Starfish: A high throughput BFT protocol on uncertified DAG with linear amortized communication complexity

Nikita Polyanskii
IOTA Foundation
nikitapolyansky@gmail.com

Sebastian Müller
Aix-Marseille Université
sebastian.muller@univ-amu.fr

Ilya Vorobyev
IOTA Foundation
ilia.vorobev@iota.org

Abstract

Current DAG-based BFT protocols face a critical trade-off: certified DAGs provide strong security guarantees but require additional rounds of communication to progress the DAG construction, while uncertified DAGs achieve lower latency at the cost of either reduced resistance to adversarial behaviour or higher communication costs.

This paper presents Starfish, a partially synchronous DAG-based BFT protocol that achieves the security properties of certified DAGs, the efficiency of uncertified approaches and linear amortized communication complexity. The key innovation is Encoded Cordial Dissemination, a push-based dissemination strategy that combines Reed-Solomon erasure coding with Data Availability Certificates (DACs). Each of the $n = 3f + 1$ validators disseminates complete transaction data for its own blocks while distributing encoded shards for others' blocks, enabling efficient data reconstruction with just $f + 1$ shards. Building on the previous uncertified DAG BFT commit rule, Starfish extends it to efficiently verify data availability through committed leader blocks serving as DACs. For large enough transaction data, this design allows Starfish to achieve $O(n)$ amortized communication complexity per committed transaction byte. The average and worst-case end-to-end latencies for Starfish are rigorously proven to be bounded by 7.5δ and 11δ in the steady state, where δ denotes the actual network delay.

Experimental evaluation against state-of-the-art DAG BFT protocols demonstrates Starfish's robust performance under steady-state and Byzantine scenarios. Our results show that strong Byzantine fault tolerance, high performance, and low communication complexity can coexist in DAG BFT protocols, making Starfish particularly suitable for large-scale distributed ledger deployments.

1 Introduction

State Machine Replication (SMR) is a fundamental problem in distributed computing, referring to the task of implementing a fault-tolerant service by replicating state across multiple servers and coordinating client interactions. Byzantine Fault-Tolerant (BFT) consensus protocols address this problem in the presence of adversarial behavior, where Byzantine participants may deviate arbitrarily from the protocol. The partial synchrony model [14] serves as a practical foundation for many BFT protocols, balancing the trade-offs between synchronous and asynchronous communication. In a system with $n = 3f + 1$ validators, BFT protocols can tolerate up to f Byzantine validators in the partial synchrony model.

Recent advancements in blockchain and distributed ledger technologies (DLTs) have fueled the development of scalable BFT consensus protocols that minimize latency and communication complexity per consensus decision. While traditional leader-based BFT protocols rely on sequential block production, DAG-based BFT consensus has emerged as a promising alternative, improving throughput by allowing parallel block proposals and validation.

Two primary classes of DAG-based protocols have emerged: certified DAGs, e.g., [2, 11, 20, 31, 34, 35, 32], which rely on reliable broadcast primitives (RBC) to guarantee data availability and prevent equivocation, and uncertified (optimistic) DAGs [3, 21], which forego such guarantees to reduce latency. Certified DAGs ensure robust security by requiring a quorum of validators to certify each block before it is included in the DAG. This provides strong guarantees of equivocation protection and data availability for each block but increases latency for the DAG construction. Additionally, communication-efficient RBCs such as [12, 7] allow to achieve a linear amortized communication complexity for certified DAGs. In contrast, uncertified DAGs, e.g. [21, 3], rely on best-effort broadcasting with either unknown-history-pushing mechanisms or optimistic pull block dissemination. This reduces the latency in the steady state either at the cost of the quadratic amortized communication complexity or increased vulnerability in adversarial settings.

This paper introduces *Starfish*[1], a novel DAG-based BFT consensus protocol that bridges the gap between certified and uncertified DAGs. By leveraging a new commit rule and a layered approach to data availability, Starfish achieves the robustness of certified DAGs without requiring pre-certification of individual blocks. Starfish modifies the efficient commit rule of uncertified DAGs by decoupling data dissemination from the ordering task. Transaction data is sequenced only after forming Data Availability Certificates (DACs) directly on the DAG. Together with the use of Reed-Solomon erasure coding for the transaction data, Starfish significantly decreases the amortized communication costs per sequenced transaction.

Theoretical Guarantees of Starfish. Byzantine Fault-Tolerant (BFT) consensus and Byzantine Atomic Broadcast (BAB) are equivalent problems, e.g., [6, 10, 14]. However, BAB provides a more natural framework for DAG-based protocols like Starfish, where transaction ordering emerges from the DAG structure rather than explicit leader-driven voting rounds. Starfish operates under the partial synchrony model [14], which assumes that after an unknown Global Stabilization Time (GST), all messages are delivered within a bounded delay Δ, known to the validators. We denote δ as the actual (unknown) message transmission latency such that $\delta \leq \Delta$ after GST. We estimate the **amortized communication complexity**, defined as the average number of bits sent through the network by honest validators per one bit of sequenced transactions. In addition, we analyse two types of latencies: **consensus (block) latency**, defined as the time between the creation of a block by an honest validator and the moment when the transaction data associated with that block is sequenced by all honest validators; **end-to-end (transaction) latency**, defined as the consensus latency plus the time it takes for a transaction to be included in a block.

Theorem 1. *Starfish is a Byzantine Atomic Broadcast protocol in the partial synchrony model. Assuming each validator receives a continuous stream of transactions and creates blocks, including the transaction data of size M, Starfish satisfies the following properties:*

***Amortized communication complexity**: Amortized communication complexity is $O\left(\frac{n^3}{M}\kappa + n\right)$, where κ is the hash size. For $\kappa = O(1)$ and $M = \Omega(n^2)$, the amortized complexity is $O(n)$.*

***Average latency**: if all validators are honest, then the average consensus and end-to-end latencies are upper bounded by 7δ and 7.5δ, correspondingly.*

***Worst-case latency**: after GST, the worst-case consensus and end-to-end latencies are bounded as shown in the table below, depending on the number and type of Byzantine validators:*

	n **honest**	**1 failure**[2]	**1 Byzan.**[3]	f **failures**	f **Byzan.**
block lat.	9δ	$12\delta + 2\Delta$	$12\delta + 4\Delta$	$n\delta + 2f\Delta + O(\Delta)$	$n\delta + 4f\Delta + O(\Delta)$
e2e lat.	$+2\delta$	$+2\delta$	$+2\delta$	$+2\delta$	$+2\delta$

[1] A notable trend in DAG-based BFT protocols is naming them after marine species, e.g. Narwhal [11], Bullshark [35], Mysticeti [3], Shoal [34], Sailfish [31], and many more. Following this, we name our protocol Starfish. Like its namesake, Starfish survives damage—losing a limb (or transaction data) doesn't halt it. Some starfish regenerate entirely from one arm, akin to our protocol reconstruction transaction data from encoded shards.

	RBC used	Avg. block latency	Avg. e2e latency	Amort. comm. complexity	One failure leader
Bullshark [35]	[12]	14δ	$+2\delta$	$O(n)$	$+(8\Delta + 8\delta)$
Shoal [34]	[12]	12δ	$+2\delta$	$O(n)$	$+(8\Delta + 4\delta)$
Shoal++ [2]	[1]	5δ	$+0.5\delta$	$O(n^2)$	$+(8\Delta + 4\delta)$
Sailfish 1 [31]	[12]	9δ	$+2\delta$	$O(n)$	$+(8\Delta + 2\delta)$
Sailfish 2 [31]	[1]	5δ	$+\delta$	$O(n^2)$	$+(4\Delta + 2\delta)$
Cordial Miners [21]	No	5δ	$+0.5\delta$	$O(n^2)$	$+6\Delta$
Mysticeti [3]	No	4δ	$+0.5\delta$	$O(n^2)$	$+4\Delta$
Starfish	**No**	5δ	$+0.5\delta$	$O(n)$	$+2\Delta$

Table 1: Comparison of partially synchronous DAG-based BFT protocols. Latencies are computed for the steady state after GST; validators are assumed to be synchronized, and each non-faulty validator references all blocks from the previous round.

The proofs for the safety (totality, agreement, and total order) of Starfish are similar to other uncertified DAG BFTs [3, 21] and given in Appendix C. As a part of our key contribution, we address several flaws, e.g. see [31], in the liveness (validity) of the existing uncertified DAG-based BFT protocols [3, 21] by proposing a new round advancement mechanism in Section 3 that allows us to formally prove the liveness for Starfish in Appendix D. The amortized communication complexity is addressed in Section 4, where we also propose Starfish-Pull, a variant of Starfish that further improves communication efficiency in practice. Latency analysis is provided in Appendix D.1.

Comparison with State-of-the-Art DAG-Based BFT Protocols. The latency bounds like the ones provided in Theorem 1 are often missing in the literature of DAG-based BFT protocols primarily because it is non-trivial to accurately estimate the consensus and end-to-end latency in terms of δ due to the parallelism of a DAG and the fact that every block does not necessarily reference all blocks from the previous round. There exist bounds on commit latencies for leader blocks in terms of RBC latency and δ [31], consensus latency in terms of rounds [21, 3], and transaction latency in terms of message delay [2]. For this particular comparison, Table 1 attempts to align existing DAG-based partially synchronous protocols by making a strong *Assumption*[4] on how a DAG is constructed: "there is a perfect synchronization across validators, i.e., all validators advance into the new round at the same moment; there is a link from any block from round r to any block from round $r-1$; all messages are delivered within time δ".

We include in this comparison certified DAG-based protocols such as Bullshark [35], Shoal [34], Shoal++ [2], Sailfish [31] and uncertified DAG-based protocols such as Cordial Miners [21] and Mysticeti [3]. Sailfish [31], Shoal++ [2] and Mysticeti [3] allow for supporting multiple leaders every round. However, none of these multi-leader versions was rigorously proven to improve the latency compared to their single-leader versions in the Byzantine environment. For this reason, we depict in the table the results for (at most) one leader in every round.

Block and transaction latencies are computed in the steady state when no Byzantine validator is present in the network. The amortized communication complexity is computed for the worst-case scenario when f

[3] A failure is a Byzantine validator that fails to create its blocks

[3] For the results with 1 Byzantine validator, it is required $n \geq 6$.

[4] While this assumption is not realistic, it allows to make a rough estimate on how the respective protocols perform. A similar assumption was used in Sailfish [31], Shoal++ [2].

Byzantine strategy	Cordial Miners	Mysticeti	Starfish Pull	Starfish	
Baseline	22.65	**1.05**	**1.05**	3.98	Bandwidth efficiency
(all honest validators)	0.721	**0.521**	0.671	0.690	Latency (in sec)
Chain Bomb Attack	16.81	**1.03**	1.62	3.05	Bandwidth efficiency
(8 attackers)	4.944	23.146	1.731	**1.106**	Latency (in sec.)
Equivocating Chains Bomb	21.94	**1.08**	1.85	4.67	Bandwidth efficiency
(1 attacker)	0.967	18.450	2.484	**0.776**	Latency (in sec.)

Table 2: Performance comparison of the state-of-the-art uncertified DAG-based BFT protocols. We consider honest validator scenarios and two Byzantine attacks with $n = 25$ nodes distributed over 10 regions and 40,000 txs/sec load. First value shows bandwidth efficiency ratio (average validator bandwidth divided by total sequenced data), and second shows p50 end-to-end latency (in seconds). See Section 6 for detailed setup description.

Byzantine validators are present in the network.

By employing Reed-Solomon (RS) erasure codes for transaction data, the amortized communication complexity in Starfish becomes linear when the transaction data has size $\Omega(n^2)$, whereas it is linear for certified DAGs by employing RS-based RBC from [12] for transaction data of size $\Omega(n)$. This difference primarily stems from the push-based dissemination strategy of Starfish which has to guarantee the reliable delivery of all blocks in the causal history of a given block, whereas an RBC is applied in certified DAGs for each individual block. In other words, certified DAG BFTs that achieve linear communication complexity treat erasure coding as a separate optimization within the RBC protocol and do not integrate it directly into the DAG structure as we do in Starfish.

The last column in the table quantifies the additional latency incurred when a single Byzantine validator fails to create blocks during its leader turn, demonstrating Starfish's superior resilience with only $+2\Delta$ additional delay compared to $+4\Delta$ or higher in other protocols.

Our analysis reveals three distinct protocol categories with their associated trade-offs:

- Certified DAG protocols (Bullshark, Shoal, Sailfish with RBC from [12]) can achieve linear amortized communication complexity but at the cost of significantly higher latencies (11δ or higher compared to 5.5δ in Starfish).
- Certified DAG protocols (Shoal++, Sailfish with fast RBC from [1]) offer a comparable low latency but have quadratic amortized complexity.
- Uncertified DAG protocols (Mysticeti, Cordial Miners) provide lower or comparable latency but face quadratic communication complexity[5].

A detailed discussion of Table 1 can be found in Appendix E.

Empirical Performance Evaluation. Existing uncertified DAG-based consensus protocols, such as Mysticeti [3] and Cordial Miners [21], face certain challenges. Cordial Miners employs a push-based "cordial dissemination" mechanism, where validators broadcast differences in their local DAGs to peers. However, it introduces significant communication overhead. The implementations of Mysticeti [3] and certified DAG protocols, e.g., [11, 35, 34, 2, 31] adopt a pull-based strategy, leveraging references to previous blocks to request step-by-step all missing blocks. A widely accepted assumption in the field [11] is that push-based dissemination incurs quadratic communication complexity, making it impractical for high-throughput systems.

[5]Even though theoretical amortized communication complexity for Mysticeti is quadratic in the worst case, we observe near-optimal bandwidth efficiency in some practical scenarios, see Table 2.

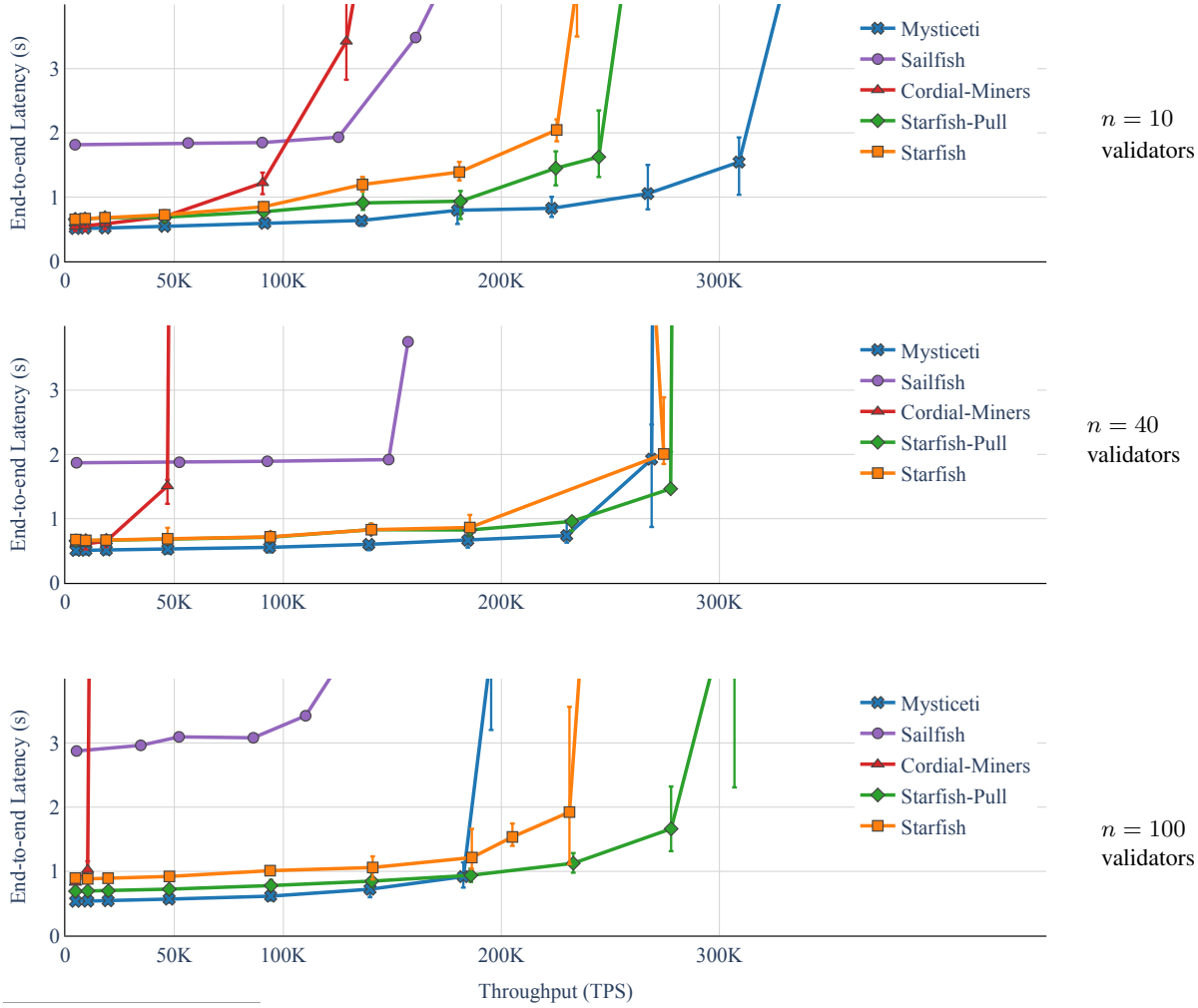

Figure 1: Summary of performance results (e2e latency vs throughput in txs/sec with 512B-sized transactions) for 10, 40 and 100 validators in a geo-distributed network over 10 regions in the steady state. In comparison, we include Sailfish, a state-of-the-art certified DAG-based protocol, Mysticeti and Cordial Miners, two state-of-the-art uncertified DAG-based protocols, and two versions of Starfish. See Section 6 for a detailed setup description.

Starfish challenges this fundamental assumption about push-based dissemination by introducing erasure encoding for transaction data, demonstrating that push-based strategies can be efficient in practice. Instead of broadcasting blocks with entire transaction data, validators distribute smaller encoded shards along with Merkle proofs, allowing efficient transaction reconstruction with minimal redundancy. This approach significantly reduces communication overhead while maintaining strong data availability guarantees and low latency. Moreover, we argue that while pull-based strategies can be robust for certified DAGs, they may increase latency in adversarial settings when used with uncertified DAGs due to repeated requests for missing blocks from potential Byzantine nodes; see Table 2 and Section 6.

In large-scale deployments with a high frequency of blocks, natural inconsistencies in data availability emerge, causing increased delays and performance degradation in pull-based protocols. As shown in Figure 1, while Starfish is slightly less performant than Mysticeti in steady-state scenarios under low stress, its robustness becomes evident in a high-throughput large-scale regime. We refer to Section 6 for a detailed performance evaluation and Starfish's computation overhead estimates.

2 Preliminaries

System Model. We consider a message-passing system $\mathcal{V} := \{v_1, \ldots, v_n\}$ consisting of $n = 3f + 1$ validators or more generally parties. An adversary may control up to f validators referred to as *Byzantine* or *faulty*. These validators may deviate arbitrarily from the protocol. A validator that follows the protocol throughout the execution is considered *correct* or *honest*. A standard assumption is that the adversary is computationally bounded. This ensures that standard cryptographic properties, such as the security of hash functions and digital signatures, hold. A message x digitally signed by validator v using its private key is denoted as $\langle x \rangle_v$. The size of a hash evaluation is assumed to be κ, whereas a signature size is $O(\kappa)$.

We adopt the partial synchrony model of Dwork et al. [14]. In this model, the network initially operates in an asynchronous state, during which an adversary can arbitrarily delay messages sent by honest parties. However, after an unknown time called the *Global Stabilization Time* (GST), the adversary loses this ability, and all messages sent by honest parties are delivered to their intended recipients within a bounded delay Δ, which is known to the parties. Let δ be an (unknown) upper bound on the actual message delay and assume that $\delta \leq \Delta$ holds after GST. Furthermore, we assume that local clocks of parties exhibit *no clock drift* and are free from *arbitrary clock skew*.

Byzantine Reliable Broadcast and Atomic Broadcast. Let us recall the setting and definition of a Byzantine Reliable Broadcast (RBC) as a building block for certified DAGs. In an RBC, a designated sender v_k invokes $r\_bcast_k(m, r)$ to propagate its input m in some round $r \in \mathbb{N}$. Each party v_i attempts to output value $r\_deliver_i(r, v_k)$, where v_k is the designated sender and r is the round number in which v_k sent the message. The reliable broadcast primitive [5] satisfies the following properties:

- **Validity.** If an honest party v_k calls $r\_bcast_k(m, r)$, then every honest party v_i eventually outputs $m = r\_deliver_i(r, v_k)$.
- **Totality** If some honest party outputs a value, then eventually all honest parties will output a value.
- **Agreement.** If two honest parties v_i and v_j output their values for a given round r and designed sender v_k, then the output values are the same $r\_deliver_i(r, v_k) = r\_deliver_j(r, v_k)$.

Byzantine Atomic Broadcast (BAB) extends RBC by ensuring a total order of message delivery across all honest parties. The properties include:

Definition 1 (Byzantine Atomic Broadcast)**.** *Each honest validator $v_i \in \mathcal{V}$ can call $a\_bcast_i(m, r)$ and attempt to output $a\_deliver_i(r, v_k)$, where $v_k \in \mathcal{V}$. A Byzantine atomic broadcast protocol satisfies reliable broadcast properties (validity, totality, and agreement) as well as:*

- ***Total order.*** *If an honest party v_i outputs $a\_deliver_i(r, v_k)$ before $a\_deliver_i(r', v_\ell)$, then no honest party v_j outputs $a\_deliver_j(r', v_\ell)$ before $a\_deliver_j(r, v_k)$.*

Reed-Solomon Codes. For efficient communication, Starfish uses erasure-correcting codes that achieve maximal-distance separable properties [33]. Specifically, we use the standard Reed-Solomon (RS) codes [29]. An (n, k) RS code in Galois Field $\mathbb{F}_q$ with $n \leq q$, encodes k data symbols from $\mathbb{F}_q$ into a codeword of n symbols from $\mathbb{F}_q$. We will consider a systematic encoding, meaning the first k encoded symbols coincide with the input data symbols. Let $\text{RSEnc}(m, n, k)$ be an encoding algorithm of RS codes that takes as input a message m consisting of k field symbols and outputs k information symbols and $n-k$ parity symbols. Similarly, let $\text{RSDec}(t, n, k)$ be the RS erasure decoding procedure. RSDec takes as input an n-length codeword t, which was corrupted by $n - k$ erasures (at any locations) and outputs the whole codeword of the (n, k) RS code. We primarily use q as a power of two. Concrete efficient (with complexity $O(n \log n)$) encoding and erasure decoding instantiations of RS codes include [23].

In Starfish, we encode messages (transaction data) of size larger than k symbols of $\mathbb{F}_q$. For this purpose,

we divide the original message m consisting of M bits into k equal-sized (zero-padded if necessary) parts, called *shards*. Then, each shard of size M/k is treated as $M/(k \log_2 q)$ symbols of $\mathbb{F}_q$. After that, all ith symbols in the k shards are encoded using RSEnc. This results in k *information* and $n-k$ *parity* shards. With a slight abuse of notation, we will use the same notation of RSEnc(m, n, k) for this encoding of large messages m. The encoding and decoding complexity for messages of size M is then $O(M \log n)$.[6]

Block Structure. Each block B in Starfish consists of a *block header* B_{header} and a *block body*. The header is signed by a validator v_i proposing it and defined as:

$$B_{\text{header}} = \langle \text{Round}, \text{Ancestors}, \text{Acknowledgments}, \text{MerkleRoot} \rangle_{v_i},$$

where Round is the round number r; the set Ancestors contains n hash references to the latest blocks created at or before round $r-1$ from distinct validators such that it always includes at first position a hash reference to the previous block of validator v_i and there are blocks from at least $2f+1$ validators created at round $r-1$; MerkleRoot is the Merkle root [26] of the n encoded shards RSEnc(TransactionData, n, $f+1$); Acknowledgments are hash references to blocks from rounds at most $r-1$ for which the transaction data is available to v_i (and was not acknowledged by v_i before). The hash reference of a block is the hash of the block header.

The block body optionally contains the transaction data and an encoded shard of the transaction data, along with Merkle proof for verification. A full block is not signed and defined as: $B = \langle B_{\text{header}}, \text{Option(TransactionData)}, \text{Option(E}$ where EncodedShard is a Reed-Solomon encoded shard of the transaction data, and MerkleProof ensures shard integrity. The creator of the block header v_i includes the transaction data in its block but not the encoded shard. Any other validator $v_j \neq v_i$ includes the jth encoded shard in RSEnc(TransactionData, n, $f+1$) along with the corresponding Merkle proof but not the original transaction data. Unlike the block header, the content of the body of a block depends on who sends the block and whether the transaction data is available to the sender.

The Block DAG. Each block in Starfish references at least $2f+1$ blocks from the previous round, ensuring that new blocks are causally linked to prior proposals. We model the protocol's execution as a directed acyclic graph (DAG), where each block header serves as a unique vertex, and a directed edge from block header B to block header B' indicates that B explicitly references B'. Every honest validator proposes a signed block header and broadcasts it with the transaction data, while Byzantine validators may attempt to equivocate or withhold their proposals or the transaction data. A validator advances to the next round only after creating the block in the previous round and receiving at least $2f+1$ blocks from the previous round. This structure ensures that the DAG serves as a logical clock, where time progresses as blocks accumulate.

Atomic Broadcast and the Starfish DAG. Atomic broadcast requires participants to agree on a consistent order of message receipt, ensuring that even validators recovering from failures can learn and comply with the established order. In Starfish, the DAG structure enforces a globally consistent block order with its commit rules. For correctness, the message m in $a\_bcast_i(m, r)$ must correspond to the transaction data of validator v_i at round r. The block header acts as verifiable proof that v_i initiated this broadcast.

3 Overview of Starfish

In this section, we informally describe the protocol and give some insights on why it works. To allow a better comparison, we stay as close as possible to the notation of Cordial Miners [21] and Mysticeti [3]. We refer to Appendix A for a more formal description.

[6] The implications of this additional computational complexity are discussed in Section 6.

Advance round $r \to r+1$	**Create block of round** r	**Broadcast history**
A1: received $2f+1$ blocks of round r **AND** **A2:** created block of round r	**C1:** waiting for leader and votes: **L1:** received leader block of round $r-1$ **AND** **L2:** received $2f+1$ blocks voting for leader of round $r-2$ **OR** there is a *skipped pattern* for leader of round $r-2$ **OR** **C2:** wait timeout $\delta_{TO} = 2\Delta$ after entering round r **OR** **C3:** received $2f+1$ blocks of round r	**B1:** block is created **OR** **B2:** round is advanced

Table 3: Conditions for advancing rounds, creating blocks and broadcasting unknown history in Starfish

High-level goal. In Starfish, the DAG progresses in rounds using threshold clocks. We call every 3 consecutive rounds a *wave* and assign a unique *leader* validator to the wave. For every $r \in \mathbb{N}$, a wave $w(r) := r$ consists of three rounds: *proposer* round r, *voting* round $r+1$, and *certifying* round $r+2$. Validators agree beforehand on the round-robin leader scheduler: validator v_i proposes leader blocks in rounds $r = i, i+n, i+2n \ldots$. By interpreting the DAG structure, the validators decide which leader blocks are *committed* or *skipped* and derive a consistent sequence of committed leader blocks. Using this sequence, the validators find for which prior blocks the committed leader blocks are Data Availability Certificates (DACs), i.e., a DAC must reach $2f+1$ acknowledgements for the transaction data of a past block, see Figure 2. This allows us to derive a consistent order of transaction data.

Cordial Dissemination. The DAG structure, particularly the causal history of a block, can be used to improve the reliable dissemination of the blocks. For instance, a validator creating a block B indicates that it knows the ancestor blocks and the whole ancestral history. When all validators are honest and the moment of GST has already occurred, dissemination can be realized simply by each validator sending each new own block to all other validators. However, Byzantine validators may fail to do so, possibly intentionally, and send new blocks only to some validators.

The principle of cordial dissemination [30] is: "Send to others blocks you know and think they need." More precisely, a validator broadcasts the unknown history to other validators when one of the events occurs

B1: it creates a new block; **B2:** it enters a new round.[7]

When triggered by **B1**, cordial dissemination ensures that validators can add the new block to their local DAG upon receipt. When triggered by **B2**, it allows validators to sync information about the round advancement of their DAGs, which in turn guarantees timely cordial progression of the DAG construction. The downside of this "push" strategy is that it increases communication complexity.

Erasure Coding and Encoded Dissemination. In Starfish, ($n = 3f+1$, $k = f+1$) Reed-Solomon (RS) coding is employed to reduce the communication complexity of cordial dissemination. The transaction data of each block is first divided into $f+1$ *information shards* (containing raw transaction data). Then, they are encoded by an RS code to obtain the $n-f-1$ *parity shards* (for redundancy). A *Merkle tree* is constructed over all shards to ensure data integrity, with the *Merkle root* included in the block header. Any $f+1$ shards verified against the Merkle root using the Merkle proofs allow for the reconstruction of the original transaction data. After the transaction data gets locally available, a validator v_i acknowledges that in its next block and shares its encoded shard with others.

When the broadcast history event, see Table 3, is triggered, v_i broadcasts the block headers of unknown

[7] This second condition to send the unknown history is not present in the original description of Cordial Miners [21].

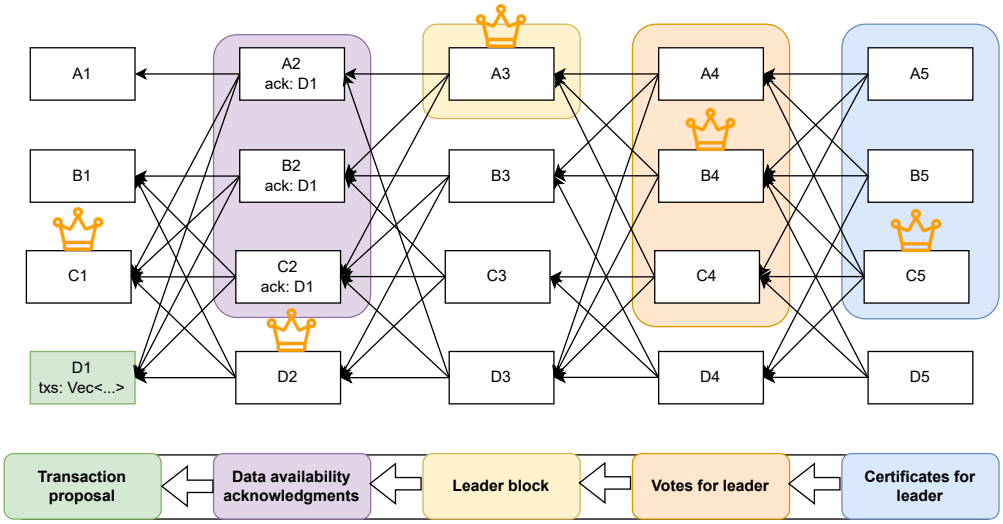

Figure 2: Commit rule in Starfish. Four validators, A, B, C, and D, created blocks in five rounds. Transactions in a block (D1) become ready for sequencing when its data acknowledgements form a quorum (A2, B2, C2) reachable from a committed leader block (A3) serving as a Data Availability Certificate. A leader block (A3) is directly committed when the DAG contains a quorum of certificates (A5, B5, C5), where each certificate observes a quorum of blocks (A4, B4, C4) voting for that leader. There are (at least) 5 rounds to sequence transactions: 1 round: transaction proposal, 2 round: acknowledgements, 3 round: leader block, 4 round: votes for the leader, 5 round: certificates for the leader. Transactions can be sequenced when a) the decision (to-skip or to-commit) for all previous leader blocks has been made and b) all ready-to-be sequenced transaction data up to the leader block is locally available.

history to other validators. For own blocks, v_i includes the transaction data in the block body. For blocks of others, v_i includes the ith encoded shards (if available), together with Merkle proofs in the block body.

We note that a given validator can broadcast the same block of another validator twice: once without any encoded shard and once with the encoded shard after the transaction data becomes available.

Advancing Rounds and Creating Blocks. We consider one leader per round. In Starfish, we need at least three rounds to commit a leader block. Let δ_{TO} be a protocol parameter that describes the timeout for waiting for conditions to be satisfied after entering a round and creating a block.

The timeout δ_{TO} is set to 2Δ, where Δ is a known upper bound for the message delivery time after GST. The rules on how to advance rounds and when to create blocks are given in Table 3.

The conditions for round creation and block advancement are similar to those in Mysticeti and Cordial Miners' protocols, but there are some essential differences. Condition **C3** is new compared to previous protocols and ensures a faster synchronization of the validators. We also changed the condition **L2**, allowing not to wait for voting blocks if it is clear that the corresponding leader block can't gather a quorum of votes. With this new condition, if the leader block of the round r was not created, we will wait for timeout only in round $r + 1$, whereas without it, we would also need to wait in round $r + 2$. The Mysticeti paper does not formally describe advancing rounds and creating block logic, so it is hard to point out the differences. Cordial Miners' protocol works with non-intersecting waves and has one leader in 3 rounds, whereas we have one leader for every round. Cordial Miners have a condition similar to our condition **C1**, but their validators wait for one of 3 subconditions **L1**, **L2** or **L3** to be satisfied, depending on the round offset. Their additional condition, **L3**, requires the validator to wait until it receives $2f + 1$ certificates for the last leader block. We do not use the condition **L3** since, in our case, it can only improve the latency in terms of the number of rounds, not derived in the actual network latency δ. Their condition **L2** does not have a part about the skipping pattern, which increases the latency in case of a faulty leader. We also note that in Cordial

Miners' protocol, advancement to round r and creating the round-r block happen at the same time, whereas in our protocol, there can be a time interval between these two events. However, both the advancement of the round and the creation of the block trigger the broadcast of unknown history in Starfish.

Liveness Proof. As pointed out in [31], Mysticeti's liveness proof has some issues. It is claimed in Lemma 8 in [3] that after GST, all honest validators will enter the same round within Δ, but there are gaps in the proof of this lemma. In Cordial Miners' protocol, there are also gaps in the proof of Proposition 38, where it is assumed without explanation that all honest miners are in the same round after GST. Our new condition **C3**, together with condition **B2**, helps to ensure that this property holds. Intuitively, if the validator v received $2f + 1$ round-r blocks with their history, but the condition **C1** is still not satisfied, then at least $f + 1$ honest validators advanced from round r to round $r + 1$ after their timeouts expired. Then, we allow the validator v to advance to the next round without waiting for its timeout. This condition allows the validators who are behind others to catch up quickly. Formally, the new condition allows us to prove that after GST, honest validators enter any round within a time interval of length Δ and create their blocks in round r within a time interval of length Δ.

Commit Rule for Leader Blocks and Safety. The commit rule for leader blocks remains similar to that in Mysticeti [3] or Cordial Miners [21]. In the simplest case, a leader block is directly committed when a quorum of certificates appears in the local DAG; see the example in Figure 2. A certificate for a leader block is a block header in the certifying round that observes a quorum of blocks in the voting round supporting the leader block. A leader in a proposing round could also be directly skipped if one finds a *skipped pattern* on the DAG: no potential leader block in that round could get a quorum of votes. Like Mysticeti, we rely on indirect decision rules for leader blocks when the direct decision rule leaves the state undecided; see detailed description in Appendix A.2.

The consistency in the order of committed leaders in Starfish can be obtained similarly to the previous work. Intuitively, since a certificate requires a quorum of votes, no two equivocating leaders could have certificates due to quorum intersection by an honest validator. Similarly, a leader directly committed by one validator can not be skipped by another due to a quorum intersection by an honest validator.

Data Availability Certificate and Sequencing Transactions. One challenge in Starfish is that the transaction data of a block might not be available for a given validator. This availability could be guaranteed when the block body is required to contain the transaction data as it is done in the previous uncertified DAG BFTs. Starfish introduces an additional rule for sequencing the transaction data of blocks. Validators include acknowledgments of transaction data availability in their blocks. Specifically, validators explicitly express the availability of the transaction data of a previous block, say B, in their blocks. We sequence B if there are $2f + 1$ acknowledgments *reachable* from the committed leader block, which we call a data availability certificate (DAC) for B. Note that the commitment of leader blocks is consistent across validators as it relies on the block headers, which are signed; therefore, the ordering of the transaction data is also consistent.

Unlike a DAG with certified block headers, e.g. Narwhal [11], block headers in Starfish can reference the leader block even when not all of its transaction data (or transaction data of prior blocks) is available to a quorum of validators and these transactions can be sequenced later than the commitment of the leader blocks. This enables Starfish to commit leader blocks faster. Moreover, a Starfish DAG might contain blocks of faulty validators whose transactions are never sequenced due to the lack of DACs.

Intuition for Latency Analysis. As mentioned, compared to Cordial Miners, we separate two events: round advancement and block creation. For each of these events, we trigger broadcasting of unknown history to ensure synchronized actions across all honest validators. Formally, we prove the following: for any $r > r_{\max}$ (the largest round among honest validators at the moment of GST), all honest validators create their r-round blocks within an interval of duration δ and enter round r within an interval of duration δ (the unknown bound of the actual network delay).

However, the above fact is not sufficient to prove bounds for latencies. To this end, we make use of an extreme-case analysis and examine how fast the *slowest* honest validators create their blocks. Specifically, we prove and then heavily rely on the following fact: after GST and assuming the leaders of rounds $r-1$ and r are honest, the slowest honest validators in rounds r and $r+1$ create their blocks with a delay of at most δ. This means that the DAG grows *on average* at a pace of at least one round per time δ in the steady state. This implies, in particular, that a transaction ends up in a block in time $\delta/2$ on average.

One Byzantine validator at round r can increase the longevity of rounds $r+1$ and $r+2$ by the timeout duration $\delta_{TO} = 2\Delta$, whereas a simple failure can do this only in round $r+1$ due to the skip pattern in condition **L2**. For the worst-case analysis, one can also show that the maximum time between the creation of two consecutive blocks by a given validator is at most 2δ when no Byzantine nodes are present in the previous two rounds. In particular, this means that a transaction ends up in a block at the latest after time 2δ if no Byzantine node was a leader in a few past rounds.

4 Starfish-Pull: A More Efficient and Scalable Variant

Communication complexity of Starfish. Each validator in Starfish broadcasts its own block to every other validator only once, whereas it broadcasts the blocks of other validators at most twice (without and with encoded shards). The data availability acknowledgement for a given block is included at most once by every validator. Assuming the size of each block is M, the hash size is κ and a signature has size $O(\kappa)$, the size of an own block for a validator can be estimated on average as $O(1) + \kappa n + \kappa n + \kappa + O(\kappa) + M$ (round, hash references, acknowledgements, Merkle root, signature, transaction data). Similarly, the size of a block from another validator is estimated as $O(1) + \kappa n + \kappa n + \kappa + O(\kappa) + M/(f+1) + \kappa \log n$ (the last two terms stand for the encoded shard and the Merkle proof). Therefore, the average communication costs for one round across all honest validators are upper bounded as:

$$2n(n-1)^2 \left(O(1) + 2\kappa n + O(\kappa) + M/(f+1) + \kappa \log n\right) + n(n-1)\left(O(1) + 2\kappa n + O(\kappa) + M\right).$$

This behaves as $O(\kappa n^4) + O(Mn^2)$ as $n = 3f+1$. For transaction data of size M (the total transaction size for one round is at least $M(2f+1)$ across honest validators), this leads[8] to the amortized costs: $O(\kappa n^3/M) + O(n)$. Assuming a constant hash size κ and large enough transaction data $M = \Omega(n^2)$, this leads to a linear amortized communication complexity. When all validators are honest and a message m between $v_i \to v_j$ arrives faster than through the path $v_i \to v_k \to v_j$, the leading coefficient in front of a linear term of the communication complexity can be estimated as $1 + \frac{n}{f+1}$, which behaves as $4 + o(1)$ for large n.

Starfish-Pull. As a practical trade-off between bandwidth and latency, we propose an alternative version of the Starfish protocol called Starfish-Pull, which aims to decrease the factor 4 in the steady state. In this strategy, instead of pushing the unknown history validator v_i uses best-effort broadcast to disseminate its own block. If a peer is not aware of the full history of this block, it makes a missing history request for this block to v_i. Only after receiving such a request does v_i push the unknown causal history. The information in this pushed missing history contains the block headers and the corresponding encoded shards of v_i if available. We use the same commit rule as in Starfish. It is now possible that some transactions are ready to be sequenced by the Starfish commit rule but are still unavailable locally. For this reason, a validator sends a special message requesting the unavailable transaction data to the validators who have acknowledged that the data is available. Starfish-Pull requires extra rounds of communications in case of Byzantine attacks; however, it consumes less bandwidth, requires less CPU usage, and handles a higher throughput in the

[8]The arguments here assume that no equivocation happens. However, one can address possible equivocators similarly to how it is done in Cordial Miners [21]. Once a validator observes an equivocator, it must acknowledge this once and stop directly referencing its blocks and broadcasting encoded shards of its transaction data (unless the commit rule sequences the transaction data). After some moment after GST, honest validators will not have any communication costs incurred by the transaction data of equivocators.

steady state, see Figure 1. In practice, the constant in front of the linear term can be very close to 1, see detailed performance evaluation setup in Section 6 and Table 2.

5 Implementation

Starting with the Mysticeti codebase[9], we implement and open-source[10] our prototype of the Starfish protocol in Rust. It uses tokio[11] for asynchronous networking and utilizes TCP sockets for communication without relying on any RPC frameworks. For cryptographic operations, we rely on the ed25519-consensus[12] signature scheme and blake3[13] for cryptographic hashing. For the serialization and deserialization of protocol messages and internal data structures, we utilize bincode[14]. Unlike the originally implemented WAL (Write-Ahead Logging) storage, we stick with RocksDB[15] for persistent storage of the consensus data, such as blocks and commits. For encoding and decoding the transaction data, we employ the Reed-Solomon-SIMD[16] crate that implements Reed-Solomon codes over the field $\mathbb{F}_{2^{16}}$ with erasure decoding based on the Fast-Fourier transform. It also leverages SIMD instructions when working with shards that could be larger than two bytes (the field size $q = 2^{16}$). Our prototype does not contain the execution and ledger storage components to measure solely the consensus performance, similar to how it is done in all other past works; see [35, 2, 31, 3].

Core thread and connection tasks. Similar to the Mysticeti implementation[17], we use one dedicated thread for most critical operations such as updating the local DAG with received blocks, creating new blocks, applying the commit rule of Starfish and sequencing transactions. The preprocessing and verification of blocks received from a given validator is performed by a separate thread that handles all incoming and outgoing network messages with that peer. Preprocessing starts with checking whether the block was not yet included in the local DAG and then continues with encoding the transaction data, hashing the encoded data, constructing the Merkle tree and verifying the signature. Suppose a block with transaction data is verified. In that case, the connection thread creates two copies of the block together with their serializations: one contains the full transaction data and aims for storage, and another contains only one encoded shard of the transaction data and is used for potential further transmission. With one dedicated thread, this approach allows for the effective optimization of the critical path and the avoidance of race conditions.

6 Performance Evaluation

We implemented a prototype in Rust to run a validator node for both versions of the Starfish protocol. We refer to Appendix 5 for more implementation details. To compare apples-with-apples, we include in our testbed the state-of-the-art uncertified DAG-based BFT protocols: we implement Cordial Miners [21] (whose implementation was not yet available) and include the Mysticeti [3] implementation[18] with several target optimizations and modifications. Recall that Mysticeti is currently deployed on the Sui[19] and IOTA[20] blockchains. In addition, for our baseline comparison, we used a publicly available code[21] for Sailfish [31],

[9] https://github.com/asonnino/mysticeti/tree/paper
[10] https://github.com/iotaledger/starfish
[11] https://tokio.rs
[12] https://docs.rs/ed25519-consensus/
[13] https://docs.rs/blake3/
[14] https://docs.rs/bincode/
[15] https://rocksdb.org/
[16] https://crates.io/crates/reed-solomon-simd
[17] https://github.com/MystenLabs/mysticeti
[18] https://github.com/MystenLabs/mysticeti
[19] https://sui.io/mysticeti
[20] https://docs.iota.org/about-iota/iota-architecture/consensus
[21] https://github.com/nibeshrestha/sailfish

	USE1	USW1	CAC1	EUW1	EUS2	EUN1	SAE1	APS1	APSE2	APNE1
USE1	1	65	14	68	104	110	112	201	198	146
USW1	65	1	78	127	163	172	175	226	137	108
CAC1	14	78	1	67	106	103	122	189	196	142
EUW1	68	127	67	1	29	38	176	125	254	199
EUS2	104	163	106	29	1	50	215	143	281	238
EUN1	110	172	103	38	50	1	220	148	268	245
SAE1	112	175	122	176	215	220	1	299	309	254
APS1	201	226	189	125	143	148	299	1	150	140
APSE2	198	137	196	254	281	268	309	150	1	101
APNE1	146	108	142	199	238	245	254	140	101	1

Table 4: Round-trip latencies between regions (milliseconds)

that was shown to improve the end-to-end latency of existing certified DAG BFTs such as Bullshark [35] (was adopted by the Sui blockchain before June 2024) and Shoal [34] by 10-25%. We note that Sailfish is implemented on top of Narwhal [11], which uses an additional worker layer to continuously disseminate transaction data. This allows for a higher throughput, but increases the resulting latency[22]. As demonstrated in [2], including transaction data directly to blocks allows for decreasing the latency by an additional 10-20%. [23]

In our evaluation, we seek to answer the following questions:

Q1: How well can a push-based dissemination scale with increasing validator count and transaction load?

Q2: How does Starfish's performance compare to state-of-the-art DAG-based BFT protocols in terms of latency and throughput?

Q3: What performance advantages does push-based dissemination with encoding offer over Mysticeti's pull-based approach and Cordial Miners' push-based approach without encoding under Byzantine attacks?

Q4: What computational overhead does Starfish imply compared to Mysticeti due to encoding and decoding operations, hashing more data, and Merkle tree computations?

Experimental Setup. We use the Amazon Web Services (AWS) to mimic the deployment of a globally decentralized network. We evaluate performance using Amazon EC2 m5d.4xlarge instances, each equipped with 16 vCPUs, 64 GiB RAM, 10 Gbps network bandwidth, and 600 GB of NVMe SSD storage. Our testbed consists of $\{10, 25, 40, 100\}$ validators spread evenly[24] across ten global regions: US East (Virginia, USE1), US West (California, USW1), Canada Central (Quebec, CAC1), EU West (Ireland, EUW1), EU South (Madrid, EUS2), EU North (Stockholm, EUN1), South America East (São Paulo, SAE1), Asia Pacific South (Mumbai, APS1), Asia Pacific Southeast (Sydney, APSE2), and Asia Pacific Northeast (Tokyo, APNE1). Before starting the simulation, we have measured round-trip latencies between different regions. To construct the latency matrix, Table 4, we deploy virtual machines in selected AWS regions and execute ping tests to measure round-trip times (RTT). Each value is averaged over 10 measurements and rounded up. The gathered latencies provide inter-region communication delays for our specific measurements. The network round-trip times (RTTs) range from 14ms between nearby regions to 309ms between distant regions, with cross-continental latencies typically between 200-300ms.

[22]We note that in the concurrent work of Sailfish++ [32], the Narwhal layer is removed and a signature-free implementation is used. This allowed the authors to significantly improve the end-to-end latency.

[23]We do not include Shoal++ [2] in our comparison due to the lack of a publicly available codebase at the time we carried out our simulations.

[24]In case of 25 validators, we use three validators for each of the first five regions and two for each of the remaining regions.

Protocol	$n = 10$	$n = 40$	$n = 100$
Sailfish	1.815	1.870	2.874
Mysticeti	0.510	0.505	0.533
Cordial-Miners	0.540	0.610	0.826
Starfish-Pull	0.654	0.655	0.691
Starfish	0.654	0.674	0.895

Table 5: Baseline (all honest) e2e latency (in sec.) at minimal load for different validator counts.

Each honest validator is connected to one client, which issues a continuous stream of random transactions, each consisting of 512 bytes. The end-to-end latency for validator v_i is measured as the time between when a transaction arrives to a validator v_j who includes it in the block and when the transaction is eventually sequenced and ready to be executed by v_i. We average the latency measurements across all validators. When referring to throughput, we mean the number of sequenced transactions over the entire duration of the run divided by the duration in seconds. We run each experiment for over three minutes.

We use a timeout $\delta_{TO} = 2\Delta$ of 600ms in each round for Starfish, Cordial Miners, Mysticeti; Sailfish by default uses a 3s timeout. We note that 600ms is much larger than the largest RTT (see Table 4). However, the algorithm assumes an instant reaction, which is not possible in case of a high load due to the overhead in the block creation and flushing all consensus data to the disk. Nevertheless, we deem the chosen timeout is sufficient for all experiments we provided. For Mysticeti, we enabled only one leader in each round since multiple leaders in each round do not show any benefit in the case of Byzantine nodes. In addition, validators in our modification of the Mysticeti implementation request missing parents not from a limited number of other validators like it is done in the original testbed[25], but whenever a block has this missing parent. This modification allows for obtaining more reasonable results for Mysticeti under Byzantine attacks.

Baseline Performance. Figure 1 illustrates throughput and end-to-end p50 latency for 10, 40, and 100 validators under steady-state conditions (no Byzantine nodes and stable latencies between validators). We include error bars indicating p25 and p75 latencies for all protocols except Sailfish. Table 5 shows the baseline p50 latency under minimal load.

For **Q1** (scalability of push-based dissemination), Starfish demonstrates robust scaling behavior, maintaining sub-second latency up to approximately 150K TPS across all validator counts (10, 40, and 100). The consistent performance patterns across different validator counts show that push-based dissemination scales well with increasing network size. However, Cordial Miners, with its push-based broadcast approach *without* encoding, fails to achieve comparable performance due to significantly higher communication complexity, resulting in increased bandwidth requirements and CPU load.

For **Q2** (comparison with state-of-the-art), Starfish achieves competitive performance against other DAG-based protocols. While Mysticeti (uncertified DAG BFT) shows lower baseline latency under small load (see Table 5), Starfish maintains more stable latency scaling for large validator counts (40 and 100) and high throughput. This advantage likely stems from high-frequency block regimes, where validators process blocks out of expected order, potentially causing missing ancestors or history requests in Mysticeti and Starfish-Pull. Unlike Starfish-Pull, a single request may prove insufficient in Mysticeti, leading to delayed block creation and increased core thread task (see Section 5). Compared to Sailfish (certified DAG BFT), which shows higher baseline latency (1.8-2.8s), Starfish consistently delivers significantly better latency across all throughput ranges. Both Starfish variants substantially outperform Cordial Miners beyond 50K TPS due to Cordial Miners' scaling limitations.

[25] `https://github.com/MystenLabs/mysticeti`

Performance Under Attack Scenarios. To evaluate **Q3** (advantages of push-based dissemination with encoding), we analyze performance under Byzantine attacks with $n = 25$ validators and 40,000 tx/sec load. Table 2 reports end-to-end latency (seconds) and bandwidth efficiency ratio—defined as average validator bandwidth divided by (total sequenced transactions × transaction size). This metric approximates $A(n)/n$, where $A(n)$ is the amortized communication cost per byte of sequenced transaction.

We implement two closely related attacks: *Chain Bombs* and *Equivocating Chains Bomb*. In a Chain Bombs attack, Byzantine nodes operate without equivocation but exploit the leader schedule and use a special broadcasting strategy. For each $i \in \{1, \ldots, 8\}$, Byzantine node v_{3i} occupies position $3i$ in the leader schedule. This arrangement represents the worst-case scenario for both Mysticeti and Starfish commit rules, as their indirect decision rules require three consecutive honest leader blocks in the DAG to break the dependency between potentially undecided states of Byzantine leader blocks. Each Byzantine node i employs a selective broadcasting strategy: when acting as a leader, it pushes its chain of blocks exclusively to the validator scheduled as leader at position $3i + 1$, while ignoring all other communication and block requests.

The Equivocating Chains Bomb attack differs by concentrating malicious behavior in a single Byzantine validator that creates $n - 1$ equivocating blocks per round. For each $i \in \{1, ..., n - 1\}$, the ith equivocating block references only its counterpart (the ith block) from the previous round. The Byzantine validator strategically times its attack by sending each chain of ith blocks to the ith validator immediately before the round when that validator becomes the leader.

Under both attacks, Starfish's push-based dissemination with encoding demonstrates key advantages:

- *Latency Resilience*: Starfish maintains the lowest latency among all protocols. In contrast, Mysticeti's pull-based approach, while bandwidth-efficient (ratio close to 1), suffers significant latency increases and potential liveness issues under adversarial network conditions.
- *Bandwidth Efficiency*: Starfish achieves a bandwidth efficiency ratio around 4, which aligns with the theoretical estimate in Section 4. This is substantially better than Cordial Miners' 16-23. This efficiency stems directly from encoding: Cordial Miners, using uncoded push-based broadcast, requires processing significantly more data, resulting in higher latencies.
- *Balanced Design*: Unlike Mysticeti's pull-based optimization that trades latency for bandwidth, Starfish's encoded push-based approach maintains both low latency and reasonable bandwidth usage under attack.

These results demonstrate that Starfish's push-based dissemination with encoding effectively balances performance metrics under Byzantine conditions, addressing limitations of both pull-based (Mysticeti) and uncoded push-based (Cordial Miners) approaches.

Computational overhead for processing data. To evaluate **Q4** (computational overhead), we first estimate the computation overhead in theory and then measure the performance of different components of our implementation of Mysticeti and Starfish.

As mentioned in Section 2, encoding and decoding complexities for messages of size M using Reed-Solomon codes of length n (number of validators) is $O(M \log n)$. Assume that hashing of data of size M takes time $O(M)$, signature generation and verification are $O(1)$. Merkle tree construction from encoded shards then takes $O(M) + O(n)$ time, whereas Merkle proof verification for a given shard takes time $O(M/n) + O(\log(n))$. Block creation in Starfish is then $O(M \log n) + O(n)$ compared to $O(M)$ in Mysticeti. Block verification of a block from an honest validator is $O(M \log n)$ compared to $O(m)$ in Mysticeti. Block verification and reconstruction of a block from a Byzantine validator could take $O(M \log n) + O(n \log n)$ in the worst case in Starfish (after receiving block headers and block shards from all other validators). We note that in the current architecture that we used, where all communication with different

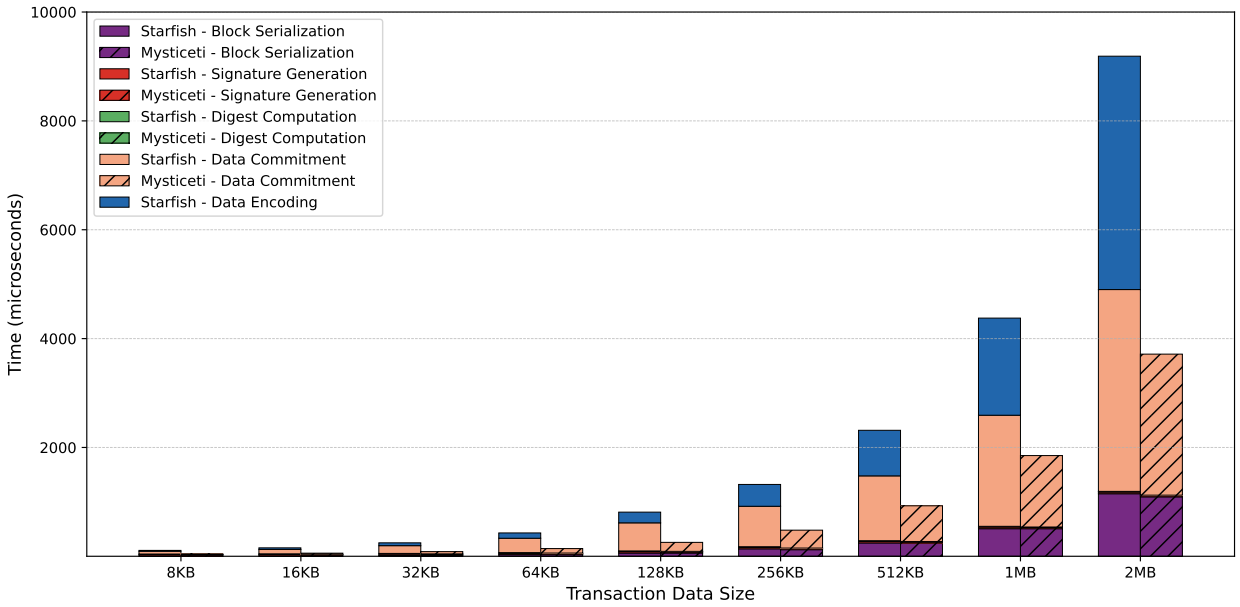

Figure 3: Timing breakdown for block creation in Mysticeti and Starfish for different transaction data sizes and 100 validators. Only operations with significant time are represented in the plots: encoding transaction data (converting transactions to encoded shards), computing data commitment (hashing and Merkelization of encoded shards in Starfish or hashing serialized transaction data in Mysticeti), digest computation, and signature generation.

peers is processed concurrently (not sequentially), verification of a block in Mysticeti could take (across all connections) time up to $O(Mn)$ as the same block could be processed by coming from different peers.

In Figure 3, we provide a timing breakdown for block creation for different transaction data sizes. We use 100 validators for all computations in this case. One can observe that encoding takes a significant portion of time in Starfish. In addition, computing data commitment for Starfish requires hashing more data and building a Merkle tree, compared to simple hashing of the actual transaction data in Mysticeti.

In Figure 4 and Figure 5, we present the key functions in the Starfish and Mysticeti protocols in two scaling regimes. **Network scaling**: the network size increases from 4 to 145 validators, with a fixed transaction data size of 512 KB. **Transaction data scaling**: the number of validators is set to 100 and the transaction data increases from 512B to 1MB. The plots includes timings for block creation (`CreateBlock`), block with full transaction data verification (`VerifyBlock`), block with one shard verification (`VerifyShard`), and block decoding from $f+1$ shards (`DecodeBlock`) for Starfish, and `CreateBlock` and `VerifyBlock` for Mysticeti, measured in microseconds. In the network scaling regime, these functions remain mostly constant (with a very slight increase to more hash references in blocks) for Mysticeti; Starfish requires more time for creating and verifying blocks, aligning with expected $O(M \log n)$ behaviour. Note that some jumps are due to library-related RS encoding and decoding optimizations. We also note that `VerifyShard` time decreases as committee size grows, due to smaller shard sizes with more validators. In the transaction data scaling regime, all functions for both Mysticeti and Starfish grow linearly (in M) as expected.

To put things in perspective, we consider one example: the target throughput 128.000 txs/sec with 100 validators. In this case, each validator creates (approximately) 10 blocks every second, i.e., each block contains on average 128 transactions, thereby having transaction data of size 64KB. Block creation takes 430 µs in Starfish vs 142 µs in Mysticeti. Block verification of the block with full transaction data takes 418 µs in Starfish vs 130 µs in Mysticeti. Decoding takes 560 µs and verification of one shard 64 µs, out of which Merkle proof verification takes 7 µs. It is possible for $f = 33$ Byzantine validators to distribute data only among $f + 1$ honest validators, forcing the remaining validators to decode transaction data. Decoding is

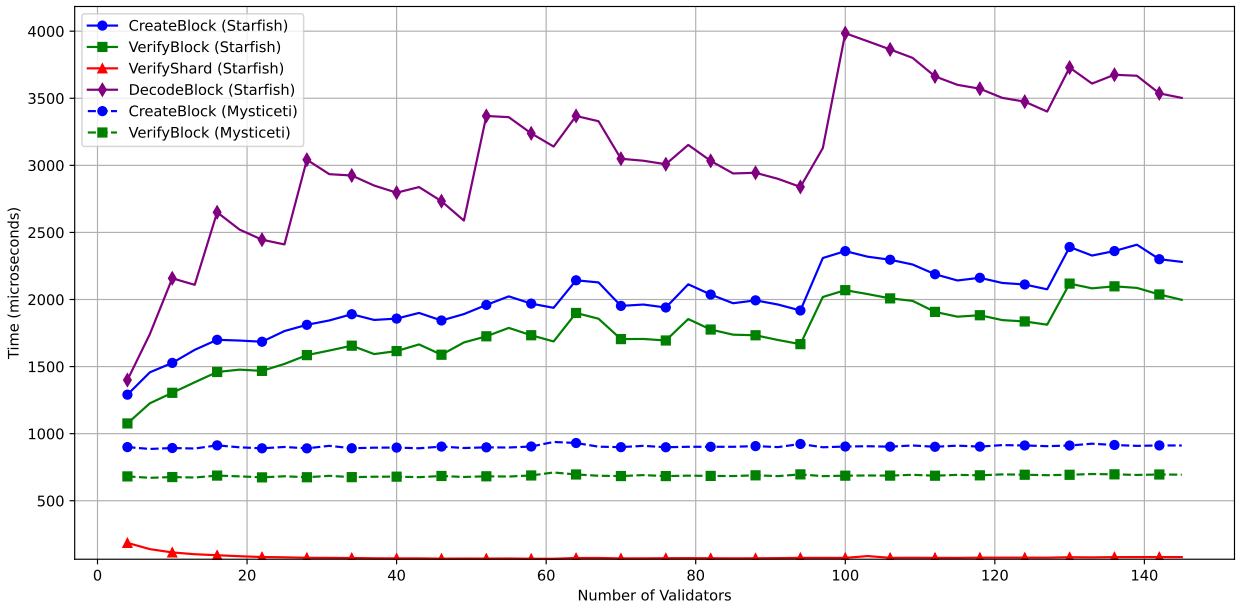

Figure 4: Performance of key functions in Starfish and Mysticeti across increasing committee sizes (4 to 145 validators) with a fixed transaction data size of 512KB. The plot shows `CreateBlock` and `VerifyBlock` for both protocols, and `VerifyShard` and `DecodeBlock` for Starfish only.

guaranteed to be correct since at least one honest validator had the original data and made verification of the data commitment. In the worst case, the decoding takes 560 (time to decode) $\times$ 33 (Byzantine validators) $\times$ 10 (blocks per second) = 184.4 ms, which can be handled by a single thread. Block with one shard verification has to be parallelized as it takes at most 64 µs (time to verify a block with a shard) $\times$ 100 (total number of shards) $\times$ 33 (Byzantine validators) $\times$ 10 (blocks per second) = 2.112 seconds.

In summary, Starfish's encoding-based approach incurs a computational overhead in steady-state operation, with a theoretical complexity increase of $O(\log n)$ for block creation and verification due to Reed-Solomon encoding, transaction data reconstruction due to Reed-Solomon decoding. Empirically, in typical blockchain scenarios, this overhead manifests as a multiplicative factor of 3-4. In worst-case adversarial scenarios, where Byzantine validators control network delays and concurrent block processing is employed, Starfish's computational complexity can be better than the one of Mysticeti, which may scale to $O(Mn)$ due to redundant block processing. This trade-off underscores Starfish's suitability for security-critical distributed systems, balancing higher CPU usage with robust performance under adversarial conditions.

7 Related Work

Byzantine Fault-Tolerant (BFT) consensus protocols have been extensively studied, e.g., [8, 22, 18, 19, 24, 11, 16, 13] with DAG-based protocols promising to improve throughput, scalability, and latency. In this section, we focus on the research most relevant to Starfish. In particular, we focus on round- and leader-based DAG protocols under the lens of certified vs. uncertified DAGs, data availability mechanisms, security considerations, and communication complexity and latency trade-offs. Round-based DAG-based protocols structure consensus by allowing validators to advance rounds only when a quorum of blocks from the current round is observed. A protocol can additionally be leader-based to optimize latency by introducing special commit rules for leader blocks, forming a "backbone sequence" that partitions the DAG into slices and enables deterministic sequencing.

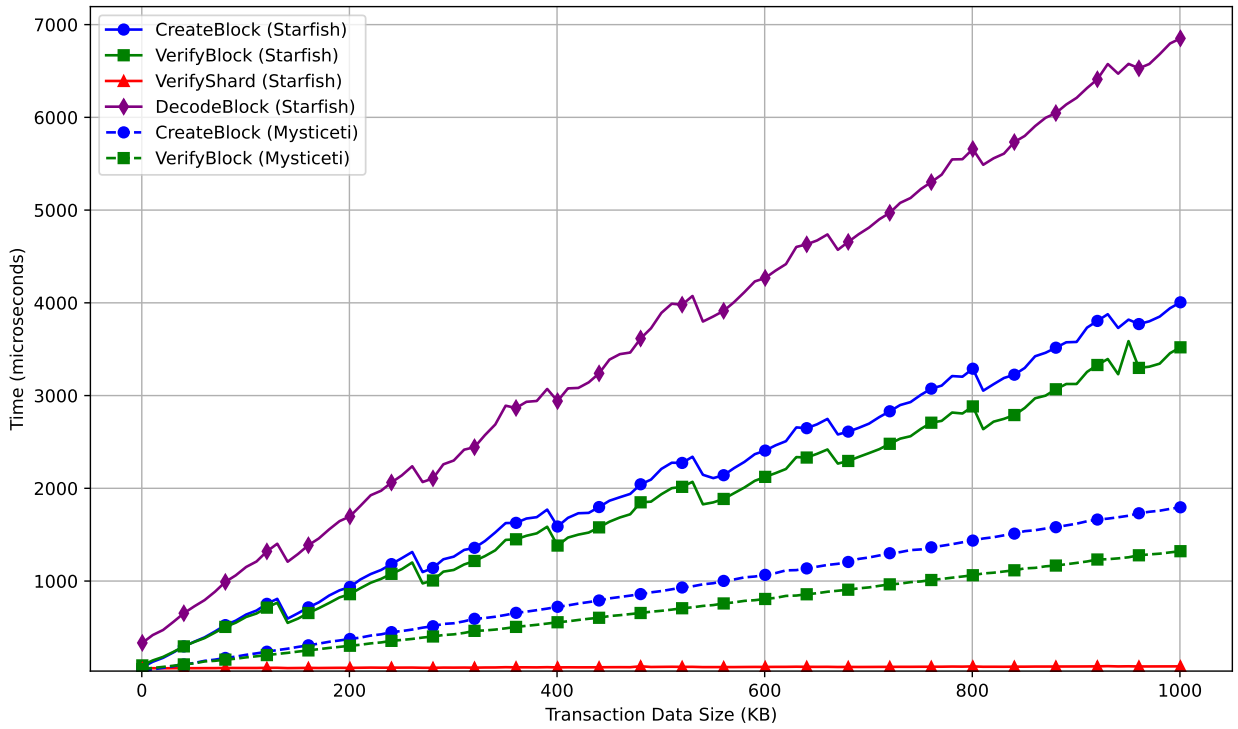

Figure 5: Timing performance of key functions in Starfish and Mysticeti across increasing transaction data sizes (0.5 KB to 1 MB) with a fixed committee size of 100 validators. The plot shows `CreateBlock` and `VerifyBlock` for both protocols, and `VerifyShard` and `DecodeBlock` for Starfish only, as Mysticeti does not involve sharding or decoding in this context.

Certified vs. uncertified DAGs. DAG-based BFT protocols such as Aleph [18], DAG-Rider [20], Tusk [11], Bullshark [35], Shoal [34], Shoal++ [2] Sailfish [31, 32] make use of Byzantine Reliable Broadcast (RBC) to guarantee data availability and to prevent equivocation. This ensures that validators construct a *certified DAG*, where every block is signed by a quorum, preventing conflicts but increasing communication overhead. Bullshark uses disjoint waves of two rounds and assigns one leader to each wave. By local interpretation of edges in the DAG, validators in Bullshark can commit the leader block and sequence all blocks in the causal past. Shoal [34] reduces the latency of non-leader blocks by interleaving two instances of Bullshark. BBCA-Chain [25] introduced a hybrid approach, applying RBC only to leader blocks while using Best-Effort Broadcast (BEB) for other blocks. However, when the Byzantine validators "selectively" send their blocks only to the leader, additional latency can be incurred to download the missing blocks, and the leader gets responsible for propagating a linear number of blocks. Sailfish [31] and Shoal++ [2] are two certified DAG-based protocols that similarly optimize their commit rules for leader blocks (by using first messages instead of certified blocks to count votes for leader blocks) and support multiple leaders in every round, improving the consensus latency. Sailfish additionally proposes using a no-vote certificate (an analogue of skip pattern in the uncertified DAG approach) to faster skip leader blocks from validators that fail to create their blocks in leader rounds and improve the latency in case of such misbehaviour. Shoal++, in turn, supports multiple instances of DAGs that improve end-to-end transaction latency. We note, however, that neither the multi-leader versions of these protocols nor the multi-DAG version of Shoal++ have proven to have performance improvements in the case of Byzantine nodes.

Hashgraph [4] pioneered the uncertified DAG-based approach, using Best-Effort Broadcast (BEB) for block dissemination, resulting in *uncertified* or *optimistic* DAGs where equivocating blocks may appear, and data availability has to be guaranteed by the consensus protocol. We note that all further uncertified DAG-based

protocols inherit similar mechanisms for resolving equivocations from Hashgraph: validators always have to reference their own last blocks, and they sequence only blocks certified by $2f + 1$ validators. Here, a block B *certifies* another block L in its causal past if B can reach blocks from $2f + 1$ validators such that each block V from these $2f + 1$ *votes* for L, i.e., L is reachable from V and V can't reach any equivocation of the validator that created L. Hashgraph sequences certified blocks in a leaderless way using received timestamps of the respected blocks. Cordial Miners [21] introduced disjoint 3-round waves, and one leader was assigned in the first round of each wave. By committing leader blocks and deciding on the sequence of committed leader blocks, validators agree on how to slice the DAG and order blocks. Such a leader-based commit rule significantly reduced the commit latency. Mysticeti [3] is built on top of Cordial Miners and suggests pipelining waves with leaders, further reducing the latency. It was also proposed to use multiple leaders every round; however, the multi-leader version does not guarantee any improvement in latency in the case of Byzantine validators and, instead, may lead to a larger latency due to the undecided state of Byzantine leader blocks. We refer to [28] for a more detailed description and comparison between certified and uncertified approaches.

Starfish builds on top of Cordial Miners and Mysticeti with just pipelined leaders. Compared to the prior work, we separated block creation and round advancement events, each triggering broadcast of the unknown history to other validators, and added a new block creation condition **C3**. This design allows us to formally prove the liveness property for an uncertified DAG-based approach.

Data Availability and Dissemination. Certified DAGs outsource the data-availability problem to the additionally used RBC primitive, as blocks need to be certified before entering the consensus protocol. In addition, implementation of many certified DAG-based protocol makes use of Narwhal [11], which separates transaction dissemination from consensus using quorum-based availability certificates. Blocks in certified DAGs, in this case, become lightweight since they contain only digests of certified transaction batches. However, this separate layer increases the resulting end-to-end latency due to additional network trips for forming certificates for transaction batches. To improve the latency, Raptr [36] introduces a hybrid approach that integrates transaction certification into the consensus, allowing nodes to vote on a prefix of (uncertified) transaction batches anchored by the proposed leader block. Autobahn [17] similarly separates transaction data dissemination, by allowing each validator to create its own transaction "lane", a chain of transaction batches, and other validators' votes can shortly acknowledge the availability of a prefix of the lane.

For the uncertified DAG approach, Cordial Miners [21] proposed the "cordial dissemination" where validators "push" possibly unknown history of the DAG to other validators. This approach has quadratic communication complexity in theory; furthermore, it reaches practical limitations due to high bandwidth usage. Mysticeti uses a "pull" strategy, where validators can request missing history from their peers. This reduces bandwidth usage but might increase latency in bad network conditions due to these "missing ancestors" requests.

Using a new block structure, Starfish decouples transaction data dissemination from the consensus task. In Starfish, the transaction data from blocks is included in the final ordering only after creating data availability certificates ($2f + 1$ acknowledgments) directly on the DAG. This transaction dissemination concept is different from Narwhal, Raptr, and Autobahn. Compared to Starfish, in Narwhal, validators vote on blocks that include already "well-disseminated" certified transaction batches - this increases the end-to-end latency. Both Autobahn and Raptr can operate with uncertified transaction batches. To sequence transactions, they require Proofs-of-Availability consisting of $f + 1$ acknowledgments, as it guarantees that at least one honest validator has transaction data locally available. Starfish requires $2f + 1$ acknowledgments primarily because of the usage of Reed-Solomon codes - f Byzantine validators could lie, and $k = f + 1$ encoded shards are required for reconstruction (having this parameter k smaller increases the redundancy of the code, resulting in higher bandwidth requirements). PoAs in Starfish rely on the block's signatures. Autobahn leaves the

task of constructing PoAs to the data dissemination layer. Both Starfish and Raptr integrate constructing PoAs into the consensus layer, but Starfish allows for a more granular transaction commitment. We note that a similar idea of encoding with RS codes could be applied to both Autobahn and Raptr - this would allow achieving linear amortized communication complexity for the transaction dissemination layer in the Byzantine environment.

Security and Adversarial Strategies. As pointed out in [28], uncertified DAGs are more vulnerable to Byzantine behaviour than certified DAGs, as the additional RBC prevents attacks focusing on equivocations and data availability.

Equivocating attack: Uncertified DAG-based protocols such as Cordial Miners, Mysticeti, and Starfish come with security proofs and their own method to handle equivocating blocks. Cordial Miners suggested how to avoid handling blocks from equivocators: once an equivocator is detected, an honest validator first references two equivocating blocks and then stops directly referencing blocks from it. Once it has learnt about the equivocation, any other honest validator stops directly referencing blocks from the equivocator. However, there could be malicious validators that do not create equivocation but reference equivocators, allowing multiple blocks from the same validator from the same round to exist in the resulting DAG. Fortunately, this kind of malicious behaviour can be handled similarly, and after GST, one can effectively exclude all equivocators from the consensus by processing $O_f(1)$ equivocating blocks. Nevertheless, equivocations can happen in practice by validators unintentionally, e.g., after crashing and restarting the consensus module, and such a strict punishing equivocators might be dangerous for the liveness of the protocol.

Data availability attack: The paper [18] discusses the chain-bomb attack, where spam forces honest validators to process excessive data, risking system overload.[26] It argues that DAG-based protocols require reliable broadcast to prevent such attacks, claiming that liveness is not guaranteed otherwise. In Adelie [9], such attack vectors are discussed in the context of Mysticeti and new validity rules for blocks were proposed. While this was shown to work in practice in the steady-state network, the suggested implicit validation of prior own blocks can impose serious liveness risks when no validator is able to create its block. In addition, the above papers present no quantitative results for actual attack scenarios. In this paper, we are the first to implement specific attack strategies and compare their influence on uncertified DAG-based BFT protocols.

Communication Complexity and Erasure Coding. The use of erasure and error-correcting codes in constructing Reliable Broadcast primitives is well established [12, 7]. Notably, Honeybadger BFT [27] applies erasure coding [7] to Bracha's RBC [5], reducing its communication complexity to achieve linear amortized communication complexity. Similarly, DAG-Rider [20] and Dumbo-NG [15] leverage erasure codes to optimize RBC communication costs. Certified DAG-based BFT protocols rely on an additional RBC layer that can be implemented using the RBC from [12], leading to a linear amortized communication complexity. These approaches, however, treat erasure coding as a separate optimization within the RBC protocol and do not integrate it directly into the DAG structure as we do in Starfish.

Our usage of RS codes is similar to [7] for RBC. While using RS codes tolerating *errors* (not *erasures*) in [12] allows getting rid of extra $\log n$ factor in the RBC communication complexity by avoiding the usage of Merkle trees, we stick with the erasure decoder for RS codes. First, the error correcting decoder does not help to improve the requirement for the message size in Starfish to achieve linear amortized complexity. Second, the existing error-correcting decoders have a larger constant factor in front of $n \log n$ in decoding complexity.

[26] This attack is referred to as the "Fork-Bomb Attack" in [18].

8 Conclusion and Outlook

We have presented Starfish, a novel uncertified DAG-based BFT consensus protocol that relies on a new commit rule for transaction data and an efficient data availability mechanism. Starfish achieves linear communication complexity in the worst case for large enough transaction data while maintaining low end-to-end transaction latency. Encoded Cordial Dissemination ensures that push-based broadcast remains scalable, addressing concerns about the inefficiency of naive push-based approaches. Our performance evaluation demonstrates that Starfish outperforms state-of-the-art certified DAG protocols and performs better than existing uncertified DAG protocols under adversarial conditions.

Implicit certification of own blocks. While Starfish allows for pushing the unknown history from the theoretical point of view, it can practically meet situations when a history to be pushed is over the bandwidth and cannot be received in full after Δ. An interesting research direction to further strengthen uncertified DAG protocols is to introduce implicit certification on a chain of blocks of a given validator. Specifically, each validator could be required to implicitly certify one of its own previous blocks when creating a new block. This constraint would naturally limit block flooding attacks; see a similar attempt in [9]. Such a mechanism could potentially bridge the security-performance gap between certified and uncertified DAGs without compromising the linear communication complexity. The theoretical analysis of this approach, particularly its impact on liveness, remains an open question.

Unlink transaction data from block header. It is clear that if transaction data is disseminated without binding its content to a block (using the Merkle root in the case of Starfish), it could reduce the end-to-end latency. Indeed, in this way, validators can make their acknowledgments faster since blocks in Starfish are created only after triggering one of the events **C1**, **C2**, or **C3**. To overcome this issue, one could either allow creating blocks more frequently, i.e., a chain of blocks by one validator in one round, or allow disseminating transaction data without associating it with any block. The first approach is similar to a multi-DAG version of Shoal++ [2]. The second approach can be implemented in a similar fashion as it is done in Raptr [36] or Autobahn [17] with proofs-of-availability being created directly on the DAG.

A Detailed Description of Starfish Protocol

In this section, we provide a detailed description of Starfish. Pseudocodes of the core algorithms used for block creation, round advancement, commit rules, and sequencing can be found in Appendix B.

History of Known Block Headers. We denote $\mathsf{hist}(B)$ as the causal *history* (sometimes called the *past cone*) of a block B, representing the set of all block headers that are transitively reachable from B via its hash references (in Ancestors). Every validator may have a different perception of the current DAG. Let us denote by DAG_i the local DAG of validator v_i. Due to network latency, a validator does not necessarily know the local DAGs of the other validators. However, every block expresses knowledge about the existence of other blocks in its causal history. Therefore, we can define the history of known block headers known to a validator v_j from the point of view of validator v_i as follows:

$$\mathsf{hist}_i(j) = H_{i,j} \cup P_{i,j}, \tag{1}$$

where $H_{i,j} = \bigcup \mathsf{hist}(B)$ is the union over all blocks B in DAG_i created by validator v_j and $P_{i,j}$ is all blocks that are sent (or pushed) from v_i to v_j. The latter is added to the union $H_{i,j}$ since v_i already pushed the unknown history even though some of the pushed blocks might not have yet reached v_j.

A.1 Identifying DAG Patterns

Proposer Slot. We use the concept of *proposer slot* as in Mysticeti. A proposer slot represents a tuple (validator, round) and can be either empty or contain the validator's proposal(s) for the respective round. Starfish uses a round-robin mechanism to assign proposer slots to validators. Validator v_i has its proposer

slot in rounds $i, i+n, i+2n \ldots$.

State of Proposer Slots. Every proposer slot is in one of the following states: to-commit, to-skip, or undecided. Initially, all slots are set to be undecided. The goal of the commit rule of Starfish is to ensure that each validator, by interpretation of local DAG, makes a decision to-commit or to-skip for every proposer slot.

The to-commit state allows to commit the leader block in a proposer slot. The crucial state is the undecided, which forces all subsequent proposer slots to wait, mitigating the risk of non-deterministic commitments due to network asynchrony without the need for a buffer round as prior work [20, 11, 35, 15]. Finally, the to-skip state allows to exclude proposer slots that will not get enough votes or certificates to be skipped.

Proposing, Voting and Certifying Round. If a proposer slot at round r is fixed, we call round r *proposing*, round $r+1$ *voting*, and round $r+2$ *certifying*. The latter two are used to make the decision for the proposer slot.

Voting for a Leader Block. Recall that each block header from validator v_i created at round $r+1$ contains a set Ancestors. This set includes hash references to blocks from $2f+1$ validators created at round r. Each block from voting round $r+1$ can vote on a leader from the proposer round r. A block B from round $r+1$ is *voting* on a leader block L from round r if L is the first block in Ancestors of B from the leader of the proposer slot of round r.

Patterns for Leader Blocks. Starfish, as its predecessors, operates by interpreting the structure of the DAG and finding patterns that create certificates or skip the attempt to find certificates for leader blocks:

1. The *skip pattern*: blocks from at least $2f+1$ validators at round $r+1$ *do not* vote for a leader block from round r. Note that there might be no proposal for a leader slot or more than one proposal for a leader slot in the case of equivocations. The slot is skipped if, for any block proposal, we observe $2f+1$ blocks in the next round that do not vote for it.
2. The *certificate pattern*: blocks from at least $2f+1$ validators at round $r+1$ *vote* for a leader block B of round r. We then say that B is *certified*. Any block from certifying round $r+2$ that contains in its history such a pattern is called a *quorum certificate* (QC) for the block B.

Quorum Certificates for Leader Blocks. Using these patterns, we derive quorum certificates implicitly for leader blocks by interpreting the DAG structure. Certification ensures that at most one leader block from a given proposer slot can be certified. This prevents conflicts caused by equivocation, as conflicting blocks cannot simultaneously meet the $2f+1$ quorum requirement.

Pattern for Data Availability. In Starfish, we decouple block headers from their transaction data. This means that unlike Cordial Miners and Mysticeti, we can't immediately sequence all transactions by *slicing* the DAG using the committed leader blocks. One has to check the data availability of the transaction data within the slices by inspecting Acknowledgment fields of blocks in the slices.

Similar to the *certificate pattern* above, we introduce

3. The *acknowledgment pattern*: blocks from at least $2f+1$ validators at rounds $> r$ *acknowledge the availability of the transaction data of a block* B from round r.

We say that B is *certified for data availability*. In contrast to the certificate pattern, we do not require the acknowledgment pattern to appear in one specific round $r+1$ but allow it to appear at any later point after round r.

Data Availability Certificates for Blocks. The *Data Availability Certificate (DAC)* ensures that the transaction data $\mathsf{tx}(B)$ associated with a block header B is available to a quorum of validators. Any subsequent

block that contains in its causal history such an acknowledgment pattern is called a *Data Availability Certificate (DAC)* for the block B.

A.2 Sequence of Committed Leader Blocks

This section describes the mechanism to find a sequence of committed leader blocks; it is the same as in Mysticeti.

All proposer slots are initially in the undecided state. The goal is to mark all proposer slots as either to-commit or to-skip by detecting certificate or skip patterns or relying on an *indirect decision rule*. The whole decision rule operates in three steps:

Step 1: Direct Decision Rule. The validator applies the following *direct decision rule* to attempt to determine the status of the proposer slot.

The validator marks a proposer slot as to-commit if it observes QCs from at least $2f + 1$ validators for that slot. In case of equivocation and multiple proposals for one slot, the proposal or block header that received a required quorum of QC is committed.

The direct decision rule marks a proposer slot as to-skip if it observes a *skip pattern* for that slot.

If the direct decision rule fails to mark a slot as either to-commit or to-skip, the slot remains undecided and the validator resorts to the *indirect decision rule* presented in step 2 below.

Step 2: Indirect Decision Rule. If the direct decision rule leaves a slot undecided, the validator resorts to the indirect decision rule to attempt to decide. This rule operates in two stages. It initially searches for an *anchor*, which is defined as the first proposer slot with the round number ($r' \geq r + 3$) that is already marked as either undecided or to-commit.

If the anchor is marked as undecided the validator marks the slot as undecided. Conversely, if the anchor is marked as to-commit, the validator marks the slot either as to-commit if the anchor contains a path to a QC for a leader block from the proposer slot or as to-skip otherwise.

Step 3: Commit Sequence. After processing all proposer slots, the validator derives an ordered sequence of leader blocks. Subsequently, the validator iterates over that sequence of proposer slots, committing all slots marked as to-commit and skipping all slots marked as to-skip until it reaches the first slot marked as undecided.

We will prove, in Lemma 12, that, eventually, every proposer slot will be decided.

A.3 Sequencing Transaction Data

In Starfish, we need to specify how the actual transactions are ordered. For each block B let us denote $\mathsf{hist}_{DA}(B)$ the set of all blocks D in $\mathsf{hist}(B)$ such that B is a Data Availability Certificate for $\mathsf{tx}(D)$ in $\mathsf{hist}(B)$. Given the sequence of committed leader blocks $C_1, \ldots, C_N$, derived in Appendix A.2, and the fixed topological ordering of block headers in the DAG, this allows to construct a total ordering over the sequenced transaction data in $\mathsf{hist}_{DA}(C_N)$ as follows

$$\mathsf{sort}(\mathsf{hist}_{DA}(C_1)), \mathsf{sort}(\mathsf{hist}_{DA}(C_2) \setminus \mathsf{hist}_{DA}(C_1)), \ldots, \mathsf{sort}(\mathsf{hist}_{DA}(C_N) \setminus \bigcup_{i}^{N-1} \mathsf{hist}_{DA}(C_i)). \quad (2)$$

This gives a total order of the blocks with availability guarantees. One has to remove potential equivocating blocks from this ordering by preferring the first block of a given validator from a given round. By removing equivocation, we guarantee the integrity property of Starfish as BAB and establish a total ordering of the transactions contained in the remaining blocks.

A.4 Encoded Cordial Dissemination

Compared to push-based cordial dissemination, Cordial Miners [21], we propose modifying the broadcast mechanism and introducing a more communication-efficient method. Our approach consists of two steps aimed at reducing communication overhead. First, we decouple the broadcast of the "block header" from the dissemination of transaction data. Second, we employ an erasure code to broadcast encoded parts of the transaction data for blocks of others' validators.

Erasure Code and Merkle Tree Construction. For the transaction data $\mathsf{tx}(B)$ of a given block B, we use a systematic Reed-Solomon code of length $n = 3f + 1$, dimension $k = f + 1$, and alphabet $\mathbb{F}_q$ of size q, which is a power of two. We also make use of Merkle trees to ensure integrity verification for encoded transaction data.

1. *Encoding the Transaction Data:* The transaction data $\mathsf{tx}(B)$ is encoded into $n = 3f + 1$ shards using the Reed-Solomon encoding function:

$$c = (c_1, c_2, \ldots, c_n) = \text{encRS}(\mathsf{tx}(B), n = 3f + 1, f + 1).$$

   The first $f + 1$ shards, $c_1, c_2, \ldots, c_{f+1}$, are called information shards. They represent the transaction data; the other pieces are parity shards. The original transaction data can be recovered from any $f + 1$ shards.

   More precisely, by setting $k = f + 1$, the transaction data is represented as

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,k} \\ \ldots & \ldots & \ldots & \ldots \\ x_{t,1} & x_{t,2} & \ldots & x_{t,k} \end{pmatrix}$$

   where each $x_{i,j} \in \mathbb{F}_q$ and $t = \frac{M}{(f+1)\log_2 q}$ with M being the size (in bits) of $\mathsf{tx}(B)$[27]. Every row is treated as an information sequence and encoded into a length-n sequence of elements from $\mathbb{F}_q$ with the Reed-Solomon code

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,n} \\ \ldots & \ldots & \ldots & \ldots \\ x_{t,1} & x_{t,2} & \ldots & x_{t,n} \end{pmatrix}.$$

   Every column is called a *shard*, the first k are information shards, which represent the transaction data, the last $n - k$ are parity shards. In other words, k shards $c_1, \ldots, c_k$ were encoded into n shards $c_1, \ldots, c_n$ by computing parity shards $c_{k+1}, \ldots, c_n$.

2. *Creating the Merkle Tree:* After encoding the data, we compute the hash of every shard:

$$h_i = \text{hash}(c_i) \quad \text{for} \quad i = 1, \ldots, n.$$

   These hashes are used to construct a Merkle tree, where the leaf nodes are the hashes of the shards. The root of the Merkle tree, h, is computed as:

$$h = \text{MerkleRoot}(h_1, h_2, \ldots, h_n).$$

   The Merkle root h acts as a commitment to the entire set of encoded shards and is used by other validators to verify the integrity of the data.

3. *Merkle proofs:* To validate the integrity of an individual shard c_i, the ith validator provides a Merkle

[27] In case, M is not divisble by $(f + 1)\log_2 q$ we use the ceiling operation and zero padding

proof. A Merkle proof

$$p_i = \text{MerkleProof}(c_i)$$

is a sequence of sibling hashes from the Merkle tree that allows a validator to reconstruct the Merkle root h using only the shard c_i and its corresponding proof p_i.

Broadcasting Unknown History. Whenever the broadcast history event is triggered, see Table 3, a validator v_i broadcasts to v_j a batch of blocks. This batch contains all blocks that are currently in DAG_i but not in $\mathsf{hist}_i(j)$, see (1). In addition, let A_i denote the set of all blocks whose transaction data is available to v_i. The batch also includes all blocks that were not in A_i when the previous unknown history event was triggered

Recall the structure of blocks, see Section 2. Each block consists of a block header and a block body, where the block header

$$B_{\text{header}} = \langle \text{Round}, \text{Ancestors}, \text{Acknowledgments}, \text{MerkleRoot} \rangle_{v_i}$$

is signed. The block body optionally contains the transaction data and an encoded shard of the transaction data, along with Merkle proof for verification. A full block is not signed and defined as:

$$B = \langle B_{\text{header}}, \text{Option(TransactionData)}, \text{Option(EncodedShard, MerkleProof)} \rangle.$$

Specifically, all block headers in the set difference of DAG_i and $P_i(j)$ are sent in a batch. The block bodies of the corresponding blocks are formed depending on whether the block was created by v_i or not.

Block Body for Own Blocks. Once the block creator has completed the encoding and Merkle tree construction, the Merkle root is added to the block header, which is then signed. The original transaction data is added to the block body. The block creator broadcasts this block within the batch.

Block Body of Blocks from Other Validators. There are two options for the block body depending on whether a block was received from the block creator or not:

Opt 1: If the original transaction data of block B is not yet available to validator v_i, then v_i does not include anything in the block body and v_i broadcasts just the block header of B. In this case, B was not in the history of known block headers $\mathsf{hist}_i(j))$ before this transmission, see (1).

Opt 2: If validator v_i receives or reconstructs the full transaction data of a block B, then it performs the encoding and constructing the Merkle root described above to calculate the tuple (c_i, p_i), where

- c_i is the ith encoded shard derived from the transaction data.
- p_i is the Merkle proof that allows any receiving validator to verify the integrity of c_i.

This tuple is then added to the block body for block B and sent to all validators that have not yet acknowledged the availability of $\mathsf{tx}(B)$.[28] In this case, B was included to A_i after the previous unknown history event was triggered.

Transaction data verification. The transaction data $\mathsf{tx}(B)$ for a block B can be reconstructed or verified by validators in two distinct scenarios:

1. *Verification of Transaction Data Broadcasted by the Block Creator*:
   - The block creator directly broadcasts the original transaction data $\mathsf{tx(B)}$ to the validator.

[28] We note, that, it is possible that one validator v_i first broadcasts to v_j the block header of block B and after a while the block B including in its block body the encoded shard. In this case block B became a part of $\mathsf{hist}_i(j)$ when the first broadcast was called. Nevertheless, validators *always* are required to send their encoded shards once the transaction data becomes available to them.

- Upon receiving tx(B), the validator encodes transaction data, computes the Merkle root h from the encoded data, and compares it to the Merkle root included in the block header.
- If the computed root matches the one in the block header, the integrity of the transaction data is verified. The validator will participate in the cordial dissemination and include acknowledgments of availability for tx(B) in the next block headers.

2. *Reconstruction from Encoded Shards*:
   - A block sent by a block creator is not reached by a validator; instead, the validator collects encoded shards. In this case the validator collects $f + 1$ encoded shards c_i, each accompanied by its Merkle proof p_i.
   - For each shard c_i, the validator uses the proof p_i to verify its integrity against the Merkle root h from the block header of B. Only valid fragments are retained for reconstruction.
   - Once $f + 1$ valid shards $d_1, \ldots, d_{f+1}$ are collected and arranged to the required positions, the codeword of the RS code is reconstructed using the decoding function:

$$\hat{c} = (\hat{c}_1, \ldots, \hat{c}_n) = \text{decRS}((d_1, d_2, \ldots, d_{f+1}), n, f + 1).$$

   - The validator will also participate in the cordial dissemination of that block (even though the block header could have already been broadcasted) by including the corresponding encoded shard. The validator includes the acknowledgment of availability for tx(B) in the next block header.

Benefits of Merkle-Based Verification. Using Merkle-based verification allows us to deal with erasures instead of errors. We note that [12] made improvements over [7] in the communication complexity of the constructed RBC primitives by using RS *error* decoder instead of RS *erasure* decoder. We note that this idea would not improve the communication complexity for Starfish due to its push-based dissemination strategy. In addition, we stick with the erasure decoder due to its more computationally efficient implementation.

B Algorithms

This section presents a selection of more detailed algorithms for Starfish. Let B be a block with the header

$$B_{\text{header}} = \langle \text{Round}, \text{Ancestors}, \text{Acknowledgments}, \text{MerkleRoot} \rangle_{v_i}.$$

For simplicity of notation, we will write in algorithm descriptions $B.author$ for validator index i, $B.round$ for round number Round, $B.ancestors$ for hash references of the parents Ancestors, $B.acknowledgments$ for hash references of transaction data availability acknowledgments Acknowledgments.

Starfish uses several global variables shown at the beginning of Algorithm 1. This algorithm illustrates the overview and core mechanics of the Starfish protocol, detailing how validators execute rounds, create blocks, advance rounds and participate in encoded cordial dissemination. Conditions of these steps are also given in Table 3. Such an overview was not given in Mysticeti [3]. We also provide algorithms for consensus helper functions and algorithms for commit rule and sequencing transactions in Algorithm 2 and Algorithm 3. For better comparability with prior work, we highlight important changes in procedures and functions. When a function is similar to its counterpart in Mysticeti, the differences are marked in blue, while entirely Starfish-specific functions and procedures are highlighted in green.

Algorithm 1: Starfish Execution for Validator v_i

Global variables and constants:
- $\delta_{TO} \leftarrow 2\Delta$ // Timeout duration; Δ is (known) upper bound for network delay after GST
- $n \leftarrow 3f + 1$ // Number of validators; f is maximum number of Byzantine validators
- $r_{decided} \leftarrow 0$ // Latest round of block in the sequence of decided leaders
- $r_{highest} \leftarrow 0$ // Highest round of block in local DAG
- $r_{last} \leftarrow 0$ // Round of latest own block
- DAG $\leftarrow$ GENESISBLOCKS() // Local DAG of v_i initialized with predefined blocks of round 0

procedure EXECUTESTARFISH() ◇ *Execute Starfish protocol*
- **for** $r = 1, 2, \ldots$ **do**
  - EXECUTEROUND(r)

procedure EXECUTEROUND(r) ◇ *Execute given round in Starfish*
- Step 1:
- BROADCASTUNKNOWNHISTORY() // Encoded Cordial Dissemination, Appendix A.4
- $\tau_{enter}(r) \leftarrow$ LOCALTIME() // Local time of validator v_i
- Step 2:
- **while** TRYCREATEBLOCK($r, \tau_{enter}(r)$) $=$`false` **do**
  - RECEIVEBLOCKSFROMVALIDATORS() // Updates DAG, $r_{highest}$
  - TRYDECIDE($r_{decided}, r_{highest}$) // Try commit leader blocks and sequence transactions
- Step 3:
- $r_{last} \leftarrow r$
- BROADCASTUNKNOWNHISTORY() // Encoded Cordial Dissemination, Appendix A.4
- Step 4:
- **while** TRYADVANCETONEXTROUND(r)=`false` **do**
  - RECEIVEBLOCKSFROMVALIDATORS() // Updates DAG, $r_{highest}$
  - TRYDECIDE($r_{decided}, r_{highest}$) // Try commit leader blocks and sequence transactions

function TRYADVANCETONEXTROUND(r) ◇ *Check conditions **A1** and **A2** in Table 3 to enter next round*
- **if** NUMBERVALIDATORWITHBLOCKSINROUND(r) $\geq 2f + 1$ ***and*** $r_{last} = r$ **then**
  - **return** `true`
- **return** `false`

function TRYCREATEBLOCK($r, \tau_{enter}(r)$) ◇ *Check conditions **C1**, **C2** or **C3** in Table 3 to create new block*
- **if** LEADERCONDITIONSMET(r) ***or*** RECEIVEDBLOCKS(r) $\geq 2f + 1$ ***or*** LOCALTIME() $\geq \tau_{enter}(r) + \delta_{TO}$ **then**
  - CREATEBLOCK(r)
  - **return** `true`
- **return** `false`

function LEADERCONDITIONSMET(r) ◇ *Check for given round condition **C1** in Table 3*
- L1 $\leftarrow$ "leader block of round r is received" // Boolean variable true or false
- L2 $\leftarrow$ "received $2f + 1$ blocks voting for leader of round $r - 1$" **or** "exists skipping pattern for leader of round $r - 1$"
- **return** L1 & L2

C Safety of Starfish

Starfish was largely inspired by prior uncertified DAG-based BFTs such as Cordial Miners and Mysticeti. This section contains proofs for totality, agreement, and total order, 3 out of 4 properties of BAB, see Section 2. These proofs are based on similar considerations as in the prior work.

Lemma 1. *If at a round k, $2f + 1$ blocks from distinct validators certify a leader block B from round $k - 2$,*

Algorithm 2: Helper Functions for Consensus

function GETPREDEFINEDLEADER(r) ◇ *Return index of leader validator in proposer slot with given round*
  return $(r \bmod n) + 1$

function ISDIRECTLYSKIPPEDSLOT(r) ◇ *Return* `true` *if leader slot is directly skipped*
  $leader \leftarrow$ GETPREDEFINEDLEADER(r)
  $Votes \leftarrow$ DAG$[r+1]$ // All blocks in voting round
  $Proposals \leftarrow \{B \in \text{DAG}[r] \text{ s.t. } B.author = leader\}$ // All blocks by leader in proposer slot
  for $L \in Proposals$ **do**
    $res \leftarrow |\{B.author : B \in Votes \text{ s.t. ISVOTE}(B, L) = \texttt{false}\}|$
    if $res < 2f + 1$ **then**
      return `false` // Found a block by proposer with not at least $2f+1$ of blame
  if $|Proposals| = 0$ & $|\{B.author : B \in Votes\}| < 2f + 1$ **then**
    return `false` // All blocks by the proposer have enough blame
  return `true`

function DIRECTLYCOMMITTEDLEADERBLOCK(r) ◇ *Return directly committed leader block if any*
  $leader \leftarrow$ GETPREDEFINEDLEADER(r)
  $Proposals \leftarrow \{B \in \text{DAG}[r] \text{ s.t. } B.author = leader\}$ // All blocks by leader in proposer slot
  $Certificates \leftarrow$ DAG$[r+2]$ // All blocks in certifying round
  for $L \in Proposals$ **do**
    $res \leftarrow |\{B.author : B \in Certificates \text{ s.t. ISCERT}(B, L) = \texttt{true}\}|$
    if $res \geq 2f + 1$ **then**
      return L
  return UNDECIDED(r)

function ISVOTE(B_{vote}, B_{leader}) ◇ *Return* `true` *if one block votes for another*
  RoundCondition $\leftarrow$ "$B_{vote}.round = B_{leader}.round + 1$"
  VoteCondition $\leftarrow$ "B_{leader} is a first hash reference of $B_{leader}.author$ in $B_{vote}.ancestors$"
  return RoundCondition & VoteCondition

function ISCERT(B_{cert}, B_{leader}) ◇ *Return* `true` *if one block is a certificate for another*
  $Ancestors \leftarrow \{B \in \text{DAG s.t. HASH}(B) \in B_{cert}.ancestors\}$
  $res \leftarrow |\{B.author : B \in Ancestors \text{ s.t. ISVOTE}(B, B_{leader}) = \texttt{true}\}|$
  return $res \geq 2f + 1$

function ISLINK(B_{old}, B_{new}) ◇ *Return* `true` *if one block is reachable from another*
  return $\exists$ sequence of blocks $B_1, \ldots, B_k \in$ DAG, $k \in \mathbb{N}$, s.t.
    $B_1 = B_{old}, B_k = B_{new}$ and
    $\forall j \in [2, k] : \text{HASH}(B_{j-1}) \in B_j.ancestors$

function GETREACHABLEBLOCKS(L) ◇ *Return all blocks reachable from leader block*
  return $\{B \in \text{DAG s.t. ISLINK}(B, L) = \texttt{true}\}$

function ISCERTIFIEDLINK(B_{anchor}, B_{leader}) ◇ *Return true if block can reach certificate for another block*
  $r \leftarrow B_{leader}.round$
  $Certificates \leftarrow$ DAG$[r+2]$ // All blocks in certifying round
  return $\exists B \in Certificates$ s.t. ISCERT$(B, B_{leader}) =$ `true` & ISLINK$(B, B_{anchor}) =$ `true`

function ISACK(B_{ack}, B) ◇ *Return* `true` *if block acks a tx data of past block*
  return HASH$(B) \in B_{ack}.acknowledgments$

Algorithm 3: Committer of Leader Blocks and Sequencer for Transaction Data

```
1  procedure TryDecide(r_decided, r_highest)        ◇ Update sequence of commit/skip decisions for leaders
2      sequence ← [ ]
3      for r ∈ [r_highest − 1 down to r_decided + 1] do
4          status ← TryDirectDecide(r)
5          if status = Undecided(r) then
6              status ← TryIndirectDecide(r, sequence)
7          sequence ← status||sequence
8      for status ∈ sequence in ascending round-order do
9          if status = Undecided(r) then
10             break
11         if status = Commit(L) then
12             SequenceTransactionsFromDAC(L)
13         r_decided ← r_decided + 1                                   // Update last decided round

14 function TryDirectDecide(r)                      ◇ Return direct commit/skip decision or undecided state
15     if IsDirectlySkippedSlot(r) = true then
16         return Skip(r)
17     L ← DirectlyCommittedLeader(r)
18     if L ≠ Undecided(r) then
19         return Commit(L)
20     return Undecided(r)

21 function TryIndirectDecide(r, sequence)          ◇ Return indirect commit/skip decision or undecided state
22     Anchors ← {s ∈ sequence s.t. s.round ≥ r + 3}
23     for A ∈ Anchors in ascending round-order do
24         if A = Undecided(r) then
25             return Undecided(r)
26         if A = Commit(B_anchor) then
27             leader ← GetPredefinedLeader(r)
28             Proposals ← {B ∈ DAG[r] s.t. B.author = leader}        // All blocks by leader in proposer slot
29             for L ∈ Proposals do
30                 if IsCertifiedLink(B_anchor, L) then
31                     return Commit(L)
32             return Skip(r)
33     return Undecided(r)

34 procedure SequenceTransactionsFromDAC(L)          ◇ Sequence tx data for blocks with L being their DAC
35     Reachable ← GetReachableBlocks(L)
36     for B ∈ Reachable do
37         if IsDataAvailCert(L, B) then
38             Sequence(B)                                             // Sequence transaction data of block B if not sequenced already

39 function IsDataAvailCert(B_DAC, B_check)         ◇ Return true if one block is DAC for another
40     Reachable ← GetReachableBlocks(B_DAC)
41     res ← |{B.author : B ∈ Reachable s.t. IsAck(B, B_check) = true}|   // Number of unique acknowledgments
42     return res ≥ 2f + 1
```

then all blocks at future rounds $> k$ will have a path to a certificate for B from round k.

Proof. Consider any block B' in round $r > k$. There exists one honest validator v such that v is both a creator of the round-k certificate for B and a creator of the block in the set Ancestors of B'. Since validators by the Starfish protocol are required to reference their own previous blocks, B' has a path, through the chain of blocks of validator v, to the certificate for B. □

The proof for the following lemma is different from the one provided in [3] since we use a more general condition for direct skipping of a proposer slot, see Algorithm 2.

Lemma 2. *If an honest validator commits a leader block of round r, then no honest validator decides to directly skip the proposer slot of round r.*

Proof. If an honest validator v directly or indirectly commits a leader block L of round r, then there exists a set W_L of $2f + 1$ validators that vote in round $r + 1$ for L. The set W_L includes $f + 1$ honest validators. Toward a contradiction assume that another validator v' directly skipped the proposer slot of round r. That means at the time of skipping, v' had a local DAG such that for each (if any) leader block L' of round r there were $2f + 1$ non-voters for L'. We consider two cases:
1. There was at least one leader block L' in the local DAG of v'. This means within the $2f + 1$ validators, that non-vote for L', there was at least one honest validator from W_L. Therefore, $L' \neq L$. Moreover, the local DAG of v' has to contain additionally L because of this honest non-voter. This means that there should be $2f + 1$ non-voters for L which is not possible due to quorum intersection.
2. There was no leader block of round r in the local DAG of v' and the local DAG contained blocks from $2f + 1$ validators in the voting round $r + 1$. However, one of these $2f + 1$ validators is an honest validator from W_L, implying that L was a part of the local DAG. This leads to a contradiction with the assumption of point 2. □

Lemma 3. *If an honest validator directly commits a leader block of round r, then no honest validator will decide to skip the proposer slot of round r.*

Proof. For the sake of contradiction, assume that an honest validator v directly commits a leader block B of round r, and an honest validator v' skips the proposer slot of round r. By Lemma 2 v' can not skip directly. It remains to consider the case when the validator v' skips indirectly.

Let B' be a block which was an anchor, when v' decided to indirectly skip the proposer slot from round r. Then the block B' is from round $r' > r + 2$. The validator v directly committed B, therefore $2f + 1$ blocks from different validators certify B in round $r + 2$. By Lemma 1 there is a path from B' to certificate for B, which means that B shouldn't have been skipped. □

Lemma 4. *For any proposer slot, at most, a single block will ever be certified, i.e. gather a quorum $(2f+1)$ of voting blocks.*

Proof. For contradiction's sake, assume that two block proposals for a slot gather a quorum of votes. The intersection of these quorums contains at least one honest validator that voted for two blocks for one slot, which is a contradiction. □

Corollary 1. *No two honest validators commit distinct leader blocks for the same slot.*

Lemma 5. *All honest validators have a consistent state for each proposer slot, i.e., if two validators have decided the state of the slot, then both either commit the same block or skip the slot.*

Proof. Assume that at some moment validators v_1 and v_2 have inconsistent states. Pick a slot from the latest round r, such that one validator commits the leader block in the corresponding proposer slot, and the other skips this slot. Without loss of generality, v_1 commits the leader block, and v_2 skips it. By Lemmas 2 and 3 both validators made their decisions indirectly. Let r_1 and r_2 be the rounds, from which the decisions were made by v_1 and v_2 correspondingly. Then the validator v_1 skips all slots in rounds $[r+3, r_1)$ and commits in round r_1, and v_2 skips all slots in rounds $[r+3, r_2)$ and commits in the round r_2. Since r was assumed to be the last round with different decisions for v_1 and v_2, we conclude that $r_1 = r_2$. The decision to indirectly commit or skip the leader block in round r depends only on the causal history of a committed leader block in round $r_1 = r_2$, but this committed leader block have to be the same for both v_1 and v_2 by Corollary 1. This leads to a contradiction. □

Corollary 2. *All honest validators have a consistent sequence of committed leader blocks, i.e. a sequence of committed leader blocks for one honest validator is a prefix of another or other way round.*

The following theorem is adopted for Starfish since to sequence transaction data validators use committed leader blocks and find for which blocks in the causal past they serve as DACs.

Theorem 2 (Totality, Agreement and Total Order)**.** *If one honest validator delivers the transaction data of block B then all other honest validators will also deliver the transaction data of block B. Moreover, transactions are executed in the same order for all honest validators.*

Proof. The decision to sequence the transaction data of block B after committing the leader block L depends on whether the causal history of block L contains acknowledgments from $2f + 1$ validators. This history of a given block is the same for all validators. By Corollary 2, the sequence of committed leader blocks is the same for all honest validators; thereby, the order of the transaction data will be also the same. In other words, all honest validators will deliver transactions in the same order.

We note that each validator delivers at most one block from a given round from a given validator since we remove the equivocating blocks, see Appendix A.3. □

D Liveness of Starfish

In this section, we prove a liveness property, which completes the proof that Starfish is a BAB. In Starfish, we distinguish two liveness properties. The first concerns the liveness of the sequence of leader blocks, ensuring that leader blocks proposed by honest validators are committed after GST. This is formalized in the first results, where progress in consensus relies solely on block headers. The second property, the liveness of transaction data, Theorem 4, is induced by the liveness of committed leader blocks together with the guarantees provided by the DACs.

Let $r_{\max}$ be the largest round among honest validators at the moment of GST. The first property that we prove is about synchronization of honest validators after GST, which was not shown in prior work on uncertified DAGs.

Lemma 6. *All honest validators enter any round $r > r_{\max}$ within a time interval of length Δ and create their blocks in round r within a time interval of length Δ.*

Proof. Consider round r higher than any round achieved by honest validators at the moment of GST. Consider the first honest validator v_1 that enters round r. Say that it happened at the moment t_1. By condition **B2**, this validator sends the blocks it knows to all other validators. Then at the moment $t_1 + \Delta$ all other validators have enough blocks to advance to the next round from any round $r' < r$, due to **A2**. Condition **A1** is fulfilled since for any round $r' < r$ they create their blocks since the condition **C3** is satisfied. Indeed, for any round $r' < r$ the validator v_1 has at least $2f + 1$ blocks, and for any other validator v these blocks

(except those which were already known to v from v_1's point of view) have been sent to v at the moment t_1. So, at the moment $t_1 + \Delta$ every honest validator has created and disseminated its blocks for all rounds until $r-1$ and advanced to round r. In other words, all honest validators will enter round r in time interval $[t_1, t_1 + \Delta]$.

Let v_2 be the first honest validator that creates a block B in round r and it happens at the moment t_2 (it could be that $v_2 \neq v_1$). The creation of this block could not have been triggered by condition **C3**, since v_2 is the first among honest validators and could have obtained at most f round-r blocks from Byzantine validators. If the creation of the block was triggered by timeout condition **C2**, then all other validators will also create their blocks during the time interval $[t_2, t_2 + \Delta]$. Indeed, the difference between their entry (to the round r) times is at most Δ, and the difference between the moments when their timeouts expire is also at most Δ. If some honest validator creates its block before its timeout expires, then this moment is still in the time interval $[t_2, t_2 + \Delta]$, since v_2 is the first one by definition. If the creation of the block was triggered by condition **C1**, then not later than at moment $t_2 + \Delta$ all other validators will receive the block from v_2 together with the unknown history. They will be able to create their own blocks since the condition **C1** will be satisfied for them. □

Lemma 7. *For any $r > r_{\max}$ all blocks of honest validators from round $r-1$ will be delivered to every honest validator v before the round-r timeout of v expires.*

Proof. If GST happens at the moment t_{GST}, and the last block of honest validator from round $r-1$ was created at moment t_{last}, then at the moment $t_{deliver} = \max(t_{GST}, t_{last}) + \Delta$ all blocks of honest validators from round $r-1$ will be delivered to every validator v. Let t_v be a moment when the validator v enters round r. Since $r > r_{max}$, this happens after GST, so $t_v \geq t_{GST}$. Also by Lemma 6, $t_v \geq t_{last} - \Delta$. Recall that the timeout $\delta_{TO} = 2\Delta$, see Table 3. The two latter inequalities imply $t_v + \delta_{TO} \geq \max(t_{GST}, t_{last}) + \Delta = t_{deliver}$. Hence, all blocks of honest validators from round $r-1$ will be delivered to every honest validator v before the round-r timeout of v expires. □

Lemma 8. *Suppose that for some round r, each honest validator has received before its r-round timeout expires the same leader block from round $r-1$. Then all round-r blocks created by honest validators reference the same leader block from round $r-1$.*

Proof. If the round-r block creation for some honest validator v was triggered by the condition **C1** then the created block of round r by v references the leader block of round $r-1$. If the block creation is instead triggered by the timeout condition **C2**, the validator is assumed to have already received the leader block before the timeout and will reference it.

It remains to explain the case when condition **C3** triggered the block creation for some validator v. For this case, we consider $f+1$ honest validators, which are the fastest to create a block in round r. Denote the set of these $f+1$ honest validators by W. For these validators the block creation couldn't have been triggered by condition **C3** since there are not enough round-r blocks for it. Therefore, their block creation was triggered by **C1** or **C2** and have referenced the leader block by the above arguments. In the case that the block creation for validator v was triggered by **C3**, i.e., v received $2f+1$ round-r blocks. There is at least one block B from W among them, and this block references the leader of the previous round. The validator v received the round-$(r-1)$ leader block (or block header) together with B and will reference it in its block. □

Lemmas 7 and 8 imply the following corollary.

Corollary 3. *Suppose an honest validator occupies the proposer slot of round $r-1$ such that $r > r_{\max}$. Then all round-r blocks of honest validators reference the leader block of honest validator from round $r-1$.*

Using similar arguments as in the proof of Lemma 8, one can prove the following statements about voting blocks.

Lemma 9. *Suppose that for some round r, each honest validator has received before its r-round timeout expires $2f + 1$ blocks voting for a leader block of round $r - 2$. Then all round-r blocks created by honest validators reference the $2f + 1$ blocks voting for the leader block of round $r - 2$.*

Proof. If the round-r block creation for some honest validator v was triggered by the condition **C1** then the block references the $2f+1$ blocks voting for the leader block of round $r-2$ by definition. If it was triggered by the timeout condition **C2** then validator v has, by assumption, already received $2f + 1$ blocks voting for the leader block of round $r - 2$ and will reference them.

It remains to explain the case when condition **C3** triggered the block creation. In this case, we consider $f+1$ honest validators, which are the fastest to create a block in round r. The block creation couldn't have been triggered by condition **C3** since there are not enough round-r blocks for it. Therefore, they have referenced $2f + 1$ blocks voting for the leader block of round $r - 2$ by the above arguments. Denote the set of these $f+1$ honest validators by W. The block creation for validator v was triggered by **C3**, i.e., v received $2f+1$ round r blocks. There is at least one block B from W among them, and this block references $2f + 1$ blocks voting for the leader block of round $r - 2$. The validator v received them together with B and will reference them in its block. □

Corollary 3 and Lemmas 7 and 9 imply the following corollary:

Corollary 4. *Suppose an honest validator occupies the proposer slot of round $r-2$ such that $r > r_{\max} + 1$. Then all round-r blocks of honest validators will be certificates for the leader block of the honest validator from round $r - 2$.*

Lemma 10. *Any leader block created by an honest validator in round $r \geq r_{\max}$ will be marked to-commit by the direct decision rule.*

Proof. Consider a leader block created by an honest validator v in round $r \geq r_{\max}$. By Corollary 4 all honest validators will certify this leader block in round $r + 2$. These $\geq 2f + 1$ certificates will be obtained by every validator, and every honest validator will mark the leader block of round r to-commit by the direct decision rule. □

The next two lemmas are analogues of Lemmas 11 and 12 in Mysticeti [3] with minor adjustments. We provide the proofs for completeness.

Lemma 11. *The round-robin leader schedule ensures that in any $n + 2 = 3f + 3$ consecutive rounds, there are three consecutive rounds in which their leaders are honest.*

Proof. There are $3f + 1$ groups of three consecutive rounds. Due to the round-robin schedule, each of the honest validators must be a leader in exactly 3 of these groups. As there are at least $2f+1$ honest validators, due to the pigeonhole principle, one group must contain $\lceil \frac{3(2f+1)}{3f+1} \rceil = 3$ honest leaders. □

Lemma 12. *After GST any undecided leader block will be decided, i.e., marked as to-commit or to-skip.*

Proof. Consider an arbitrary undecided leader block from round r. By Lemma 11 there will be 3 consecutive rounds k, $k + 1$, $k + 2$ with honest leaders in the corresponding proposer slots for some k such that $\max(r, r_{\max}) \leq k \leq \max(r, r_{\max}) + n - 1$. The leader blocks in these 3 rounds will be marked to-commit by Lemma 10. Let us iterate over all undecided leader blocks before round k in the order of decreasing rounds and apply the indirect decision rule. For each undecided block, the corresponding anchor

will be marked to-commit, so by the indirect decision rule the undecided block will be marked to-commit or to-skip. □

Lemmas 10 and 12 imply the liveness of the consensus.

Theorem 3. *Any leader block created by an honest validator in round $r \geq r_{\max}$ will be committed.*

Lemma 13. *Transaction data of any block created by an honest validator will eventually get a data availability certificate that will be committed.*

Proof. A block B created by an honest validator at t_1 will be received by every honest validator at latest after $\max(t_{GST}, t_1) + \Delta$. Each honest validator will acknowledge the data availability of the transaction data in a next block. Let t_2 be a moment when a slowest honest validator includes the acknowledgment. Every leader block, created after $t_2 + \Delta$ will be a DAC for the transaction data of B. By Lemmas 10 and 12, such a leader block will be eventually decided and committed; thereby, the transaction data of block B will be sequenced at latest after deciding such a leader block. □

Lemma 14. *If the transaction data of a block has a data availability certificate in the view of one honest validator, then the transaction data of that block will be eventually available by every honest validator.*

Proof. Since validators in Starfish employ the push-based dissemination strategy, every other honest validator will recognize the same DAC in time Δ after the next broadcast unknown history event is triggered. At least $f + 1$ honest validator, contributing to the DAC, will share their encoded shards at the time when they make acknowledgments. The parameters of the $(n, f + 1)$ Reed-Solomon code allow for the successful reconstruction of the original transaction data. □

Theorem 4 (Validity)**.** *Transaction data of every block by an honest validator will be eventually sequenced.*

Proof. Consider a block B created by an honest validator v. By Lemma 13, this block will get a DAC L that is decided and committed. When attempting to sequence the transaction data for this block, one needs to check data availability for all blocks that got DACs in the sequence of committed leader blocks ended by L. By Lemma 14, the transaction data for all such blocks will be eventually available by every honest validator. □

D.1 Latency after GST

Now we discuss the situation when the actual transmission latency after GST is upper bounded by $\delta < \Delta$, where the upper limit Δ is known to the validators, and δ is unknown. At first we note that in this case Lemma 6 holds with upper limit δ instead of Δ. We rewrite it with the corresponding change.

Lemma 15. *All honest validators enter any round $r > r_{\max}$ within a time interval of length δ and create their blocks in round r within a time interval of length δ.*

Lemma 16. *Assume that the leaders of rounds $r - 1$, r, $r + 1$ are honest, and $r > r_{\max}$. Then round-r leader block will be marked to-commit by all honest validators not later than 4δ after the moment of the creation of this leader block.*

In case when all validators are honest and $r > r_{\max} + 2$ the round-r leader block will be committed not later than after 4δ.

Proof. Suppose a leader block was created by an honest validator at the moment t in round r. By Lemma 15 all other honest validators will create their round-r blocks not later than at $t + \delta$. At the moment $t + 2\delta$ all round-r blocks created by honest validators will be obtained by everyone. Every honest validator will be

able to enter round $r+1$ at the moment $t+2\delta$. Every honest validator will create its block in round $r+1$ at the moment $t+2\delta$ or earlier. Indeed, at the moment $t+2\delta$ every honest validator will have received the round-r leader block and all other round-r blocks from honest validators. Since the leader of round $r-1$ is honest by Corollary 3 every round-r block from honest validator will reference the leader of the round $r-1$. Hence, the condition **C1** will be satisfied at $t+2\delta$, and every honest validator will create its block in round $r+1$ at the moment $t+2\delta$ or earlier.

All blocks from round $r+1$ created by honest validators will be obtained by everyone at $t+3\delta$ or earlier. So, at the moment $t+3\delta$ or earlier all honest validators will enter the round $r+2$. Since the leader of the round $r+1$ is honest, the leader block of round $r+1$ will be received at $t+3\delta$ or earlier. All honest blocks from round $r+1$ will reference the leader block from round r by Corollary 3. So, at the moment $t+3\delta$ the condition **C1** in round $r+2$ will be satisfied for all honest validators, so they create their blocks at $t+3\delta$ or earlier. By Corollary 4 these blocks will certify the leader of the round r.

Every honest validator will receive at least $2f+1$ certificates at the moment $t+4\delta$ or earlier and mark the round-r leader block to-commit.

The leader block can be marked to-commit but not committed if some of the previous leader blocks are undecided. If all validators are honest, then from Lemma 10, we conclude that the leaders of round $r-2$, $r-1$, r will be marked to-commit. Similarly to Lemma 12 all previous leaders will be decided after that, and round-r leader will be committed. □

Lemmas 17 to 20 serve as useful tools for estimating latencies in various scenarios.

Lemma 17. *After GST, the time interval between the creation of blocks by an honest validator v in round r and $r+1$ is*

1. *at most 2δ if the leaders of rounds r and $r-1$ are honest.*
2. *at most $2\delta+2\Delta$ otherwise.*

Proof. If a block in round r was created at the moment t, then all other honest validators will also create their blocks not later than at $t+\delta$ by Lemma 15. Then, at the moment $t+2\delta$ or earlier, the validator v will receive round-r blocks from all honest validators. It is enough to enter round $r+1$. Then, at the moment $2\delta+2\Delta$, the timeout will expire, and the validator v will create a round-$(r+1)$ block if it was not created before.

If the leaders of round r and $r-1$ are honest, then condition **C1** is satisfied at the moment $t+2\delta$ by Corollary 3, and the block will be created at moment $t+2\delta$ or earlier. □

Lemma 18. *After GST, the time that an honest validator v spends in round r is*

1. *at most 3δ if the leaders of rounds $r-1$ and $r-2$ are honest.*
2. *at most $2\delta+2\Delta$ otherwise.*

Proof. If an honest validator v enters round r at moment t, then all other honest validators broadcast their blocks from round $r-1$ and enter round r not later than at $t+\delta$ by Lemma 15. The validator v and all other validators obtain all blocks from honest validators not later than at $t+2\delta$. If leaders of round $r-1$ and $r-2$ are honest, then condition **C1** is satisfied, and all honest validators, including v, create their blocks at $t+2\delta$ or earlier. All validators receive round-r blocks from honest validators not later than at $t+3\delta$, and can advance to the next round.

If at least one of the leaders of rounds $r-1$ and $r-2$ is Byzantine, then the validator v may need to wait until its timeout expires. Anyway, at the moment $t+2\Delta$ or earlier, the validator v create and broadcast their round-r blocks. Applying Lemma 15, other validators will do the same at latest at $t+2\Delta+\delta$

Therefore, at moment $t+2\Delta+2\delta$ or earlier, all honest validators receive round-r blocks from all honest validators and advance to round $r+1$.

□

The following statement is key for establishing bounds for the latency for a long enough range of rounds, e.g. average and worst-cast end-to-end latencies with 0 or f Byzantine nodes. It says that the DAG progresses in rounds for honest validators at pace at least one round per time δ when there is no Byzantine validators in the schedule. Any Byzantine validator in the leader schedule can affect the DAG contruction by additional 2Δ or even 4Δ depending whether it creates its block in the respected round.

Lemma 19. *Let $t_{last,r}$ denote the moment when the last honestly created block[29] of round r was created. Then after GST, $t_{last,r+1}-t_{last,r}$ is upper bounded by*

1. *at most δ if the leaders of rounds r and $r-1$ are honest.*
2. *at most $\delta+2\Delta$ if the leader of round r or $r-1$ is Byzantine.*
3. *at most δ if the leader of round $r-1$ is Byzantine and the leader of round r is honest, and the leader of round $r-1$ fails to create its block in round $r-1$.*

Proof. If the last round-r block from an honest validator was created at the moment t then at the moment $t+\delta$ all honest validators will obtain all round-r blocks from honest validators. It is enough to enter round $r+1$. If leaders of rounds r and $r-1$ are honest then the condition **C1** is satisfied, and they can create and broadcast their blocks from round $r+1$.

If at least one of the leaders of round r and $r-1$ is Byzantine, then they may need to wait their timeouts. In that case all honest validators create their blocks from round $r+1$ not later than at $t+\delta+2\Delta$.

If the leader of round $r-1$ is Byzantine and *failed* to create its block and leader of round r is honest, then round-r blocks from honest validators create a skip pattern for the proposer round $r-1$ and the condition **C1** is satisfied at $t+\delta$. □

Lemma 20. *If all validators are honest, then after GST, the time interval between the creation of two (consecutive) leaders' blocks is at most 2δ.*

Proof. If the leader block of round r was created at the moment t, then at the moment $t+\delta$ all other blocks from round r are created, and at moment $t+2\delta$ they are received by every validator. Then the leader block for round $r+1$ will be created not later than at $t+2\delta$ since the condition **C1** is satisfied. □

Theorem 5 (Worst-case latency, all honest validators)**.** *Suppose all validators are honest. After GST, the worst-case consensus latency is at most 9δ. The worst-case end-to-end latency is at most 11δ.*

Proof. Assume that a given transaction was included in a block B created at the moment t. Then, at the moment $t+\delta$, all validators will receive the block B. By Lemma 17, at the moment, not later than $t+3\delta$, all validators will create their blocks with acknowledgement for the block B. At the moment, not later than $t+4\delta$, all validators will have in their local DAGs $\geq 2f+1$ acknowledgements for the block B. The next leader block L with $\geq 2f+1$ acknowledgements for B in its history will be created in some round, say r.

[29] Block created by a slowest honest validator

By Lemma 17, all blocks from round r will be created not later than at $t + 6\delta$. By Lemma 19 the block of a slowest validator from round $r + 1$ and $r + 2$ will be created not later than at $t + 7\delta$ and $t + 8\delta$. All honestly created blocks from round $r + 2$ will certify the block L, and these blocks will be obtained by all validators not later than at $t + 9\delta$. So, the leader block L will be committed not later than at $t + 9\delta$, and the transaction data of block B will be sequenced.

To compute end-to-end latency, we must add the time the transaction waits to be included in the block, which is at most 2δ by Lemma 17. □

Theorem 6 (Average latency, all honest validators)**.** *Suppose all validators are honest. After GST, the average consensus latency is at most* 7δ*. The average end-to-end latency is at most* 7.5δ*.*

Proof. We divide the end-to-end latency into three subintervals I1, I2 and I3. I1 is the time interval from when a transaction gets available to a validator until it gets included in a block by the validator. I2 is the time interval for a block B from the time of its creation till the time creation of last round-r block such that all validators at or before round r acknowledged the availability of the transaction data of block B. The last interval I3 is from the end of I2 till a leader block serving as a DAC is committed by every honest validator.

By Lemma 19, the slowest validator creates blocks with a delay between these blocks at most δ. Since transactions are arriving continuously in time, the average time for interval I1 can be upper bounded by 0.5δ.

Let $t_{first,r}$ denote the moment when a first round-r block is created or the time creation of the block of a (round-r) fastest honest validator (since all validators are assumed to be honest in this statement). All blocks that are created at or before $t_{first,r} - \delta$ are guaranteed to receive a quorum of acknowledgments from round-r blocks. Moreover, by Lemma 15, the slowest validator creates round-r block at moment $\leq t_{first,r} + \delta$.

Now we are ready to upper bound the duration of I2 on average. By Lemma 19, one can show that the average value of $t_{first,r} - t_{first,r-1}$ is bounded by δ. Therefore, the duration of I2 is on average at most $\delta + (t_{first,r} + \delta) - (t_{first,r} - \delta) = 3\delta$.

Regarding I3, round-$(r + 1)$ leader block L is guaranteed to be a DAC for the respected transaction data as all validators at or before round r made acknowledgment. All honestly created blocks at round $r + 3$ will certify L. Thus, applying Lemma 19, I3 is bounded by the progress of slowest blocks in the DAG from round r to round $r + 3$ and the delivery of round-$(r + 3)$ blocks, i.e. 4δ.

To sum up, the average end-to-end latency is bounded by 7.5δ. □

Theorem 7 (Worst-case latency, f Byzantine validators)**.** *If* $f < \frac{n}{3}$ *validators are Byzantine, then after GST, the worst case end-to-end latency of any transaction sent to an honest validator is upper bounded by* $4f\Delta + n\delta + O(\Delta)$*. If Byzantine validators fail to create their blocks in rounds when they are leaders in the their proposer slots, then the latency can be estimated as* $2f\Delta + n\delta + O(\Delta)$

Proof. Assume that a given transaction is sent to an honest validator v at the moment t. We divide time interval until this transaction is sequenced by every validator into three subintervals I1, I2, I3. I1 is the time until there is a quorum of acknowledgments for this transaction in the local DAG of every honest validator. I2 is the time from the end of I1 until there are three consecutive leaders from honest validators. Finally, I3 is the time until the third leader in the sequence of the three consecutive leaders is committed ensuring that the transaction is sequenced.

First estimate the duration of I1. By Lemma 17, at moment $\leq t+2\delta+2\Delta$ the validator v creates the block B which includes the transaction. At moment $\leq t+3\delta+2\Delta$ all honest validators receive the block B. Applying again Lemma 17, at moment $\leq t+5\delta+4\Delta$ all honest validators create blocks with acknowledgement for B.

At the moment $\leq t+6\delta+4\Delta$ all honest validators have in their local DAGs a quorum of acknowledgements for the transaction data of block B. Any honest leader committed after this moment will sequence the block B.

Second we bound the duration of I2. By Lemma 11 there will be 3 consecutive honest leaders in $n + O(1)$ rounds, which will be marked to-commit by Lemma 10. After this all previous leaders that were undecided will be marked to-commit or to-skip. The time interval until three consecutive honest leaders is $n\delta + 4f\Delta + O(\Delta)$ by Lemma 19. Indeed, two honest leaders in the scheduler allow the slowest honest validator to create blocks with delay between them at most δ; one Byzantine leader among a window of two rounds can increase this to $\delta + 2\Delta$; if Byzantine leader does not create its block it can affect the latency in only one round, resulting in $\delta + 2\Delta$.

Finally it remains to bound the duration of I3. It takes up to $O(\Delta)$ to commit one leader among the three consecutive honest leaders.

Therefore, the total time to sequence the transaction can be bounded by $n\delta + 4f\Delta + O(\Delta)$ in the worst case if f Byzantine nodes are active and $n\delta + 2f\Delta + O(\Delta)$ if they fail to produce their blocks when they are leaders. □

Theorem 8 (Worst-case latency, 1 Byzantine validator)**.** *Suppose the number of validators is at least 6. If one validator is Byzantine, then after GST the worst-case end-to-end latency of a transaction sent to the honest validator is at most* $14\delta + 4\Delta$*. If the Byzantine validator fails to create its block, the latency is bounded by* $14\delta + 2\Delta$*.*

Proof. Assume a given transaction was received by honest validator v at moment t_1. At the moment t_1 the last block created by v was from round r. The validator v creates its block B for round $r+1$ which includes the given transaction at the moment t_2. Applying Lemma 17, if the Byzantine validator was a leader in proposer rounds r or $r-1$ then $t_2 \leq t_1 + 2\delta + 2\Delta$, otherwise $t_2 \leq t_1 + 2\delta$.

At the moment $t_3 = t_2 + \delta$ all honest validators receive the block B and add it to their local DAGs. All blocks created by honest validators that contain acknowledgement for B are from round $\geq r + 2$. Let $\hat{r}$ be a minimal round, such that all honest validators created blocks with acknowledgement for B in round $\hat{r}$ or earlier. Note that $\hat{r} \geq r + 2$. Consider a validator $\hat{v}$ that created its block with acknowledgement for B in round $\hat{r}$. At the moment t_3 it already has created the block for round $\hat{r} - 1$. By Lemma 15 all honest validators created their blocks in round $\hat{r} - 1$ at $t_3 + \delta$. Denote the moment of creation of last round $\hat{r}$ block by honest validator as t_4. By Lemma 19 $t_4 \leq t_3 + 2\delta$ if the leaders of rounds $\hat{r} - 1$ and $\hat{r} - 2$ are honest; if at least one of them is Byzantine, then $t_4 \leq t_3 + 2\delta + 2\Delta$.

At the moment $t_5 = t_4 + \delta$ all validators will receive from every honest validator a block with acknowledgement for the block B. Therefore, any committed leader block created by an honest validator after t_5 will be a DAC for block B. Say that the last leader block created before t_5 was from round $r' \geq \hat{r}$. Then by Lemma 15 the slowest honest validator will create his block in round r' at $t_6 \leq t_5 + \delta$.

We consider different cases depending on whether the Byzantine node is a leader at rounds $r' - 1$, r', $r' + 1$ or $r' + 2$:

1. The Byzantine node is not a leader at rounds $r' - 1$, r' or $r' + 1$, $r' + 2$. In this case, the blocks of honest validators from round $r' + 3$ will be certificates for the honest leader block of round $r' + 1$. Moreover, by arguments from Lemma 12, marking the three constitutive leader blocks as TO-COMMIT results in the sequencing the transaction data of block B. By Lemma 19, the slowest honest validator will create its block at round $r' + 3$ at moment not later than $t_6 + 3\delta$. After time δ, all honest validators will receive all honestly created round-$(r' + 3)$ blocks. All together the resulting end-to-end latency is at most $11\delta + 4\Delta$.

2. The Byzantine node is a leader at round $r' - 1$. We note that in this case waiting for the transaction to be included in block B takes at most 2δ (but not $2\delta + 2\Delta$) since $r' \geq \hat{r} \geq r + 2$. In the worst case, transactions from B will be sequenced when all honestly created leader blocks from round r', $r' + 1$ and $r' + 2$ are marked as TO-COMMIT. This will happen after all honestly created blocks from round $r' + 4$ are delivered. By Lemma 19, the slowest honest validator will create its block at round $r' + 4$ at moment not later than $t_6 + 4\delta + 2\Delta$. After time δ, all honest validators will receive all honestly created round-$(r' + 4)$ blocks. All together the resulting end-to-end latency is at most $12\delta + 4\Delta$.

3. The Byzantine node is a leader at round r'. In this case $t_6 - t_1 \leq 7\delta$ since $r' \geq \hat{r} \geq r + 2$. In the worst case, transactions from B will be sequenced when all honestly created leader blocks from round $r' + 1$, $r' + 2$ and $r' + 3$ are marked as TO-COMMIT. This will happen after all honestly created blocks from round $r' + 5$ are delivered. By Lemma 19, the slowest honest validator will create its block at round $r' + 5$ at moment not later than $t_6 + 5\delta + 4\Delta$. After time δ, all honest validators will receive all honestly created round-$(r' + 5)$ blocks. All together the resulting end-to-end latency is at most $13\delta + 4\Delta$.

4. The Byzantine node is a leader at round $r' + 1$. In this case $t_6 - t_1 \leq 7\delta$ since $r' \geq \hat{r} \geq r + 2$. In the worst case, transactions from B will be sequenced when all honestly created leader blocks from round $r' + 2$, $r' + 3$ and $r' + 4$ are marked as TO-COMMIT. This will happen after all honestly created blocks from round $r' + 6$ are delivered. By Lemma 19, the slowest honest validator will create its block at round $r' + 6$ at moment not later than $t_6 + 6\delta + 4\Delta$. After time δ, all honest validators will receive all honestly created round-$(r' + 6)$ blocks. All together the resulting end-to-end latency is at most $14\delta + 4\Delta$.

5. The Byzantine node is a leader at round $r' + 2$. In this case $t_6 - t_1 \leq 7\delta$ since $r' \geq \hat{r} \geq r + 2$. In the worst case, transactions from B will be sequenced when all honestly created leader blocks from round $r' - 1$, r' and $r' + 1$ are marked as TO-COMMIT. This will happen after all honestly created blocks from round $r' + 3$ are delivered. By Lemma 19, the slowest honest validator will create its block at round $r' + 3$ at moment not later than $t_6 + 3\delta + 2\Delta$. After time δ, all honest validators will receive all honestly created round-$(r' + 3)$ blocks. All together, the resulting end-to-end latency is at most $11\delta + 2\Delta$.

If the Byzantine validator fails to create its own blocks, then the timeout will affect the resulting latency at most once. □

E Discussion of Table 1

The average block and transaction latencies are computed under the following *Assumption*:

- All validators are honest[30].
- All messages are delivered within time interval δ.
- Any block of round r references all blocks from round $r - 1$.
- All validators are perfectly synchronized, i.e., they enter the round r at the same time.
- Transactions are coming to validators continuously at a uniform rate.

Recall that we include in this comparison certified DAG-based protocols such as Bullshark [35], Shoal [34], Shoal++ [2], Sailfish [31] and uncertified DAG-based protocols such as Cordial Miners [21], Mysticeti [3] and Starfish.

We provide the results for one leader per round, even though some protocols from the table allow to have multiple leaders per round. For example, Mysticeti protocol can support n leaders per round and under the above assumption this could improve the latency. However, without such strict and unrealistic assumption

[30]For the last column of the table, we consider one faulty validator

we didn't find quantitative results confirming that this could improve the latency in the Byzantine environment. On a practical side, the up-to-date production Mysticeti code [31] uses one leader per round.

We start the analysis with Cordial Miners. Note that the latencies are different for the leader blocks and non-leader blocks. In addition, the latencies are different for non-leader blocks with different round offsets. Specifically, the block latency for the leader block from round r is 3δ, for blocks from round $r-1$ it is 4δ, for blocks from round $r-2$ it is 5δ, for non-leader blocks from round $r-3$ it is 6δ. Average block latency is

$$\frac{3\delta + 4\delta \cdot n + 5\delta \cdot n + 6\delta \cdot (n-1)}{3n} = 5\delta - \frac{\delta}{n}.$$

We omit the terms of order $O(n^{-1})$ and write 5δ. Since blocks are created at pace one block at time δ and the transactions are coming continuously, we add 0.5δ for the average time for a transaction to be included in a block. For one leader failure, one needs to add 6Δ for the latency as 2Δ is the timeout of a validator before the creation of a block and one such leader at round r triggers waiting the timeout of all honest validators at rounds $r+1$, $r+2$ and $r+3$.

Now we analyse Mysticeti with pipelined leaders and a single leader at round. Unlike Cordial Miners, the block latency for non-leader block is 4δ. Therefore, the average block latency is $\frac{3\delta+4\delta(n-1)}{n} = 4\delta - \delta/n$. Similarly, 0.5δ is added to the transaction latency. For one leader failure, we added 4Δ while the Mysticeti paper does not explicitly explain when the rounds are advanced and blocks are created. Nevetherless, it was mentioned that validators at round r are waiting for the leader block of round $r-1$ and votes of the leader of round $r-2$, thereby one failure can trigger waiting the timeout twice. We update the condition **L2** with including the skipped pattern[32] for leader of round $r-2$, which allows validators in Starfish to wait the timeout only at one round.

Let us turn to analysing the latency for Starfish. Unlike Mysticeti, one needs to form a DAC for transaction data of block B in a committed leader block in order to sequence transaction data of B. This requires one more round, resulting in 5δ block latency and 5.5δ transaction latency. Now assume that the leader of round r is a failure and the leaders of the next rounds are honest. Consider the non-leader block B of the honest validator from round r. Block B will be acknowledged in round $r+1$, leader block L of the round $r+2$ will see these acknowledgments in its history, block L will receive votes in round $r+3$, certificates in round $r+4$, and then it will be marked to-commit. The faulty leader will be marked to-skip, since it will not receive any votes. So the time from the creation of the block B until the execution of its transactions is 5δ plus the time which was spent by honest validators on waiting. In the round $r+1$ honest validators waited for their timeout 2Δ, but in all other rounds, they did not need to wait. In round $r+2$ they see a skip pattern for the faulty leader, in other rounds the block creation depends only on the blocks from honest validators.

The values for Bullshark, Shoal and Sailfish are taken from the Sailfish paper [31], where a similar analysis was performed and leader and non-leader block latencies are computed. For Bullshark, a leader block is committed in time 8δ if one uses RS-based RBC from [12]. Non-leader blocks can incur extra 4δ or 8δ latency depending on the round offset, resulting in average $\frac{8\delta+12\delta n+16\delta(n-1)}{2n} = 14\delta + O(1/n)$. Since blocks are created in Bullshark with such RBC at pace one block per 4δ, it results in extra 2δ delay for the transaction latency. In Shoal, non-leader blocks can incur extra 4δ, which results in block latency 12δ and transaction latency 14δ. Depending on whether communication efficient RBC is used, Sailfish requires 3δ (5δ) to commit leader blocks and extra 2δ (4δ) for non-leader blocks. This results in average block latency 5δ (9δ) and average transaction latency 6δ (11δ).

Shoal++ achieves similar results as Sailfish, except two things. It was suggested to use multiple DAGs in parallel to improve the transaction latency (reducing waiting time for a transaction until included in a block).

[31] `https://github.com/MystenLabs/sui/releases`, release: MAINNET-V1.42.2

[32] A similar skipped pattern was proposed in Mysticeti, but was not integrated to the block creation logic

While k can be infinitely large, Shoal++ suggested to use a moderate value of $k = 3$, so the additional time for the transaction latency was computed in Shoal++ [2] as $1.5\delta/k = 0.5\delta$. In addition, the waiting time in case of a faulty validator is larger than the one in Sailfish due to the lack of non-vote certificates.

Then, we provide a column with amortized communication complexity. It is measured in the number of bytes we need to transmit over the network per 1 byte of transactions. The complexity is computed for the worst-case scenario with f Byzantine nodes and sufficiently large transaction data. Protocols on certified DAGs can achieve that by employing RS-based RBC from [12]l however, the resulting latencies are higher that provided by Starfish. Alternatively, for Sailfish and Shoal++, it is possible to use a faster RBC, but this leads to a quadratic communication complexity in the worst case. For the protocols with uncertified DAGs, such as Mysticeti and Cordial Miners, the latencies can be lower, however, the amortized communication complexity is quadratic. Our proposed protocol, Starfish, simultaneously has linear communication complexity and relatively low latencies.

References

[1] Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Good-case latency of byzantine broadcast: A complete categorization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC). pp. 331–341 (2021)

[2] Arun, B., Li, Z., Suri-Payer, F., Das, S., Spiegelman, A.: Shoal++: High throughput dag bft can be fast! In: Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI) (2025)

[3] Babel, K., Chursin, A., Danezis, G., Kokoris-Kogias, L., Sonnino, A.: Mysticeti: Reaching the limits of latency with uncertified dags. In: Proc. of the 2025 the Network and Distributed System Security Symposium (NDSS) (2025)

[4] Baird, L.: The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep **34**, 9–11 (2016)

[5] Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. Journal of the ACM (JACM) **32**(4), 824–840 (1985)

[6] Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)

[7] Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05). pp. 191–201. IEEE (2005)

[8] Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OsDI. vol. 99, pp. 173–186 (1999)

[9] Chursin, A.: Adelie: Detection and prevention of byzantine behaviour in dag-based consensus protocols (2024), https://arxiv.org/abs/2408.02000

[10] Correia, M., Neves, N.F., Veríssimo, P.: From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. Comput. J. **49**(1), 82–96 (Jan 2006). https://doi.org/10.1093/comjnl/bxh145, https://doi.org/10.1093/comjnl/bxh145

[11] Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the 17th European Conference on Computer Systems (EuroSys). pp. 34–50 (2022)

[12] Das, S., Xiang, Z., Ren, L.: Asynchronous data dissemination and its applications. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2705–2721 (2021)

[13] Doidge, I., Ramesh, R., Shrestha, N., Tobkin, J.: Moonshot: Optimizing block period and commit latency in chain-based rotating leader bft. In: 2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 470–482 (2024). https://doi.org/10.1109/DSN58291.2024.00052

[14] Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)

[15] Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. p. 1187–1201. CCS '22, Association for Computing Machinery, New York, NY, USA (2022)

[16] Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In: Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers. p. 296–315. Springer-Verlag, Berlin, Heidelberg (2022)

[17] Giridharan, N., Suri-Payer, F., Abraham, I., Alvisi, L., Crooks, N.: Autobahn: Seamless high speed bft. In: Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles. pp. 1–23 (2024)

[18] Gągol, A., Leśniak, D., Straszak, D., Świętek, M.: Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT). pp. 214–228 (2019)

[19] Golan Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., Seredinschi, D.A., Tamir, O., Tomescu, A.: Sbft: A scalable and decentralized trust infrastructure. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 568–580 (2019). https://doi.org/10.1109/DSN.2019.00063

[20] Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC). pp. 165–175 (2021)

[21] Keidar, I., Naor, O., Shapiro, E.: Cordial miners: A family of simple, efficient and self-contained consensus protocols for every eventuality. In: Proc. of the 37th International Symposium on Distributed Computing (DISC) (2023)

[22] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. p. 45–58. SOSP '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1294261.1294267

[23] Lin, S.J., Al-Naffouri, T.Y., Han, Y.S., Chung, W.H.: Novel polynomial basis with fast fourier transform and its application to reed–solomon erasure codes. IEEE Transactions on Information Theory **62**(11), 6284–6299 (2016)

[24] Malkhi, D., Nayak, K.: Hotstuff-2: Optimal two-phase responsive bft. Cryptology ePrint Archive (2023)

[25] Malkhi, D., Stathakopoulou, C., Yin, M.: Bbca-chain: One-message, low latency bft consensus on a dag. In: Proc. of the 2024 Financial Cryptography and Data Security (FC) (2024)

[26] Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the theory and application of cryptographic techniques. pp. 369–378. Springer (1987)

[27] Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 31–42. CCS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978399, https://doi.org/10.1145/2976749.2978399

[28] Raikwar, M., Polyanskii, N., Müller, S.: Sok: Dag-based consensus protocols. In: 2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–18. IEEE (2024)

[29] Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics **8**(2), 300–304 (1960)

[30] Shapiro, E.: Grassroots cryptocurrencies: A foundation for a grassroots digital economy. arXiv preprint arXiv **2202** (2022)

[31] Shrestha, N., Kate, A., Nayak, K.: Sailfish: Towards improving latency of dag-based bft. In: Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP) (2025)

[32] Shrestha, Nibesh Yu, Q., Kate, A., Loas, G., Nayak, K., Wang, X.: Optimistic, signature-free reliable broadcast and its applications. arXiv preprint arXiv:2505.02761 (2025)

[33] Singleton, R.: Maximum distance q-nary codes. IEEE Transactions on Information Theory **10**(2), 116–118 (1964)

[34] Spiegelman, A., Aurn, B., Gelashvili, R., Li, Z.: Shoal: Improving dag-bft latency and robustness. In: Proc. of the 2024 Financial Cryptography and Data Security (FC) (2024)

[35] Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: Dag bft protocols made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2718 (2022)

[36] Tonkikh, A., Arun, B., Xiang, Z., Li, Z., Spiegelman, A.: Raptr: Prefix consensus for robust high-performance bft. arXiv preprint arXiv:2504.18649 (2025)