Constant latency and finality for dynamically available DAG

Hans Schmiedel
The University of Sydney
hans.mailbox@tutamail.com

Runchao Han
Babylon Labs
me@runchao.rocks

Qiang Tang
The University of Sydney
qiang.tang@sydney.edu.au

Ron Steinfeld
Monash University
ron.steinfeld@monash.edu

Jiangshan Yu
The University of Sydney
jiangshan.yu@sydney.edu.au

***Abstract*—Directed Acyclic Graph (DAG) based protocols have shown great promise to improve the performance of blockchains. The CAP theorem shows that it is impossible to have a single system that achieves both liveness (known as dynamic availability) and safety under network partition. This paper explores two types of DAG-based protocols prioritizing liveness or safety, named *structured dissemination* and *Graded Common Prefix* (GCP), respectively.**

For the former, we introduce the first DAG-based protocol with constant expected latency, providing high throughput dynamic availability under the sleepy model. Its expected latency is 3Δ and its throughput linearly scales with participation. We validate these expected performance improvements over existing constant latency sleepy model BFT by running prototypes of each protocol across multiple machines.

The latter, GCP, is a primitive that provides safety under network partition, while being weaker than standard consensus. As a result, we are able to obtain a construction that runs in only 2 communication steps, as opposed to the 4 steps of existing low latency partially synchronous BFT. In addition, GCP can easily avoid relying on single leaders' proposals, becoming more resilient to crashes. We also validate these theoretical benefits of GCP experimentally.

We leverage our findings to extend the Ebb-and-Flow framework, where two BFT sub-protocols allow different types of clients in the same system to prioritize either liveness or safety. Our extension integrates our two types of DAG-based protocols. This provides a hybrid DAG-based protocol with high throughput, dynamical availability, and finality under network partitions, without running a standard consensus protocol twice as required in existing work.

1. Introduction

Bitcoin [1] introduced the use of blockchain to enable nodes in a permissionless environment to agree on an ordered set of transactions. In this setting, nodes are free to join or leave the network at any time. Consequently, Bitcoin guarantees liveness in a permissionless environment, referred to as dynamic availability. However, to ensure security, the network must be synchronous, meaning messages must be delivered within a known time bound. It follows that Bitcoin does not provide safety in the face of network partitions, where the network becomes asynchronous and message delivery time is unknown. A further known limitation of Bitcoin and its derivatives is performance: latency ranges from minutes to hours, and throughput is fewer than 10 transactions per second.

To remedy the limited performance of Bitcoin's blockchain consensus, traditional Byzantine Fault Tolerant State Machine Replication (BFT-SMR) techniques have been considered for the blockchain setting. Some BFT-SMR protocols are able to remain secure even under network partitions and asynchronous communication networks [2], [3], [4], [5]. However, a trade-off with BFT-SMR is that the number of active nodes in the system must be known, limiting dynamic availability. To address this challenge, the sleepy model [6] was introduced, formalizing the concept of permissionless networks. Recent advances in BFT within the sleepy model have achieved constant expected latency [7], [8] by regularly estimating for the current participation level.

While, ideally, both dynamic availability (liveness) and security during network partitions (safety) are achieved within a single protocol, the CAP theorem demonstrates this to be impossible [9]. Consequently, two distinct approaches have emerged: one prioritizes liveness with dynamic availability, and the other focuses on asynchronous safety.

Originally, both classes of protocols include transactions in a linear sequence of batches or blocks. However, utilizing a Directed Acyclic Graph (DAG) structure, blocks are processed concurrently which has significantly increased throughput for both approaches. This innovation allows throughput to scale with network capacity [10], [11], [12].

We highlight the following research gaps regarding DAG based protocols in each line of work.

DAGs when prioritizing liveness. As noted earlier, proposals for BFT in the sleepy model have achieved constant latency. Meanwhile, recent advances in DAG-based protocols have primarily focused on asynchronous and partially synchronous networks [11], [12], leaving the modern use of DAGs in BFT for the sleepy model largely unexplored.

Although DAG-based proposals expanding on Bitcoin's longest chain idea exist [10], [13], their latency is linear in certain security parameters. Achieving constant latency for high throughput DAG protocols within the sleepy model remains a significant open challenge.

DAGs when prioritizing safety. DAG-based protocols such as Narwhal/Tusk [11] and Bullshark [12] disseminate an unconfirmed DAG of blocks as a mempool, of which a subset is finalized. However, there are multiple approaches for the finalization process, and previous work has not provided a generic security analysis of finalization approaches in a given DAG protocol. For instance, one way to finalize Narwhal is by using a separate consensus primitive such as Hotstuff to agree on a block and all its ancestors, as presented by the authors. Alternatively, Tusk, which builds on top of Narwhal, selects leaders using a common coin but does not independently solve consensus. Similarly, Bullshark employs a round-robin leader schedule to finalize a subset of the DAG, but the finalization mechanism in isolation is not a consensus protocol in itself.

We observe that the common thread in the two latter protocols is a finalization mechanism that leverages the DAG structure to obtain consensus in the overall system. A key advantage of this approach is that it avoids the need to layer a full BFT protocol on top of the DAG dissemination mechanism. However, there is no composable treatment of DAG finalization in the literature beyond constructions specific to monolithic DAGs.

Contributions

In this work we address the research gaps previously highlighted, and leverage our findings to propose a hybrid protocol simultaneously catering to clients favoring liveness and clients favoring safety.

We introduce the first constant latency DAG based protocol in the sleepy model. Our protocol can commit a set of DAG blocks in 3Δ latency, where Δ represents the synchronous network's message delivery bound. This result not only advances upon existing constant latency sleepy model BFT, but also improves on existing DAG protocols whose latency linearly scales with Δ. We provide a summary comparison of our protocol with other dynamic availability protocols in Table 1. Additionally, we experimentally show the improved performance of our DAG protocol compared to Mahlki et al. constant latency protocol. We can observe that our DAG protocol indeed linearly scales throughput with the number of nodes, achieving drastic throughput improvements, and that it has lower expected latency.

Secondly, we introduce a generic primitive for DAG finalization, named *graded common prefix*, and an efficient construction. Graded common prefix allows a set of nodes to quickly agree on the subset of a DAG common to all honest nodes. Crucially, our primitive is lighter than standard consensus, enabling simpler protocols. For example, our construction runs in only 2 communication steps, compared to the 4 expected steps per decision required by low latency partially synchronous BFT [2], [3], [18]. It is also resilient to crashes within the supermajority bound, unlike many existing leader-based protocols for partially synchronous networks [2], [3], [5], [18]. We experimentally observe the resulting improved latency of our construction over a leader-based BFT, Hotstuff [5], with and without crashes. Graded common prefix can also be defined generically with sets of values instead of block-DAGs, which may be of independent interest for distributed protocols.

Thirdly, we propose a hybrid protocol that consists of two sub-protocols, namely a dynamically available sub-protocol and a partially synchronous finality sub-protocol. The former provides dynamic availability for clients prioritizing liveness, and the latter provides more conservative clients asynchronous safety. To accomplish this, we adapt the Ebb-and-Flow framework [19], which, roughly speaking, considers the sleepy model with a *Global Awake Time* (GAT), the time after which all honest nodes participate actively. GAT guarantees that the finality sub-protocol eventually progresses, as dynamic availability is not required after GAT. We show that graded common prefix is sufficient for finality after GAT, avoiding the need to perform standard consensus twice as in existing schemes. Complementing our hybrid protocol, we further relax the GAT assumption by introducing *Quorum Awake Time* (QAT). Unlike GAT, which assumes that all honest nodes are awake after this period, QAT requires only that a sufficient number of honest nodes are awake. To quantify the number of awake nodes after QAT, we leverage the recently introduced Mobile Crash Adaptive Byzantine (MCAB) model [20] that enables a fine-grained treatment of the upper bound on crashed/sleepy nodes for a protocol.

The rest of this paper is organized as follows. Section 2 introduces the system model and preliminaries used in this work. Section 3 presents our dynamically available DAG protocol and the formal definitions and properties it achieves. Section 4 introduces the graded common prefix primitive and a construction for it. Section 5 synthesizes our results into the hybrid protocol described previously, then presents QAT, the generalization of GAT. Section 6 contains the formal security proofs of our theorems and lemmas. Section 7 provides the experimental evaluation of our DAG protocol. Section 8 presents the literature related to this work. The conclusion summarizes our findings.

In summary:

- We propose a constant latency dynamically available DAG BFT protocol with 3Δ latency. Its high throughput is demonstrated experimentally.
- We introduce *graded common prefix* (GCP), a primitive that allows a set of nodes to agree on the subset of a DAG in common to honest nodes. Additionally, we provide a construction of GCP that runs in 2 communication steps. Its low latency and resilience to crashes is shown experimentally.
- We show that GCP is sufficient as a finality sub-protocol combined with a dynamically available sub-protocol, to allow clients to prioritize dynamically available liveness or asynchronous safety. In addi-

TABLE 1: Comparison of optimal resilience SMR in the sleepy model. k is the security level and a is the fraction of awake nodes.

	DAG structure*	Expected latency
Bitcoin [1]	✗	$O(k\Delta/a)$
Ouroboros/ Snow White [14], [15]	✗	$O(k\Delta/a)$
Pass et al. [6]	✗	$O(k\Delta/a)$
Goyal et al. [16]	✗	$O(k\Delta)$
Prism [10]	✓	$O(\Delta/a)$
Garay et al. [17]	✓	$O(\Delta/a)$
Momose et al. [7]	✗	16Δ
Mahlki et al. [8]	✗	4Δ
(This work)	✓	3Δ

* A protocol uses a DAG structure if it commits multiple blocks of the same height.

tion, we introduce and discuss a generalization of *Global Awake Time* of Ebb-and-Flow [19], the time after which honest nodes are assumed to be awake to allow asynchronous safety. Our generalization, *Quorum Awake Time*, relaxes the GAT assumption by leaving the possibility of a parametrized bound on the number of sleepy nodes.

2. System model and preliminaries

2.1. System model

We consider a set $\mathcal{N}$ of nodes, each equipped with a public key and a unique identifier. This includes the Proof-of-Stake setting where the smallest discrete amount of stake is represented by one node. Time advances in discrete slots and, for simplicity, nodes have access to synchronized clocks. Our results can include bounded clock drifts using a round transformation technique [7] at the cost of small added latency.

Node corruptions. Nodes may be controlled by an *adversary*, selected during the execution of a protocol. Nodes controlled by the adversary are considered Byzantine, and may behave arbitrarily. The remaining nodes are *honest*, and behave according to protocol specifications.

Network communication. Nodes communicate with each other through authenticated messages, within the limitations set by one of the following network assumptions.

- *Synchronous network.* All messages sent between honest nodes are delivered within a known number of time slots Δ. The order in which they arrive can be determined by the adversary.
- *Partially synchronous network.* Message delivery is determined by the adversary, as long as all messages sent between honest nodes are eventually delivered. Messages between honest nodes cannot be dropped or modified. After a *Global Stabilisation time* (GST), the network behaves as a synchronous network. GST is decided by the adversary, and unknown to honest nodes.

Sleepy model. The adversary may cause honest nodes to become *sleepy* at any time slot, during which the protocol is not executed and messages are delayed until the next slot where the node is *awake* again. Awake nodes behave according to protocol specifications. The number of awake nodes at any time slot t is denoted n_t. The unknown time after which all honest nodes in $\mathcal{N}$ are awake is called *Global Awake time* (GAT). GAT is decided by the adversary, and unknown to honest nodes.

Parameterized dynamic participation. We use the extended sleepy model [8] that allows corrupted nodes' participation to be dynamic. We define $f(T_f, T_b, t)$ as the number of corrupted nodes around time t for a time T_f forward and a time T_b backwards. For a system model's fraction β of corrupted nodes, we have $f(T_f, T_b, t) < \beta \cdot n_t$. It has been shown in existing work that $T_f = \infty$ and $T_b = O(\Delta)$ is necessary for consensus under dynamic participation without PoW [8]. We therefore consider $T_f = \infty$ and $T_b = O(\Delta)$.

Two system models. As it is impossible to achieve dynamic availability and asynchronous finality under the same protocol [9], we consider two different system models to allow clients to prioritize liveness or safety, as proposed by the Ebb-and-Flow framework [19]. A system model is defined by an adversary $\mathcal{A}$, an environment $\mathcal{Z}$, and a fraction β of corrupted nodes.

System model $(\mathcal{A}_1(\beta), \mathcal{Z}_1)$: In this system model, the network is partially synchronous and there exist a bounded GAT and a bounded GST.

System model $(\mathcal{A}_2(\beta), \mathcal{Z}_2)$: In this system model, the network is always synchronous (GST=0) and GAT can be unbounded (GAT $\to \infty$).

2.2. Preliminaries

Definition 1 (State machine replication protocol.)**.** *A state machine replication protocol* ***P*** *takes transactions as input, and outputs a ledger LOG. It is secure if it satisfies the following properties for a polynomial function T_{fin} of security parameter κ.*

- Safety*: if at time t an awake honest node reports a ledger LOG, then for every honest node that reports a ledger LOG$'$ after time $t' \leq t$, LOG $\preceq$ LOG$'$ or LOG$'$ $\preceq$ LOG where $\preceq$ denotes a prefix relation.*

- Liveness*: a transaction given as input to all honest nodes continuously from time t to time $t+T_{fin}$ will be reported in an honest node's output LOG after time $t+T_{fin}$.*

Definition 2. *A flexible protocol is a pair of SMR protocols ($\boldsymbol{P}_1$,$\boldsymbol{P}_2$) taking the same transactions as inputs and outputting the ledgers LOG_1 and LOG_2 respectively.*

Negligible function. A negligible function refers to a function $negl(x)$ where for every $y \in \mathbb{N}$, there exists an $z \in \mathbb{N}$ such that $negl(x) < 1/x^y$ for all $x \geq z$.

Definition 3 (Ebb-and-flow protocol. [19])**.** *A (β_1, β_2)-secure Ebb-and-flow protocol $\boldsymbol{P}$ is a flexible protocol ($\boldsymbol{P}_{da}$,$\boldsymbol{P}_{fin}$) with the same input transactions and outputs an available ledger LOG_{da} and a finalized ledger LOG_{fin}, such that for security parameter κ:*

- *Finality: Under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$, LOG_{fin} is safe at all times, and there exists a constant C such that LOG_{fin} is live after $C(\max\{GST, GAT\} + \kappa)$ except with probability negligible in κ.*
- *Dynamic availability: Under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$, LOG_{da} is a secure SMR output except with probability negligible in κ.*
- *Prefix: For any honest node i and time $t > 0$, $LOG^t_{fin,i}$ is a prefix of $LOG^t_{da,i}$.*

3. Dynamically available DAG

This section presents the first constant latency dynamically available DAG-based protocol in the sleepy model, introducing the possibility of high throughput while maintaining latency comparable to existing sleepy model BFT [8]. Our construction is secure under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$. We first summarize Synchronous Hotstuff [21] and the intuition behind its design that our protocol is based on.

Protocol flow. A leader proposes a block and honest nodes vote for it if its valid. Honest nodes can then commit after waiting 2Δ, provided no conflicting vote was received. **Security intuition.** Since a synchronous network is considered, it is guaranteed that if an honest node commits 2Δ after voting, all honest node receive the proposal and there is no conflicting commitment.

Considering the sleepy model. Waiting 2Δ for conflicts instead of collecting a fixed number of votes, a quorum, to advance is an especially interesting technique in the sleepy model, where fixed quorums are not useful. However, in Synchronous Hotstuff, potentially faulty leaders must still be dealt with, in a process called view-change that uses fixed quorums.

From single leader to DAG proposals. For our protocol, we leverage Synchronous Hotstuff's technique of waiting 2Δ after voting to detect conflicts in dynamic participation and guarantee consistency. By having every node propose a block to the block-DAG, we are able to avoid the fixed quorum based view change of Synchronous Hotstuff, guaranteeing progress in the sleepy model.

3.1. Block-DAGs and structured dissemination

We define a Directed Acyclic Graphs of blocks (block-DAG) which we use as a foundation for our DAG based protocols. Formally defining block-DAGs allows us to reason about a generic DAG protocol without specifying a specific algorithm. Block-DAGs can be seen as the generalization of the blockchain data structure.

Definition 4. *block-DAG. A block-DAG $D = \{\mathcal{E}, \mathcal{B}\}$ comprises a set $\mathcal{E}$ of directed edges and a set $\mathcal{B}$ of vertices, as follows:*

- *A directed edge $(B \rightarrow B') \in \mathcal{E}$ represents a transitive relation of B referencing B', where $B, B' \in \mathcal{B}$. B' is a parent of B, denoted $B' \in B.parents$.*
- *A vertex $B \in \mathcal{B}$ is called a block. Every block references at least one other vertex, except the genesis block $B_{genesis}$ where $B_{genesis}.parents = \varnothing$. The height of a vertex is $B.height = h + 1$, where $h = \max(B.parents.height)$ and $B_{genesis}.height = 0$. B is a tip if $\nexists B' \in \mathcal{B}$ s.t. $B \in B'.parents$. Each block B has an author $B.author$ and $B_{genesis}.author = \varnothing$.*
- *Extension: A block-DAG $D' = \{\mathcal{E}', \mathcal{B}'\}$ is an extension of D, denoted $D \subseteq D'$, if $\mathcal{E} \subseteq \mathcal{E}'$ and $\mathcal{B} \subseteq \mathcal{B}'$.*
- *Height: The height h of D is $\max(\{B.height | B \in \mathcal{B}\})$.*
- *Conflict: D and D' are conflict if $(D \nsubseteq D') \wedge (D' \nsubseteq D)$, denoted $D \nsim D'$. Otherwise, we use $D \sim D'$.*

A block B, contains referenced blocks $B.parents$ which recursively reference blocks until $B_{genesis}$, representing a block-DAG whose highest height block is B. We can therefore uniquely represents a sub-graph of D with height h by the set of identifiers for blocks in D of height h.

We define properties for structured dissemination, guaranteeing that a distributed set of nodes maintaining a block-DAG is able to obtain a consistent, growing view of the block-DAG.

Definition 5 (Structured dissemination.)**.** *Structured dissemination with parameters (k, τ, t, μ) takes transactions as input and outputs a block-DAG. A structured dissemination protocol is secure if it satisfies the following properties:*

- *δ-Consistency: If a block B of height h is considered consistent (called confirmed) in an honest node's block-DAG at time t, B and its ancestors are confirmed in any honest node i's block-DAG D_i at time $t+\delta$, and no honest node confirms a block of height h after $t+\delta$.*
- *τ, t-DAG growth: any honest node's view of the block-DAG at time $t'+t$ must include $\tau \cdot t$ additional blocks proposed by honest nodes compared to its view at time t'.*
- *μ-DAG quality: any honest node's view of the block-DAG includes at least a fraction μ of blocks proposed by honest nodes.*

- *External Integrity: If a block B is consistent, $condition(B) = 1$ holds for the protocol-specific external integrity condition.*

Obtaining total order. With δ-consistency among honest nodes, confirmed blocks can be ordered by height. In the blockchain special case, an ordered ledger is obtained trivially since we have only one confirmed block per height. In a block-DAG, a height h may have multiple confirmed blocks. However, with δ-consistency, honest nodes confirm the same set $\mathcal{B}$ of blocks for height h. Once blocks in $\mathcal{B}$ are confirmed, a simple rule is therefore already enough to totally order them, for example ordering based on block proposers' identifiers.

Designing more intricate block-DAG ordering rules that may achieve application-specific fairness requirements is orthogonal to this work.

Example in the literature. To illustrate our block-DAG and structured dissemination definitions in the context of existing DAG based protcols, we show that the popular Narwhal [11] DAG mempool layer is a block-DAGs that satisfies the properties of structured dissemination under $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT in Lemma 1. Recall that $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT has a synchronous network and honest nodes are awake, enabling Narwhal to achieve structured dissemination properties while not guaranteeing total order in its original, asynchronous network system model. To translate Narwhal's nomenclature into a block-DAG, consider the following: a block is considered confirmed when it is *certified*, a block b extends a block b' if it includes its certificate, the height of a block is the round in belongs to.

Lemma 1. *Narwhal run by n nodes is a structured dissemination of a block-DAG with $\Delta-$consistency, $\frac{2n}{9\Delta}$, 3Δ-DAG growth, and $1/2$-DAG quality under $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT where $\beta_2 < 1/3$.*

Proof. After GAT, the system model assumptions are stronger to Narwhal's original model, and the above lemma can easily be proven using Narwhal's established properties [11]. See Appendix C for the full proof.

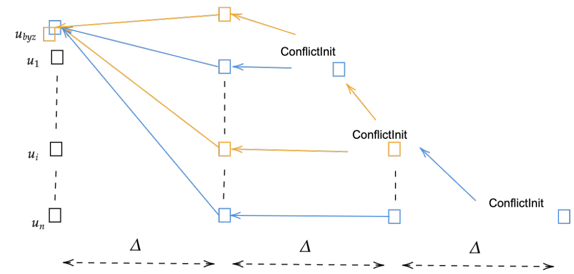
3.2. Overview

We outline the main components of our protocol and provide brief intuition behind its components

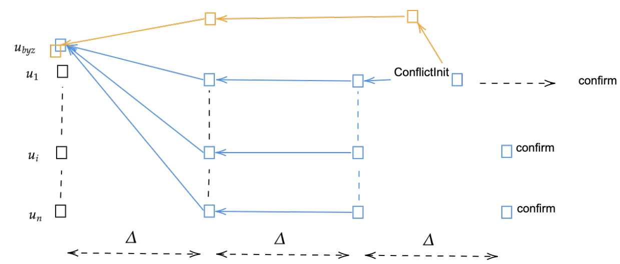
Propose. Every Δ slots, nodes propose a block of transactions sent by clients.

Vote. When proposing a block of height $h + 1$, nodes include as references all received blocks with height h, and start their confirmation timer for the set of referenced blocks at height h.

Confirm. A block of height h proposed at time t is considered confirmed 2Δ slots after the confirmation timer for height h has started if no conflicts are detected for blocks of height h. The confirmation timer starts at time $t+\Delta$ when proposing a block of height $h + 1$, such that, by the time $t + 3\Delta$, it is confirmed and received by all honest nodes.



(a) Conflict is first received before 2Δ.



(b) Conflict is first received after 2Δ.

Figure 1: Two cases of conflicting proposals.

Order blocks. As mentioned in the previous section, confirmed blocks of the same height h can be deterministically ordered by their proposer's identifiers.

Resolving conflicts. Consider a Byzantine node's conflicting block proposals B and B'. Simply discarding both is not possible as some honest nodes may consider B confirmed as illustrated in Figure 1b. If both are discarded by node u_1, node u_i may confirm the blue block.

When an honest node broadcasts a conflict message including B and B', v is provably Byzantine. To handle equivocating proposals by a Byzantine node $B.author$, nodes concurrently run a generic Byzantine agreement (BA, recalled in Appendix A) sub-protocol ByzAgreement with the first received block B as input. $B.author$ serves as an identifier for the Byzantine agreement protocol instance. Blocks of a height h are only confirmed once all potential instances of ByzAgreement for height h are completed.

ByzAgreement can be instantiated with existing sleepy model protocols from the literature [7], [8], [22], and benefits from future advances in Byzantine agreement in the sleepy model. Additional timers may be required for the specific ByzAgreement implementation.

To understand why all conflicts are resolved, consider the two cases depicted in Figure 1. In Figure 1a, the first honest node u_1 receiving the conflict does so before 2Δ, guaranteeing that all honest nodes receive the conflict before honest nodes confirm a proposal. Some nodes may have received the orange block first while others may have received the blue block first. ByzAgreement then guarantees agreement for the blue or orange block.

In Figure 1b, the first honest node u_1 receiving the conflict does so after 2Δ. Honest nodes have seen blue first when entering BYZAGREEMENT, and some may have confirmed blue already. As all honest nodes input blue to BYZAGREEMENT, blue is the output.

Handling late blocks A node u may also receive a block B of height h at time $t' \geq t + \Delta$ from a Byzantine node where t is the time that u proposes a block of height h. Here, u knows that $B.author$ is Byzantine, but unlike in conflicts, it is not straightforward to prove that u hasn't received h before $t+2\Delta$. Node u broadcasts an UNRECEIVED message for B including all received blocks of height $h+1$. Upon receiving an UNRECEIVED message for B, nodes forward the message and discard B if it is referenced by a minority of known blocks for height $h+1$.

A node u checks all blocks of height $h+1$ received by time $t+2\Delta$ and if none reference B, broadcasts the set $\mathcal{B}_{h+1}$ of all received blocks of height $h+1$ received by time $t+3\Delta$ in a UNRECEIVED message. Upon receiving an UNRECEIVED message, a node rebroadcasts the UNRECEIVED message and discards B if it is referenced by a minority of blocks received for height $h+1$ when considering $\mathcal{B}_{h+1}$ in addition to their received blocks.

A Byzantine node can not force honest blocks of height h to be discarded as they are referenced by all honest nodes at height $h+1$, the majority. A Byzantine block can achieve a majority only if at least one honest node references it in a block of height $h+1$, guaranteeing that it is shared among all honest nodes before $t+2\Delta$.

3.3. Full protocol

Protocol flow We provide an overview of $\mathbf{P}_{sd}$ from Algorithm 1. The local view of the full block-DAG D and confirmed block-DAG D_{con} are initialized by the genesis block (Line 1-3). Each time slot, network messages are processed. Newly received blocks are processed according to Lines 25-40 described below and conflict messages according to Lines 41-61 describe below (Line 5-8).

Every Δ time slots, RESETBLOCKTIMER(T) resets the block timer, which can be accessed by BLOCKTIMER(). Upon RESETBLOCKTIMER(T), nodes broadcast a block of height k extending all received blocks of height $k-1$ (Line 9-13). The timer for consistency is started, and the height to confirm is saved (Line 14-15). Nodes consider a block B of height h proposed by node u consistent Δ time after referencing B in its own block proposal at height $h+1$, unless a CONFLICTINIT message is received. If a CONFLICTINIT message is received for a block of height h, the consistency timer is reset to allow time to resolve conflicts (Line 17-21).

New received blocks are processed in SDPROCESSNEWBLOCKS. If a block $B' \neq B$ of height h proposed by u is received before receiving B, they are broadcast in a CONFLICTINIT message including B' and B, and enter a BYZAGREEMENT(B') instance with input B' as B' was received first (Line 26-29). Otherwise a valid block whose height is at least the consistency height is added to the block-DAG (Line 30-33). A block B is valid if $condition(B) = 1$ holds for the application-specific external integrity condition $condition(B)$. If a block of height h is received 2Δ time after proposing a block of height h, an UNRECEIVED message is broadcast including B and all blocks of height $h+1$.

CONFLICTINIT and UNRECEIVED messages received at slot t are processed in PROCESSCONFLICTMESSAGESSLOT For a CONFLICTINIT message containing B and B', nodes forward CONFLICTINIT and enter the black-box BYZAGREEMENT protocol (Line 42-45).

For an UNRECEIVED message containing the block B and the set $\mathcal{B}_{B.height+1}$ of blocks, a node adds blocks in $\mathcal{B}_{B.height+1}$ to its set $\mathcal{B}'_{B.height+1}$ of blocks of height $B.height+1$ and forwards the UNRECEIVED message with $\mathcal{B}'_{B.height+1}$ replacing $\mathcal{B}_{B.height+1}$ (Line 48-52). If B is referenced in a minority of blocks in $\mathcal{B}'_{B.height+1}$, B is discarded from the block-DAG D (Line 53-54).

If a BYZAGREEMENT outputs a block B_{out}, all other blocks of height $B_{out}.height$ by $B_{out}.author$ are discarded from the block-DAG D (Line 57-60).

Algorithm 1 Dynamically available structured dissemination protocol $\mathbf{P}_{sd}$

1: **procedure** MAIN()
2: $D, D_{con} \leftarrow B_{genesis}$ // initialize block-DAG D and confirmed block-DAG D_{con} with genesis block
3: $h_{con} \leftarrow 0$ // initialize to-confirm block height
4: **for** time slot $t \in \mathbb{N}$ **do**
5: PROCESSINCOMINGNETWORKMESSAGES()
6: $\mathcal{B}_{new} \leftarrow$ GETLATESTBLOCKS()
7: $D \leftarrow$ SDPROCESSNEWBLOCKS($D, \mathcal{B}_{new}, height_{con}$) // see line 25
8: $D \leftarrow$ PROCESSCONFLICTMESSAGESSLOT(t, D) // see line 36
9: **if** BLOCKTIMER(t) $= 0$ **then**
10: $B \leftarrow$ CREATEBLOCK($clienttransactions$)
11: $B.parents \leftarrow$ { All blocks of height $D.height$ }
12: BROADCAST($< Propose, B >$)
13: RESETBLOCKTIMER(t)
14: $h_{con} \leftarrow B.height - 1$
15: STARTCONSISTENCYTIMER(t, h_{con}) // start timer for height h_{con}
16: **end if**
17: **if** (CONSISTENCYTIMER($t, \mathcal{B}_{con}$) $= 0$) $\wedge$ ($\exists$BYZAGREE with blocks in $\mathcal{B}_{con}$ **then** // check for ongoing conflict resolution instances
18: RESETCONSISTENCYTIMER($t, \mathcal{B}_{con}$)
19: **else if** CONSISTENCYTIMER($t, \mathcal{B}_{con}$) $= 0$ **then**
20: $D_{con} \leftarrow D_{con} \cup \mathcal{B}_{con}$
21: **end if**
22: **end for**
23: **end procedure**

24: **procedure** SDPROCESSNEWBLOCKS($D, \mathcal{B}_{new}$)
25: **for** $B \in \mathcal{B}_{new}$ **do**
26: **if** $\exists B' \in D$ s.t. $B'.author = B.author \wedge B'.height = B.height$ **then**
27: // block B' received first
28: BROADCAST($<$ CONFLICTINIT$, B', B >$)
29: BYZAGREEMENT(B') // input B' to BA
30: **else if** $condition(B) = 1$ **and** $B.height \geq D.height$ **then**
31: // check external integrity condition on B
32: // honest blocks of lower height are indirectly included by

```
                newer blocks
                D ← D ∪ B
            else if Δ_{B.height} > 2Δ where Δ_{B.height} denotes the time since block B of height B.height was proposed then
                ℬ_{B.height+1} ← All blocks in D of height B.height + 1.
                BROADCAST(< UNRECEIVED, B, ℬ_{B.height+1} >)
            end if
        end for
        return D
end procedure

procedure PROCESSCONFLICTMESSAGESSLOT(t, D)
        for all new < CONFLICTINIT, B', B > messages do
            if CONFLICTINIT not sent for B'.author and B'.height then
                BROADCAST(< CONFLICTINIT, B', B >
                BYZAGREEMENT(B')
            end if
        end for
        for all new < UNRECEIVED, B, ℬ_{B.height+1} > messages do
            B'_{B.height+1} ← All blocks of height B.height + 1
            if UNRECEIVED not sent for B then
                BROADCAST(< UNRECEIVED, B, ℬ'_{B.height+1} >)
            end if
            if B referenced by minority of blocks in B'_{B.height+1} then
                D ← D \ B
            end if
        end for
        upon BYZAGREEMENTOUTPUT() do
            B_out ← GETBYZAGREEMENTVALUE()
            ℬ_del ← {All blocks of height B_out proposed by B_out.author}
            D ← D \ ℬ_del
        return D
end procedure
```

The security of Algorithm 1 comes from the guarantee that conflicts will be detected for all honest nodes and handled with BYZAGREEMENT. Similarly, UNRECEIVED guarantee that honest nodes can discard or keep blocks that were not properly broadcast. The security analysis is given in Section 6.1.

Theorem 1. *$\boldsymbol{P}_{sd}$ is a structured dissemination protocol under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$ and $T_b = 15\Delta$.*

Proof. See Section 6.1.

We also show that Algorithm 1 can recover security after GST even if it was previously not secure due to an unstable network.

Theorem 2. *$\boldsymbol{P}_{sd}$ is a structured dissemination protocol under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_2 < 1/2$ after GST.*

Proof. See Section 6.1.

4. Graded Common Prefix

In this section, we utilize the block-DAG structure to introduce a novel primitive, the Graded Common Prefix. This primitive is weaker than standard consensus but still achieves finality for a subset of the blocks in a block-DAG, in partially synchronous networks. Unlike Graded Byzantine Agreement [23] that considers single values, graded common prefix considers block-DAGs, represented by sets of values, and outputs the common prefix amongst honest node's block-DAGs.

Intuition. To weaken standard consensus, agreement is relaxed by introducing grades 1 and 2. An output of grade 2 must be obtained with grade ≥ 1 for all honest nodes and no output of grade 2 can conflict. As outputs are block-DAGs, an output may be a subset of another, hence different but not conflicting.

By running GCP sequentially, a block-DAG D of grade 2 can be considered final even if some honest nodes have only obtained grade ≥ 1, as all honest nodes will propose D in the subsequent instance.

4.1. Definition

We define graded common prefix that is specific to block-DAG inputs and outputs. We separate the *weak validity* property from validity as it is achievable before $\max\{GST, GAT\}$ by a construction such as Algorithm 2 while validity is achieved after $\max\{GST, GAT\}$.

Definition 6 (Graded Common Prefix. (GCP))**.** *In a graded common prefix protocol, every node $i \in 1, ..., n$ can input a block-DAG D_i, and outputs a graded block-DAG where each block a grade $\{1, 2\}$. A graded common prefix protocol satisfies the following properties:*

- *Termination: Every honest node terminates.*
- *Validity: If all honest nodes propose a block-DAG containing block-DAG D, $D' \supseteq D$ holds for any output D' by an honest node with grade 2.*
- *Weak Validity: If all honest nodes propose a block-DAG containing block-DAG D, $D' \supseteq D$ holds for any output D' by an honest node with grade ≥ 1.*
- *Agreement: If an honest node outputs grade 2 for a DAG D then all honest nodes output D and with grade ≥ 1, and no honest node outputs a DAG $D' \not\sim D$ with grade 2.*

According the the properties of graded common prefix, different honest nodes may output different block-DAGs and different grades for the same blocks.

We additionally provide a generic version where nodes input a set of arbitrary values, such that the output is a graded set in Appendix B that may be of independent interest for BFT.

4.2. Partially synchronous graded common prefix

We propose a protocol to solve graded common prefix in a partially synchronous network for $n > 3f$ nodes, defined as pseudocode in algorithm 2.

Protocol flow. We provide a description of the protocol from Algorithm 2. First, a node u broadcasts its input D_u to all other nodes in a *propose* message (Line 2). Second, nodes wait for at least $n - f$ *propose* messages, and broadcast a *vote* with a block-DAG D_{vote} including every block-DAG contained in at least $n - f$ propose messages, or containing $\varnothing$ otherwise (Line 3-9). Thirdly, honest nodes wait for at least

$n-f$ vote messages. An honest node outputs a block-DAG D_2 with grade 2 that includes every block-DAG contained in at least $n-f$ vote messages (Line 11-12). An honest node outputs a block-DAG D_1 with grade 1 that includes every block-DAG contained in at least 1 vote message (Line 13-14). Both D_1 and D_2 may be $\varnothing$.

Security intuition. We provide some intuition on the components of our protocol that guarantee GCP agreement. For D to be included in the output D_2 with grade 2, a quorum of $n-f$ vote messages containing D is required. With $n > 3f$ nodes, the quorum implies that all honest node receive at least 1 vote message containing D even when the network is asynchronous before GST. A single vote should therefore imply an output of grade 1 to guarantee agreement at all times.

The quorum requirement also ensures that a quorum for conflicting block-DAG is not obtainable, achieving the second agreement condition. However Byzantine nodes may attempt to broadcast votes for an arbitrary block-DAG D', that would also obtain grade 1. To prevent this scenario, a valid vote for D contains the set of $n-f$ received propose messages whose block-DAGs include D.

Theorem 3. *Algorithm 2 achieves the properties of graded common prefix as follows: validity after* $\max\{GST, GAT\}$*, termination in* 2 *network delays after* GAT*, and agreement and weak validity for* $n \geq 3f+1$ *nodes and up to* f *Byzantine nodes in a partially synchronous network.*

Proof. To show the security of Algorithm 2, we use quorum intersection arguments for each property of GCP. See Section 6.2 for the full proof.

Algorithm 2 Partially synchronous graded common prefix for a node u with input D_u.

1: **procedure** PsGradedCommonPrefix(D_u)
2: Broadcast($<$ Propose, $D_u >$)
3: **upon** receiving $n-f$ Propose messages **do**
4: **if** $\nexists D$ s.t. at least $n-f$ Propose messages include D **then**
5: Broadcast($<$ Vote, $\varnothing >$)
6: **else**
7: $D_{vote} \leftarrow \bigcup\{D | D$ included in at least $n-f$ Propose messages $\}$
8: $\Sigma_{vote} \leftarrow \{$All Propose messages including D s.t. $D \subseteq D_{vote}\}$
9: Broadcast($<$ Vote, $D_{vote}, \Sigma_{vote} >$)
10: **end if**
11: **upon** receiving $n-f$ Vote messages **do**
12: $D_2 \leftarrow \bigcup\{D | D$ included in at least $n-f$ Vote message $\}$
13: OutputGrade2(D_2)
14: $D_1 \leftarrow \bigcup\{D | D$ included in at least 1 Vote message $\}$
15: OutputGrade1(D_1)
16: **end procedure**

5. Hybrid protocol with block-DAG

Simultaneously achieving dynamic availability and partition tolerance is a known impossibility due to the CAP theorem [9]. However, it is possible to enable clients to decide between prioritizing liveness or safety by running two sub-protocols concurrently. We adapt the Ebb-and-Flow framework [19] to obtain such a hybrid protocols. Proposed constructions run a dynamically available protocol for liveness-prioritizing clients, and a partially synchronous BFT protocol for safety-prioritizing clients. In total, consensus is performed twice, once in each setting.

We present the class of secure Ebb-and-Flow protocols $\mathbf{P}$ that leverages a partially synchronous graded common prefix protocol $\mathbf{P}_{gcp}$ such as Algorithm 2, in parallel with a standard black box dynamically available protocol $\mathbf{P}_{da}$, of which many instantiations are available in the literature [1], [10], [14], [15]. $\mathbf{P}_{da}$ is not limited to longest chain protocols, and can also be instantiated with potentially high throughput structured dissemination protocols presented in Section 3.1.

We therefore avoid performing standard consensus twice, as $\mathbf{P}_{gcp}$ leverages the information contained in the block-DAG (blockchains being special case of block-DAGs) created by $\mathbf{P}_{da}$.

Protocol instantiation. We assume the ledger LOG_{da} produced by $\mathbf{P}_{da}$ to be secure in the system model $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ and in the system model $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ only after GST. This is the case for our structured dissemination protocol in Section 3.1 as shown in Theorem 1 and Theorem 2. Alternatively, longest chain protocols have also been shown to be secure in both system models [19].

We also assume $\mathbf{P}_{gcp}$ to solve graded common prefix in the system model $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ and output a grade 1 block-DAG D_{g1} and a grade 2 output D_{fin}.

Protocol flow. Transactions are input to $\mathbf{P}_{da}$, which outputs LOG_{da} and a confirmed block-DAG D^t_{da}. If all calls to $\mathbf{P}_{gcp}$ have terminated, and $D^t_{da}.height > D_{fin}.height$ for the finalized block-DAG D_{fin}, nodes input the union of D^t_{da} and the last grade 1 output D_{g1} to the graded common prefix protocol $\mathbf{P}_{gcp}$. A grade 2 output of $\mathbf{P}_{gcp}$ becomes the finalized ledger LOG_{fin}, and a grade 1 output is saved in local state as LOG_{g1} (Line 13-16).

Security Intuition.

The secure finality with $\mathbf{P}_{gcp}$ comes from its agreement and weak validity properties. By guaranteeing that all honest nodes output D with grade ≥ 1 if D is output with grade 2 due to agreement, all honest nodes will include D in their input for next instance. As a result, weak validity ensures that any future output will include D. The security of $\mathbf{P}_{da}$ is easier as Ebb-and-Flow protocols with a secure dynamically available protocol have already been proven [19]. The full proof is included in Section 6.3.

Theorem 4. $\mathbf{P}$ *is a secure* $(\frac{1}{3}, \frac{1}{2})$*-Ebb-and-Flow protocol*

Proof. We prove that $\mathbf{P}$ is a secure Ebb-and-Flow protocol, analyzing the security of each sub-protocol is shown under both system models. See Section 6.3 for the full proof.

Algorithm 3 $\mathbf{P}$ protocol using a dynamically available protocol $\mathbf{P}_{da}$ and graded common prefix $\mathbf{P}_{gcp}$

1: **procedure** GcpSlot(LOG)
2: **if** $D_{fin}.height < D.height$ **and** AllGcpTerminated() **then**
3: PsGradedCommonPrefix(D)
4: **end if**
5: **end procedure**

procedure MAIN()
  if $D_{fin}.height < D.height$ **and** ALLGCPTERMINATED() **then**
    PSGRADEDCOMMONPREFIX(D)
  end if
    DASLOT(t)
    $D^t_{da} \leftarrow$ DACONFIRMEDDAG(t)
    GCPSLOT(D^t_{da})
    $D_{g1} \leftarrow$ GCPGRADE1DAGS(t)
    $D_{fin} \leftarrow$ GCPGRADE2DAGS(t)
  end for
end procedure

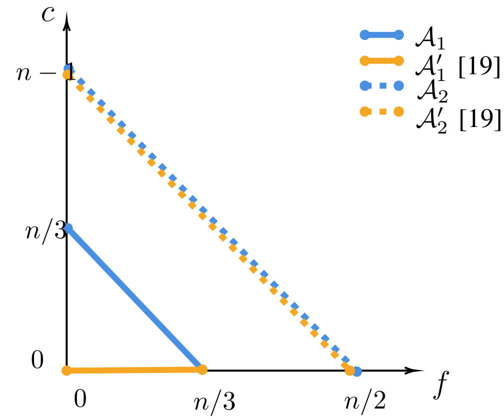


Figure 2: The highest number c of mobile crash faults and the number f of Byzantine faults allowed for protocols against $\mathcal{A}'_1$ after QAT and $\mathcal{A}'_2$ in our work, and their respective equivalents from Ebb-and-Flow $\mathcal{A}_1$ after GAT and $\mathcal{A}_2$ [19]

.

5.1. Generalizing GAT with the MCAB model

In a real system, the assumption that GAT exists, the time after which all honest nodes are awake, may be considered too strong. This is notably the case in public blockchain deployments over the internet where nodes as public servers may always be targeted by denial-of-service attacks.

From the perspective of protocol design, the GAT assumption guarantees that enough honest nodes participate, for example to reach a quorum of votes. We highlight that *enough* awake honest nodes does not mean that *all* honest nodes must be awake.

To weaken the GAT assumption while still providing a time with *enough* participation, we define a system model using the mobile crashes of the MCAB model [20]. We obtain a more fine-grained treatment of GAT, the time from which participation is lower bounded, which we call Quorum Awake Time (QAT).

5.1.1. Definitions. Mobile crashes. Nodes that are mobile crashed are nodes whose outgoing and incoming messages are not delivered. Mobile crashed nodes may become cured, causing all previously undelivered outgoing and incoming messages to deliver.

Node corruptions. Up to $f+c$ nodes may be corrupted by the adversary, where f is the number of adaptive Byzantine faults, nodes gradually chosen by the adversary during protocol execution that may behave arbitrarily, and c is the number of mobile crashed nodes chosen by the adversary for every slot. A node that was crashed in slot t and not in slot $t+1$ eventually receives all incoming messages from slot t. The remaining nodes are *honest*, and behave according to protocol specifications. For ease of notation w.l.o.g. we consider nodes numbered with $i \in [1, n-(f+c)]$ to be honest.

Quorum Awake Time. There is an unknown time after which $f+c \leq n \cdot \alpha$ called *Quorum Awake time* (QAT) for a resilience fraction α. QAT is decided by the adversary, and unknown to honest nodes.

Two system models. System model $(\mathcal{A}_1(\alpha)', \mathcal{Z}_1)$: In this system model, the network is partially synchronous, $c \leq n-1$ and $f \leq (n-c) \cdot \alpha$, and there exists a bounded QAT and a bounded GST.

System model $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$: In this system model, the network is always synchronous (GST=0), $c \leq n-1$ and $f \leq (n-c) \cdot \alpha$, and QAT can be unbounded (QAT $\to \infty$).

Ebb-and-Flow with MCAB liveness We say that a (β_1, β_2)-secure Ebb-and-Flow protocol has *MCAB liveness* if finality holds under $(\mathcal{A}_1(\alpha)', \mathcal{Z}_1)$ and $(\mathcal{A}_1(\alpha), \mathcal{Z}_1)$, and dynamic availability holds under $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$ and $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$.

5.1.2. Comparison with Ebb-and-Flow. Mobile crashes behave in the same way as adaptive sleepiness from the sleepy model [6]: both allow the adversary to prevent a node's messages from delivering, and allow a node to be cured/awoken to become honest again. In the sleepy model, there is no parameter for the number of simultaneous sleepy nodes, while the MCAB model [20] provides a fine-grained number of mobile crashes at any time.

As a result, the key difference with the system models of Ebb-and-Flow [19]'s is that after QAT in our models, there is still the possibility of mobile crashes, unlike after global awake time where the all nodes are either honest or Byzantine. Note that $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$ describes the same model as $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ if QAT $\to \infty$, and that setting $c = 0$ after QAT for $(\mathcal{A}'_1(\alpha), \mathcal{Z}_1)$ results in $(\mathcal{A}_1(\alpha), \mathcal{Z}_1)$ We visualize the difference between $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$, $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$, $(\mathcal{A}_1(\alpha), \mathcal{Z}_1)$, and $(\mathcal{A}_1(\alpha)', \mathcal{Z}_1)$ in Figure 2.

5.1.3. Our protocols in the MCAB model. Deterministic leader based protocols such as Hotstuff have been shown to lose liveness in the presence of mobile crashes [20], implying that even after $\max(QAT, GST)$, Snap-and-Chat protocols using Hotstuff or Streamlet are not able to guarantee liveness. In contrast, every step in our constructions for GCP involve every node performing the same protocol step. It is easy to see that these fit the criteria for *abundant roles*, shown to be sufficient for liveness in the MCAB model [20]. Our protocol $\mathbf{P}$ is therefore an Ebb-and-Flow protocol with MCAB liveness, extending our results in Theorem 1 to $\mathcal{A}'_2$.

Similarly, Theorem 4 can be extended to $\mathcal{A}_1'$ and $\mathcal{A}_2'$ as long as the sub-protocols $\mathbf{P}_{da}$ and $\mathbf{P}_{fin}$ only include protocol steps that are *abundant roles* or *concealed roles*, which guarantees liveness in the MCAB model [20].

6. Security analysis

6.1. Constant latency dynamically available structured dissemination

To analyze the security of $\mathbf{P}_{sd}$, we analyze how conflicts and late blocks are resolved under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$ before showing that our full protocol achieves structured dissemination.

Lemma 2. *Unreceived message agreement. If an honest node discards a block B of height h after receiving an* UNRECEIVED *message, all honest nodes discard B.*

Proof. We proceed by contradiction. Assume u discards B and, w.l.o.g., is the first honest node that broadcasts an UNRECEIVED message at time $t_u \geq t + 2\Delta$ where t is the time B was proposed. Also assume that v does not discard B until it is consistent. Since v does not discard B, we have two cases:

- Case 1: v has not received any UNRECEIVED message, a contradiction as u rebroadcasts UNRECEIVED message and it is received within Δ slots.
- Case 2: v has received at least $f_{t+\Delta} + 1$ blocks of height $h + 1$ referencing B by time $t + 2\Delta$, where t is the time honest nodes propose blocks of height h. This implies that at least one honest node has proposed a block of height $h + 1$ at time $t + \Delta$, received by all honest nodes before $t + 2\Delta$ by the synchronous assumption. As a result, u receives B before $t + 2\Delta < t_u$, and does not send a UNRECEIVED message, a contradiction.

Lemma 3. *Timing of conflict instance. For any two honest nodes starting* BYZAGREEMENT *for an equivocating node $B.author$ at times t, t', we have $|t - t'| \leq \Delta$.*

Proof. Honest nodes always broadcast a CONFLICTINIT message before starting BYZAGREEMENT for an equivocating node $B.author$. By assumption, all honest nodes receive the CONFLICTINIT message in at most Δ slots. All honest nodes will therefore start BYZAGREEMENT at most Δ slots after the first honest node that starts BYZAGREEMENT.

Lemma 4. *Security of* BYZAGREEMENT *Any instance of* BYZAGREEMENT *is secure in Algorithm 1 under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$ and $T_b = 15\Delta$.*

Proof. Our conflict resolution protocol BYZAGREEMENT can be instantiated with a single view of the atomic broadcast protocol [8] in sleepy model with backward time $T_b = 11\Delta$. The confirmation of blocks of the same height h depends on the proposals at height h sent synchronously at time t, on the messages sent between t and $t + 3\Delta$, and all messages of BYZAGREEMENT instances where $B.height = h$. All honest nodes start instances of BYZAGREEMENT where $B.height = h$ between t and $t + 4\Delta$ by Lemma 3. We therefore have $T_b = 4\Delta + T_{bm} = 15\Delta$.

6.1.1. Full protocol.

Lemma 5. *Consistency. $\mathbf{P}_{sd}$ achieves $\delta-$consistency of a structured dissemination protocol under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$, $\delta = 3\Delta$, and $T_b = 15\Delta$.*

Proof. Part 1: We proceed by contradiction in two parts. Assume a block B proposed by an arbitrary node $B.author$ of height h is consistent in an honest node u's block-DAG, but in another honest node v's block-DAG, no block by $B.author$ of height h is consistent. Since B is consistent for u, u has waited 2Δ time slots after proposing a block B_u of height $h+1$ such that $B_u \to B$ at time slot t_{h+1}. Since v is honest, v has proposed a block B_v at height $h+1$ at time t_{h+1}. If B was received before t_{h+1} by v, B_v references B, and considers B δ-consistent after time $t_{h+1} + 2\Delta$, a contradiction. If v has not received B before t_{h+1}, it is received before $t_{h+1} + \Delta$ from u through the synchronous network. v therefore broadcasts an UNRECEIVED message, received by all honest nodes before $t_{h+1}+2\Delta$. By Lemma 2, if an honest node discards B due to the UNRECEIVED message, all nodes discard B a contradiction. If no honest node discards B, v will confirm B at time $t_{h+1} + 2\Delta$, a contradiction. This completes the first part of our proof.

Part 2: Assume a block B proposed by an arbitrary node $B.author$ of height h is consistent in an honest node u's block-DAG, but in another honest node v's block-DAG, a block $B' \neq B$ by $B.author$ of height h is consistent. Consider two cases for time t when u extends B with a proposal and time t' when v receives B'. We assume w.l.o.g. that u is the first honest node receiving B and v is the first honest node receiving B':

- Case 1: $t' > t + 2\Delta$. All honest nodes received B as a parent of u's block of height $h + 1$ at time $t + \Delta$. As a result, when CONFLICTINIT message is broadcast after receiving B', all honest nodes propose B in BYZAGREEMENT and, by validity, B is output for all honest nodes and B' is discarded. BYZAGREEMENT is secure by lemma 4.
- Case 2: $t \leq t' \leq t + 2\Delta$. If all honest nodes receive B first, this is the same case as case 1. If some nodes may receive B first while others receive B' first, we will show that all honest nodes enter BYZAGREEMENT and that no two honest nodes will exclusively discard B, B' respectively. By time $t' + \Delta$, all honest nodes have received B from u and B' from v by the synchronous network assumption. Since $t' + \Delta \leq t + 2\Delta$, u receives B' before the confirmation timer for B runs out, and enters BYZAGREEMENT. By the agreement property of BYZAGREEMENT, all honest nodes obtain the same output B or B'.

Part 3: Assume a block B of height h is consistent at time t in an honest node u's block-DAG, but another honest node v confirms a block $B' \neq B$ of height h after t. Since B is confirmed for u, all instances of BYZAGREEMENT for blocks in height h have terminated. When receiving B', v broadcasts an UNRECEIVED message. However, as we have shown in Part 1 of the proof, if no honest node has confirmed B' by time t, B' will be discarded. Otherwise, B' is confirmed by all honest nodes by time t, a contradiction.

Theorem. *$\boldsymbol{P}_{sd}$ is a structured dissemination protocol under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$ and $T_b = 15\Delta$.*

Proof. δ-Consistency: Follows from Lemma 5.

τ, t-DAG growth: An honest node's block timer $BlockTimer()$ triggers every Δ_b time, upon which an honest node broadcasts a new block. Within t time, there at least $\frac{t}{\Delta_b} \cdot (n^*)$ additional blocks proposed by the n^* nodes that are honest and awake during t. In cases with conflicts, the termination of BYZAGREEMENT guarantees that blocks of any height with honest proposals are eventually confirmed or discarded. We therefore have τ, t-DAG growth where $\tau = \frac{n^*-f}{\Delta_b}$.

μ-DAG quality: We first show by contradiction that at any height k for an honest node, there are up to n blocks proposed by distinct nodes. Assume that at height k, an honest node u includes $n' > n$ blocks. This either implies that there are n' distinct nodes, a contradiction, or that u has included more than one block of height k proposed by the same node. This is also a contradiction as the protocol is consistent.

External Integrity: External integrity is guaranteed as honest nodes only add a block B to their local block-DAG if $condition(B) = 1$ holds.

6.1.2. Self-healing DAG after GST. We defer the proofs for $\mathbf{P}_{sd}$'s security after GST under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ to Appendix C.1.

Lemma 6. *Consistency of blocks before GST after GST. All blocks proposed before GST under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 < 1/2$ are eventually discarded or consistent for all honest nodes after GST.*

Proof. See Appendix C.1.

Theorem. *$\boldsymbol{P}_{sd}$ satisfies a structured dissemination protocol's properties under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 < 1/2$ after GST.*

Proof. See Appendix C.1.

6.2. Graded Common Prefix

Theorem. *Algorithm 2 achieves the properties of graded common prefix as follows: validity after $\max\{GST, GAT\}$, termination in 2 network delays after GAT, and agreement and weak validity for $n \geq 3f + 1$ nodes and up to f Byzantine nodes in a partially synchronous network.*

Proof. Validity. For a block-DAG D, that is a subset of all honest proposals, it is included in $n - f$ $Proposal$ messages after GAT and which are received by all honest nodes after waiting Δ time slots after GST. All nodes therefore include D in their VOTE messages, and all honest nodes therefore receive $n - f$ vote messages containing D after Δ time slots, hence including D in their grade 2 output.

Weak Validity: We proceed by contradiction. Assume that a block-DAG D is a subset of all honest proposals and an honest node u outputs D' with grade ≥ 1 such that $D \nsubseteq D'$. To output D' with grade ≥ 1, u has received at least one VOTE message where $n-f$ proposals don't include D. This is a contradiction as all honest nodes include D in their proposal, and only up to f nodes are Byzantine.

Agreement. We proceed by contradiction. Assume that an honest node u has included a block-DAG D in its grade 2 output, and that another honest node doesn't include D in its output ≥ 1 at protocol termination. For u to include D in its grade 2 output, at least $f+1$ honest nodes have sent a VOTE messages whose block-DAG $D_i \supseteq D$ $i \in \{1, ...f+1\}$, of which at least 1 is included by u in its set of $2f+1$ vote messages. This is a contradiction as the honest node v would include D with grade at least 1.

Assume that an honest node u has decided D with grade 2, that another honest node v decides $D' \nsim D$ with grade 2. To decide D with grade 2 and not D', u has received $n-f$ votes that contain D and not D'. By quorum intersection, v can not obtain $n - f$ votes that also contain D' unless $D' \sim D$, a contradiction.

Termination: Termination is guaranteed in 2 network delays, or $2 \cdot \Delta$ after GST, as honest nodes receive at least $n-f$ proposals and $n-f$ votes since $n-f$ nodes are honest and awake.

6.3. Ebb-and-Flow using Graded Common Prefix

We prove that $\mathbf{P}$ is a secure Ebb-and-Flow protocol, using a similar proof structure as for snap-and-chat protocol [19].

Theorem. $\mathbf{P}$ *is a secure $(\frac{1}{3}, \frac{1}{2})$-Ebb-and-Flow protocol*

Proof. We first show the safety of LOG_{fin} under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 = \frac{1}{3}$. Due to agreement of graded common prefix, when an honest node outputs a final block-DAG B with grade 2, all honest nodes output B with grade at least 1. As a result, every honest node i will input $B_i \supset B$ to the next instance of graded common prefix. By validity, $B \subseteq D$ for a block-DAG output D of $B \nsubseteq D$ with grade 2.

For the liveness of LOG_{fin} after time $\max\{GST, GAT\}$ under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$, consider the following: After time $\max\{GST, GAT\}$, $\mathbf{P}_{da}$ is safe and live and all nodes input the same latest confirmed block-DAG B from $\mathbf{P}_{da}$. The validity property of $\mathbf{P}_{gcp}$ then ensures that B is included in the output with grade 2.

To show the security of $\mathbf{P}$ under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 = \frac{1}{2}$, $\mathbf{P}_{da}$ must be secure and its output LOG_{da} must be consistent with the finalized output LOG_{fin}. The security of $\mathbf{P}_{da}$ follows from our black box secure dynamically available protocol assumption.

By the external integrity property, $\mathbf{P}_{gcp}$ can only output a block-DAG D_{fin} such that $D_{fin} \subseteq D_{da}$ for the confirmed block-DAG output D_{da} of $\mathbf{P}_{da}$. The security of $\mathbf{P}_{da}$ guarantees that D_{da} is confirmed for all honest nodes by the security of $\mathbf{P}_{da}$. LOG_{fin} is therefore consistent with LOG_{da} as they are directly extracted from D_{fin}, D_{da} respectively.

7. Experimental evaluation

We evaluate our protocols experimentally to show the following:

1) Our DAG protocol $\mathbf{P}_{sd}$ for the sleepy model throughput scales with the number of nodes, while current state-of-the-art sleepy model BFT does not. In addition, our protocol's common case latency is lower.
2) The latency of our graded common prefix protocol is not affected by crashed nodes, unlike existing leader based BFT such as Hotstuff [5].

7.1. Setup

We implemented both our protocols and their respective baselines in Haskell using the *distributed-haskell* library[1] to instantiate asynchronous nodes and their communication network. The code is available online[2]. We ran our experiments on a national research cloud, using 10 m3.xsmall instances which have with 2GB of RAM and one vCPU. Each instance ran up to 6 nodes for a total of 60 nodes in our experiments.

In each protocols, arbitrary data is proposed by nodes, and we consider 8B of data as 1 operation. For each committed block of operations, its latency and the current throughput is tracked. Throughput measures the number of committed operations divided by the elapsed time of the experiment. Latency measures the elapsed time between the moment a operation is proposed by a node, until the moment a client receives its confirmation. As different nodes may commit a operation at slightly different times, the average is computed.

7.2. Dynamically available protocols

Since constant-latency BFT in the sleepy model has no publicly available implementation to our knowledge, we implemented a prototype for sleepy model BFT based on Mahkhi et al.'s sleepy BFT [8] as our baseline.

To show that our protocol scales with participating nodes, we increase the number of nodes from 10 to 60 in steps of 10 (1 additional node per instance), and run each protocol for 600 seconds. Blocks are set to contain 5 operations, for a total of $200B$ per block. The protocol's timers (Δ) were set to 1 second, such that nodes collect messages for 1 second before proceeding to the next step.

Results. As depicted in Figure 3a, the baseline protocol's throughput remains relatively stable and obtains no increase with more participating nodes. The performance of the baseline protocol does not noticeably degrade either due to the relatively large timeout set to 1 second, meaning that messages have more than enough time to deliver between protocol steps.

In contrast, in our dynamic DAG protocol, new blocks are proposed every $\Delta = 1s$ which inherently contributes a x4 increase in throughput, compared to the baseline that proposes a block every 4Δ. There are also n more blocks proposed, for n nodes, for a theoretical increase of x$4n$ in throughput. Our observed linear throughput growth matches our intuition. Note that a pipelined version of the baseline could potentially mitigate the x4 factor difference, proposing every Δ instead.

At 60 nodes, our instances' modest resources (CPU, RAM, Network) were saturated and throughput drops. We expect stronger machines to be able to support more throughput before saturation, as is the case in DAG BFT works with access to such resources [11], [12].

As shown in Figure 3b, our protocol exhibits slightly lower latency, almost 1 second less than the baseline. With timers set to $\Delta = 1$ second, these results align with the theoretical expectations outlined in Table 1, where our protocol commits in 3Δ, compared to the baseline's 4Δ. For reference, in simulations for DAGs with participation-dependent latency, 22 seconds was the best case in Prism [10].

Variance in our latency measures can be explained by temporary clock de-synchronization between our instances as the protocols run, causing occasional drift in the latency measurement. However, the highest statistical standard deviation of our latencies was ~ 0.1s, and these inacurracies do not prevent us from observing the expected 1 second lower latency.

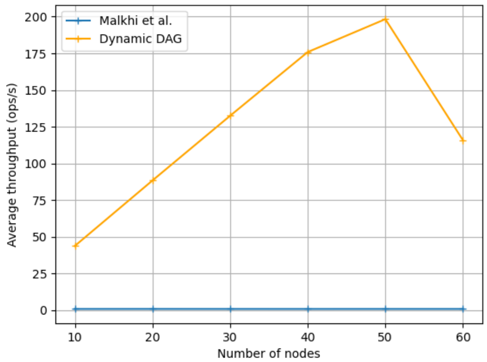
7.3. Partially synchronous protocols

To fairly compare our GCP protocol to a partially synchronous BFT, we implemented Hotstuff [5] under the same Haskell framework and *distributed-haskell* library.

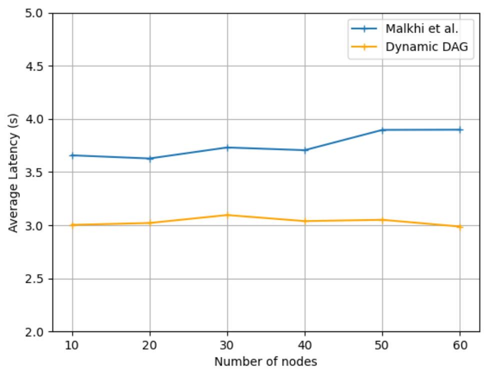
To show the impact of crashes on latency for our GCP protocol and a leader based BFT like Hotstuff, we increase the number of crashes from 0 to 6 in steps of 1, and run each protocol for 300 seconds. Each of the 10 instances run 2 nodes for a total of 20 nodes. Hotstuff blocks are set to contain 10 operations, while GCP proposals contain 10 block-DAG identifiers, represented by 10 operations of the same size, for a total of 200B per block or proposal for each protocol. Hotstuff's timeout was set to 1 second.

Results. As depicted in Figure 4, in the baseline protocol that relies on leaders, crashes increasingly affect the average latency due to the expensive leader change operation, and waiting for timeout timers.

1. https://github.com/haskell-distributed
2. Code will be made available after paper acceptance, and upon request by chair



(a) Throughput vs. number of nodes.



(b) Latency vs. number of nodes.

Figure 3: Throughput and latency evaluation with a gradually increasing number of nodes for our proposed DAG protocol and Malkhi et al.'s constant latency sleepy BFT [8].

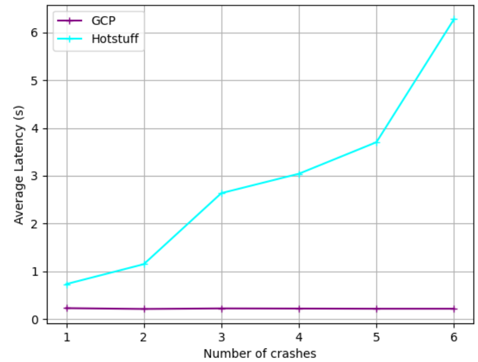


Figure 4: Latency evaluation with a gradually increasing number of crashes for our proposed GCP protocol and Hotstuff [5].

In contrast, the latency of our GCP is faster without crashes and remains stable as the crashes increase, matching our theoretical expected benefits of GCP.

8. Related work

Sleepy model. Protocols for the sleepy model such as Ouroboros [24] and Snow White [15] circumvent Bitcoin's need for Proof of Work (PoW) with Proof of Stake (PoS). However, they inherit the long latency of longest chain protocols and low participation further lengthens latency. The feasibility of constant expected latency has been achieved in follow-up works [7], [8], [22] by adapting design principles from classical BFT, but their throughput is still limited by their single block output.

DAG protocols. Prism [10] is a low latency, high throughput, dynamically available PoW protocol. It improves upon previous PoW DAG attempts [13], [25] to obtain a provably secure PoW DAG protocol achieving network capacity throughput and low latency. As discussed for Ebb-and-Flow [19], it can provide a high performance available ledger for Snap-and-Chat protocols, but requires high participation to obtain the desired low latency.

The asynchronous and partially synchronous DAG based BFT line of work [11], [12] offers high throughput BFT-SMR. Narwhal notably finalizes a partially consistent asynchronous DAG using BFT, similarly to our Ebb-and-Flow protocol that finalizes a structured dissemination protocol's block-DAG. The crucial difference is that no dynamic availability can be offered by asynchronous DAGs that use fixed quorums during dissemination. As Prism shows for PoW DAGs, Narwhal also demonstrates the throughput benefit of separating transaction dissemination in a DAG and consensus, motivating our work to expand from throughput-restricted single blockchains to generic block-DAG constructions in the dynamic availability setting.

Ebb-and-Flow. Ebb-and-Flow [19] introduces optimal hybrid protocols with dynamic availability and full finalization using two concurrent protocols, a longest chain and BFT protocols. Additionally, the Snap-and-Chat class of protocols is introduced, solving Ebb-and-Flow with a longest chain and BFT running concurrently. Our work extends the Ebb-and-Flow framework by offering protocols with low latency and a lighter finalization primitive, graded Byzantine agreement or graded common prefix.

Graded agreement. Our graded common prefix primitive is similar to graded agreement for arbitrary values [23]. However, graded agreement considers single arbitrary values and not sets of values that can take advantage of the underlying data structure, as graded common prefix does with block-DAGs. For example, if all honest nodes propose a different block-DAG but where a common prefix is available, graded agreement can not easily guarantee that the overall protocol will progress, while graded prefix can as we show in Section 4.

Graded agreement for blockchains [8] has also been introduced, and is specific to the blockchain data structure, while we allow a generic integrity condition and block-DAG data structure. This version of graded agreement foregoes the termination property, and allows nodes to gradually output multiple, potentially conflicting blocks. In contrast graded common prefix has a single output set before terminating. Although the block-DAG output is also represented by a set of blocks, terminating with a single output set allows nodes to finalize blocks, without later potentially receiving another block in the output for the same instance, especially for non-synchronous constructions.

Constructions for both works focus on graded agreement in a synchronous network, while we provide a partially synchronous construction of graded common prefix.

9. Conclusion

This work introduces the first constant latency dynamically available DAG protocol, and a new primitive, graded common prefix, to finalize block-DAGs in the partially synchronous setting without resorting to standard consensus. These findings are synthesized in a hybrid protocol consisting of the dynamically available DAG protocol, coupled with GCP. The hybrid protocol lets clients choose prioritizing liveness or safety. By leveraging the DAG structure in each sub-protocol, we obtain high throughput and finalization without the redundant work of running standard consensus again for asynchronous safety.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[3] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[4] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 31–42.

[5] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. ACM, 2019, pp. 347–356.

[6] R. Pass and E. Shi, "The sleepy model of consensus," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 380–409.

[7] A. Momose and L. Ren, "Constant latency in sleepy consensus," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. ACM, 2022, pp. 2295–2308.

[8] D. Malkhi, A. Momose, and L. Ren, "Towards practical sleepy bft," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 490–503.

[9] S. Gilbert and N. A. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002. [Online]. Available: https://doi.org/10.1145/564585.564601

[10] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 585–602.

[11] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

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

[13] Y. Sompolinsky, S. Wyborski, and A. Zohar, "Phantom ghostdag: a scalable generalization of nakamoto consensus: September 2, 2021," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, 2021, pp. 57–70.

[14] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 913–930.

[15] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *Financial Cryptography and Data Security: 23rd International Conference, (FC), Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer, 2019, pp. 23–41.

[16] V. Goyal, H. Li, and J. Raizes, "Instant block confirmation in the sleepy model," in *International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2021, pp. 65–83.

[17] J. Garay, A. Kiayias, and Y. Shen, "Proof-of-work-based consensus in expected-constant time," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 96–125.

[18] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, (DSN) 2022, Baltimore, MD, USA, June 27-30, 2022*. IEEE, 2022, pp. 54–66.

[19] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 446–465.

[20] H. Schmiedel, R. Han, Q. Tang, R. Steinfeld, and J. Yu, "Modeling mobile crash in byzantine consensus," *IEEE 37th Computer Security Foundations Symposium*, 2024.

[21] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 106–118.

[22] F. D'Amato and L. Zanolini, "Streamlining sleepy consensus: Total-order broadcast with single-vote decisions in the sleepy model," *arXiv e-prints*, pp. arXiv–2310, 2023.

[23] A. Momose and L. Ren, "Optimal communication complexity of authenticated byzantine agreement," in *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[24] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros chronos: Permissionless clock synchronization via proof-of-stake," *Cryptology ePrint Archive*, 2019.

[25] A. Kiayias and G. Panagiotakos, "On trees, chains and fast transactions in the blockchain," in *Progress in Cryptology–LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20–22, 2017, Revised Selected Papers 5*. Springer, 2019, pp. 327–351.

Appendix A.

Definition 7 (Byzantine Agreement (BA).)**.** *In a Byzantine agreement protocol, each node i has an input value v_i, and* *outputs a value v. A Byzantine agreement protocol satisfies the following properties:*

- *Termination: Every honest node outputs a value and terminates.*
- *Validity: If all honest nodes propose the same value v, all honest nodes output v.*
- *Agreement: If an honest node outputs v then all honest nodes output v.*

Appendix B.

Definition 8 (Generic Graded Common Subset.)**.** *A generic graded common prefix protocol takes as input a set $\mathcal{D}_i$ of values for every node $i \in 1, ..., n$, and outputs either a set of values with grade 1 or a set of values with grade 1 and a set of values with grade 2. A generic graded common prefix protocol satisfies the following properties:*

- *Termination: Every honest node terminates.*
- *Validity: If all honest nodes propose a set of values containing the value v, $v \in \mathcal{D}'$ holds for any output $\mathcal{D}'$ by an honest node with grade 2.*
- *Weak Validity: If all honest nodes propose a set of values containing the value v, $v \in \mathcal{D}'$ holds for any output $\mathcal{D}'$ by an honest node with grade ≥ 1.*
- *Agreement: If an honest node u outputs grade 2 for a set $\mathcal{D}$ of values then all honest nodes output $\mathcal{D}$ and with grade ≥ 1, and no honest node outputs a set $\mathcal{D}'$ with grade 2 where $(\mathcal{D}' \nsubseteq \mathcal{D}) \wedge (\mathcal{D} \nsubseteq \mathcal{D}')$.*

Appendix C.

Lemma. *Narwhal Narwhal run by n nodes is a structured dissemination of a block-DAG with $\Delta-$consistency, $\frac{2n}{9\Delta}, 3\Delta$-DAG growth, and $1/2$-DAG quality under $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT where $\beta_2 < 1/3$.*

Proof. *Δ-Consistency:* We proceed by contradiction. Assume an honest node u considers a block b with digest d consistent, but another honest node v does not. Since u considers b confirmed at time t, it is certified and the *write(d,b* operation has completed. By the block-availability property of Narwhal and the synchronous network assumption, *read(d)* completes by time $t + \Delta$. By the integrity property of Narwhal, d outputs the same block b for all honest nodes, completing our contradiction.

Assume a block b of round r and height h is consistent at time t in an honest node u's block-DAG, but another honest node v confirms a block $b' \neq b$ of round r and height h after $t+\Delta$. Since the network is synchronous, b' is received after b is certified, implying that honest blocks of round r are also certified. This further implies that honest nodes are at least in round $r+1$ since they certified $2f+1$ honest blocks of round r. b' is therefore not valid for honest nodes due to the round number, and unable to obtain a quorum certificate, a contradiction.

$\frac{2n}{9\Delta}, 3\Delta$*-DAG growth:* We proceed by induction. For the first round of the protocol, at least $2n/3$ honest node propose a block certified within 3Δ time. For a round $r+1$, every honest node has received at least $2n/3$ blocks from honest nodes at round r and is able propose a block delivered within Δ time. We therefore have $\frac{2n}{3}$ blocks certified in 3Δ time, and $\tau = \frac{2n}{9\Delta}$, completing the induction.

$1/2$*-DAG quality:* $1/2$-DAG quality directly follows from Narwhal's $1/2$-chain quality property.

External Integrity: External integrity for $condition(B)$ can be trivially satisfied by adding $condition(B) = 1$ as a requirement for honest nodes to certify blocks. If $condition(B) \neq 1$ for a block B, B can not obtain a certificate from at least $2f+1$ nodes' signatures.

C.1. Structured dissemination

Lemma. *Consistency of blocks before GST after GST. All blocks proposed before GST under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 < 1/2$ are eventually discarded or consistent for all honest nodes after GST.*

Proof. At time $t > GST$, consider the set $\mathcal{C}$ containing every pair of blocks B and $B' \neq B$ of the same height where $B.author = B'.author$, and B is confirmed for an honest node u while B' is confirmed for another honest node v. We also consider the set $\mathcal{M}$ containing every block B of height h such that B is confirmed in an honest node u's block-DAG, but in another honest node v's block-DAG, no block by $B.author$ of height h is confirmed.

For pairs in $\mathcal{C}$, consider the following proof by contradiction. Assume a block B proposed by an arbitrary node $B.author$ of height h is consistent in an honest node u's block-DAG, but in another honest node v's block-DAG, a block $B' \neq B$ by $B.author$ of height h is consistent. Since the network is synchronous after GST, Lemma 3 holds and all honest nodes enter an instance of ByzAgreement. Lemma 4 also holds since ByzAgreement is started after GST and the network is synchronous. By the agreement property of ByzAgreement, all honest nodes obtain the same output B or B', a contradiction.

Honest nodes therefore eventually agree on a consistent block for all pairs in $\mathcal{C}$.

For blocks in $\mathcal{M}$, consider the following recursive proof once $\mathcal{C} = \varnothing$. We order the blocks in $\mathcal{M}$ by height such that the block with the largest height has index $i = 0$. Blocks of the same height have neighboring indexes.

For $i = 0$, we denote the block B_0 of height h_0. There are no blocks of larger height in $\mathcal{M}$ by definition, and blocks of height $h_0 + 1$ are confirmed since $\mathcal{C} = \varnothing$. Once an honest node broadcasts an Unreceived message after receiving B_0 after GST at time t_u, all honest nodes receive the Unreceived message after $t + \Delta$ and either discard or keep B_0 after $t + 3\Delta$. We prove by contradiction that all honest nodes discard B_0 if an honest node u discads B_0. Since u discards B_0, it has received blocks of height h_0+1 such that a majority does not reference B_0. Since $\mathcal{C} = \varnothing$ and B_0 has the highest height in $\mathcal{M}$, the blocks of height h_0+1 are confirmed for all honest nodes, a contradiction as they would all discard B_0 too. As a result, all honest nodes either discard or confirm B_0 after GST. For the recursion proof, we now replace B_0 with the empty element $\varnothing$ in $\mathcal{M}$, maintaining the same indexes for the other elements.

For $i+1$, blocks of height h_i in $\mathcal{M}$ are either discarded or confirmed. The block B_{i+1} is therefore the block with the largest height in $\mathcal{M}$. Using the same proof as for $i=0$, all honest nodes either discard or confirm B_{i+1} after GST, completing the recursive proof.

Theorem. *$\boldsymbol{P}_{sd}$ satisfies a structured dissemination protocol's properties under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 < 1/2$ after GST.*

Proof. Before GST, δ-Consistency, τ, t-DAG growth, and μ-DAG quality are not guaranteed and honest nodes may have differing views of confirmed block-DAGs, with no guaranteed growth rate or fraction of honest blocks. External integrity is still guaranteed as honest nodes only add a block B to their local block-DAG if $condition(B) = 1$ holds, irrespective of network conditions.

We now analyze each property after GST. Compared to the proofs for Theorem 1, the system model is equivalent as both networks are synchronous, and the fraction of Byzantine nodes is bounded by $1/2$. The only difference with the proofs for Theorem 1 is therefore that honest nodes don't start with the same consistent block-DAG. We now show that δ-Consistency, τ, t-DAG growth, and μ-DAG quality eventually hold after GST.

τ, t-DAG growth: The proof is the same as for Theorem 1.

δ-Consistency: Since before GST, honest nodes have differing confirmed block-DAGs, and are eventually discarded or confirmed as shown by Lemma 6. For blocks proposed after GST, the proof is the same as for Lemma 5.

μ-DAG quality: The proof is the same as for Theorem 1, since we have shown that the protocol is consistent after GST.

C.2. Graded common prefix

Theorem. *Algorithm 2 achieves the properties of graded common prefix as follows: validity after* $\max\{GST, GAT\}$, *termination in* 2 *network delays after* GAT, *and agreement and weak validity for* $n \geq 3f+1$ *nodes and up to* f *Byzantine nodes in a partially synchronous network.*

Proof. Validity. For a block-DAG D, that is a subset of all honest proposals, it is included in $n-f$ $Proposal$ messages after GAT and which are received by all honest nodes after waiting Δ time slots after GST. All nodes therefore include D in their VOTE messages, and all honest nodes therefore receive $n-f$ vote messages containing D after Δ time slots, hence including D in their grade 2 output.

Weak Validity: We proceed by contradiction. Assume that a block-DAG D is a subset of all honest proposals and an honest node u outputs D' with grade ≥ 1 such that $D \nsubseteq D'$. To output D' with grade ≥ 1, u has received at least one VOTE message where $n-f$ proposals don't include D. This is a contradiction as all honest nodes include D in their proposal, and there are only up to $f < n-f$ Byzantine nodes.

Agreement. We proceed by contradiction. Assume that an honest node u has included a block-DAG D in its grade 2 output, and that another honest node doesn't include D in its output ≥ 1 at protocol termination. For u to include D in its grade 2 output, at least $f+1$ honest nodes have sent a VOTE messages whose block-DAG $D_i \supseteq D$ $i \in \{1, ...f+1\}$, of which at least 1 is included by u in its set of $2f+1$ vote messages. This is a contradiction as the honest node v would include D with grade at least 1.

Assume that an honest node u has decided D with grade 2, that another honest node v decides $D' \nsim D$ with grade 2. To decide D with grade 2 and not D', u has received $n-f$ votes that contain D and not D'. By quorum intersection, v can not obtain $n-f$ votes that also contain D' unless $D' \sim D$, a contradiction.

Termination: Termination is guaranteed in 2 network delays, or $2 \cdot \Delta$ after GST, as honest nodes receive at least $n-f$ proposals and $n-f$ votes since $n-f$ nodes are honest and awake.