Shoal: Improving DAG-BFT Latency And Robustness

Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li

Aptos Labs

Abstract. The Narwhal system is a state-of-the-art Byzantine fault-tolerant (BFT) scalable architecture that involves constructing a directed acyclic graph (DAG) of messages among a set of validators in a Blockchain network. Bullshark is a zero-overhead consensus protocol on top of the Narwhal's DAG that can order over 100k transactions per second. Unfortunately, the high throughput of Bullshark comes with a latency price due to the DAG construction, increasing the latency compared to the state-of-the-art leader-based BFT consensus protocols.
We introduce *Shoal*, a protocol-agnostic framework for enhancing Narwhal-based consensus. By incorporating leader reputation and pipelining support for the first time in DAG-BFT, Shoal significantly reduces latency. Moreover, the combination of properties of the DAG construction and the leader reputation mechanism enables the elimination of timeouts in all but extremely uncommon scenarios in practice, a property we name "prevalent responsiveness" (it strictly subsumes the established and often desired "optimistic responsiveness" property for BFT protocols).
We integrated Shoal instantiated with Bullshark in an open-source Blockchain project and provide experimental evaluations demonstrating up to 40% latency reduction in the failure-free executions, and up-to 80% reduction in executions with failures against the vanilla Bullshark implementation.

1 Introduction

Byzantine fault tolerant (BFT) systems, including consensus protocols [29,12,22,23] and state machine replication [6,47,43,9,26], have been a topic of research for over four decades as a mean of constructing reliable distributed systems. The recent advent of Blockchains has underscored the significance of high performance, leading to a race among committee-based blockchains [44,39,40,45,41,42] to deliver a scalable BFT system with the utmost throughput and minimal latency.

A recent throughput breakthrough stemmed from the realization that data dissemination, especially in the leader-based approach, is the primary consensus bottleneck and it can benefit from parallelization [15,38,37,46,4]. The leaderless Narwhal&Tusk system [15] achieves a throughput of 160000 transactions per second by parallelizing the networking layer and abstracting it away from the consensus logic [27,35]. Narwhal is a perfectly load-balanced system that efficiently constructs a directed acyclic graph (DAG) of messages, where each message is a vertex that contains, among other things, transaction information.

Tusk is a local asynchronous consensus algorithm, in which validators observe their local views of the DAG and agree on the total order of all vertices without any extra communication.

The drawback, however, of the Narwhal&Tusk system is its high latency compared to the partially synchronous leader-based approach. To improve latency, the Bullshark paper, which together with Narwhal is currently run in production in Sui Blockchain [45] and under development in Aptos [39], trades the ability to tolerate asynchrony and reduces Tusk's latency by 30% while preserving its high throughput. However, the Bullshark latency is still high compared to latency-optimized leader-based protocols such as the two-chain Jolteon [20] and others [25,21].

Our goal is therefore to explore the latency bottlenecks in Narwhal-based consensus protocols and provide a general solution to drastically reduce them.

We present Shoal: a framework that incorporates leader reputation and pipelining mechanisms into all Narwhal-based consensus protocols (i.e., DAG-Rider [27], Tusk [15], and Bullshark [35,36]), and in addition, eliminates timeouts in all except extremely rare scenarios therein.

1.1 Latency bottlenecks

Narwhal provides a round-based DAG in which each validator contributes one vertex per round, and each vertex links to $n - f$ vertices in the preceding round. Each vertex (message) is disseminated via an efficient reliable broadcast, ensuring that malicious validators cannot distribute different vertices to different validators within the same round. The idea shared by Narwhal-based consensus protocols is to interpret the DAG structure as the consensus logic [27,15,35,36]. Although the protocols differ in the networking assumptions and the number of rounds required for vertex ordering, they all share a common structure.

Prior to the protocol initiation, there is an a-priori mapping from specific rounds to leaders shared among all validators (in the asynchronous protocols, this mapping is hidden behind threshold cryptography). We use the term *anchor* to refer to the vertex associated with the round leader in each relevant round.

The DAG local ordering process is divided into two phases. First, each validator determines which anchors to order. Then, the validators sequentially traverse the ordered anchors (the rest are skipped), deterministically ordering all DAG vertices contained within the causal histories of the respective anchors. Consequently, the main considerations that affect the protocol latency are as follows:

1. **Bad leaders.** When a leader fails to broadcast the anchor fast enough, validators advance the DAG without including the round anchor. In this case, unordered vertices in previous rounds can only be ordered as a part of a causal history of a future anchor, directly impacting their latency.
2. **Timeouts.** Moreover, in the partially synchronous version of Bullshark, a crashed leader causes a significant latency penalty as validators, in this case, must wait for a timeout expiration before moving on to the next round.

3. **Sparse anchors.** In Narwhal-based consensus protocols, not every round includes an anchor. Consequently, vertices located farther from the next anchor must wait for additional rounds before they can be ordered.

1.2 Our algorithmic contribution

On the algorithmic side, Shoal presents a generic way to introduce leader reputation and pipelining mechanisms into all Narwhal-based consensus protocols.

Leader reputation is an often overlooked concept in theoretical research, yet it holds crucial importance for practical performance. In practice, Byzantine failures are rare due to robust protection and economic incentives for validators to adhere to the protocol. Moreover, the non-equivocation property provided by the Narwhal-based DAG construction significantly reduces the range of potential Byzantine behavior, basically eliminating the need for the consensus protocols that interpret the DAG to deal with Byzantine behavior. Thus, the most common failure scenarios in Blockchain (esp. in Narwhal-based) systems involve validators who struggle to keep up, which can occur due to temporary crashes, slower hardware, or geographical distance. If unresponsive validators repeatedly become leaders, progress is inevitably impeded and degrades system performance. The leader reputation schemes select leaders based on the history of their recent activity, as introduced in Diem [43] and later formalized in [14].

As for pipelining, in the context of Narwhal-based consensus, it means having an anchor in every round (as apposed to every 2 and 4 rounds in Bullshark and DAG-Rifer, respectively), which would improve latency for non-anchor vertices.

The main challenge. The leader reputation problem is simpler to solve for monolithic BFT consensus protocols. While the validators may disagree on the history that determines the next leader's identity, the worst that can happen is a temporary loss of liveness until view synchronization, i.e. the quorum of validators can eventually recover by agreeing on a fall-back leader. This exact method was utilized in [14], electing the fall-back leaders by a simple round-robin.

In contrast, in Narwhal-based protocols, all communication is done upfront for building the DAG. Therefore, their safety relies on a key property of the local computation that all validators will decide to order the same set of anchors.

For pipelining, even if all validators agree on the mapping, they also must agree on whether to order or skip each anchor. Our attempts to solve the problem by delving into the inner workings of the protocol and exploring complex quorum intersection ordering rules have not been fruitful. Intuitively, this is because consensus requires a voting round after each anchor proposal and the next anchor should link to the decisions (votes) on the previous one (as done in Bullshark).

Our solution. In Shoal, we lean into the power of performing computations on the DAG. In particular, the ability to preserve and re-interpret information from previous rounds. More concretely, Shoal combines multiple instances of the instantiated protocol in a suitable manner, where the trick is to agree on the

switching point based on the following observation:

For any Narwhal-based consensus protocol, since all validators agree on which anchors to order vs skip, they in particular agree on the first ordered anchor.

With this observation in mind, each validator can start locally interpreting its view of the DAG by running an instance of its favorite protocol until it determines the first ordered anchor. Since validators agree on this anchor, they can all deterministically start a new protocol instance in the following round. Note that this too, happens locally, from a validator's perspective, as a part of re-interpreting the DAG. As a result, Shoal ensures the following

1. **Leader reputation:** validators select new anchors for future rounds based on the information available in the causal history of the ordered anchors.
2. **Pipelining:** allocate an anchor in the first round of every instance. That way, if the first anchor in every instance is ordered, we get an anchor in every round, providing the pipelining effect and reducing overall latency.

1.3 Our system contribution

We implemented Shoal in the open-source codebase of one of the live Blockchain networks and instantiated it with the partially synchronous version of Bullshark, resulting in a *Shoal of bull sharks*.

Prevalent responsiveness. With the help of the leader reputation mechanism, we discovered a way to eliminate timeouts in all except extremely rare scenarios, a property we refer to as prevalent responsiveness. The motivation to avoid timeouts in as many situations as possible comes from a purely practical point of view, as (1) when timeouts are common, the duration affects the system performance, but in a way that is non-trivial to configure in an optimal way as it is highly environmentally (network) dependent; and (2) timeout handling is known to add significant complexity to the implementation logic for managing potential state space of validators. In addition, our evaluations demonstrate that eliminating timeouts significantly improves performance.

Leader-based BFT protocols use timeouts to trigger protocol progress whenever a leader is faulty or slow. The optimistic responsiveness property, popularized by HotStuff [47], effectively eliminates timeout implications in ideal scenarios when the network is synchronous and there are no failures. However, when failures do occur or the network experiences asynchronous periods all validators must still wait until the timeout expires before transitioning to the next leader.

Utilizing the inherent properties of the DAG construction, and leader reputation mechanism, we ensure that Shoal makes progress at network speed under a much larger set of scenarios than optimistically responsive protocols would. While the FLP [17] impossibility result dictates that there has to be a scenario that requires a timeout, Shoal design aligns this FLP scenario to be extremely improbably in practice (requires an adversary that fully controls the network).

All available Bullshark implementations use timeouts when advancing rounds to ensure honest validators wait for slow anchors. By eliminating timeouts, Shoal drastically reduces latency when a leader is faulty, as the corresponding anchors would never be delivered and it is best to advance to the next round as fast as possible. However, if the leader is just slower, validators may skip anchors that they could order if they waited a little bit longer. This is where the leader reputation mechanism of Shoal shines, filtering out slow validators that constantly delay new rounds and allowing the DAG to proceed at network speed.

In addition, since the DAG construction provides a notion of an asynchronous clock [18], we avoid the issue faced by leader-based protocols during asynchrony – the impossibility of distinguishing a slow leader from a crashed one – which might cause time outing good leaders, decreasing "responsiveness". DAG consensus protocols, in contrast, do not require a view-change and thus can afford waiting until $2f+1$ vertices are eventually delivered, resulting in network speed progress.

Evaluation Our experimental evaluation demonstrates up to 40% reduction in latency against vanilla Bullshark protocol implementation when there are no failures in the system, and up to 80% reduction in latency when there are failures. In addition, we compare Shoal to Jolteon [20] and get comparable latencies.

In summary, the paper focuses on improving latency and robustness in DAG-Based protocols. It provides Shoal, a framework to enhance any Narwhal-based consensus protocol with:

1. Leader reputation mechanism that prevents slow, isolated, or crashed validators from becoming leaders.
2. Pipelining support that ensures every round on the DAG has an anchor.
3. The ability to eliminate timeouts in most cases.

2 DAG BFT

2.1 Background

The concept of DAG-based BFT consensus, initially introduced by HashGraph [5], decouples the network communication layer from the consensus logic. Each message consists of a collection of transactions and references to previous messages. These messages collectively form an ever-growing DAG, with messaging serving as vertices and references between messages serving as edges.

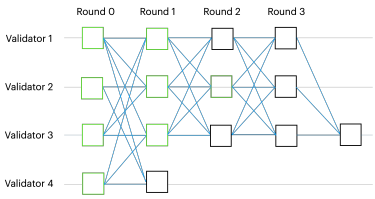

Fig. 1: A causal history is highlighted in green.

In Narwhal, the DAG is round-based, similar to Aleph [19]. Each vertex within the DAG is associated with a round number. In order to progress to round r, a validator must first obtain $n-f$ vertices in round $r-1$ (from distinct validators). Every validator can broadcast one vertex per round, with each vertex referencing a minimum of $n-f$ vertices from the previous round. The *causal*

history of a vertex v refers to the sub-graph that starts from v. Figure 1 illustrates a validator's local view of a round-based DAG.

To disseminate messages, Narwhal uses an efficient reliable broadcast implementation, and all in all the DAG construction guarantees the following:

Validity: if an honest validator has a vertex v in its local view of the DAG, then it also has all the causal history of v.

Eventual delivery: if an honest validator has a vertex in round r by validator p in its local view of the DAG, then eventually all honest validators have a vertex in round r by validator p in their local views of the DAG.

Non-equivocation: if two honest validators have a vertex in round r by validator p in their local views of the DAG, then the vertices are identical.

Inductively applying Validity and Non-equivocation, we get:

Completeness: if two honest validators have a vertex v in round r by validator p in their local views of the DAG, then v's causal histories are identical in both validators' local view of the DAG.

In simple words, Narwhal construction guarantees that:

1. All validators eventually see the same DAG; and at any given time
2. Any two validators that locally have the same vertex v, also have and agree on the whole causal history of v (content and structure).

Narwhal-based consensus protocols. DAG-Rider, Tusk, and Bullshark are all algorithms to agree on the total order of all vertices in the DAG with no additional communication overhead, completely eliminating the need for complex mechanisms like view-change or view-synchronization. Each validator independently looks at its local view of the DAG and orders the vertices without sending a single message. This is done by interpreting the structure of the DAG as a consensus protocol, where a vertex represents a proposal and an edge represents a vote.

DAG-Rider [27] and Tusk [15] are randomized protocols designed to tolerate full asynchrony, which necessitates a larger number of rounds and consequently, a higher latency. Bullshark [36] also provides a deterministic protocol variant with a faster ordering rule, relying on partial synchrony for liveness. While the specific details are not required to understand this paper, next we explain the high-level structure of these protocols and define a property they all share.

For space limitation, a detailed related work section is deferred to Appenidx A.

2.2 Common framework

Narwhal-based consensus protocols share the following abstract structure:

1. **Pre-determined anchors.** Every few rounds (the number depends on the protocol) there is a round with a pre-determined leader. The vertex of the leader is called an *anchor*. In the partially synchronous version of Bullshark, the leaders are a-priori known. In the asynchronous protocols, the leaders are hidden and revealed during the DAG construction.

2. **Order the anchors.** All validators independently decide which anchors to skip and which to order. The details differ among the protocols, although they all rely on quorum intersection in the DAG structure.
3. **Order causal histories.** Validators process their list of ordered anchors one by one, and for each anchor order all previously unordered vertices in their causal history by some deterministic rule.

An illustration of the ordering logic appears in Figure 2. In this example, the validator orders the red and yellow anchors, while the green (which is not in the DAG) anchor is skipped. To order the DAG, the validator deterministically orders the yellow anchor's causal history after the red anchor's causal history.

The key correctness argument for all the above mention consensus protocols relies on the fact that all validators agree on which anchors to order and which to skip. In particular, they agree on the first anchor to order. More formally, our Shoal framework relies on is the following:

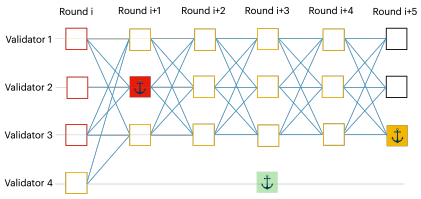

Fig. 2: A possible local view of the DAG in the partially synchronous Bullshark protocol. Filled squares represent the pre-defined anchors.

Property 1. Given a Narwhal-based protocol $\mathcal{P}$, if all honest validators a priori agree on the mapping from rounds to leaders, then they will agree on the first anchor to order during the execution of $\mathcal{P}$.

The proof follows immediately from Proposition 2 in DAG-Rider [27] and Corollary C in Bullshark [35].

3 Shoal

Shoal is protocol agnostic and can be directly applied to all Narwhal-based protocols. It makes no changes to the protocols but rather combines instances in a "black-box" manner. The proof can be derived solely from Property 1.

3.1 Pipelining

Bullshark already halved DAG-rider's latency for ordering anchors from 4 rounds to 2 by adding an optimistic path under the partially synchronous network communication assumption. Intuitively, it is hard to imagine latency lower than 2 rounds as in the interpretation of the DAG structure as a consensus protocol, one round is needed to "propose" the anchor, while another is needed for "voting". However, only anchors can be ordered in 2 rounds. The rest of the vertices are ordered as part of anchors' causal histories and thus require more rounds. The vertices in a "voting" round require (minimum) 3 rounds, while vertices that share a round with an anchor have to wait for at least the next anchor to be ordered, thus requiring (minimum) 4 rounds. See Figure 3 for illustration.

Ideally, to reduce the latency we would like to have an anchor in every round. This would allow for non-anchor vertices to be ordered as a part of some anchor's causal history in each and every round, making latency and throughput of the protocol less spiky. In Bullshark, it would become possible for every non-anchor vertex to be ordered in 3 rounds.

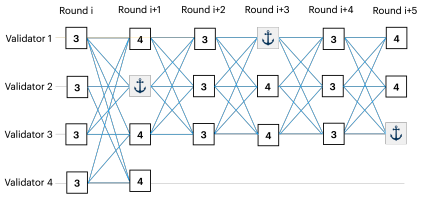

Fig. 3: The number in each vertex represents its minimum latency according to Bullshark. For example, the anchor in round $i+1$ can be ordered in round $i+2$, but the other vertices in this round require at least 4 rounds.

Solution Let $\mathcal{P}$ be any Narwhal-based consensus protocol. The core technique in Shoal is to execute $\mathcal{P}$ until it, as a consensus protocol, guarantees agreement on some part of the DAG for all validators. Starting from the round following the agreed part of the DAG, all validators can switch over and start executing a new instance of $\mathcal{P}$ (or a different Narwhal-based consensus protocol, if desired) from scratch. While the instances are not executing concurrently, this scheme effectively pipelines the "proposing" and "voting" rounds. As a result in Shoal, in a good case an anchor is ordered in every round.

Algorithm 1 Pipelining in Shoal

1: $current\_round \leftarrow 0$
2: $F : R \rightarrow A$ ▷ deterministic rounds to anchors mapping
3: **while** *true* **do**
4: execute $\mathcal{P}$, select anchors by F, starting from $current\_round$ until the first ordered (not skipped) anchor is determined.
5: let A be the first ordered anchor and let r be its round
6: order A's causal history according to $\mathcal{P}$
7: $current\_round \leftarrow r + 1$
8: update F according to A's causal history

The pseudocode appears in Algorithm 1 (ignore line 8 for now). In the beginning of the protocol, all validators interpret the DAG from round 0, and the function F is some pre-defined deterministic mapping from rounds to leaders. Each validator locally runs $\mathcal{P}$, using F to determine the anchors, until it orders the first anchor, denoted by A in round r. The key is that, by the correctness of $\mathcal{P}$ as stated in Property 1, all validators agree that A is the first ordered anchor. Consequently, each validator can re-interpret the DAG from the next round $(r+1)$ according to a new instance of the protocol $\mathcal{P}$ (or another Narwhal-based protocol) executing from scratch. Note that without re-interpreting the DAG, the next anchor according to the instantiated protocol would appear in a strictly later round (e.g. $r+4$ for DagRider and $r+2$ for Bullshark). The above process can continue for as long as needed. To order the DAG, like in the original $\mathcal{P}$, the validators deterministically order A's causal history, and by the Completeness property, arrive at the same total order of vertices. See illustration in Figure 4.

Note that in Algorithm 1, function F is fixed and used by each instance of protocol $\mathcal{P}$. In a true "black-box" implementation, the round numbers could be different from the perspective of the executing protocol instance (i.e. start from 0 for each new instance). However, F is fixed and always assigns the same anchor to any given round r in Shoal regardless of the protocol instance used for this round.

Note that with Shoal, ordering an anchor vertex requires 2 rounds, while all other vertices require 3. In Section D we discuss a potential direction to reduce the latency for non-anchor vertices by treating all vertices as anchors. Intuitively, we can use Property 1 to instantiate a binary agreement to decide whether to commit each vertex individually.

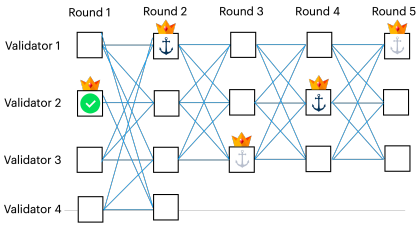

Fig. 4: Shoal's pipelining integrated into Bullshark. The vertices that are fixed to be anchors by F are marked by a crown. The protocol starts by interpreting the DAG with anchors in rounds $1, 3$, and 5. Bullshark determines that the anchor in round 1, marked by a green checkmark, is the first to be ordered. Then, a new instance of Bullshark starts at round 2 with the anchors in rounds 2 and 4.

3.2 Leader Reputation

BFT systems are designed to tolerate Byzantine failures in order to provide as strong as possible worst-case reliability guarantees. However, actual Byzantine failures rarely occur in practice. This is because validators are highly secured and have strong economic incentives to follow the protocol. Slow or crashed leaders are a much more frequent occurrence which can significantly degrade the system performance. In Narwhal-based BFT, if the leader of round r crashes, no validator will have the anchor of round r in its local view of the DAG. Thus, the anchor will be skipped and no vertices in the previous round can be ordered until some later point due to an anchor in a future round.

The way to deal with missing anchors is to somehow ensure that the corresponding leaders are less likely to be elected in the future. A natural approach to this end is to maintain a reputation mechanism, assigning each validator a score based on the history of its recent activity. A validator who has been participating in the protocol and has been responsive would be assigned a high score. The idea is then to deterministically re-compute the pre-defined mapping from rounds to leaders every time the scores are updated, biasing towards leaders with higher scores. In order for validators to agree on the new mapping, they should agree on the scores, and thus on the history used to derive the scores.

Such a mechanism was previously proposed in [14] and implemented in the Diem Blockchain [43] to enhance the performance of Jolteon [20], a leader-based consensus protocol. One important property Jolteon is that Safety is preserved even if validators temporarily disagree on the identity of the leader. In Narwhal-based BFT, however, if validators disagree on the anchor vertices, they will order the DAG differently, violating safety.

Solution. Surprisingly, leader reputation in Shoal can be naturally combined with pipelining as they both utilize the same core technique of re-interpreting the DAG. In fact, the full pseudocode for Shoal requires only 1 extra line (marked in gray) on top of the pipelining pseudocode in Algorithm 1. The idea is that the validators simply need to compute a new mapping, starting from round $r+1$, based on the causal history of ordered anchor A in round r (which they are guaranteed to agree on by Property 1). Then, validators start executing a new instance of $\mathcal{P}$ from round $r+1$ with the updated anchor selection function F. See Figure 5 for the leader reputation illustration.

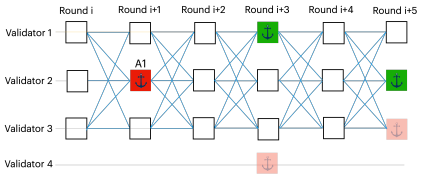

Fig. 5: Shoal's leader reputation integrated into Bullshark (no pipelining). First, the DAG is interpreted via the red anchors. The anchor in A1 is the first ordered anchor. Then, based on A1's causal history, new anchors are selected for future rounds (marked in green). Finally, after ordering A1's causal history, a new instance of Bullshark starts at round $i+2$ based on the green anchors.

For space limitation, the correctness proof can be found in Appendix B.

4 Implementation and Prevalent Responsiveness

We have implemented Narwhal and the partially synchronous version of Bullshark as part of a publicly available open-source blockchain project (undisclosed for anonymity requirement). This blockchain is live and the process of productionizing our implementation is underway. We used Rust, utilizing Tokio for asynchronous runtime, BLS [7] implemented over BLS12-381 curves for signatures, RocksDB for persistent storage, and Noise for authentication. Our implementation of Bullshark is according to [35] and we refer to it as *Vanilla Bullshark*.

4.1 Eliminating Timeouts

Validators in Bullshark must observe $n-f$ vertices in a round to advance to the next round. Even rounds have anchors, while vertices in odd rounds determine the "voting" pattern. The full Bullshark protocol uses, in addition, the following conditions to advance rounds:

- Even-round: Wait until timeout or the delivery of the round anchor.
- Odd-round: Wait until timeout or the delivery of $2f+1$ vertices that link to the previous round anchor.

Shoal eliminates these timeouts in a way that significantly improves latency and simplifies implementation and maintenance. From now on we will refer to even-rounds as *anchor rounds* and to odd-rounds as *vote rounds*.

Vanilla Bullshark w/o vote Timeout. In the full Bullshark $2f + 1$ votes are required to order anchors. Therefore, without timeouts in vote rounds, a Byzantine adversary can prevent the fast path from making progress even during synchrony. As long as Byzantine validators deliberately do not link to the anchor, and even 1 of their vertices get delivered among the first $2f + 1$ to an honest validator in a vote round, then the honest validator will not be able to order the anchor.

However, we realized that timeouts in vote rounds in the partially synchronous variant of Bullshark are redundant. The anchor ordering rule in this case is $f + 1$ votes [36]. As a result, even if f out of the first $2f + 1$ vertices delivered to a validator in a round are from Byzantine validators, the remaining $f + 1$ vertices will link to the anchor due to the anchor round timeout.

Baseline Bullshark. The FLP impossibility result [17] dictates that any deterministic protocol providing liveness under partial synchrony must use timeouts. In Bullshark, without timeouts in the even rounds, an honest leader that is even slightly slower than the fastest $2f + 1$ validators will struggle to get its anchor linked by other vertices. As a result, the anchor is unlikely to be ordered. The timeout, therefore, ensures that all honest validators link to anchors during periods of synchrony, provided they are associated with honest leaders.

Even though timeouts are unavoidable in the worst case, we observe that the DAG construction combined with the leader reputation mechanism allows avoiding them in the vast majority of cases. This is in contrast to leader-based monolithic consensus protocols, where timeouts are the only tool to bypass views with bad leaders. Without timeouts, a monolithic protocol could stall forever as there is no other mechanism to stop waiting for a crashed leader.

In contrast, the DAG construction provides a "clock" that estimates the network speed. Even without timeouts, the rounds keeps advancing as long as $2f+1$ honest validators continue to add their vertices to the DAG. As a result, the DAG can evolve despite some leaders being faulty. Eventually, when a non-faulty leader is fast enough to broadcast the anchor, the ordering will also make progress.

In the partially synchronous Bullshark, an anchor needs $f + 1$ votes (links) out of the $3f + 1$ vertices. Therefore, as our evaluation demonstrates, in the failure-free case, most of the anchors are ordered in the next round. The benefit are even more pronounced when there are failures. This is because a crashed validator causes a timeout to expire, stalling the protocol for the entire duration. Without a timer, however, the DAG will advance rounds at network speed and the Bullshark protocol will immediately move to the next anchor.

Timeouts as a fallback. To guarantee liveness, i.e., to avoid the extremely unlikely adversarial schedule of events that can prevent all anchors from getting enough votes to be ordered, Shoal falls back to using timeouts after a certain (pre-defined) amount of consecutive skipped anchors.

4.2 Shoal of Bullsharks

A realistic case in which timeouts may help the latency of a Narwhal-based consensus protocol is when the leader is slower than the other validators. In this

case, waiting for a slow anchor may be faster than building the next two DAG rounds in network speed (and committing the next anchor). In Shoal, however, the leader reputation mechanism excludes (or at least significantly reduces the chances of) slow validators from being selected as leaders. This way, the system takes advantage of the fast validators to operate at network speed.

Prevalent Responsiveness. Shoal provides network speed responsiveness under all realistic failure and network scenarios, a property we name *Prevalent Responsiveness*. Specifically, compared to optimistic responsiveness, Shoal continues to operate at network speed, without artificial delays, even during asynchronous periods or if leaders fail for a configurable number of consecutive rounds.

We implemented leader reputation and pipelining on top of the Baseline Bullshark. We provide the logic of leader reputation score assignment in Appendix C.

5 Evaluation

We evaluated the performance of the aforemententioned variants of Bullshark, Shoal, and Jolteon on a geo-replicated environment in Google Cloud. In order to show the improvements from pipelining and leader reputation independently, we also evaluate Shoal PL, which is a Shoal instantiation with only pipelining enabled, and Shoal LR, which is a Shoal instantiation with only Leader Reputation enabled. With our evaluation, we aim to show that (i) Shoal maintains the same throughput guarantees as Bullshark. (ii) Shoal can provide significantly lower latency than Bullshark and its variants. (iii) Shoal is more robust to failures.

For completeness, we also compare against Jolteon [20], which is the current consensus protocol of the production system we use. Jolteon combines the linear fast path of Tendermint/Hotstuff with a PBFT style view-change, and as a result, reduces Hotstuff latency by 33%. To mitigate the leader bottleneck and support high throughput, the implementation uses the Narwhal technique to decouple data dissemination via a pre-step component (called Quorum Store).

Experimental Setup. Our experimental setup consists of `t2d-standard-32` type virtual machines spread equally across three different Google Cloud regions: us-west1, europe-west4, asia-east1. Each virtual machine has 32 vCPUs, 128GB of memory, and can provide up to 10Gbps of network bandwidth. The round-trip latencies are: 118ms between us-west1 and asia-east1, 251ms between europe-west4 and asia-east1, and 133ms between us-west1 and europe-west4. The experiments involve three different values of N (the number of validators): 10, 20, and 50, tolerating up to 3, 6, and 16 failures, respectively.

The transactions are approximately 270B in size and the maximum batch size is 5000 transactions. We measure *latency* as the time elapsed from when a vertex is created from a batch of client transactions to when it is ordered by a validator. The timeouts, when applicable, are set to 1s.

5.1 Baseline Performance

First, we evaluate the performance of the Bullshark variants, namely Vanilla Bullshark, Vanilla Bullshark w/ Anchor Timeouts, and Baseline Bullshark, to align on a baseline performance to evaluate Shoal in the rest of the experiments. The results are in Figures 6 and 7.

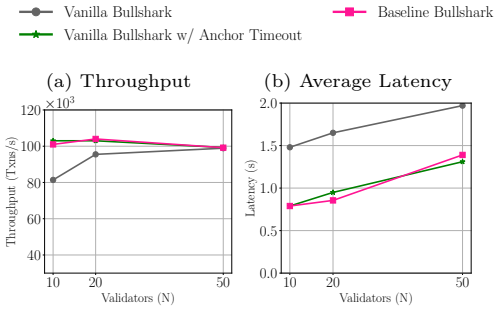

Fig. 6: Baseline performance under no failures

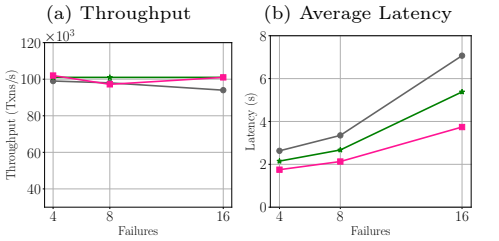

Fig. 7: Baseline performance w/ failures (N=50)

Figure 6 shows the throughput and average latencies of the three Bullshark variants as the system size increases. The presence of timeouts in Vanilla Bullshark forces it to build the DAG slowly, which combined with the fact that fewer validators contribute vertices to the DAG when $N = 10$, results in lower throughput than other variants, which have fewer or no timeouts. The latencies for Vanilla Bullshark is up to 88% higher due to the timeouts. Interestingly, the latencies are similar for baseline Bullshark and Vanilla Bullshark w/o Vote timeout in the normal case because there is a trade-off between building a DAG at network-speed while skipping an anchor and waiting slightly longer for the anchor to be part of the votes.

We also evaluated the vanilla variants and the baseline for $N = 50$ with varying the number of failures, in Figure 7. The Baseline Bullshark provides lower latency than other variants by virtue of being able to build the DAG at network speed skipping failed anchors and ordering using the alive ones. Therefore, in the rest of the section, we use Baseline Bullshark as the baseline to evaluate Shoal.

5.2 Performance of Shoal under fault-free case

We now evaluate the Shoal variants against the baseline under the normal case where there are no failures. The results are in Figure 8. As expected, the throughput of the Shoal variants is similar as the number of validators increases. It can be observed that each variant of Shoal decreases the latency leading to full Shoal protocol. In summary, we observe that the Shoal's average latency decreases by up to 20% compared to Baseline Bullshark.

On the other hand, Jolteon [20], despite its use Narwhal's data dissemination decoupling, is only able to achieve a peak throughput of less than 60k, about

40% lower than Shoal. This is because under high load leaders are again the bottleneck as they are not able to deal with the required network bandwidth, and as a result, unable to drive progress before timeouts expire.

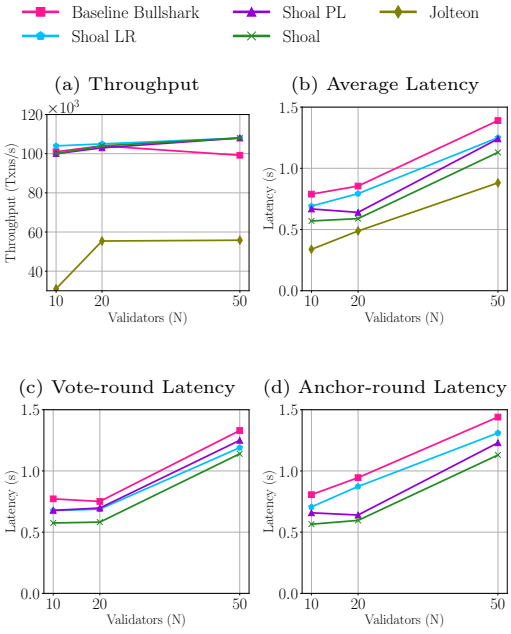

Fig. 8: Shoal performance under no failures

This is because under high load leaders become the bottleneck again as they are not able to deal with the required network bandwidth, and as a result, unable to drive progress before timeouts expire. Furthermore, in terms of latency, Jolteon is ≈50% better than Vanilla Bullshark, but only ≈20% better than Shoal. Note that the latencies presented do not include the pre-step Quorum Store's latencies to decouple data from metadata, because all the compared protocols include this optimization. However, in the Shoal case, this latency can be avoided by merging Quorum Store into the DAG construction, as done in Narwhal. This will further close the latency gap from Jolteon.

In Figures 10c and 10d, we distinguish the latencies of transactions in the vote-round vertices from that in anchor-round vertices, in order to show the effect of the pipelining approach. The vote and anchor round latencies for Shoal PL, as well as Shoal, are similar, which helps provide predictable and smooth latency for transactions in real production systems. In contrast, the vote and anchor round latencies for Baseline Bullshark and Shoal LR differ by 5-20% depending on the number of failures.

5.3 Performance of Shoal under faults

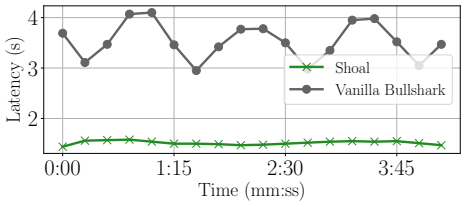

Fig. 9: Latency timeline under 8 failures with $N = 50$. The x-axis represents a part of the experiment time window.

Figure 10 shows the behavior of baseline and Shoal variants under faults. For this experiment, $N = 50$ and the failures are increased from 4 to 16 (maximum tolerated). This is the case where the Leader Reputation mechanism helps to improve the latency significantly by reducing the likelihood of failed validators from being anchors. Notice that without Leader Reputation, the latencies of Baseline

Bullshark and Shoal PL increases significantly as the number of failures increases. Shoal provides up to 65% lower latencies than Baseline Bullshark under failures.

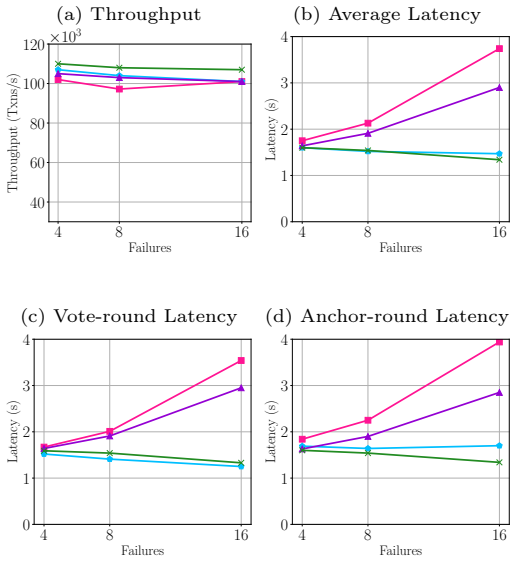

Fig. 10: Shoal performance w. failures (N=50).

Figure 9 shows the impact of skipping leaders on the latency by comparing vanilla Bullshark with Shoal on a timeline plot under failures. We have a system of 50 validators, 8 of which have failed. The x-axis represents a part of the experiment time window and the y-axis shows the latency. The presence of timeouts and the need to skip anchors causes vanilla Bullshark's latency to fluctuate. In our experiment, we observed latency jitter of approximately one second, which makes it impossible to provide predictable latency in production systems. In constrast, Shoal maintains consistent low latency without any jitter.

5.4 Summary

Shoal provides comparable latency to Jolteon, and in contrast to Vanilla Bullshark, Shoal provides up to 40% lower latency in the fault-free case and up to 80% lower latency under failures. Moreover, Shoal provides predictable latency and commits at network speed in most cases, avoiding waiting for timeouts.

6 Discussion

Shoal can be instantiated with any Narwhal-based consensus protocol to reduce latency, and can even switch between protocols during the DAG retrospective re-interpretation.

Moreover, Shoal eliminates the use of timeouts except in very rare cases, which contributes to the robustness and performance of the system. These timeouts are hard to set right, especially in dynamic networks. Too aggressive timeouts may cause skipping honest leaders, whereas too conservative timeouts cause high latency penalties in case of failures. In addition, predictable and smooth latency and throughput patterns have major practical benefits for real systems. It facilitates setting up effective monitoring and alerts for anomaly detection. This is crucial for ensuring security and quality of service by enabling timely response and any intervention necessary, be it manual or automated. Predictable

consensus throughput also facilitates pipelining the ordering of transactions with other components of the Blockchain, e.g. transaction execution and commit.

As for prevalent responsiveness, intuitively, Shoal ensures network speed progress (no artificial delays) even under failures and asynchrony.

References

1. Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. Fault-scalable byzantine fault-tolerant services. In Andrew Herbert and Kenneth P. Birman, editors, *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005*, pages 59–74. ACM, 2005.
2. Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 337–346. ACM, 2019.
3. Salem Alqahtani and Murat Demirbas. Bottlenecks in blockchain consensus protocols. *CoRR*, abs/2103.04234, 2021.
4. Balaji Arun and Binoy Ravindran. Scalable byzantine fault tolerance via partial decentralization. *Proc. VLDB Endow.*, 15(9):1739–1752, may 2022.
5. Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 2016.
6. Alysson Bessani, Joao Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362. IEEE, 2014.
7. Dan Boneh, Benjamin Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.
8. Manuel Bravo, Gregory Chockler, and Alexey Gotsman. Making byzantine consensus live. *Distributed Computing*, 35(6):503–532, 2022.
9. Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.
10. Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming.* Springer Science & Business Media, 2011.
11. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.
12. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.
13. Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, 1996.
14. Shir Cohen, Rati Gelashvili, Lefteris Kokoris Kogias, Zekun Li, Dahlia Malkhi, Alberto Sonnino, and Alexander Spiegelman. Be aware of your leaders. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 279–295. Springer, 2022.

15. George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.
16. Sisi Duan, Michael K. Reiter, and Haibin Zhang. BEAT: asynchronous BFT made practical. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2028–2041. ACM, 2018.
17. Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
18. Bryan Ford. Threshold logical clocks for asynchronous distributed coordination and consensus. *arXiv preprint arXiv:1907.07010*, 2019.
19. Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 214–228, 2019.
20. Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 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*, pages 296–315. Springer, 2022.
21. Neil Giridharan, Heidi Howard, Ittai Abraham, Natacha Crooks, and Alin Tomescu. No-commit proofs: Defeating livelock in bft. *Cryptology ePrint Archive*, 2021.
22. Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376, 2010.
23. Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.
24. Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 803–818. ACM, 2020.
25. Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.
26. Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: Resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 295–308, 2012.
27. Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *Proceedings of the 40th Symposium on Principles of Distributed Computing*, PODC '21, New York, NY, USA, 2021. Association for Computing Machinery.
28. Idit Keidar, Oded Naor, and Ehud Shapiro. Cordial miners: A family of simple, efficient and self-contained consensus protocols for every eventuality. *arXiv preprint arXiv:2205.09174*, 2022.

29. Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.
30. Leslie Lamport. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*, pages 277–317. 2019.
31. Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 129–138, New York, NY, USA, 2020. Association for Computing Machinery.
32. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.
33. Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. Lumière: Byzantine view synchronization. *CoRR*, abs/1909.05204, 2019.
34. Oded Naor and Idit Keidar. Expected linear round synchronization: The missing link for linear byzantine smr. *arXiv preprint arXiv:2002.07539*, 2020.
35. Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.
36. Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version. *arXiv preprint arXiv:2209.05633*, 2022.
37. Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. State machine replication scalability made simple. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 17–33, 2022.
38. Chrysoula Stathakopoulou, David Tudor, Matej Pavlovic, and Marko Vukolić. [solution] mir-bft: Scalable and robust bft for decentralized networks. *Journal of Systems Research*, 2(1), 2022.
39. The Aptos Team. Aptos networks, 2022. Accessed: May 2023.
40. The Avalanche Team. Avalanche networks, 2020. Accessed: May 2023.
41. The Celo Team. Celo networks, 2020. Accessed: May 2023.
42. The Definity Team. Definity networks, 2018. Accessed: May 2023.
43. The Diem Team. Diembftv4, 2021. Accessed: May 2023.
44. The Solana Team. Solana networks, 2020. Accessed: May 2023.
45. The Sui Team. Sui networks, 2023. Accessed: May 2023.
46. Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.
47. Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.

A Related work

A.1 BFT systems for Blockchains

Byzantine fault tolerance (BFT) has been an active area of research for over four decades, with a significant body of literature in both theory [10] and systems [6,12,1,29]. With the advent of Blockchain systems in recent years, the focus on performance and scalability has notably increased.

Historically, the prevailing belief has been that reducing the communication complexity of leader-based protocols was the key to unlocking high throughput, leading to the pursuit of protocols with communication complexity linear to the number of validators [47,33,8,34]. However, despite a sound theoretical premise, the practical throughput implications arguably fell short of expectations. An independent evaluation and comparison conducted by [3] revealed that the well-known HotStuff [47] protocol achieved a throughput of only 3,500 TPS on a geo-replicated network.

The practical breakthrough occurred recently with the realization that the main bottleneck in BFT systems, particularly those relying on leaders, is data dissemination. Mir-BFT [38] and [37] introduced an innovative approach by running multiple PBFT [12] instances in parallel. Independently, Narwhal [15] and later Dispersedledger [46] decoupled data dissemination from the consensus logic. These advancements showcased impressive results, with Narwhal achieving a peak throughput of 160,000 TPS.

There has been system [32,20,16,24] and theoretical [2,31,11] research in asynchronous BFT protocols. However, to the best of our knowledge, no asynchronous protocol is deployed in production in an industrial system. Another appealing property of Narwhal is the support of a partially synchronous [36] as well as asynchronous [15,27,35] (as long as randomness is available) protocols, and the ability to easily switch among them.

A.2 Timeouts and responsiveness

The FLP [17] impossibility result states that there is no deterministic consensus protocol that can tolerate a fully asynchronous network. The proof relies on the fact that it is impossible to distinguish between crashed and slow validators during asynchronous periods. The immediate application to partially synchronous networks, therefore, is that all deterministic protocols must rely on timeouts in some way to guarantee liveness against a worst-case adversary. Indeed, to the best of our knowledge, all previous deterministic BFT protocols, including the partially synchronous version of Bullshark [36], relied on timeouts to implement a simple version of a failure detector [13]. This mechanism monitors the leaders and triggers view-changes when timeouts expire, i.e. when faults are suspected.

The optimistic responsiveness property, popularized by HotStuff [47], avoids timeouts in the best-case synchrnous failure-free scenario. However, when failures do occur, all validators wait until the timeout expires before view-changing to the next leader, introducing a significant slowdown in the protocol execution.

Moreover, as discussed in Section 4, setting a proper timeout duration is a non-trivial problem in its own right since if timeouts are too short during periods of asynchrony, validators view-change good leaders.

Shoal provides prevalent responsiveness, which is a strictly better property than optimistic responsiveness as it guarantees network speed progress in case of healthy leaders and zero artificial delays in case of failures or asynchrony. Shoal achieves this by relying on the network speed "clock" inherent in the DAG construction itself [18], combined with the leader reputation mechanism. While due to the FLP result, the worst case in which a timeout would be required for maintaining the liveness of the protocol cannot completely be eliminated, Shoal successfully relegates such cases to occur in specific extremely uncommon scenarios from a practical point of view (multiple consecutive unordered anchors).

A.3 DAG-based BFT

DAG-based consensus in the context of BFT was first proposed by HashGraph [5]. The idea is to separate the network communication layer, i.e. efficiently constructing a system that forms a DAG of messages, and the consensus logic that can involve complex pieces such as view-change and view-synchronization. The consensus logic is performed locally, whereby a validator examines its local view of the DAG and orders the vertices without sending any messages. The challenge arises from the asynchronous nature of the network, which may cause different validators to observe slightly different portions of the DAG. To address this, the DAG structure is interpreted as a consensus protocol, wherein a vertex represents a proposal and an edge represents a vote.

Aleph [19] introduced a round-based DAG structure. Such a structure simplifies support for garbage collection and non-equivocation, which in turn simplifies the consensus logic to order the vertices. Narwhal implements round-based DAG, and three Narwhal-based consensus protocols have been previously proposed. The first is DAG-Rider [27], which introduced a quantum-safe asynchronous protocol with optimal amortized communication complexity and $O(1)$ latency. Tusk [15] improved latency in the best case. An asynchronous version of Bullshark [35,36] includes a fast path [35], while a stand-alone partially synchronous protocol [36] also exists and is currently deployed in production in Sui [45]. Shoal presents a framework that applies to all Narwhal-based protocols, enhancing their latency through a more efficient ordering rule and a leader reputation mechanism.

An orthogonal theoretical effort [28] reduces best-case latency by trading the non-equivocation property of the DAG construction. Instead, they use a clever quorum intersection rule on the DAG structure. Practically, however, the approach must inherently use timeouts as it requires waiting for all honest validators in every round. In addition, since validators can equivocate, the memory footprint of the DAG is much bigger and the state-sync is much more complex.

A.4 Pipelining

To the best of our knowledge, pipelining in the BFT context was first proposed by Tendermint [9], and later utilized in HotStuff [47] and Diem [43]. State machine replication (SMR) systems can be constructed from multiple instances of single-shot consensus [30], e.g. one approach to build Byzantine SMR is by running a PBFT instance [12] for each slot. Tendermint introduced the elegant idea of chaining proposals or piggybacking single-shot instances such that a value for a new slot could be proposed before the value for the previous slot was committed. In this approach, a message in the i^{th} round of the k^{th} instance can be interpreted as a message in round $i-1$ of instance $k+1$. While the latency for each instance remains unchanged, clients experience improved latency as their transactions can be proposed earlier.

In DAG-based consensus, the concept of piggybacking proposals is inherent in the design, as each vertex in the DAG links to vertices in previous rounds. However, previous protocols did not allow having an anchor in every round. Shoal framework supports having an anchor in each round in a good case for any Narwhal-based protocol, providing a "pipelining effect".

A.5 Leader reputation

Leader reputation is often overlooked in theory, yet it plays a crucial role in performance in practice. While Byzantine failures are rare as validators are highly protected, isolated, and economically incentivized to follow the protocol, more common are validators that are unresponsive. This may be because they temporarily crashed, running slow hardware, or are simply located farther away. If a leader/anchor election is done naively, unresponsive validators will unavoidably stall progress and lead to significant performance impact.

A practical approach, implemented in Diem [43] and formalized in [14], is to exclude underperforming validators from leader election. This is achieved by updating the set of candidates after every committed block based on the recent activity of validators. In a chained protocol, if all validators observe the same committed block, they can deterministically elect future leaders based on the information in the chain. However, in some cases, certain validators may see a commit certificate for a block earlier than others. This can lead to disagreements among validators regarding the list of next leaders, causing a temporary loss of liveness.

For DAG-based protocols, disagreements on the identity of round leaders can lead the validators to order the DAG completely differently. This poses a challenge for implementing leader reputation on the DAG. As evidence, a Narwhal and Bullshark implementation currently deployed in production in Sui blockchain does not support such a feature [1]. Shoal enables leader reputation in Narwhal-based BFT protocols without any additional overhead.

[1] github.com/MystenLabs/sui/blob/main/narwhal/consensus/src/bullshark.rs

B Correctness

To prove the correctness of Shoal (Algorithm 1) we assume that the underlying protocol satisfies Property 1, which we will use inductively.

Lemma 1. *Let P be a Narwhal-based DAG-BFT protocol that satisfies Property 1. Let D be a round-based DAG, and assume a known to all function F that maps rounds to anchors. Then all the locally ordered lists of anchors by honest validators executing Shoal with P according to F share the same prefix.*

Proof. Proof is by induction on the ordered anchors.

Base: We need to show that all honest validators agree on the first anchor. Since Shoal starts by running P until the first anchor is ordered, the base case follows immediately from Property 1.

Step: Assume all honest validators agree on the first k ordered anchors, we need to prove that they agree on anchor $k + 1$. First, we show that all honest validators agree on the new function F (Line 8 in Algorithm **??**). This holds because the new function F is deterministically computed according to the information in k's causal history, and by the Completeness property of the DAG, all honest validators have the same causal history of anchor k in their local view.

Next, let r be the round of anchor k. By the inductive assumption, all honest validators agree on r. Thus, all honest validators start the next instance of P in the same round $r + 1$.

Now consider a DAG D' that is identical to D except it does not have the first r rounds. By Property 1, all validators that run P with the new function F on D' agree on the first ordered anchor in D'. Therefore, all validators agree on anchor $k + 1$ in D.

Theorem 1. *Let P be a Narwhal-based DAG-BFT protocol that satisfies Property 1. Shoal with P satisfies total order.*

Proof. By Lemma 1, all validators order the same anchors. The theorem follows from the DAG Completeness property as all validators follow the same deterministic rule to order the respective causal histories of the ordered anchors.

C Leader reputation logic

As explained in Section 3.2, Shoal ensures all validators agree on the information used to evaluate the recent activity and to bias the leader selection process accordingly towards healthier validators. Any deterministic rule to determine the mapping from rounds to leaders (i.e. the logic in pseudocode Line 8 in Algorithm 1) based on this shared and agreed upon information would satisfy the correctness requirements. Next, we discuss the specific logic used in our implementation.

At any time each validator is assigned either a high or a low score, and all validators start with a high score. After ordering an anchor v, each validator

examines v's causal history H. Every skipped anchor in H is (re-)assigned a low score, and every ordered anchor in H is (re-)assigned a high score. Then, the new sequence of anchors is pseudo-randomly chosen based on the scores, with a validator with a high score more likely to be a leader in any given round. Note that while the validators use the same pseudo-randomness (so that they agree on the anchors), the computation is performed locally without extra communication.

Assigning higher scores to validators whose anchors get ordered ensures that future anchors correspond to faster validators, thus increasing their probability to be ordered. However, we ensure that the low score is non-zero, and thus underperforming validators also get a chance to be leaders. This crucially gives a temporarily crashed or underperforming validator a chance to recover its reputation.

D Multiple Anchors per Round

With pipelining, Shoal introduces an anchor in every round. As a result, in the best case, each anchor requires 2 rounds to commit and while non-anchor vertices require 3 rounds. Next, we present an approach to further optimize the latency for non-anchor vertices, which relies on retrospectively re-interpreting the DAG structure.

We could envision a protocol in which we iterate over more than one vertex in each round in a deterministic order and treat each vertex as an anchor. More specifically, for a vertex v in round r, we consider executing an instance of the underlying Narwhal-based consensus protocol $\mathcal{P}$ (i.e., DAG-Rider, Tusk, and Bullshark) starting from round r with v being the first anchor. This involves re-interpreting the existing DAG structure, and potentially letting it evolve, until a decision of whether v is ordered or skipped is locally made. If v is ordered by $\mathcal{P}$, then the causal history of v followed by v is added to the ordering determined by the new protocol. Otherwise, v is skipped and the protocol proceeds to considering a new instantiation of $\mathcal{P}$ from the next potential anchor (which may be in the same round).

A pseudocode in which all vertices are considered as anchors appears in Algorithm 2.

In the good case, each vertex that is considered as an anchor can be ordered in 2 rounds. However, the drawback of this approach is that if some validators are slow and a potential anchor takes many rounds to decide whether to skip or order, the progress of the whole protocol will be stalled. This happens because potential anchor vertices must be considered in an agreed-upon and deterministic order. As a result, a vertex that necessitates more rounds incurs a latency penalty for the subsequent vertices.

The above issue can potentially be mitigated by combining it with a leader reputation mechanism to select the vertices that are considered as potential anchors, making the bad case delays less likely. The other vertices can be ordered based on causal history as previously.

Algorithm 2 Every vertex as an Anchor

1: $r \leftarrow 0$
2: **while** *true* **do**
3: **for each** validator k **do**
4: let $v_{r,k}$ be a vertex by validator k in round r
5: let $F_{r,k} : R \rightarrow A$ be a known to all mapping from rounds to anchors such that $F_{r,k}(r) = k$
6: execute $\mathcal{P}$, select anchors by $F_{r,k}$, starting from r until the first ordered (not skipped) anchor A is determined.
7: **if** $A = v_{r,k}$ **then**
8: order A's causal history according to $\mathcal{P}$
9: $r \leftarrow r + 1$