DAG It Off: Latency Prefers No Common Coins

Ignacio Amores-Sesar
Aarhus University, Denmark

Viktor Grøndal
Aarhus University, Denmark

Adam Holmgård
Aarhus University, Denmark

Mads Ottendal
Aarhus University, Denmark

Abstract

We introduce Black Marlin, the first Directed Acyclic Graph (DAG)-based Byzantine atomic broadcast protocol in a partially synchronous setting that successfully forgoes the reliable broadcast and common coin primitives. Black Marlin achieves the optimal latency of 3 rounds of communication (4.25 with Byzantine faults) while maintaining optimal communication and amortized communication complexities. We present a formal security analysis of the protocol, accompanied by empirical evidence that Black Marlin outperforms state-of-the-art DAG-based protocols in both throughput and latency.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Atomic broadcast, DAG-based, Partial synchrony

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.5

Related Version *arXiv version*: `https://arxiv.org/abs/2508.14716`

Funding This work has been mostly funded by the Cryptographic Foundations for Digital Society, CryptoDigi, DFF Research Project 2, Grant ID 10.46540/3103-00077B, and partially funded by the Swiss National Science Foundation(SNSF) under grant agreement Nr. 188443 (Advanced Consensus Protocols) and by a grant from Avalanche, Inc. to the University of Bern.

1 Introduction

Scalability poses a significant challenge for consensus protocols. Paradoxically, increasing the number of participants in consensus degrades its performance rather than enhancing it.

Traditional consensus protocols, such as Paxos [19], PBFT [10], and HotStuff [27], primarily adopt a leader-based approach. In this model, a designated leader proposes a value, which the remaining participants validate. This design results in an uneven distribution of workload, with the leader becoming a performance bottleneck. If we add more parties to a consensus protocol, we simply increase the communication load of the leader, degrading the performance of the protocol. Furthermore, if the leader behaves in a Byzantine manner, the system must invoke a view-change mechanism to replace it. These view-changes are sensitive to timing assumptions and become progressively more expensive [8] as the number of parties increases.

To address these limitations, Keidar et al. [16] introduced DAG-Rider, an elegant asynchronous solution that leverages the common core abstraction [9]. This approach enables every participant to act as a leader in the protocol. These instances are interwoven into a directed acyclic graph (DAG), on top of which a total ordering is achieved using a common coin [8], applied at the end of every *wave*: a set of four consecutive rounds.

DAG-Rider demonstrates remarkable throughput compared to leader-based protocols, but this comes at the cost of significantly increased latency. The primary cause of this latency is the repeated use of reliable broadcast [8] across each of the four rounds required to implement the common core primitive [9]. This triggered the effort of the research community [12, 26, 17, 15, 24, 6, 25, 5] aimed at mitigating this latency overhead. These contributions generally fall into two broad categories.

On the one hand, Cordial Miners [17] replaces the reliable broadcast rounds with bare-message communications, reducing communication overhead at the cost of increasing the wave length to five rounds (three in partial synchrony). On the other hand, a majority of the works [12, 26, 15, 24, 6, 25, 5] pursue orthogonal improvements, either through more efficient use of the common coin or by shortening the length of the waves under partial synchronous assumptions. These approaches have collectively succeeded in halving the latency of DAG-based protocols, marking a significant advance in the state of the art. However, their reliance on the reliable broadcast primitive continues to be a limiting factor. Moreover, to the best of our knowledge, no existing protocol has yet successfully elected an anchor in every round.

These are precisely the two limitations addressed in this paper. We introduce Black Marlin, the first partially synchronous, DAG-based protocol that elects an anchor in every round without relying on reliable broadcast. The simplest – yet most impactful – innovation is the substitution of the common coin with a deterministic round-robin mechanism. This substitution is valid because the FLP impossibility [14] only applies in asynchronous settings.

The primary advantage of the round-robin approach is that it allows block creation to be conditioned on the reception of the anchor block from a given round. To handle scenarios in which a malicious anchor fails to behave correctly, Black Marlin employs timeouts, thereby preventing deadlocks. This conditional block creation enables Black Marlin to elect an anchor in every round, fulfilling the long-standing goal of the research on DAG-based protocols.

Moreover, by operating in a partially synchronous model, Black Marlin is able to adapt ideas from Cordial Miners [17] to eliminate the need for reliable broadcast, using only digital signatures without the need for extra rounds of communication. The trade-off for achieving anchor election without reliable broadcast is a small constant delay and a minor reduction in commitment probability under Byzantine behavior, while maintaining optimal message complexity in both cases.

Specifically, Black Marlin commits the anchor of round $r - 2$ upon completing round r with probability $4/9$ under adversarial conditions. This probability increases to 1 when all participants behave honestly. As a result, the expected latency is 4.25 communication rounds under Byzantine behavior and the optimal 3 rounds in the honest case.

Additionally, timeouts are only triggered during asynchronous periods or when facing Byzantine faults. Thus, Black Marlin remains a reactive protocol, effectively leveraging the advantages of both synchronous and asynchronous models. We prove that Black Marlin implements atomic broadcast [8] together with all the performance results mentioned above. Furthermore, we benchmark Black Marlin against Bullshark, showing its practicality.

1.1 Structure

The paper is structured as follows. Section 1 briefly describes the paper. Section 2 details the state of the art of DAG-based protocols. Section 3 sets the notation and key concepts used in the paper. Section 4 describes Black Marlin in full detail. Section 5 shows that Black Marlin implements atomic broadcast and studies its complexities. Section 6 describes a comparison of the throughput and latency of Black Marlin and Bullshark.

2 Related work

DAG-based protocols have constituted a significant transformation in the field of consensus over the past few years.

This revolution began in the permissionless setting, with protocols such as Avalanche [22] and IOTA [20]. However, these protocols were initially introduced without thorough security analyses. As a result, several vulnerabilities were later discovered [11, 4, 3], which limited their success despite recent work demonstrating their optimality [2].

DAG-based consensus protocols were later extended to the permissioned setting, most notably with DAG-Rider by Keidar et al. [16]. DAG-Rider was the first protocol to leverage the properties of the common core primitive [9] to implement atomic broadcast in asynchronous networks. It outperforms traditional leader-based protocols [19, 10, 27] in terms of throughput. However, DAG-Rider requires on average six instances of reliable broadcast [7], resulting in 18 rounds of communication. Since its introduction, several works have aimed to address the latency limitations of DAG-Rider.

On one hand, Cordial Miners [17] reduces latency to an average of 7.5 communication rounds by replacing reliable broadcast with bare-message communication. The tradeoff of this approach is that its new communication paradigm requires an additional round per wave to compensate for the properties that reliable broadcast provides. When a party creates a block in Cordial Miners, it must include any blocks the recipient may have missed. We adopt this idea in the design of Black Marlin.

On the other hand, a widely adopted strategy to improve DAG-Rider's latency has involved either assuming synchronous networks or modifying the core and coin primitives. Narwhal and Tusk [12] introduced a clever transaction preprocessing mechanism, where transactions are grouped into batches, so only the hashes of these batches are submitted to consensus. This batching is delegated to specialized *workers*, allowing the system to scale dynamically with transaction load. However, Narwhal and Tusk lack a rigorous security analysis, and liveness vulnerabilities have been identified [23].

Bullshark [26], which builds on top of Narwhal and Tusk, also introduces a variant for partial synchrony that improves latency to just three instances of reliable broadcast, about nine rounds of communication, on average. Bullshark is particularly relevant to our work, as its authors provide a public implementation [21][1], enabling a fair, side-by-side comparison with Black Marlin. It is worth noting that both Narwhal and Tusk, as well as Bullshark, rely on reliable broadcast as a fundamental building block.

More recently, several protocols have been proposed targeting latency. These works largely follow the above trajectory. Mysticeti [6] reduces latency further by employing consistent broadcast, achieving an average-case latency of only three instances [8]. Sailfish [24] also improves latency, though it continues to rely on reliable broadcast. Similarly, Shoal and Shoal++ [25] provide enhancements in anchor selection, but still depend on reliable broadcast.

To the best of our knowledge, Black Marlin is the first DAG-based protocol to achieve optimal latency in partial synchrony, while electing an anchor in every round, a crucial feature for minimizing the latency of non-anchor blocks without relying on reliable broadcast. We also highlight that Black Marlin achieves these properties while preserving simplicity.

[1] Bullshark has been chosen over more recent frameworks such as Mysticeti [18] due to its closer similarity to Black Marlin and the timing of this project.

3 Model

3.1 Notation

We consider a set $\mathcal{N} = \{0, ..., n-1\}$ of n parties running a round-based algorithm. In each round, these parties create blocks that form the vertices of a directed acyclic graph $(DAG, \mathcal{R})$ with references that constitute the edges. Edge $(B_1, B_2) \in \mathcal{R}$ if B_1 includes a reference to B_2. A block $B = [r, i, \mathit{refs}, \mathit{refs}']$ is a tuple formed by a round number r, an identifier for party i, a set of strong references *refs*, and a set of weak references refs'. For simplicity, we omit the set of transactions and the signature from the block, as these transactions play no role in the consensus mechanism. Given a block B, $\mathit{creator}(B)$ denotes the party that created B, $\mathit{round}(B)$ denotes the round in which B was created. The variable $DAG(r)$ denotes the set of block $B' \in DAG$ such that $\mathit{round}(B') = r$. A party selected by a round-robin mechanism is referred to as *anchor*, blocks produced by the anchor party are referred to as *anchor* blocks.

3.2 Adversary and Network

We model parties as *interactive Turing machines* (*ITM*). An interactive Turing machine is a Turing machine with an input and an output tape that allows the Turing machine to communicate with other Turing machines and make decisions based on the content of their input tape. The adversary is modeled as another ITM that corrupts up to f parties at the beginning of the execution, the actual number of corruptions is denoted by $t \leq f < \frac{n}{3}$. These corrupted parties obey the adversary; i.e., they may diverge from the execution of the protocol. Corrupted parties are often referred to as *Byzantine* and non-corrupted as *honest*.

We assume a partial synchronous communication model [13] on point-to-point links. The network begins in an asynchronous state and turns synchronous after a global stabilization time (GST). After this point, every message sent is delivered with a delay of up to Δ. The adversary not only decides the global stabilization time but also when the message is made available to the party within the Δ time limit. We also assume that the local computations are instantaneous, this assumption can easily be relaxed by factoring the computation of the honest parties in the network delay. We assume the existence of a public key infrastructure. We also assume that every block and a message produced by every party is signed, and that the signature scheme is secure.

3.3 Atomic broadcast

We model our protocol as *atomic broadcast*. Our atomic broadcast primitive is accessed with the events $\mathit{ab\text{-}broadcast}(B, i, r)$ and outputs events $\mathit{ab\text{-}deliver}(B, j, r)$. The event $\mathit{ab\text{-}broadcast}(B, i, r)$ can only be triggered by party i, but $\mathit{ab\text{-}deliver}(B, j, r)$ may be output by every party.

▶ **Definition 1** (Atomic broadcast)**.** *A protocol solves* atomic broadcast *if it satisfies the following properties, except with negligible probability:*

Validity. *If an honest party i invokes* ab-broadcast(B, i, r) *after the global stabilization time, then party i eventually outputs* ab-deliver(B, i, r).

Agreement. *If an honest party outputs* ab-deliver(B, i, r)*, then all honest parties eventually output* ab-deliver(B, i, r).

Integrity. *For any party i and round r, every honest party outputs* ab-deliver(B, i, r) *at most once regardless of B. Moreover, if i is honest, then i has invoked* ab-broadcast(B, i, r).

Total order. *If an honest party outputs* ab-deliver(B, i, r) *before* ab-deliver(B', j, r')*, then no honest party outputs* ab-deliver(B', j, r') *before* ab-deliver(B, i, r).

Note that our definition of atomic broadcast is adapted to the partial synchrony model; the validity property applies to blocks that are broadcast after the network becomes synchronous. However, the safety properties are satisfied during both synchrony and asynchrony.

4 Protocol

Black Marlin (Algorithm 2) proceeds in rounds. In round r, honest parties wait to receive blocks from at least $n!-!f$ parties. They then check if an anchor for round r has been received and the anchors for rounds $r-2$ and $r-1$ appear in the past of at least $n-f$ blocks from rounds $r-1$ and r, respectively. If both hold, a new block is created referencing all received blocks, and the party advances to round $r+1$. Otherwise, the party waits until the conditions are met or a timeout triggers block creation. These timeouts ensure liveness under Byzantine anchors and are only used when progress is blocked, allowing the protocol to be responsive. Figure 1 illustrates an example execution.

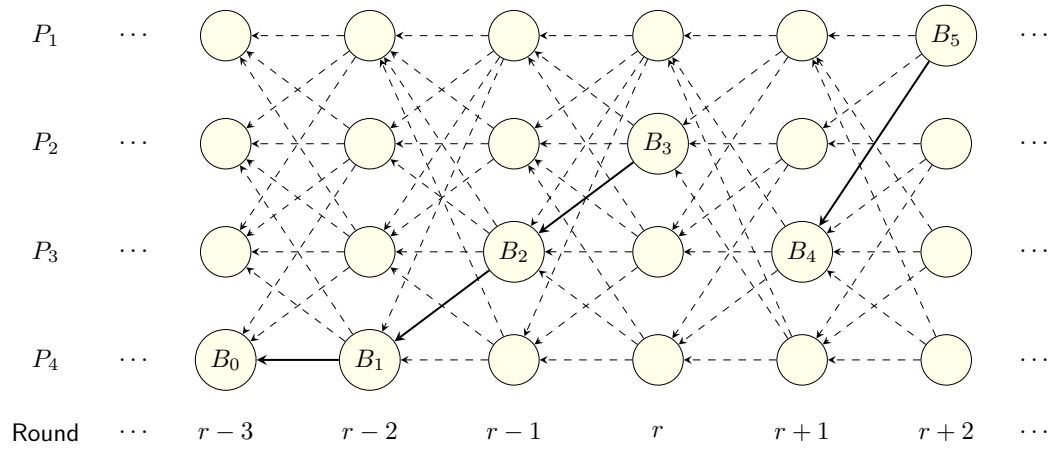


Figure 1 An example of execution of Black Marlin (Algorithm 2) with $f = 1$ and $n = 4$. Every block has references to at least 3 blocks from the previous round. The blocks $B_0, ..., B_5$ are the anchor blocks for the corresponding round. Note that blocks B_0, B_1, B_2, and B_3 have enough support (i.e., at least 3 references) to be committed. However, $B_3 \notin \mathit{strong}(B_4)$, thus B_0, B_1, and B_2, satisfy the commitment rule (L18–32) while B_3 does not. After the global stabilization time, this situation cannot occur without malicious behavior, as B_3 would satisfy $B_3 \in \mathit{past}(B_4)$. This situation has been constructed to illustrate the commitment rule.

4.1 Validity predicates

A block from round $r \neq 1$ and creator j is valid if it is signed by party j and contains references to valid blocks created by at least $n-f$ parties from round $r-1$. Since blocks in round $r = 1$ do not reference any previous blocks, a block is valid if it has been signed by its creator. A history $\mathcal{H}$ is valid if all the blocks contained in it are also valid.

When receiving a message $[\text{BLOCK}, B', \mathcal{H}', j, r']$ (L57), party i checks the validity of both the block B' and the history $\mathcal{H}'$, and adds the block B' along with every unknown block from the history $\mathcal{H}'$ to its local view (DAG). Party i also updates party j's local history (L60), used solely to optimize message complexity by sending only strictly necessary blocks.

4.2 Initialization

The first round of the protocol differs from the rest as there are no blocks from earlier rounds. Party i creates a block with an empty set of references and invokes the event *ab-broadcast*(B, i) before sending the message $[\text{BLOCK}, B, \{\}, i, 0]$ to every party and starting a timeout with time 2Δ. Party i then moves to the next round, a standard round.

4.3 Round structure

Every round beyond the first shares the same structure. Consider an honest party i in round r, i.e., it has just produced a block corresponding to round r. Party i waits to receive messages $[\text{BLOCK}, B, \mathcal{H}, j, r']$ from at least $n - f$ different parties for some round $r' \geq r$. Once sufficient messages for such a round have been received, party i checks whether it has received a message $[\text{BLOCK}, B', \mathcal{H}', j, r']$ with j being the anchor for the current round, and verifies that the previous two anchors have enough support, or if a timeout has been triggered, to conclude the round. This dual condition to start a new round allows the protocol to be responsive, i.e., it runs at the speed of the actual network delay instead of the estimated network delay Δ without deadlocking when an anchor party behaves maliciously.

Party i concludes the round by performing the following sequential operations. First, i attempts the delivery of blocks (L42); this function is described in its own paragraph below. Secondly, party i creates a new block B corresponding to the next round $r' + 1$, including strong references *refs* to every block in $DAG(r')$ (L43–50). A set of weak references *refs*$'$ to blocks from previous rounds are also included; this secondary set is time-bounded, preventing the usual garbage collection problem associated with these solutions. Party i concludes these operations by sending B to every other party, together with the history of the parties (L54). Note that party i may skip one or multiple rounds if an appropriate quorum is received.

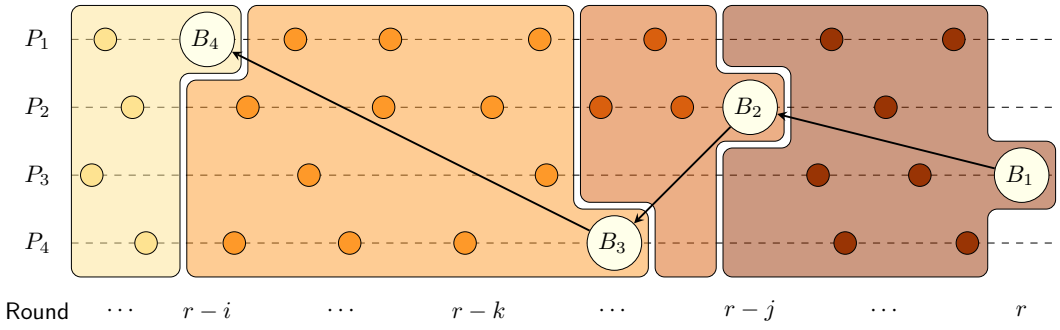


Figure 2 Illustration of the commit rule of Black Marlin. When a party invokes the commit function on an anchor block B_1, it will recursively commit earlier uncommitted anchor blocks first. The subscript numbers on the blocks indicate the order in which the commit function is called. In this process, blocks B_1 B_2, B_3, and B_4 are *ab-delivered*, in the reverse order of their commitment. Before committing any anchor block, all non-delivered blocks in their causal history are deterministically sorted and committed. In the illustration, these correspond to the smaller vertices contained within the history boxes associated with each anchor block.

4.4 Delivery

The delivery function determines which blocks are delivered and in what order, based on the local view of the DAG. Upon concluding round r, party i attempts the commitment of the anchor block B from round $r - 2$ (L14–17). B is an anchor block if its creator has been

elected by the round-robin mechanism. However, it is only committed if it is supported by at least $n - f$ parties in round $r - 1$ and including an anchor of round $r - 1$ B', and B' is also supported by at least $n - f$ parties in round r. These conditions prevent honest parties from committing different blocks when the anchor party is Byzantine.

If this check succeeds, party i calls the function $commit(B)$ (L18–32), which iterates through the local view of the DAG. Party i searches for a non-delivered block B' from a higher round reachable through strong references from B. If such a block B' exists, the function $commit(B')$ is called recursively. If no such block exists, party i deterministically orders the blocks in $past(B)$ before invoking *ab-deliver* on them. In case of conflicting blocks from the same party and round, only the first block – determined by the deterministic ordering above – is *ab-delivered*. Figure 2 shows an example of the commitment rule.

4.5 Functions

Black Marlin (Algorithm 2) uses special functions as described in the paragraphs above. For the sake of the reader, we summarize the functions together in this section.

Algorithm 1 State and functions (party i).

```
1: 𝒩                                                              set of parties
2: r ← 0                                                          current round
3: DAG ← ∅                                                        local view of the DAG
4: 𝒟 ← ∅                                                          set of delivered blocks
5: refs ← ∅                                                       set of strong references
6: refs' ← ∅                                                      set of weak references
7: history[j ∈ 𝒩] ← ∅                                             set of blocks known by party j

8: function quorum(r)
9:     return |parties(DAG(r))| ≥ n − f

10: function anchor(r)
11:     return ∃B ∈ DAG(r) : creator(B) = RR(r)

12: function suppAnchor(r)
13:     return ∃B ∈ DAG(r) : creator(B) = RR(r) ∧ supp(B) ≥ n − f

14: function delivery(r)                                          //attempt a block in r − 2
15:     B ← RR(r − 2)                                             //if multiple consider them all
16:     if supp(B) ≥ n − f ∧ ∃B' ∈ DAG(r − 1) :
            creator(B') = RR(r − 1) ∧ B ∈ strong(B') ∧ supp(B') ≥ n − f then
17:         commit(B)

18: function commit(B)
19:     𝒜 ← strong(B) \ 𝒟                                         //ignore already delivered blocks
20:     if 𝒜 ≠ ∅ then
21:         if |maxAnchor(𝒜)| = 1 then
22:             B' ← maxAnchor(𝒜)                                 //anchor from the highest round
23:         else
24:             B' ← A ∈ maxAnchor(𝒜) : |round(A) − round(maxAnchor(strong(A)))|
                is minimal and break ties deterministically
25:         commit(B')
26:     for B' ∈ τ(past(B) \ 𝒟) do                                // with τ any deterministic topological sorting
27:         if ∄B* ∈ 𝒟 : creator(B') = creator(B*) ∧ round(B') = round(B*) then
28:             ab-deliver(B', creator(B'), round(B'))
29:             𝒟 ← 𝒟 ∪ B'
30:     ab-deliver(B, creator(B), round(B))
31:     𝒟 ← 𝒟 ∪ B
32:     return
```

Given a block B, the function $creator(B)$ returns the party that created and signed B, and the function $round(B)$ returns the round in which B was created. The validity predicate $V(B)$ returns 1 if B has at least $n - f$ blocks from different parties and no two blocks from the same party corresponding to $round(B) - 1$ in its references, and B has been signed by its creator. When invoked with a set of blocks $\mathcal{H}$, the validity predicate $V(\mathcal{H})$ returns 1 if every block $B' \in \mathcal{H}$ satisfies $V(B')$. The function $past(B)$ returns the set of blocks reachable from B through strong and weak links, and the function $strong(B)$ returns the set of blocks reachable from B through strong links, B is not part of the output of either function. The function $supp(B)$ counts the number of parties i that created a block B_i in round $round(B) + 1$ such that $B \in strong(B_i)$ and no other block from the same creator and round is in $strong(B_i)$. The function $H(B)$ is a collision-resistant hash function. The operation $setTimeout(r, \rho)$ starts a timeout of duration ρ corresponding to round r, whereas $timeout(r)$ triggers when the timeout is completed and $clearTimeout(r)$ removes any active timeout corresponding to round r. The function $time()$ returns the local time of the party and when input with a round r, $time(r)$ returns the local time when the party adopted round r. In the case that the party never adopted round r, $time(r)$ returns the time when the party adopted the smallest round $r' > r$. The function $RR(r)$ takes a round as an input and returns the party by the round-robin mechanism in round r, a block from this party is also referred to as *anchor*. In case multiple blocks exist, e.g. due to Byzantine behavior, the function returns both blocks. The variable $DAG(r)$ returns the set of blocks from round r in the local view of the party. The operation $maxAnchor(\mathcal{A})$ takes a set of blocks $\mathcal{A}$ as an input and returns the anchor from the highest round. Given a round r, the function $quorum(r)$ returns 1 if there are blocks from at least $n - f$ parties in $DAG(r)$. The function $anchor(r)$ returns 1 if there is a block created by the anchor of r in $DAG(r)$, and $suppAnchor(r)$ returns 1 if there is an anchor block in r that is supported by at least $n - f$ parties in $DAG(r + 1)$.

5 Analysis

Black Marlin (Algorithm 2) is designed for partially synchronous networks, which means that it may not be alive during the asynchronous period. However, its safety properties are still guaranteed during this period. This fact is reflected in the analysis below.

5.1 Safety

The results in this section apply to every block, even during the asynchronous phase.

▶ **Lemma 2.** *Every honest message* $[\text{BLOCK}, B', \mathcal{H}', j, r]$ *received by an honest party* i *is valid, i.e., satisfies line L58 in the view of* i*.*

Proof. A message $[\text{BLOCK}, B', \mathcal{H}', j, r]$ is valid if block B' and history $\mathcal{H}'$ satisfy the validity predicate V. If the sender is honest, both B' and $\mathcal{H}'$ satisfy the validity predicate. Thus, it is only left to prove that party i has enough information to verify the validity predicate. The construction of the history $\mathcal{H}'$ guarantees this fact. ◀

Lemma 2 applies whenever an honest party sends a message. For the sake of readability, we omit explicit references to this lemma each time an honest message is sent.

▶ **Lemma 3.** *Given two honest parties* i *and* j *and blocks* B_i *and* B_j *from round* r*, if* B_i *satisfies line L16 in the view of party* i *and* B_j *satisfies line L16 in the view of party* j*, then* $B_i = B_j$*.*

Proof. Given a block B_i created by party k in round r that satisfies line L16 in the view of some party, it follows that $creator(B_i) = RR(r)$ and $supp(B_i) \geq n - f$. Recall that, by definition of the support function, an honest party supports at most one block per party per round. Therefore, if blocks B_i and B_j both satisfy line L16 in the views of parties i and j respectively, then the following condition must be satisfied $creator(B_i) = creator(B_j) = RR(r)$ and $supp(B_i), supp(B_j) \geq n - f$. Given the corruption threshold condition of $n \geq 3f + 1$, it is impossible for two distinct blocks to each have support from at least $n - f$ parties. Hence, it must be the case that $B_i = B_j$. ◀

Lemma 3 states that at most one block is honestly committed per round. Furthermore, these committed blocks create a chain in the DAG, as we prove in the following lemmas.

Algorithm 2 Black Marlin (party i).

```
33: upon init do
34:     ab-broadcast(B, i, 1)
35:     send message [BLOCK, B, {}, i, r] to every party
36:     r ← 1
37:     setTimeout(1, 2Δ)

38: upon ∃r' ≥ r : quorum(r') do
39:     setTimeout(r', 2Δ)

40: upon ∃r' ≥ r : quorum(r') ∧
        (anchor(r') ∧ suppAnchor(r' − 1) ∧ suppAnchor(r' − 2)) ∨ timeout(r')) do
41:     clearTimeout(r')
42:     delivery(r')
43:     refs ← ∅
44:     r_ℓ ← min{r' ∈ ℕ : |time() − time(r)| ≤ 3Δ}
45:     weak ← ⋃_{r*=r_ℓ}^{r'−1} DAG(r*)
46:     for j ∈ parties(DAG(r)) do
47:         B' ← B ∈ DAG(r) : creator(B') = j        if multiple B', select any
48:         refs ← refs ∪ H(B')
49:         weak ← weak \ past(B')
50:     refs' ← ∪_{B*∈weak} H(B*)
51:     r ← r' + 1
52:     B ← [r, i, refs, refs']
53:     ab-broadcast(B, i, r)
54:     for j ∈ 𝒩 do
55:         send message [BLOCK, B, DAG \ history[j], i, r] to party j
56:         history[j] ← DAG

57: upon receiving message [BLOCK, B', 𝓗', j, r'] do
58:     if V(B') ∧ V(𝓗') then
59:         DAG ← DAG ∪ {B'} ∪ 𝓗'
60:         history[j] ← history[j] ∪ 𝓗' ∪ {B'}
```

▶ **Lemma 4.** *Consider an honest party i and the maximal round r such that $DAG(r) \neq \emptyset$ in the view of party i, then $|DAG(r')| \geq n - f$ for every round $r' < r$.*

Proof. Consider the maximal round r and a block $B \in DAG(r)$. Since B is a valid block, there exists valid blocks from at least $n - f$ parties in $strong(B)$ corresponding to round $r - 1$. Thus, $|DAG(r - 1)| \geq n - f$. Now, consider any block in $DAG(r - 1)$ and iterate recursively. We conclude that $|DAG(r')| \geq n - f$ for every round $r' < r$. ◀

In contrast to traditional leader-based protocols, a block in DAG-based protocols must be in the past of an overwhelming majority of blocks to ensure its delivery. The following result formalizes this requirement for committed blocks.

▶ **Lemma 5.** *If a block B from round r satisfies line L16 in the view of some honest party i, then every valid block B' from round $r' \geq r+2$ satisfies $B \in past(B')$.*

Proof. By definition of line L16, $supp(B) \geq n-f$, thus $B \in past(B')$ of blocks B' from at least $n-f$ different parties in round $r+1$. We proceed by induction over $\rho := r' - r$:

- Let B' be any block from round $r+2$. By construction, there are blocks from at least $n-f$ in $DAG(r+1)$, and up to f of them could be Byzantine. Block B is in the past of blocks from round $r+1$ from at least $n-f$ different parties, thus $B \in past(B')$. We conclude $B \in past(B')$ of any block B' from round $r+2$.
- Assume that every block B_0 from round $r+\rho_0$ satisfies $B \in past(B_0)$ and let B' be a block from round $r' = r+\rho_0+1$. We have that $B \in past(B')$, since there are at least $n-f$ blocks in $past(B')$ and every block B_0 from round $r+\rho_0$ fulfills $B \in past(B_0)$.

We conclude that any block B from round r that satisfies line L16 in the view of some honest party, then every block B' from round $r' \geq r+2$ satisfies $B \in past(B')$. ◀

▶ **Lemma 6.** *If honest parties i and j commit blocks B_i and B_j respectively, then $B_i = B_j$, $B_i \in past(B_j)$, or $B_j \in past(B_i)$.*

Proof. Let $round(B_i)$ and $round(B_j)$ denote the rounds in which parties i and j commit blocks B_i and B_j, respectively. We analyze three cases based on the values of these rounds:

- **Case 1:** $round(B_i) = round(B_j)$.
  By Lemma 3, at most one block can be committed per round. Hence, $B_i = B_j$.
- **Case 2:** $round(B_i) < round(B_j)$. We further distinguish two sub-cases:
  - **Case 2a:** $round(B_j) = round(B_i) + 1$.
    By the second condition in line L16, there exists a block $B'_j \in DAG(round(B_j))$ such that $creator(B_j) = creator(B'_j)$ and $B_i \in strong(B'_j)$. Since Byzantine parties may create multiple blocks, we cannot immediately conclude that $B_j = B'_j$. However, the third condition of line L16 ensures that both B_j and B'_j satisfy $supp(B_j), supp(B'_j) \geq n-f$, which, by Lemma 3, implies $B_j = B'_j$. Thus, $B_i \in past(B_j)$.
  - **Case 2b:** $round(B_i) + 1 < round(B_j)$.
    In this case, Lemma 5 guarantees that $B_i \in past(B_j)$.
- **Case 3:** $round(B_i) > round(B_j)$.
  This is symmetric to Case 2 and follows by applying the same reasoning, yielding $B_j \in past(B_i)$.

We conclude that any two blocks committed by honest parties satisfy that both parties commit the same block or blocks in the past of each other. ◀

This chain of committed blocks is common across honest parties, thus defining a partial order common to every honest party.

5.2 Liveness

Liveness is guaranteed only after the global stabilization time. For this reason, the majority of the results presented here apply only to blocks that are *ab-broadcast* after time *GST*.

▶ **Lemma 7.** *If an honest party invokes ab-broadcast(B, i, r) at time $T \geq GST$, then every honest party j that invokes ab-broadcast(B', j, r), does it before time $T + 3\Delta$.*

Proof. Assume that party i invokes *ab-broadcast*(B, i, r) at time $T \geq GST$. Then, according to line L55, i sends a message of the form $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ to every party. An honest party j receives this message by time $T + \Delta$. Since the message includes all the

strong references of B, party j's view satisfies $|DAG(r-1)| \geq n-f$. As a result, j can create a block for round r once the anchors have sufficient support or a timeout of duration 2Δ triggers. Therefore, j can create its block by time $T+3\Delta$. ◀

Lemma 5 guarantees that committed anchor blocks are in the past of future blocks. The result below generalizes this property to all honest blocks created after the global stabilization time *GST*.

▶ **Lemma 8.** *If honest party i invokes ab-broadcast(B,i,r) at time $T \geq GST$, then $B \in past(B')$ of every honestly ab-broadcast(B') block B' after time $T+\Delta$*

Proof. Assume that party i invokes *ab-broadcast*(B,i,r) at time T, then i sends a message of the form $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ to every party (L55). Any block B' created by party j after receiving the message $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ satisfies $B \in past(B')$. Lemma 7 guarantees that the message $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ is not garbage collected, thus proving the statement. ◀

▶ **Lemma 9.** *Given an honest party i and a round $r \in \mathbb{N}$, $\exists r' \geq r$ such that i invokes ab-broadcast(B,i,r') for some block B.*

Proof. Let r be a round, and define $r' \geq r$ as the first round such that no honest party has executed it before time *GST*, and $RR(r') = i$. We show that party i eventually invokes *ab-broadcast*(B,i,r') for some block B.

Consider line L40 as executed in round r' by any honest party j. Since i is the leader, no honest party concludes round r without a block from i unless its timeout triggers. Consider j to be the first honest party that concluded round $r'-1$, the duration of its timeout is 2Δ. This guarantees that party i receives the quorum of round $r-1$ sent by j (at most Δ delay) before any honest party concludes round r', thus party i invokes ab-broadcast(B,i,r') ◀

▶ **Lemma 10.** *At time $T \geq GST$, if an honest party concludes a round r (L40) such that the anchors from rounds $r-2$ and $r-1$ are honest, then suppAnchor$(r-2) \geq n-f$ and suppAnchor$(r-1) \geq n-f$.*

Proof. At time $T \geq GST$, assume an honest party concludes round r (L40) and that the anchors from rounds $r-2$ and $r-1$ are also honest. By construction, if a party creates a block for round r at time T', then every honest party receives this block by time $T'+\Delta$ and subsequently creates its own block, which is received by all honest parties by time $T'+2\Delta$.

Given that both anchors from rounds $r-2$ and $r-1$ are honest, it follows that *suppAnchor*$(r-2) \geq n-f$ and *suppAnchor*$(r-1) \geq n-f$, since there are at least $n-f$ honest parties contributing to support anchors in each round. ◀

Lemma 10 states that after the *GST*, the timeout is not triggered with honest anchors.

▶ **Lemma 11.** *After the global stabilization time, given a round r, the probability that honest party i commits a block in r is at least $\frac{(n-t)^2}{n^2}$. Thus the expected number of rounds until a block is committed is at most $\frac{n^2}{(n-t)^2}$.*

Proof. Given a round r, party i commits a block $B \in DAG(r-2)$ (L16) if and only if the following conditions are satisfied:

- The creator of B has been elected by the round-robin mechanism in round $r-2$, i.e., *creator*$(B) = RR(r-2)$, and the block has sufficient support, *supp*$(B) \geq n-f$.
- There exists a block $B' \in DAG(r-1)$ such that $B \in strong(B')$, *creator*$(B') = RR(r-1)$, and *supp*$(B') \geq n-f$.

Assume that the anchors from rounds $r-2$ and $r-1$ are both honest; this implies the existence of $B \in DAG(r-2)$ and $B' \in DAG(r-1)$, as described above. According to Lemma 10, both blocks have enough support: $supp(B) \geq n-f$ and $supp(B') \geq n-f$. Since there are $n-t$ honest parties, the anchors from rounds $r-2$ and $r-1$ are both honest with probability $\frac{(n-t)^2}{n^2}$. Hence, party i commits a block in round r with probability at least $\frac{(n-t)^2}{n^2}$ and commits every a block every at most $\frac{n^2}{(n-t)^2}$ rounds on average. ◀

This lemma shows that, under maximum number of corruptions, honest parties commit a block every at most 9/4 rounds. Nonetheless, a clever adversary may delay commitment for a short period of time (t rounds), at the cost of allowing the network to commit optimally (every round) outside this period.

▶ **Lemma 12.** *Given two honest parties i and j and a block B, then i commits B if and only if j eventually commits B.*

Proof. Assume that an honest party i commits a block B. By Lemma 11, any other honest party j will eventually commit a block B' such that $round(B') \geq round(B)$. Lemma 6 applied to blocks B and B' implies that either $B = B'$ or $B \in past(B')$. Since $round(B') \geq round(B)$, it cannot be the case that $B' \in past(B)$ unless $B = B'$. If $B' = B$, then party j directly commits the same block. If $B \in past(B')$ and $B \neq B'$, then by construction of the delivery function, party j must have also committed B. Therefore, we conclude that if an honest party i commits a block B, then every honest party j eventually commits B. ◀

5.3 Main Result

Previous lemmas are sufficient to show that Black Marlin implements atomic broadcast.

▶ **Theorem 13.** *Black Marlin (Algorithm 2) implements atomic broadcast.*

Proof. We proceed property by property:

Validity. Assume an honest party i invokes $ab\text{-}broadcast(B, i, r)$ after the global stabilization time GST. According to the delivery function (L14), party i outputs $ab\text{-}deliver(B, i, r)$ the first time it observes a block B' such that $B \in past(B')$ and B' satisfies line L16. Lemma 8 guarantees that B is eventually in the past of every valid block. Lemmas 9 and 11 ensure that B' is eventually committed, hence party i outputs $ab\text{-}deliver(B, i, r)$.

Agreement. Suppose an honest party i outputs $ab\text{-}deliver(B, j, r)$. By Algorithm 2, this means a block B^* with $B \in past(B^*)$ satisfies line L16. For any other honest party j, Lemmas 9 and 11 guarantee that j eventually commits a block B' with $B^* \in past(B')$, and thus $B \in past(B')$. Therefore, party j eventually $ab\text{-}delivers(B, j, r)$, ensuring agreement.

Integrity. The delivery function (L16) ensures that honest party j keeps track of all previously delivered blocks. Consequently, j outputs $ab\text{-}deliver(B, i, r)$ at most once per party i and round r. Furthermore, if i is honest, j outputs $ab\text{-}deliver(B, i, r)$ only if i previously invoked $ab\text{-}broadcast(B, i, r)$ as Byzantine parties cannot impersonate honest parties.

Total order. By definition of the delivery function, the order of delivery of non-anchor blocks is determined by the order in which the anchor blocks are committed (L16). Thus, it suffices to show that honest parties commit anchors in the same order. Lemma 12, guarantees that every honest party commits the same set of blocks. The delivery function called with block B guarantees that any anchorin the past of B is committed before B, Lemma 6 guarantees that any committed blocks (possibly by different parties) are in the past of each other. Thus, honest parties commit anchors in the same order. ◀

5.4 Communication and Time Complexity

Previously, we showed that Algorithm 2 is both safe and live in a partial synchronous network. In particular, we demonstrated that blocks are eventually delivered, avoiding any concrete analysis of the communication and time complexity of the algorithm.

5.4.1 Communication complexity

Each round, a party sends at most one message of the form $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$. We assume that the round number can be expressed in logarithmic size $\mathcal{O}(\ln(n))$ (note that the round number grows more slowly than the number of blocks). The identifier of the party can also be expressed in $\mathcal{O}(\ln(n))$ number of bits. For the computation of the size of $DAG \setminus history[j]$ we assume that the Byzantine parties do not send multiple equivocating blocks, as these blocks include signatures that implicate the party, thus the size of $DAG \setminus history[j] \leq \mathcal{O}(n)$. We conclude that the amount of bits sent per honest party and round is $\mathcal{O}(n(|B| + n))$, hence packing n transactions of constant size in B we obtain a communication complexity of $\mathcal{O}(n^2)$ with an amortized complexity of $\mathcal{O}(n)$, both optimal [1].

5.4.2 Time complexity

Lemma 11 gives an upper bound on the expected number of rounds of communication between anchor blocks are committed. An anchor block is committed on average every $\frac{n^2}{(n-t)^2}$ rounds in the presence of t corruptions: in the worst case, every 9/4 round of communication, and every round of communication in the good case. Note that in round r, an anchor from round $r-2$ is committed. Considering latency to be the number of rounds of communication since an anchor block is broadcast until it is delivered we obtain 4.25 and 3 rounds of communication in the average and good case, respectively. It is important to remark that in contrast to most DAG-based protocols, we are measuring in rounds of communication and not instances of reliable broadcast. Thus, Black Marlin not only achieves $\mathcal{O}(1)$ time complexity in expectation, but also the best concrete time complexity concerning DAG-based protocols [24, 25, 5, 15, 6], as their use of reliable broadcast hinders the protocols.

6 Benchmarking

We evaluate the performance of our implementation with respect to the metrics of throughput and latency for a varying number of parties and network delays.

6.1 Experiments

We conducted all benchmarks locally on a MacBook Pro equipped with an M1 Pro chip, 16 GB of RAM, 10 cores (8 performance and 2 efficiency), and running macOS Sequoia Version 15.0.1. To ensure a fair comparison, we modified the benchmarking framework developed for Bullshark [26, 21][2] to incorporate our implementation of Black Marlin. The biggest difference between Black Marlin and Bullshark is the use of common coins by Bullshark. However, their implementation also uses round-robin[3]. This approach allows a close performance

[2] Bullshark has been chosen over more recent frameworks such as Mysticeti [18] due to its closer similarity to Black Marlin and the timing of this project.

[3] `https://github.com/facebookresearch/narwhal/blob/bullshark/consensus/src/lib.rs` L202.

comparison between Black Marlin and Bullshark and also demonstrates that Bullshark's preprocessing techniques can be adapted to Black Marlin. Note that the actual performance of Bullshark is worse due to the lack of common coins in the implementation.

To emulate realistic network conditions despite running benchmarks on a single machine, we modified both the Black Marlin and Bullshark implementations to introduce configurable network delays. Specifically, messages are delivered based on a Poisson distribution with the expected value $\Delta/2$, where Δ represents the maximum network delay assumed by the protocol. This design ensures that with high probability, the actual message delay remains below Δ. For instance, when $\Delta = 400$ ms, the average delivery time for a message is 200 ms.

For each experimental run, we measure *throughput* as the total number of transactions delivered divided by the total duration of the run. The latency of a transaction *tx* is defined as the time elapsed from the creation of the block containing *tx* until it is *ab-delivered*. The *latency* for a run is the average latency of all *ab-delivered* transactions in the run. Every party behaves honestly during the benchmarking.

Each data point in Figure 3 represents the average throughput or latency over 10 independent runs, each lasting 60 seconds. Error bars indicate two standard deviations. We report results across varying numbers of parties (4, 10, and 13) and network delays ($\Delta = 200$, 400, 600, 800, and 1000 ms). All other parameters remained constant throughout the experiments: one worker per party, a transaction size of 512 bytes, and a transaction injection rate of 50,000 transactions per second.

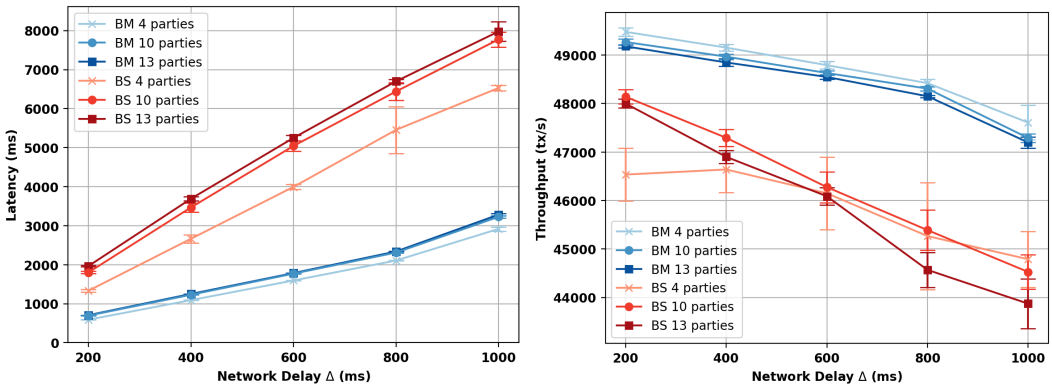
6.2 Results

The results of the experiments described above are summarized in Figure 3, Black Marlin is represented by blue tones and Bullshark by red ones. The two figures of merit that we consider are throughput and latency.

Black Marlin consistently achieves lower latency than Bullshark across all configurations. The latency improvement ranges from a factor of 2 to 3, depending on the network delay. For example, with 10 parties and a network delay of $\Delta = 200$ ms, Black Marlin achieves a latency of 696 ± 5 ms, compared to 1800 ± 54 ms for Bullshark. At $\Delta = 1000$ ms, Black Marlin's latency is 3226 ± 55 ms, while Bullshark's latency increases to 7773 ± 360 ms. This improvement is expected: Black Marlin eliminates the need for reliable broadcast, reducing latency by approximately a factor of 2. Additionally, its more frequent anchor selection contributes to a constant-factor latency reduction, further widening the latency gap.

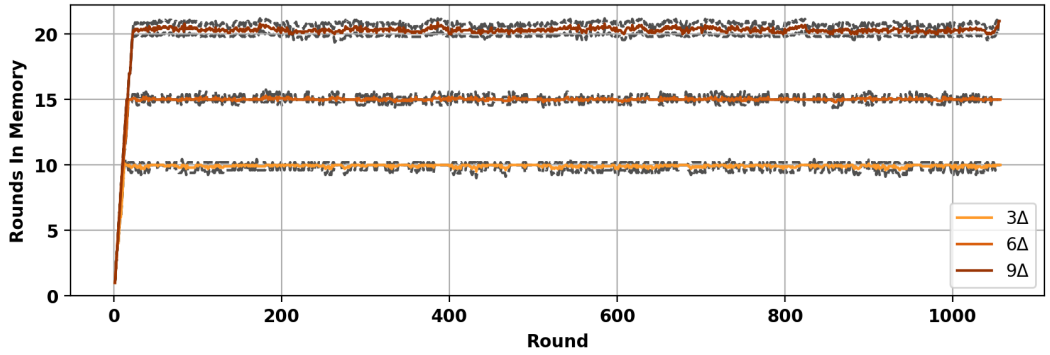
Black Marlin also achieves consistently higher throughput than Bullshark across all network delay scenarios. In contrast to the more pronounced latency gains, the throughput improvement is by a constant margin. This behavior can be explained by the way Bullshark constructs blocks: a party running Bullshark accumulates batches of transactions from workers until it is ready to broadcast a new block. As network delay increases, blocks become larger, which helps mitigate throughput loss. Therefore, the $\sim 2\times$ latency improvement observed with Black Marlin does not directly translate into an equivalent gain in throughput – unless the number of workers or the network delay is significantly increased. However, these adjustments were beyond the scope of this study. We were unable to increase the number of workers due to hardware limitations, and we excluded network delays greater than 1 second, as such conditions are considered unrealistic. This block size phenomenon also explains the erratic throughput results observed for Bullshark under configurations with 4 parties and low network delays. In these cases, small block sizes limited Bullshark's throughput.

Figure 3 also demonstrates that Black Marlin's garbage collection mechanism is robust to overestimations of the actual network delay. Specifically, a constant-factor overestimation leads to only a constant increase in the number of rounds that parties must retain in memory. This implies that even for arbitrarily long executions, Black Marlin only requires $\mathcal{O}(n)$ memory, provided the network delay is overestimated by a constant factor.



(a) Latency of Black Marlin and Bullshark.

(b) Throughput of Black Marlin and Bullshark.



(c) Garbage collection as a function of the estimated network delay.

Figure 3 Black Marlin is shown in blue, Bullshark in red. Darker tones indicate higher numbers of parties: $n = 4$ (cross), $n = 10$ (circle), $n = 13$ (square). Black Marlin outperforms Bullshark in latency and throughput: latency improves by a factor of 2–3; throughput sees a smaller but consistent gain. Bullshark's block creation strategy explains this modest throughput gap – it produces larger blocks to compensate for delays, up to a size limit. This also accounts for Bullshark's irregular throughput at $n = 4$ under low delay, where small blocks constrain performance. Plot (c) shows the number of rounds stored in memory by Black Marlin as a function of estimated network delay. The actual delay is $\Delta = 600$ ms, modeled as a Poisson distribution with mean 300 ms. We vary the garbage collection threshold from 3Δ to 6Δ and 9Δ, observing linear growth in retained rounds. As each round contains $\mathcal{O}(n)$ blocks, memory usage remains $\mathcal{O}(n)$.

Overall, Black Marlin outperforms Bullshark in our implementation, which uses the same codebase and experimental conditions for both protocols. Unfortunately, source code for more recent DAG-based protocols is not publicly available, preventing a direct empirical comparison. However, as we demonstrate in Section 5.4, Black Marlin also surpasses these protocols at a theoretical level in terms of communication and latency complexities while allowing implementations with $\mathcal{O}(n)$ memory.

References

1 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: a complete categorization. In *PODC*, pages 331–341. ACM, 2021. doi:10.1145/3465084.3467899.

2 Ignacio Amores-Sesar and Christian Cachin. We will DAG you. In *ESORICS Workshops (1)*, volume 15263 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2024. doi:10.1007/978-3-031-82349-7_19.

3 Ignacio Amores-Sesar, Christian Cachin, and Philipp Schneider. An analysis of avalanche consensus. In *SIROCCO*, volume 14662 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2024. doi:10.1007/978-3-031-60603-8_2.

4 Ignacio Amores-Sesar, Christian Cachin, and Enrico Tedeschi. When is spring coming? A security analysis of avalanche consensus. In *OPODIS*, volume 253 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.OPODIS.2022.10.

5 Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast and robust! In *NSDI*, pages 813–826. USENIX Association, 2025. URL: https://www.usenix.org/conference/nsdi25/presentation/arun.

6 Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the latency limits with uncertified dags. In *NDSS*. The Internet Society, 2025. URL: https://www.ndss-symposium.org/ndss-paper/mysticeti-reaching-the-latency-limits-with-uncertified-dags/.

7 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985. doi:10.1145/4221.214134.

8 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

9 Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51. ACM, 1993. doi:10.1145/167088.167105.

10 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999. URL: https://dl.acm.org/citation.cfm?id=296824.

11 Andrew Cullen, Pietro Ferraro, Christopher K. King, and Robert Shorten. On the resilience of dag-based distributed ledgers in iot applications. *IEEE Internet Things J.*, 7(8):7112–7122, 2020. doi:10.1109/JIOT.2020.2983401.

12 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

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

14 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

15 Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed BFT. In *SOSP*, pages 1–23. ACM, 2024. doi:10.1145/3694715.3695942.

16 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

17 Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In *DISC*, volume 281 of *LIPIcs*, pages 26:1–26:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.26.

18 Mysten Labs. Github, accessed on April 2025. URL: https://github.com/asonnino/mysticeti/tree/paper.

19 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998. doi:10.1145/279227.279229.

20 Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the tangle. *Comput. Ind. Eng.*, 136:160–172, 2019. doi:10.1016/J.CIE.2019.07.025.

21 Facebook Research. Bullshark implementation. Github, accessed on April 2025. https://github.com/facebookresearch/narwhal/tree/bullshark.

22 Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless BFT consensus through metastability. *CoRR*, abs/1906.08936, 2019. arXiv:1906.08936.

23 Victor Shoup. Blue fish, red fish, live fish, dead fish. *IACR Cryptol. ePrint Arch.*, page 1235, 2024. URL: https://eprint.iacr.org/2024/1235.

24 Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving latency of dag-based BFT. *IACR Cryptol. ePrint Arch.*, page 472, 2024. URL: https://eprint.iacr.org/2024/472.

25 Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 92–109. Springer, 2024. doi:10.1007/978-3-031-78676-1_6.

26 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *CCS*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

27 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.