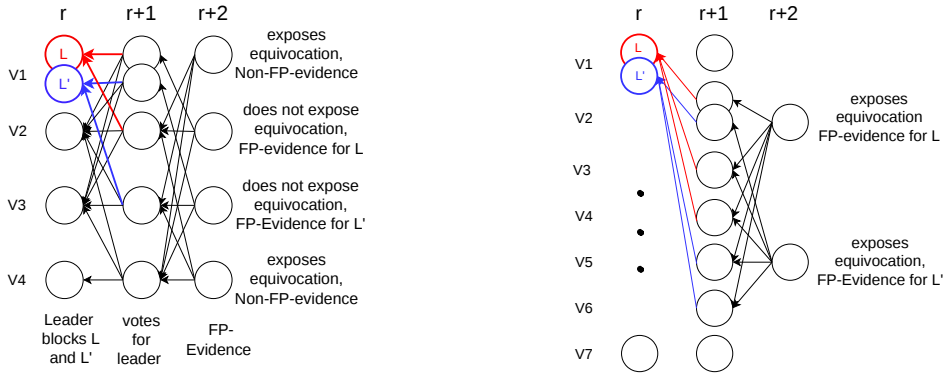


Figure 1 FP-evidence and Non-FP-evidence examples under equivocation. V_1 is an equivocating leader in round r, and votes in round $r+1$ are colored according to the leader block they support.
(a) FP-evidence and Non-FP-evidence examples illustrating the cases where equivocation is exposed and not exposed with $f = p = 1$. If a block in round $r+2$ exposes the equivocation, it qualifies as FP-evidence for L if it has at least two $(f+p)$ parents voting for L and at most one $(f+p-1)$ parent voting for L'. If the block does not expose the equivocation, a single (f+p-1) parent voting for L is sufficient to qualify as FP-evidence for L. Symmetrically, the same conditions apply for L'. Any block that neither qualifies as an FP-evidence for L nor for L' is a Non-FP-evidence block.
(b) FP-evidence for conflicting leader blocks while exposing equivocation, with $f = 2, p = 1$. Both round-$(r+2)$ blocks expose the equivocation. The round-$(r+2)$ block created by V_2 has three $(f+p)$ parents voting for L and two $(f+p-1)$ parents voting for L', and therefore qualifies as an FP-evidence for L. Symmetrically, the round-$(r+2)$ block created by V_5 qualifies as an FP-evidence for L'.

- **Non-equivocating case:** b' does not expose equivocation of L_r. It references at least $f+p-1$ parent blocks that vote for b.

Finally, a block at round $r+2$ is a *Non-FP-evidence* if it is not an FP-evidence for any block proposed by L_r.

Examples of FP-evidence and Non-FP-evidence blocks are in Figure 1.

A key property we prove in Lemma 4 below is that if a leader block of round r is directly committed via the fast path, then every block at round $r+2$ in any DAG is an FP-evidence block for b.

FinWhale relies on the same definitions of SP-certificate and SP-skip patterns as Mysticeti. Recall that in our setting, a quorum of size $\left\lceil \frac{n+f+1}{2} \right\rceil$ equals $2f+p$. With the additional definitions of FP-evidence blocks and Non-FP-evidence blocks, we are now ready to formally define FinWhale's decision rules. Here again, each validator iterates over rounds from the highest to the lowest undecided round and applies the decision rules.

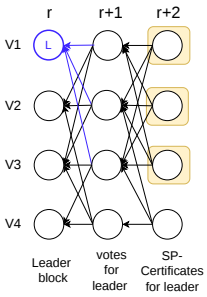
Direct decision rules: The validator marks a leader slot of round r as `to-commit` if it observes *one of* the following cases:

1. There exists a quorum of $\left\lceil \frac{n+f+1}{2} \right\rceil = 2f+p$ SP-certificates from distinct validators in round $r+2$ for a block b of the slot. b is then *committed*.
2. There exist $n-p$ blocks from distinct validators in round $r+1$ that vote for a block b of the slot. b is then *committed*.

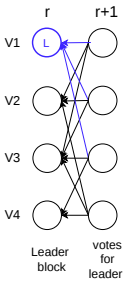
The validator marks a slot as `to-skip` when *both of* the following conditions hold:

1. The SP-skip pattern is observed in round $r+1$.
2. There are at least $\left\lceil \frac{n+f+1}{2} \right\rceil = 2f+p$ Non-FP-evidence blocks from distinct validators in round $r+2$.

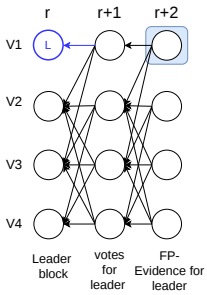
Indirect decision rule: If during the reverse pass over undecided round, the direct decison rules did not decide a given slot, the validator tries to apply the indirect decsion rules: First



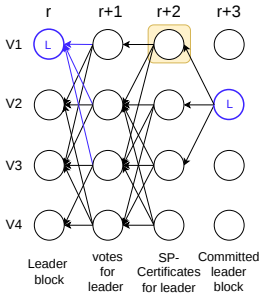
(a) Direct commit via the slow path. There is a quorum of three SP-certificates in round $r+2$, each containing a quorum of votes for the leader block.



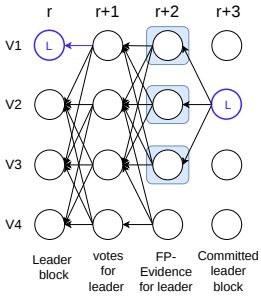
(b) Direct commit via the fast path. There are three $(n-p=3)$ votes in round $r+1$.



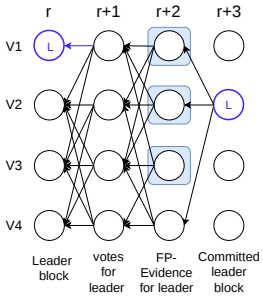
(c) Direct skip: There is a quorom of non-voters in $r+1$ and a quorum of Non-FP-evidence blocks for the leader in $r+2$.



(d) Inrirect commit: There is a path from anchor to an SP-certificate for the leader block.



(e) Indirect commit: There are paths from the anchor to a quorum of FP-evidence blocks for the leader block.



(f) Indirect skip: No path from the anchor to an SP-certificate. No paths to a quorum of FP-evidence blocks for the leader block.

Figure 2 FinWhale decision rules for $f=p=1$. Votes for the leader block are colored in blue. In this setting, there are four validators and the quorum size is 3.

it searches for an anchor, which is defined as in Mysticeti, to be the first slot with round number greater than $r+2$ that is marked either `to-commit` or `undecided`. If the anchor is `undecided` then the slot is marked `undecided` as well. If the anchor is marked `to-commit`, the validator marks the slot as `to-commit` if one of the following conditions holds:

1. There exists a path from the anchor to an SP-certificate for a block b of the slot, in which case b is *committed*.
2. There exist paths from the anchor to a quorum of $\left\lceil \frac{n+f+1}{2} \right\rceil = 2f+p$ FP-evidence blocks from distinct validators, all for the same block b of the slot, in which case b is *committed*.

Otherwise, the slot is marked `to-skip`.

The decision rules are illustrated in Figure 2. Note that both of the above conditions may hold simultaneously for conflicting blocks of the same slot. In such a case, one of the blocks is selected for commitment according to a deterministic rule. Such a pattern cannot arise if any validator can decide the slot directly, either by direct skip or direct commit: a direct skip rules out the conditions for an indirect commit, and a direct commit rules out the conditions for an indirect commit of a conflicting block. Hence, all validators must decide the slot indirectly, in which case they all reach the same decision. The patterns induced by direct decisions, as well as the consistency of indirect decisions, are described in the safety intuition and proved in the safety section.

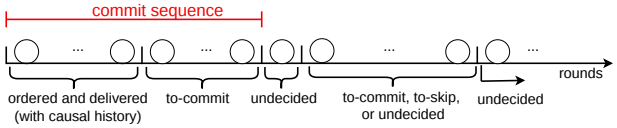


Figure 3 Leader slots over a timeline of rounds (r, r+1, r+2, ...); the state is annotated beneath each corresponding interval. The commit sequence is extended by committed leader blocks up to the first undecided slot.

Commit sequence: The algorithm proceeds similarly to Mysticeti. It examines leader slots from the highest round down to the lowest undecided round. For each slot, the validator applies the commit rules to determine its state. The commit sequence is then extended in ascending order of round numbers by including slots marked `to-commit` together with their committed blocks, while omitting slots marked `to-skip`. The process stops upon reaching the first `undecided` slot. Figure 3 illustrates how the commit sequence is extended by including committed leader blocks until reaching the first `undecided` slot.

Total order: As in Mysticeti, the final total order is obtained by deterministically sorting the causal histories of the committed leader blocks. For each ordered block b, all (m, id) pairs in its payload are delivered as $(m, id, b.\text{author})$, while filtering out any such tuples that have already been delivered.

Safety intuition: validators cannot reach conflicting decisions: they either both commit the same block for a leader slot s or both skip s. We rely on the following structural properties of the protocol:

1. A slow-path direct commit of block b yields a path from the anchor to an SP-certificate for b in every DAG (Lemma 3).
2. A fast-path direct commit yields a path from the anchor to a quorum of FP-evidence blocks for b in every DAG (Lemma 5).
3. An SP-certificate for b is also an FP-evidence for b (Lemma 2).

We claim that if a validator directly skips a leader slot s, no other validator can commit a block for the slot (Lemma 6). A direct skip requires collecting a quorum of blocks not voting for any block of s together with a quorum of Non-FP-evidence blocks. By quorum intersection, this precludes the existence of an SP-certificate or a quorum of FP-evidence blocks for any block of slot s in any other DAG. Since either of these is necessary for an indirect commit, the latter is impossible. Further, the presence of a quorum of non-voters for any leader blocks also rules out a direct commit in any other DAG.

If a validator directly commits a block b of slot s, then no other validator can skip s (Lemma 7) or commit a conflicting block (Lemma 9 and Lemma 10). When a validator directly commits b, depending on whether the slow or fast path is taken, any other DAG contains either a path from the anchor to an SP-certificate for b or paths from the anchor to a quorum of FP-evidence blocks for b (Properties 1 and 2, respectively). This already rules out indirect skipping in any other DAG by definition. To show consistency with respect to a conflicting block b', observe that a direct commit of b implies a quorum of voters for b, which by quorum intersection rules out the existence of an SP-certificate for b'. It also rules out a quorum of FP-evidence blocks for b'. Indeed, if b is committed via the fast path, then by Property 2 every DAG contains a quorum of FP-evidence blocks for b. If b is committed via the slow path, the SP-certificates for b form a quorum, which in turn induces a quorum of FP-evidence blocks for b by Property 3. In both cases, quorum intersection prevents the existence of an FP-evidence quorum for b'. The absence of either an SP-certificate or

a quorum of FP-evidence blocks for b' precludes an indirect commit of b'. Moreover, by a simple quorum intersection argument, a direct commit of b' is also impossible.

Finally, if validators decide s indirectly, they rely on the same committed anchor block, and therefore observe the same causal history and reach the same decision (Lemma 12).

Pacemaker conditions: Similarly to Mysticeti, we define pacemaker conditions for FinWhale that ensure both liveness and the creation of valid blocks.

Advancing rounds: An honest validator advances from round $r-1$ to round r when two conditions are satisfied: **A1'** and **A2**. Condition **A2** is inherited from Mysticeti and is restated here for completeness. Condition **A1'** determines the threshold on the number of round $r-1$ blocks (from distinct validators) that are required for advancement, as in Mysticeti's **A1**. However, here the value depends on whether a leader-consistent set is observed.

A1' If there exists a set of round $r-1$ blocks that is leader-consistent with respect to L_{r-2} and contains a block from each validator whose round $r-1$ block has been received, then the threshold is $n-f$; otherwise, the threshold is $n-f$, where blocks created by L_{r-2} are not counted.

A2 The validator has created its own block in round $r-1$.

Block creation: A block in round r is created if at least one of the following conditions is satisfied:

C1 The following two requirements are satisfied:
- **L1** The local DAG contains a block of L_{r-1}.
- **L2** The local DAG contains either (i) a quorum of voters from distinct validators for L_{r-2} block, or (ii) contains an SP-skip pattern for L_{r-2}

C2 The timeout $\delta_{LT} = 2\Delta$ has expired.

C3 The local DAG contains $n-f$ blocks from distinct validators in round r.

Selecting the parents: When a block is created in round r, the validator must select its parent set from round $r-1$ blocks. Byzantine validators may produce multiple blocks in round $r-1$, but the validator includes at most one parents per validator, while selecting the blocks that satisfy conditions **L1** and **L2** if condition **C1** triggers block creation.

Whenever the parent set is not leader-consistent with respect to L_{r-2}, the round-$(r-1)$ block created by L_{r-2} must be excluded. Accordingly, the parent selection algorithm needs to exclude this block whenever the resulting parent set violates leader-consistency with respect to L_{r-2}. However, when there are round-$(r-1)$ blocks from exactly $n-f$ validators, excluding this block would leave only $n-f-1$ parents, violating the requirement that a parent set contains at least $n-f$ blocks. Therefore, in this case, we must guarantee that multiple round-$(r-1)$ blocks produced by Byzantine validators do not prevent the construction of a leader-consistent parent set. Condition **A1'** ensures the existence of such a leader-consistent set of size $n-f$.

Formally, the parent selection proceeds as follows:

1. Initialize the candidate set with the blocks in the local view of $DAG[r-1]$.
2. If the round-$(r-1)$ block proposed by L_{r-2} is present and round-$(r-1)$ blocks from exactly $n-f$ distinct validators are available, construct a leader-consistent set with respect to L_{r-2} that contains at least one block from each of these validators. Such a set exists by condition **A1'**.
3. Construct the parent set from the candidate set such that: (1) for each validator v, the parent set contains exactly one block authored by v, and (2) if condition **C1** triggers block

creation, all blocks in the candidate set that satisfy condition **L1** and **L2** are included in the parent set.

4. If the current parent set is not leader consistent w.r.t. L_{r-2}, exclude the round-$(r-1)$ parent block proposed by L_{r-2} from the parent set, even if it was previously included (by the previous step).

Finally, the validator includes blocks from rounds older than $r-1$ in its parent set as follows: the latest block from each author that has not yet been referenced by a previous block of the validator is included in the parent set. This ensures that every block created by an honest validator appears in the causal history of some honest leader block, and is therefore ordered and its payload delivered.

Broadcasting history: A validator broadcasts its unknown history under the same **B1** and **B2** conditions as in Mysticeti.

Liveness intuition: Liveness in FinWhale is based on the liveness of the slow path. After GST, every leader block created by an honest validator is eventually included in the commit sequence, and its causal history is delivered. When conditions allow the fast path to be taken, leader blocks can be marked `to-commit` more quickly, thereby accelerating total ordering and delivery.

Additional Discussion Points: It is possible to simplify FinWhale by relaxing the resilience threshold from the lower bound of $n = 3f + 2p - 1$ validators to $n = 3f + 2p + 1$. For instance, this would enable a simpler DAG construction, in which it is not necessary to wait for an additional block in round $r + 1$ when the round-r leader is proven to be Byzantine. Moreover, eliminating the need to distinguish between equivocating and non-equivocating cases would simplify the definition of FP-evidence.

Under the relaxed assumption $n = 3f + 2p + 1$, there exist scenarios in which Byzantine leader blocks may be directly committed despite equivocation. In contrast, under the tighter bound of $n = 3f + 2p - 1$, enforcing that at most $f - 1$ Byzantine blocks are selected as parents prevents such direct commits. These scenarios are rare and arise only under highly specific conditions.

Mysticeti [3] targets direct skipping as a mechanism to efficiently handle benign crashed leaders by promptly skipping them and reducing their impact on performance. When the leader is Byzantine, however, it may attempt to prevent direct skipping, thereby leaving the slot undecided. In particular, to preclude the formation of a quorum of non-voters after GST, a Byzantine leader must ensure that its block is disseminated to sufficiently many honest validators so that they produce voting blocks; otherwise, a quorum of non-voters may form, leading to a direct skip. On the other hand, once honest validators create voting blocks, they broadcast their history, which may cause other honest validators to receive the leader's block and form an SP-certificate, resulting in a direct commit. Therefore, a Byzantine leader must carefully and selectively disseminate its block so that enough honest validators vote for it, while ensuring that their history broadcasts do not propagate in time to enable SP-certificate formation. This leaves the leader slot undecided. In FinWhale, however, direct skipping requires a quorum of non-FP-evidence blocks in addition to a quorum of non-voters. This makes the Byzantine attack simpler in the optimal-resilience setting. In particular, the leader may withhold its block from all honest validators, while all f Byzantine validators send their round-$r + 1$ blocks voting for the leader. When $p = 1$, these f Byzantine votes are sufficient to cause honest validators to produce FP-evidence blocks, thereby preventing a direct skip. As in Mysticeti, when the system is configured with $n = 3f + 2p + 1$ validators, this simple attack is no longer feasible. It remains an open question whether achieving the lower bound on n inherently facilitates this simple attack that prevents direct skipping.

FinWhale differs from Mysticeti in the timing at which the direct-skip decision is made. In our protocol, a direct skip can occur only at the end of round $r + 2$, whereas in Mysticeti, a direct skip may occur already in round $r + 1$. However, this difference does not actually delay the commit sequence. The purpose of the direct skip rule is to decide slots as early as possible, so as not to delay the commit of subsequent leaders. Since in Mysticeti the next leader can be committed only at round $r + 3$, our modified direct skip decision does not introduce any additional delay in the overall commit sequence.

4.2 Algorithm and Pseudo Code

The pseudocode for FinWhale is given in Algorithms 1–5. It is based on the Mysticeti design, with the necessary modifications to enable the fast path. Algorithms 1 and 2 describe the construction of the DAG, including executing rounds, creating blocks, advancing rounds, and broadcasting unknown history. Algorithm 3 presents the helper functions, while Algorithm 4 introduces new helper functions required specifically for enabling the fast path. Algorithm 5 describes the logic for deciding leader slots, ordering blocks in the DAG, and delivering payloads. In the code, we use $DAG[r]$ to refer to the set of all blocks from round r stored in the local DAG.

5 An Example of FinWhale Execution

Figure 4 illustrates an example of the DAG with four validators at a specific moment. The protocol iterates the rounds from highest, round 5, to the lowest undecided round, round 1. Initially they are all `undecided`. The leader of round 5 remains `undecided` as there are no blocks in subsequent rounds voting for it. The leader of round 4 is directly committed via the fast path, since it has more than $n - p = 3$ voters in round 5.

The leader of round 3 is directly skipped due to the following two conditions: (a) it has a quorum (3) of non-voters in round 4; and (b) in round 5, there is a quorum of Non-FP-evidence blocks: each such block references no voters for the round-3 leader block and therefore qualifies as a Non-FP-evidence block.

The leader of round 2 remains `undecided`. It has only two voters, and is therefore neither directly skipped (which requires at least three non-voters) nor directly committed (which requires at least three voters). Moreover, its anchor in round 5 is still `undecided`.

The leader block of round 1 cannot be decided directly. It is not directly committed because there is only one vote for it in round 2. It is also not directly skipped because, in round 3, there are no Non-FP-evidence blocks. All round-3 blocks are FP-evidence for the round-1 leader block, as each references the round-2 block of V_1, which votes for that leader block. In the absence of equivocation, referencing $f + p - 1 = 1$ voter is sufficient to qualify as FP-evidence.

However, the leader block of round 1 can be indirectly committed. The round-4 leader block is marked `to-commit` and therefore serves as the anchor. Moreover, it has paths to a quorum (3) of round-3 blocks that are FP-evidence for the round-1 leader block. Thus, the round-1 leader block is indirectly committed.

The commit sequence is extended to include all leader blocks committed up to the first `undecided` slot. Therefore, the leader of round 1 is included in the commit sequence, all blocks in its causal history are ordered (if they have not already been ordered by a previous leader), and their payloads are delivered.

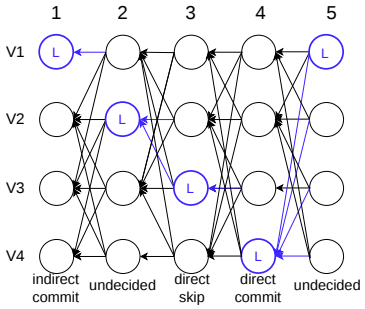


Figure 4 Example of FinWhale execution with four validators. Votes for the leader block are colored in blue.

6 FinWhale's Safety Proof

In this section, we establish the *safety* of our protocol. Specifically, we prove that the commit sequence is consistent among all honest validators, and consequently that the integrity and total order properties of Byzantine Atomic Broadcast (BAB) hold. To account for the fast path, we adapt existing statements and proofs where necessary, introducing the additional arguments required for fast-path commits while preserving the overall proof structure.

▶ **Lemma 2.** *Any SP-certificate for block b is also an FP-evidence for b.*

Proof. Whenever there is an equivocation, an FP-evidence for b includes at least $f+p$ parents voting for b and at most $f+p-1$ voting for any conflicting block b'. An SP-certificate for b has $\lceil \frac{n+f+1}{2} \rceil = 2f+p$ parents voting for b. Consequently, there can be at most $f+p-1$ parents voting for any conflicting block b'. In the absence of equivocation, the requirement for an FP-evidence, i.e., at least $f+p-1$ parents voting for b, is trivially satisfied. ◀

▶ **Lemma 3.** *Let b be a leader block in round r. If there are $\left\lceil \frac{n+f+1}{2} \right\rceil = 2f+p$ SP-certificates for b from distinct validators in round $r+2$, then every block created in any round $r' > r+2$ (in any DAG) will have a path to an SP-certificate for b from round $r+2$.*

Proof. Consider, first, blocks in round $r+3$. Each such block has at least $n-f$ references to blocks from round $r+2$. Since $n-f \geq 2f+p$, by quorum intersection the set of validators whose round-r+2 blocks are referenced intersects the set of validators contributing the 2f+p SP-certificates for b in at least one honest validator. Hence every block in round $r+3$ references at least one SP-certificate for b from round $r+2$. Now consider any block in a round $r' > r+3$. Its causal history contains at least $n-f$ blocks from round $r+3$, and by the previous case each of these blocks already references an SP-certificate from round $r+2$. Therefore, any block in round $r' > r+3$ has a path to an SP-certificate for b from round $r+2$. ◀

▶ **Lemma 4.** *Let b be a leader block in round r. If a validator v_i observes $n-p$ blocks from distinct validators voting for b in round $r+1$, then every block created in round $r+2$ (in any DAG) will be an FP-evidence for b.*

Proof. Every block in round $r+2$ references at least $n-f$ blocks from round $r+1$. Now suppose validator v_i observed $n-p$ blocks from distinct validators voting for b in round $r+1$:

- If the round-$r+2$ block exposes equivocation by the round-r leader, it does not reference the $r-1$ block of the Byzantine leader, so at most $f-1$ of its parents may be Byzantine. Consequently, it references at least $n-p-f-(f-1) = f+p$ blocks voting for b in round $r+1$, and at most $f+p-1$ blocks voting for a conflicting block. This satisfies the equivocating case of the FP-evidence definition.
- If the round-$r+2$ block does not expose equivocation, then among its $n-f$ references it must include at least $n-p-2f = f+p-1$ blocks voting for b in round $r+1$, which falls under the Non-equivocating case of the FP-evidence definition.

In either case, every block in round $r+2$ qualifies as an FP-evidence for b. ◀

▶ **Lemma 5.** *Let b be a leader block in round r. If a validator v_i observes $n-p$ blocks from distinct validators voting for b in round $r+1$, then every block created in round $r' > r+2$ (in any DAG) will link to at least $n-f$ FP-evidence blocks for b from round $r+2$.*

Proof. Every block in a round $r' > r+2$ contains in its causal history at least $n-f$ blocks created by distinct validators in round $r+2$. By Lemma 4, each block in round $r+2$ serves as an FP-evidence for block b. Therefore, any block in rounds greater than $r+2$ must reference at least $n-f$ FP-evidence blocks for b. ◀

▶ **Lemma 6.** *If an honest validator v_i commits a leader block b of slot s, no honest validator directly skips s.*

Proof. Suppose, by contradiction, that an honest validator v_j decides to directly skip slot s. Then both of the following conditions hold:

1. For each leader block of s (if any) in the local DAG of v_j, there exists a quorum of $2f+p$ blocks from distinct validators in round $r+1$ that do not vote for that leader block.
2. At least $2f+p$ blocks from distinct validators in round $r+2$ are Non-FP-evidence blocks.

Validator v_i can commit b in one of two ways:

- **Direct commit:**
  If v_i observes either $n-p$ blocks voting for b or $2f+p$ SP-certificates for b, then in both cases there exists a quorum of blocks from distinct validators voting for b.
  Any quorum of non-voters obtained by v_j intersects this quorum in at least one honest validator. Hence, b must appear in the local DAG of v_j.
  Therefore, v_j gathered a quorum of blocks that do not vote for b, while v_i observed a quorum voting for b, which is impossible by quorum intersection.
- **Indirect commit:**
  v_i observes a path from the anchor either to an SP-certificate for b, or to a quorum of FP-evidence blocks for b from distinct validators.

  - If there exists an SP-certificate for b, then a quorum of blocks from distinct validators vote for b. The contradiction then follows exactly as in the direct commit case.
  - If there exists a quorum of FP-evidence blocks for b, then by quorum intersection, at least one of the Non-FP-evidence blocks observed by v_j is itself an FP-evidence block for b.
    Hence, b appears in the local DAG of v_j. Since Non-FP-evidence is defined with respect to all leader blocks of slot s in the local DAG of v_j, including b, this yields a contradiction.

Therefore, if a validator commits a leader block, either directly or indirectly, no other validator can directly skip the slot. ◀

▶ **Lemma 7.** *Let b be a block of leader slot s in round r. If a correct validator v_i directly commits b, then no correct validator decides to skip slot s.*

Proof. By Lemma 6, if v_i directly commits block b of leader slot s, no other validator can directly skip s. The remaining case is that a validator attempts to *indirectly skip* s. This would require that an anchor block in round $r' > r + 2$ is marked *to-commit* and satisfies the following two conditions:

- it has no path to to an SP-certificate for b, and
- it has no paths to any quorum of FP-evidence blocks for b produced by distinct validators.

If v_i directly committed b via the slow path, then by Lemma 3 there will be a path from the anchor to an SP-certificate for b in round $r + 2$, which contradicts the first bullet. If v_i directly committed b via the fast path, then according to Lemma 5 the anchor block has paths to at least $n - f$ FP-evidence blocks for b (from distinct validators), which contradicts the second bullet above (as $n - f \geq quorum$). Hence, no correct validator can directly or indirectly skip slot s once v_i has directly committed b. ◀

▶ **Lemma 8.** *For any leader slot s, at most one block can gather a quorum ($2f + p$) of votes from distinct validators.*

Proof. Suppose, by way of contradiction, that two conflicting leader blocks gather a quorum of blocks from distinct validators that vote for each of them. The two quorums intersect in at least one honest validator. However, an honest validator cannot vote for two conflicting blocks of the same slot. A contradiction. ◀

▶ **Lemma 9.** *Consider a block b of slot s that is directly committed via the fast path (i.e., it gathers $n - p$ voters from distinct validators). No other block b' of the same slot s can be committed.*

Proof. Suppose, for the sake of contradiction, that there exists another block b' of slot s that is also committed. We consider all possible ways in which b' can be committed:

1. b' is committed directly via either the fast or the slow path, each of which requires at least $2f + p$ votes from distinct validators.
   Since $n - p$ blocks from distinct validators vote for b, and $n - p \geq 2f + p$, b' cannot gather a sufficient number of votes, by Lemma 8.
2. b' is indirectly committed via a path to an SP-certificate for b'.
   An SP-certificate for b' requires at least $2f + p$ votes from distinct validators. Since b already gathered at least $2f + p$ votes from distinct validators, this is impossible by Lemma 8.
3. b' is indirectly committed via a path to a quorum of FP-evidence blocks for b' from distinct validators.
   Since b was directly committed via the fast path by validator v_i, any honest validator can produce FP-evidence blocks only for b, and not for the conflicting block b', by Lemma 4.

In all cases, we reach a contradiction. Hence if a leader block is directly committed via the fast path, no other block of the same slot can be committed. ◀

▶ **Lemma 10.** *Consider a block b of slot s that is directly committed via the slow path. No other block b' of slot s can be committed.*

Proof. Suppose, for the sake of contradiction, that there exists another block b' of slot s that is also committed. We consider all possible ways in which b' can be committed:

1. b' is directly committed via either the fast or the slow path.
   Since b is directly committed via the slow path, there exist at least $2f + p$ votes for b from distinct validators.
   Any direct commit of b' also requires at least $2f + p$ votes from distinct validators, which is impossible by Lemma 8.
2. b' is indirectly committed via a path to an SP-certificate for b'.
   An SP-certificate for b' requires at least $2f + p$ votes for b' from distinct validators.
   Since b is directly committed via the slow path, there already exist at least $2f + p$ votes for b from distinct validators. Hence, by Lemma 8, such an SP-certificate for b' cannot exist.
3. b' is indirectly committed via paths to a quorum of FP-evidence blocks for b' from distinct validators.
   Since b is directly committed via the slow path, there exist $2f + p$ SP-certificates for b from distinct validators. By Lemma 2, any set of $2f + p$ SP-certificates for b also constitutes a set of $2f + p$ FP-evidence blocks for b.
   Therefore, by quorum intersection, there cannot exist $2f + p$ FP-evidence blocks for a conflicting block b' from distinct validators.

In all cases, we reach a contradiction. Hence if a leader block is directly committed via the slow path, no other block of the same slot can be committed. ◀

Combining Lemma 9 and Lemma 10 implies the following:

▶ **Corollary 11.** *If a validator directly commits a leader block, no other correct validator commits a different block for the same slot.*

▶ **Lemma 12.** *All honest validators decide a consistent state for each leader slot. Specifically, for any leader slot s, if two honest validators decide the state of s, then they either both skip s or both commit the same leader block.*

Proof. Let v_i and v_j be two honest validators. Assume, by contradiction, that they made inconsistent decisions for some slot. Let s be the latest leader slot for which such inconsistent decisions were made, and let r be the corresponding round of s. We consider two cases:

- Either v_i or v_j decided the state of s directly.
  Suppose one of them decided to directly skip slot s. Then, by Lemma 6, the other validator cannot decide to commit any block of s.
  If one of them decided to directly commit a block b of s, then by Lemma 7, the other validator cannot skip s.
  Moreover, by Corollary 11, no conflicting block of s can be committed.
- Both v_i and v_j decided indirectly.
  Validator v_i decided according to a committed anchor block in round r_i, and validator v_j decided according to a committed anchor block in round r_j.
  By the definition of the anchor, v_i decided to skip all slots in rounds $[r + 3, r_i)$ and then committed a leader block in round r_i. Similarly, v_j decided to skip all slots in rounds $[r + 3, r_j)$ and then committed a leader block in round r_j.
  By the maximality of r, the decisions of v_i and v_j are consistent for all slots greater than r. Therefore, $r_i = r_j$, and the same leader block was committed in that round.
  Since both validators use the same anchor block, they make the same decisions, as these depend only on the causal history of that block.

In both cases, we arrive at a contradiction to the assumption that v_i and v_j made inconsistent decisions. Therefore, all honest validators decide consistently for every leader slot. ◀

▶ **Lemma 13.** *All honest validators commit a consistent sequence of leader blocks; that is, the committed leader sequence of any honest validator is a prefix of that of any other honest validator.*

Proof. By Lemma 12, all honest validators make consistent decisions for every leader slot. The commit sequence of a validator contains all committed leader blocks up to the first `undecided` slot. Therefore, the committed leader sequence of any honest validator must be a prefix of that of any other honest validator. ◀

▶ **Theorem 14** (Total Order). *FinWhale satisfies the total order property of Byzantine Atomic Broadcast.*

Proof. Honest validators order the blocks in the DAG according to an identical deterministic sort induced by the causal histories of the committed leader sequence. By Lemma 13, all validators have a consistent sequence of committed leader blocks; therefore, all blocks are ordered identically across all validators.

Each ordered block b contributes tuples $(m, id, b.author)$ derived from its payload. Each $(m, id, b.author)$ tuple is delivered exactly once, as duplicate occurrences are filtered out upon delivery. Hence, the delivery sequence is a well-defined projection of the block order onto tuples.

Since all validators apply the same block ordering and the same deterministic filtering rule, the induced tuple delivery order is identical across all validators.

◀

▶ **Theorem 15** (Integrity). *FinWhale satisfies the integrity property of Byzantine Atomic Broadcast.*

Proof. The filtering mechanism applied by each honest validator to the the causal histories of blocks in the committed leader sequence ensures that each $(m, id, author)$ tuple is delivered exactly once. ◀

7 FinWhale Liveness

In this section, we establish liveness under partial synchrony. We first show that any honest leader block is eventually committed via the slow path after GST. Since any block created by an honest validator is part of the causal history of some honest leader block, it follows that every honest block is eventually ordered and its payload is delivered.

In addition, we show that if at most p validators behave Byzantine, an honest leader block can be committed via the fast path in just two rounds (fast termination).

Since liveness is established via the slow path, the proof follows [3], which provides the original liveness argument, and [17], which further refines and completes the argument, with minor adjustments required for the fast path.

We now introduce notation used in the liveness analysis.

Let $r_{\max}$ be the largest round reached by any honest validator at GST.

▶ **Lemma 16** (Round-Synchronization). *All honest validators enter any round $r > r_{\max}$ within Δ time of each other. In addition, they create their round-r blocks within Δ time of each other.*

Proof. Let t_1 be the time at which the first honest validator advances to round $r > r_{\max}$. By condition **B2**, this validator broadcasts its unknown history to all validators. Within Δ, all validators receive this history and have at least $n - f$ blocks (from distinct validators) in their DAGs for all rounds $r' < r$.

Any validator v in round r' creates a block according to **C3**. Consequently, there are at least $n - f + 1$ blocks from distinct validators in round r', which implies that **A1'** holds, while**A2** holds since v creates its round-r' block. Therefore, v is enabled to advance to the next round.

Assuming local computation time is negligible compared to message delay, by time $t_1 + \Delta$, every honest validator has caught up with all rounds $< r$ and enters round r. It follows that all honest validators enter round r within Δ time of each other.

Regarding synchronization of block creation in round r, assume the first honest validator v_2 creates a block at time t_2. Since v_2 is the first honest validator to create a block in round r, its block could not have been triggered by **C3**, as there could be at most f round-r blocks in its DAG (created by Byzantine validators).

If the block is created due to the timeout condition **C2**, then every other honest validator also creates its round-r block within $[t_2, t_2 + \Delta]$, since all honest validators enter round r within Δ time of each other and therefore their timeout expirations occur within the same interval.

If v_2's block is created by condition **C1**, then according to condition **B1**, v_2 broadcasts its history upon block creation. All other validators receive this history by time $t_2 + \Delta$ and can therefore also create their round-r blocks according to **C1** by that time.

Therefore, all validators create their round-r blocks within Δ of each other. ◀

▶ **Lemma 17** (Timely Delivery). *For any round $r > r_{\max}$, all round-$(r-1)$ blocks created by honest validators are integrated into the DAG of every honest validator before its round-r timeout expires.*

Proof. Let t_{GST} be the time of GST, and consider a round $r > r_{\max}$. Let t_{last} be the creation time of the last honest round-$(r-1)$ block. By condition **B1**, whenever a validator creates a block, it broadcasts both the block and its history. Therefore, every honest validator integrates this block into its DAG by time $t_{integrate} \leq \max(t_{GST}, t_{last}) + \Delta$.

Let t_v be the time at which the first honest validator enters round r. By definition of $r_{\max}$, we have $t_v \geq t_{GST}$. By Lemma 16, all honest validators enter round r within Δ time of each other. Hence, the last honest validator enters round r no later than $t_v + \Delta$.

Since an honest validator advances to the next round only after creating its block for the current round (condition **A2**), it follows that $t_{last} \leq t_v + \Delta$, and therefore $t_v \geq t_{last} - \Delta$.

Combining the inequalities $t_v \geq t_{GST}$ and $t_v \geq t_{last} - \Delta$ with a timeout of 2Δ yields $t_v + 2\Delta \geq \max(t_{GST}, t_{last}) + \Delta \geq t_{integrate}$.

Thus, every honest round-$(r-1)$ block is integrated into the DAG of every honest validator before the expiration of the round-r timeout. ◀

▶ **Lemma 18** (leader references). *If the leader of round $r-1$ is honest, where $r > r_{\max}$, then every round-r block created by an honest validator references (i.e., votes for) the round-$(r-1)$ leader block.*

Proof. If a round-r block created by an honest validator v_i was triggered by **C1**, then by definition it receives the leader block and references it.

If the block was created by condition **C2**, then by Lemma 17 the honest leader block is received before the timeout, and the block references it.

If the block is created by condition **C3**, then v_i receives $n - f$ round-r blocks, at least $n - 2f$ of which are from honest validators. Consider the set H consisting of the fastest $n - 2f$ honest validators to create their round-r blocks. Their block creation is not triggered by **C3**, as there are not enough round-r blocks to trigger it. Therefore, these blocks are created either by **C1** or **C2**, and by the arguments above, they reference the leader block of round $r - 1$.

Among the $n - 2f$ honest blocks received by v_i, at least one belongs to H, and thus references the leader block of round $r-1$. Therefore, v_i receives and references the round-$r-1$ leader block.

◀

▶ **Lemma 19** (quorum of votes). *If the leader of round $r - 2$ is honest, where $r > r_{\max} + 1$, then every round-r block created by an honest validator references a quorum ($2f + p$) of blocks voting for the round-$(r - 2)$ leader block.*

Proof. First, observe that if an honest validator creating a round-r block has received $2f + p$ round-$(r - 1)$ blocks voting for the leader block of round $r - 2$, then by the parent-selection mechanism these blocks are referenced as parents. Indeed, since the leader of round $r - 2$ is honest, all round-$(r - 1)$ blocks are leader-consistent with respect to that leader. Therefore, no such block is excluded by parent selection.

By Lemma 18, every round-$(r - 1)$ block created by an honest validator votes for the round-$(r - 2)$ leader block. The number of honest validators is $n - f = 2f + 2p - 1 \geq 2f + p$, for $p \geq 1$.

We now analyze the three block-creation conditions. If a round-r block created by an honest validator v_i is triggered by **C1**, then by condition **L2**, v_i has received $2f + p$ round-$(r - 1)$ blocks from distinct validators voting for the round-$(r - 2)$ leader block. The **L2** SP-skip condition cannot hold, since all blocks created by honest validators vote for the leader, so no quorum of non-voting blocks can be formed. Hence, the $2f + p$ voting blocks are selected and referenced as parents.

If the block is created by condition **C2**, then by Lemma 17, all round-$(r - 1)$ blocks created by honest validators are received by v_i before its round-r timeout expires. Hence, v_i receives at least $2f + p$ round-$(r - 1)$ blocks voting for the leader block before timeout expiration. Therefore, the created block references these blocks as parents.

Finally, suppose the block is created by condition **C3**. Then v_i received $n - f$ round-r blocks from distinct validators, at least $n - 2f$ of which are from honest validators.

Consider the set H consisting of the fastest $n - 2f$ honest validators to create their round-r blocks. Their block creation cannot be triggered by **C3**, since there are not yet enough round-r blocks to satisfy that condition. Hence, these blocks are created either by **C1** or **C2**, and by the arguments above, each references $2f + p$ blocks voting for the round-$(r-2)$ leader block.

Among the $n - 2f$ honest round-r blocks received by v_i, at least one block b belongs to H. Since b references $2f + p$ blocks voting for the round-$(r - 2)$ leader block, these voting blocks are received together with b. Therefore, v_i receives and references these blocks as parents as well. ◀

▶ **Lemma 20** (honest leaders committed). *Every leader block created in round $r > r_{\max}$ by an honest validator is committed (i.e., its slot is marked `to-commit`).*

Proof. By Lemma 18, every block created by an honest validator in round $r + 1$ votes for the honest leader block of round r. These $n - f$ voting blocks are eventually received by all honest validators.

If $p = f$, then these $n - f$ blocks from distinct validators are sufficient for a direct fast commit.

If $p < f$, then by Lemma 19, every round-$(r+2)$ block created by an honest validator references a quorum of $2f+p$ voters for the leader block, and thus constitutes an SP-certificate for it. Since $n - f \geq 2f + p$, every honest validator eventually receives at least $2f + p$ such SP-certificates. Consequently, each honest validator marks the slot as `to-commit`, and the leader block is *committed*. ◀

▶ **Theorem 21** (Fast Termination). *FinWhale satisfies fast termination.*

Proof. By Lemma 18, after GST every block created by an honest validator in round $r+1$ votes for the honest leader block of round r. If at least $n - p$ validators behave honestly (equivalently, at most p validators are Byzantine), then the leader's block eventually gathers $n - p$ honest voters, resulting in a direct fast commit after two rounds. ◀

▶ **Lemma 22.** *The round-robin leader schedule ensures that in any window of $3f + 3$ rounds, there are three consecutive rounds with honest leaders.*

Proof. Note that the protocol schedules leaders in a round-robin manner. Since there are at least $3f + 1$ validators, in any window of $3f + 3$ consecutive rounds at most f leaders can be Byzantine, leaving at least $2f + 3$ honest leaders. The f Byzantine leaders can divide the sequence of $3f + 3$ rounds into at most $f + 1$ segments of consecutive honest leaders. If none of these segments has length at least 3, then there are at most $2(f + 1) = 2f + 2$ honest leaders in total, which contradicts the fact that there are $2f + 3$. Hence, at least one segment must contain three or more consecutive honest leaders. ◀

▶ **Lemma 23** (All leader slots decided). *After GST any undecided slot eventually gets decided.*

Proof. Assume, for the sake of contradiction, that there exists an undecided slot in some round, and let r denote the highest round in which a leader slot s remains undecided.

By Lemma 22, there eventually exist three consecutive rounds $r', r'+1, r'+2$ with $r' > r$ whose leaders are honest. We denote their slots by s', s'_1, s'_2. By Lemma 20, the slots s', s'_1, and s'_2 are marked `to-commit`.

If $r \in \{r'-3, r'-2, r'-1\}$, then one of the blocks in s', s'_1, or s'_2 becomes the first committed slot above round r, and thus serves as the committed anchor for round r. This determines slot s.

If $r < r' - 3$, then by maximality of r, all slots in rounds greater than r are decided. In particular, there exists a committed block in some round $r + 3$ or higher that can serve as the anchor. This is guaranteed by the existence of s': since the block corresponding to s' is committed, if no earlier anchor exists, it can serve as the anchor. Hence, the committed anchor block will decide slot s.

In both cases, the leader of round r is decided, yielding a contradiction.

◀

▶ **Theorem 24** (Agreement). *FinWhale satisfies the agreement property of Byzantine Atomic Broadcast.*

Proof. Suppose an honest validator v_i ordered a block b and delivered its payload. This is the result of v_i applying a deterministic sort to the causal histories of a committed leader sequence $L_0, L_1, \ldots, L_n$.

By Lemma 13, the committed leader sequence is consistent across all honest validators. Moreover, by Lemma 23, all leader slots are eventually decided, and therefore $L_0, L_1, \ldots, L_n$ become part of v_j's commit sequence.

Since v_j applies the same deterministic algorithm to the same committed leader sequence, it will also order b and deliver its payload. ◀

▶ **Lemma 25.** *Every leader block created in round $r > r_{\max}$ by an honest validator is included in the commit sequence.*

Proof. By Lemma 20, any leader block b by an honest validator is committed. By Lemma 23 any undecided slot preceding the slot of b will be decided. Consequently, b will be part of the commit sequence. ◀

▶ **Theorem 26** (Validity)**.** *FinWhale satisfies the validity property of Byzantine Atomic Broadcast.*

Proof. If an honest validator v_k calls $\mathsf{a\_bcast}_k(m, id)$, then the tuple (m, id) is integrated into the payload of some block b created by v_k.

Since every block created by an honest validator becomes part of the causal history of some honest leader block, and by Lemma 25 such a leader block is included in the commit sequence, every honest validator eventually orders b and delivers (m, id, v_k).

◀

8 Related Work

8.1 DAG-based Protocols

DAG-based BFT protocols can be broadly classified into *certified* and *uncertified* DAG protocols, according to the mechanism used to disseminate blocks. In certified DAG protocols, validators disseminate blocks using a Byzantine Reliable Broadcast (BRB) primitive. BRB prevents equivocation and ensures block availability by guaranteeing that blocks are consistently delivered to all honest validators. Typically, each block must first be certified by a quorum of validators before being included in the DAG. However, BRB-based dissemination may incur an additional latency of 2–4 message delays. In contrast, uncertified DAG protocols aim to improve latency by using best-effort broadcast (BEB) instead of BRB, where the uniqueness of a block is not guaranteed. Thus, these protocols must explicitly handle equivocations.

We first consider certified DAG-based consensus protocols. Aleph is an asynchronous protocol that constructs a round-based DAG and uses an efficient binary agreement protocol for ordering. DAG-Rider [11] is an asynchronous DAG-based BFT protocol that progresses in waves consisting of four rounds, with a single leader in each wave. Tusk [6] refines DAG-Rider by replacing conventional BRB with quorum-based BRB, modifying the commit rules to improve latency in common-case executions, and enabling efficient garbage collection.

Bullshark [24] builds upon DAG-Rider to improve commit latency during synchronous periods. Its partially synchronous variant elects one leader every two rounds. Shoal [25] interleaves two instances of Bullshark, thereby introducing a leader in every round. Sailfish [22] and Shoal++ [2] similarly optimize the commit rules for leader blocks by using the first messages of the BRB to count votes for the leader. In addition, they support multiple leaders per round, further improving commit latency.

We next discuss uncertified DAG-based protocols. HashGraph [4] builds an unstructured DAG through validators disseminating blocks containing two references to previous blocks.

It uses virtual voting by interpreting the Hashgraph structure itself, together with the notion of "strongly seeing" (similar to certificates), to handle equivocations. A binary agreement protocol is then used to order transactions.

Cordial Miners [12] introduce both asynchronous and partially synchronous DAG-based protocols. In these constructions, a single leader is assigned in the first round of each wave, where waves consist of five rounds in the asynchronous setting and three rounds in the partially synchronous setting.

Mysticeti [3] is a partially synchronous DAG protocol built upon Cordial Miners. It pipelines waves and supports multiple leaders per round in order to improve commit latency. A followup work [17] introduced four DAG protocols: The first revisits Mysticeti with an improved push-based pacemaker that resolves its liveness issues. Starfish further decouples payload dissemination from metadata dissemination, improving communication complexity. Finally, Starfish-L and Mysticeti-L combine multi-signatures with a Lazy-Push pacemaker to further reduce communication overhead. In Finwhale, we build on and adapt the revised Mysticeti protocol introduced in [17], as well as their approach of using a single leader per round. BBCA-Chain [15] introduces a design in which leader blocks are broadcast using a BBCA primitive built on top of BRB, while non-leader blocks are disseminated using BEB, where leaders are assigned in every round.

8.2 Fast Consensus

Kursawe[13] was the first to implement a fast (two-step) Byzantine consensus protocol in which two-step fast path is paired with a subprotocol in the slow path. It is able to run with $3f + 1$ validators and the fast path is taken when all validators are honest and the network is synchronous. Otherwise, a fallback subprotocol is used to ensure liveness. An asynchronous binary Byzantine consensus protocol with fast-path termination is presented in [8]. The randomized protocol, assisted by a random oracle, tolerates $n > 5f$ validators and decides within one communication round in favorable executions.

Bosco [23] is a Byzantine consensus protocol that achieves a two-step fast path with $n > 5f$ validators when all validators propose the same value. The fast path can also be achieved with $n > 7f$ when all honest validators are in pre-agreement.

FaB Paxos [16] is a fast Byzantine consensus protocol requiring $n \geq 5f + 1$ validators. The paper also presents a parameterized variant with $n \geq 3f + 2p + 1$, which achieves a two-step fast path in the common case when the leader is correct, the network is synchronous, and at most p validators are Byzantine.

The lower bound of FaB Paxos was later improved to $n \geq 5f - 1$ by [1]. An independent work [14] proved a lower bound of $n \geq 3f + 2p - 1$ for the parameterized setting. The key idea is that equivocating leaders can be detected, allowing validators to wait for $n - f$ votes while excluding the vote of the Byzantine leader.

Banyan [26] is a rotating-leader protocol based on ICC [5]. It achieves two-step termination in the partially synchronous model with $n \geq 3f + 2p - 1$ validators. As long as at most p replicas are unresponsive, termination occurs within two message delays. The dual-mode mechanism enables two-step finalization latency in the fast path, and ensures that no penalties are incurred when the fast path is not taken.

Kudzu [21] is an atomic broadcast protocol with an integrated fast path. It achieves finality in just two rounds of communication for $n = 3f + 2p + 1$ validators, provided that at least $n - p$ replicas behave honestly. Building on DispersedSimplex [20], Kudzu introduces minimal modifications to incorporate a fast path. Furthermore, the protocol uses erasure

codes to ensure that the leader can disseminate large blocks with low and well-balanced communication complexity.

9 Conclusions

In this work, we presented FinWhale, the first DAG-based Byzantine Fault Tolerant protocol to integrate a fast-path mechanism that achieves termination in just two message delays when favorable conditions occur. By carefully combining fast-path commit patterns with the existing slow-path mechanisms of Mysticeti, FinWhale enables validators to commit transactions more quickly while maintaining safety and liveness under partial synchrony with $n = 3f + 2p - 1$ validators.

Our protocol tolerates up to f Byzantine faults and ensures fast termination whenever the number of faults does not exceed p, all while preserving the foundational guarantees and high-throughput benefits of Mysticeti. Beyond achieving the theoretical lower bound for latency in BFT consensus based protocols, FinWhale demonstrates that fast-path techniques traditionally applied to leader-based protocols can be effectively adapted to DAG structures. This helps bridge the gap between high-throughput DAG designs and optimal fast-path decision-making.

Starfish, Starfish-L, and Mysticeti-L [17] are protocols closely related to Mysticeti and designed to achieve improved communication complexity by combining multi-signatures, lazy dissemination of history, and efficient information dispersal techniques. Exploring how these techniques can be incorporated into FinWhale is an interesting direction for future work.

Acknowledgements.

We thank Alberto Sonnino and Sebastian Müller for insightful discussions that helped us better understand Mysticeti and Starfish.

References

1 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *PODC*, 2021.

2 Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput dag bft can be fast! *ArXiv*, abs/2405.20488, 2024. URL: `https://api.semanticscholar.org/CorpusID:270199502`.

3 Kushal Babel, Andrey Chursin, and Alberto Sonnino. Mysticeti: Reaching the limits of latency with uncertified dags. In *Network and Distributed System Security (NDSS) Symposium 2025*. Internet Society, 2025. arXiv:2310.14821.

4 Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Technical Report SWIRLDS-TR-2016-01, Swirlds, Inc., 2016.

5 Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet computer consensus. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, page 81–91, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3519270.3538430`.

6 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022.

7 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988. `doi:10.1145/42282.42283`.

8 Roy Friedman, Achour Mostefaoui, and Michel Raynal. Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems . In *Reliable Distributed Systems, IEEE Symposium on*, pages 228–237, Los Alamitos, CA, USA, October 2004. IEEE Computer Society. URL: `https://doi.ieeecomputersociety.org/10.1109/RELDIS.2004.1353024`, `doi:10.1109/RELDIS.2004.1353024`.

9 Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, AFT '19, page 214–228, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3318041.3355467`.

10 Philipp Jovanovic, Lefteris Kokoris-Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. Mahi-mahi: Low-latency asynchronous bft dag-based consensus. In *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*, pages 549–559, 2025. `doi:10.1109/ICDCS63083.2025.00060`.

11 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021.

12 Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 26:1–26:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

13 K. Kursawe. Optimistic byzantine agreement. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 262–267, 2002. `doi:10.1109/RELDIS.2002.1180196`.

14 Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. Revisiting optimal resilience of fast byzantine consensus. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 343–353, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3465084.3467924`.

15 Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. Bbca-chain: One-message, low latency bft consensus on a dag. *arXiv preprint arXiv:2310.06335*, 2023.

16 Jean-Philippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3):202–215, July 2006. `doi:10.1109/TDSC.2006.35`.

17 Nikita Polyanskii, Sebastian Mueller, and Ilya Vorobyev. Making uncertified DAG BFT provably live with linear payload and quadratic metadata communication. Cryptology ePrint Archive, Paper 2025/567, 2025. URL: `https://eprint.iacr.org/2025/567`.

18 Longfei Qiu, Jingqi Xiao, and Zhong Shao. Mechanized Safety and Liveness Proofs for the Mysticeti Consensus Protocol under the LiDO-DAG Framework . In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1522–1541, Los Alamitos, CA, USA, May 2026. IEEE Computer Society. URL: `https://doi.ieeecomputersociety.org/10.1109/SP63933.2026.00009`, `doi:10.1109/SP63933.2026.00009`.

19 Ehud Shapiro. Brief Announcement: Grassroots Distributed Systems: Concept, Examples, Implementation and Applications. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing (DISC 2023)*, volume 281 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:7, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2023.47`, `doi:10.4230/LIPIcs.DISC.2023.47`.

20 Victor Shoup. Sing a Song of Simplex. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2024.37`, `doi:10.4230/LIPIcs.DISC.2024.37`.

21 Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. Kudzu: Fast and Simple High-Throughput BFT. In Dariusz R. Kowalski, editor, *39th International Symposium on Distributed*

Computing (DISC 2025), volume 356 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2025.42, doi:10.4230/LIPIcs.DISC.2025.42.

22 Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving latency of dag-based bft. *Cryptology ePrint Archive*, 2024.

23 Yee Jiun Song and Robbert Renesse. Bosco: One-step byzantine asynchronous consensus. In *Proceedings of the 22nd International Symposium on Distributed Computing*, DISC '08, page 438–450, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-87779-0_30.

24 Alexander Spiegelman, Neil Giridharan andF Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022.

25 Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. *CoRR*, abs/2306.03058, 2023.

26 Yann Vonlanthen, Jakub Sliwinski, Massimo Albarello, and Roger Wattenhofer. Banyan: Fast rotating leader bft. In *Proceedings of the 25th International Middleware Conference*, Middleware '24, page 494–507, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3652892.3700788.

Algorithm 1 DAG Construction for validator v_i, Part 1

```
1: Global variables:
2: buffer ← ∅
3: blocksToPropose ← ∅                                  ▷ Valid blocks of transactions from clients
4: δ_LT ← 2Δ
5: n ← 3f + 2p − 1
6: r_decided ← 0                                        ▷ The most recent round in the sequence of decided leaders
7: r_highest ← 0                                        ▷ The highest block round observed in the local DAG
8: DAG ← GenesisBlocks( )
9: ∀j : Known[j] ← GenesisBlocks( )

10: upon a_bcast_i(m, id)
11: blocksToPropose.enqueue(m, id)

12: procedure Execute
13:     for r = 1, 2, ... do
14:         ExecuteRound(r)

15: procedure ExecuteRound(r)
16:     BroadcastUnknownHistory( )
17:     τ_enter(r) ← LocalTime( )
18:     while TryCreateBlock(r, τ_enter(r)) = false do
19:         ReceiveMessages( )
20:         TryDecide(r_decided, r_highest)
21:     BroadcastUnknownHistory( )
22:     while TryAdvanceRound(r) = false do
23:         ReceiveMessages
24:         TryDecide(r_decided, r_highest)

25: procedure BroadcastUnknownHistory( )
26:     DAG_snap ← DAG
27:     for j ∈ {1, ..., n} \ {i} do
28:         Unknown ← DAG_snap \ Known[j]
29:         msgs ← Unknown
30:         Send(v_j, msgs)
31:         Known[j] ← Known[j] ∪ {msgs}

32: procedure ReceiveMessages( )
33:     b ← ReceiveNetworkMessage( )
34:     TryAddToDAG(b)

35: function TryAdvanceRound(r)
36:     if ValidatorsInRound(r) < n − f then
37:         return false
38:     if ValidatorsInRound(r) > n − f then
39:         return true
40:     consistentSet ← TryGetLeaderConsistentSet(r)
41:     if consistentSet ≠ ⊥ then
42:         return true
43:     leader_{r−1} ← GetPredefinedLeader(r − 1)
44:     if ∀b ∈ DAG[r], b.author ≠ leader_{r−1} then
45:         return true
46:     return false

47: function TryCreateBlock(r, τ_enter(r))
48:     if LeaderConditionsMet(r) or LocalTime( ) ≥ τ_enter(r) + δ_LT or ValidatorsInRound(r)
    ≥ n − f then
49:         CreateBlock(r)
50:         return true
51:     return false

52: function LeaderConditionsMet(r)
53:     leader_{r−1} ← GetPredefinedLeader(r − 1)
54:     Leaders ← {b ∈ DAG[r − 1] : b.author = leader_{r−1}}
55:     L1 ← |Leaders| > 0
56:     leader_{r−2} ← GetPredefinedLeader(r − 2)
57:     Proposals ← {b ∈ DAG[r − 2] | b.author = leader_{r−2}}
58:     Votes ← DAG[r − 1]
59:     VoteQuorum ← ∃b_leader ∈ Proposals such that
60:         |{b.author | b ∈ Votes ∧ IsVote(b, b_leader)}| ≥ 2f + p
61:     SkipPattern ← IsSPSkipPattern(r − 2)
62:     L2 ← (VoteQuorum ∨ SkipPattern)
63:     return L1 ∧ L2
```

Algorithm 2 DAG construction for validator v_i, part 2

```
1: function TryGetLeaderConsistentSet(r)
2:     leader_{r-1} ← GetPredefinedLeader(r − 1)
3:     Proposals ← { b ∈ DAG[r − 1] | b.author = leader_{r-1} }
4:     for b_leader ∈ Proposals do
5:         ConflictingBlocks ← GetConflictingBlocks(b_leader)
6:         ConflictingVotes ← {b ∈ DAG[r] | ∃b′ ∈ ConflictingBlocks : IsVote(b, b′)}
7:         NonConflictingVotes ← DAG[r] \ ConflictingVotes
8:         numValidators ← |{ b.author | b ∈ NonConflictingVotes }|
9:         if numValidators = ValidatorsInRound(r) then
10:            return NonConflictingVotes
11:    if |Proposals| = 0 then
12:        return (DAG[r])
13:    return (⊥)

14: function SelectParents(r)
15:    CandidateParents ← DAG[r − 1]
16:    leader_{r-2} ← GetPredefinedLeader(r − 2)
17:    if ∃b ∈ DAG[r − 1] s.t. b.author = leader_{r-2} and ValidatorsInRound(r − 1) = n − f then
18:        ConsistentSet ← TryGetLeaderConsistentSet(r − 1)        ▷ Such a set exists by A1'.
19:        CandidateParents ← ConsistentSet
20:    Parents ⊆ CandidateParents such that:
21:        (1) for each validator v, Parents contains exactly one block authored by v, and
22:        (2) if C1 triggered block creation, all blocks in CandidateParents satisfying L1&L2 are
       included in Parents
23:    Proposals_{r-2} ← {u ∈ DAG[r − 2] | u.author = leader_{r-2}}
24:    ProposalsWithVotes ← { u ∈ Proposals_{r-2} | ∃u′ ∈ Parents : IsVote(u′, u) }
25:    if |ProposalsWithVotes| > 1 then                ▷ Parent set is not leader-consistent
26:        ExcludedBlocks ← { b ∈ DAG[r − 1] | b.author = leader_{r-2} }
27:        Parents ← Parents \ ExcludedBlocks
28:    return Parents

29: procedure CreateBlock(r)
30:    b.payload ← blocksToPropose.dequeueAll() if non-empty, else null
31:    b.round ← r; b.author ← v_i
32:    P ← SelectParents(r)
33:    U ← { u | u ∈ DAG[r′], 1 ≤ r′ ≤ r − 2 }
34:        filtered by: u is the latest block from its validator
35:        and u is not referenced by any previous block of v_i
36:    P ← P ∪ U
37:    b.parents ← {HASH(u) | u ∈ P}
38:    SignBlock(b)
39:    TryAddToDag(b);

40: function IsValidBlock(b)
41:    VerifySignature(b)
42:    P ← {u ∈ DAG | HASH(u) ∈ b.parents}
43:    P_{r-1} ← {u ∈ P | u.round = b.round − 1}
44:    if |P_{r-1}| < n − f or ∃u ≠ v ∈ P | u.author = v.author then
45:        return false
46:    leader_{r-2} ← GetPredefinedLeader(r − 2)
47:    if ¬ExposeEquivocation(b, leader_{r-2}) then
48:        return true
49:    if ¬∃u ∈ P_{r-1} | u.author = leader_{r-2} then
50:        return true
51:    return false

52: function CausalAvailable(b)
53:    return ∀h ∈ b.parents, ∃b′ ∈ DAG s.t. HASH(b′) = h

54: function GetReachableBlocks(b)
55:    return { b′ ∈ DAG | IsPath(b′, b) = true }

56: procedure TryAddToDAG(b)
57:    buffer ← buffer ∪ {b}
58:    while ∃b′ ∈ buffer s.t. CausalAvailable(b′) = true do
59:        choose any b′ ∈ buffer s.t. CausalAvailable(b′) = true
60:        buffer ← buffer \ {b′}
61:        if IsValidBlock(b′) then
62:            DAG ← DAG ∪ {b′}
63:            r_highest ← max(r_highest, b′.round)
64:            Known[b′.author] ← Known[b′.author] ∪ GetReachableBlocks(b′)
```

Algorithm 3 Helper functions

```
1: function GetPredefinedLeader(r)
2:     return (r mod n) + 1

3: function ValidatorsInRound(r)
4:     return |{b.author | b ∈ DAG[r]}|

5: function IsPath(b_old, b_new)
6:     return ∃ b_1, ..., b_k ∈ DAG, k ∈ ℕ, s.t.
7:     b_1 = b_old, b_k = b_new and
8:         ∀j ∈ {2, ..., k} : HASH(b_{j-1}) ∈ b_j.parents

9: function IsVote(b_vote, b_leader)
10:    return Hash(b_leader) ∈ b_vote.parents

11: function IsSPCert(b_cert, b_leader)
12:    Parents ← {b ∈ DAG | HASH(b) ∈ b_cert.parents}
13:    res ← |{b ∈ Parents | IsVote(b, b_leader)}|
14:    return res ≥ 2f + p

15: function IsSPSkipPattern(r)
16:    leader ← GetPredefinedLeader(r)
17:    Votes ← DAG[r + 1]
18:    Proposals ← {b ∈ DAG[r] | b.author = leader}
19:    for b_leader ∈ Proposals do
20:        res ← |{b.author | b ∈ Votes, IsVote(b, b_leader) = false}|
21:        if res < 2f + p then
22:            return false
23:    if |Proposals| = 0 and ValidatorsInRound(r + 1) < 2f + p then
24:        return false
25:    return true

26: function IsDirectlySkippedSlot(r)
27:    B_{r+2} ← DAG[r + 2]
28:    NonFPEvidenceBlocks ← {b ∈ B_{r+2} | IsNonFPEvidence(b)}
29:    res_nonFP ← |{b.author | b ∈ NonFPEvidenceBlocks}|
30:    FP_cond ← res_nonFP ≥ 2f + p
31:    SP_cond ← IsSPSkipPattern(r)
32:    return SP_cond and FP_cond

33: function DirectlyCommittedLeaderBlock(r)
34:    leader ← GetPredefinedLeader(r)
35:    Proposals ← {b ∈ DAG[r] | b.author = leader}
36:    B_{r+1} ← DAG[r + 1]
37:    B_{r+2} ← DAG[r + 2]
38:    for b_leader ∈ Proposals do
39:        res_SP ← |{b.author | b ∈ B_{r+2}, IsSPCert(b, b_leader) = true}|
40:        res_FP ← |{b.author | b ∈ B_{r+1}, IsVote(b, b_leader) = true}|
41:        if res_SP ≥ 2f + p or res_FP ≥ n − p then
42:            return b_leader
43:    return Undecided(r)

44: function IsSPCertifiedPath(b_anchor, b_leader)
45:    r ← b_leader.round
46:    B_{r+2} ← DAG[r + 2]
47:    return ∃b ∈ B_{r+2} : IsSPCert(b, b_leader) ∧ IsPath(b, b_anchor)
```

Algorithm 4 Helper functions Fast Path

```
1: function GetConflictingBlocks(b_leader)
2:     B ← {b ∈ DAG[b_leader.round] : b.author = b_leader.author  ∧ b ≠ b_leader}
3:     return B

4: function ExposeEquivocation(b)
5:     leader_{r−2} ← GetPredefinedLeader(b.round − 2)
6:     Proposals_{r−2} ← {u ∈ DAG[r − 2] | u.author = leader_{r−2}}
7:     Parents ← {u ∈ DAG | HASH(u) ∈ b.parents}
8:     ProposalsWithVotes ← {u ∈ Proposals_{r−2} | ∃u′ ∈ Parents : IsVote(u′, u) }
9:     return |ProposalsWithVotes| > 1

10: function IsFPEvidence(b_evid, b_leader)
11:     Parents ← {b ∈ DAG | HASH(b) ∈ b_evid.parents}
12:     res_vote ← |{b ∈ Parents : IsVote(b, b_leader)}|
13:     if ¬ExposeEquivocation(b_evid) then
14:         return res_vote ≥ f + p − 1
15:     else
16:         B_conflicts ← GetConflictingBlocks(b_leader)
17:         res_conflict ← |{b ∈ Parents | ∃b′ ∈ B_conflicts, IsVote(b, b′)}|
18:         return (res_vote ≥ f + p) ∧ (res_conflict < f + p)

19: function IsNonFPEvidence(b_evid)
20:     leader_{r−2} ← GetPredefinedLeader(b_evid.round − 2)
21:     Proposals_{r−2} ← {u ∈ DAG[r − 2] | u.author = leader_{r−2}}
22:     for b_leader ∈ Proposals do
23:         if IsFPEvidence(b_evid, b_leader) then
24:             return false
25:     return true

26: function PathsToFPEvidenceQuorum(b_anchor, b_leader)
27:     r ← b_leader.round
28:     B_{r+2} ← DAG[r + 2]
29:     res ← |{b.author | b ∈ B_{r+2} ∧ IsFPEvidence(b, b_leader) ∧ IsPath(b, b_anchor)}|
30:     return res ≥ 2f + p
```

Algorithm 5 Committing and ordering

```
1: Global variables:
2: Sequenced ← ∅                                   ▷ delivered (m, id, author) tuples

3: procedure TryDecide(r_decided, r_highest)
4:     sequence ← [ ]
5:     for r ∈ [r_highest down to r_decided + 1] do
6:         status ← TryDirectDecide(r)
7:         if status = Undecided(r) then
8:             status ← TryIndirectDecide(r, sequence)
9:         sequence ← status||sequence
10:    for status ∈ Sequence in ascending round order do
11:        if status = Undecided(r) then
12:            break
13:        if status = Commit(b_leader) then
14:            OnCommitLeader(b_leader)
15:        r_decided ← r_decided + 1

16: procedure TryDirectDecide(r)
17:    if IsDirectlySkippedSlot(r) then
18:        return Skip(r)
19:    b_leader ← DirectlyCommittedLeaderBlock(r)
20:    if b_leader ≠ Undecided(r) then
21:        return Commit(b_leader)
22:    return Undecided(r)

23: procedure TryIndirectDecide(r, sequence)
24:    Anchors ← [s ∈ sequence s.t. s.round > r + 2]
25:    for a ∈ Anchors in ascending round order do
26:        if a = Undecided(r) then
27:            return Undecided(r)
28:        if a = Commit(b_anchor) then
29:            leader ← GetPredefinedLeader(r)
30:            Proposals ← {b ∈ DAG[r] | b.author = leader}
31:            SortedProposals ← Sort(Proposals)
32:            for b_leader ∈ SortedProposals do
33:                SP_cond ← IsSPCertifiedPath(b_anchor, b_leader)
34:                FP_cond ← PathsToFPEvidenceQuorum(b_anchor, b_leader)
35:                if SP_cond or FP_cond then
36:                    return Commit(b_leader)
37:            return Skip(r)
38:    return Undecided(r)

39: procedure OnCommitLeader(b_leader)
40:    Reachable ← Sort(GetReachableBlocks(b_leader))
41:    for b ∈ Reachable do
42:        AppendPayload(b)

43: procedure AppendPayload(b)
44:    if b.payload ≠ null then
45:        for (m, id) ∈ b.payload do
46:            if (m, id, b.author) ∉ Sequenced then
47:                a_deliver_i(m, id, b.author)
48:                Sequenced ← Sequenced ∪ {(m, id, b.author)}
```