Majorum: Ebb-and-Flow Consensus with Dynamic Quorums

Francesco D'Amato[1][0000−0002−8190−8664], Roberto Saltini[1][0000−0002−8912−6392], Thanh-Hai Tran[2][0000−0002−2260−6877], Yann Vonlanthen[1][0000−0001−5736−8197], and Luca Zanolini[1][0000−0003−4655−3172]

[1] Ethereum Foundation
francesco.damato@ethereum.org, roberto.saltini@ethereum.org, yann.vonlanthen@ethereum.org, luca.zanolini@ethereum.org
[2] Independent Researcher
thanhhai1302@gmail.com

Abstract. Dynamic availability is the ability of a consensus protocol to remain live despite honest participants going offline and later rejoining. A well-known limitation is that dynamically available protocols, on their own, cannot provide strong safety guarantees during network partitions or extended asynchrony. Ebb-and-flow protocols [SP21] address this by combining a dynamically available protocol with a partially synchronous finality protocol that irrevocably finalizes a prefix.
We present Majorum, an ebb-and-flow construction whose dynamically available component builds on a quorum-based protocol (TOB-SVD). Under optimistic conditions, Majorum finalizes blocks in as few as three slots while requiring only a *single* voting phase per slot. In particular, when conditions remain favourable, each slot finalizes the next block extending the previously finalized one.

Keywords: dynamic availability · ebb-and-flow · consensus.

1 Introduction and Related Work

Traditional Byzantine consensus protocols such as PBFT [4] and HotStuff [19] typically assume a fixed set of validators that remain online; in such settings, going offline is treated as a fault.

Some permissionless systems require *dynamic participation*: honest validators may go offline and later rejoin. This notion was implicit in Bitcoin [14] and formalized by Pass and Shi via the *sleepy model* [18]. We call a protocol *dynamically available* if it preserves safety and liveness under this type of churn.

A limitation is that dynamically available protocols, on their own, cannot provide strong safety guarantees during network partitions or extended asynchrony while also maintaining liveness [10]. Ebb-and-flow protocols [15] address this by combining a dynamically available chain-growth protocol with a partially synchronous finality layer that irrevocably finalizes a prefix. Ethereum follows this pattern: LMD-GHOST [20] provides dynamically available chain growth, and Casper FFG [2] finalizes a prefix (as in Gasper [3]).

Momose and Ren [13] initiated the study of *deterministically safe* dynamically available consensus via quorum techniques adapted to dynamic participation. This line of work led to multiple quorum-based dynamically available protocols with different trade-offs [11,12,9,5].

In particular, D'Amato *et al.* [7] introduced TOB-SVD, a majority-quorum-based protocol targeting practicality at scale. TOB-SVD tolerates up to 50% Byzantine faults and can decide with a single voting phase in the best case, improving over earlier multi-phase designs.

In this work we present Majorum, an ebb-and-flow protocol whose dynamically available component builds on TOB-SVD. Under optimistic conditions, Majorum achieves three-slot finality while requiring only a single voting round per slot. Relative to the only ebb-and-flow construction we are aware of that is deployed in production – namely, Ethereum's – this reduces optimistic time to finality from 64 slots to 3. We further introduce, to our knowledge for the first time, a principled integration of two complementary confirmation rules within an ebb-and-flow protocol: traditional κ-deep confirmation and *fast confirmation*.

1.1 Technical Overview

In large-scale consensus systems, each voting round is costly. With hundreds of thousands of validators, votes are typically aggregated and re-broadcast before they can influence the global view; consequently, even a single *logical* voting phase can incur an end-to-end delay closer to 2Δ than Δ. Ethereum illustrates this effect: attestations are aggregated and disseminated before contributing to fork choice, yielding an effective 2Δ latency per voting round. This motivates our first design goal: a dynamically available protocol that, under synchrony, makes progress with as few voting phases as possible.

Our second goal is to obtain the guarantees characteristic of ebb-and-flow protocols [16]. Under synchrony (even with reduced participation), the system should remain live and produce a synchronously safe chain. Under a network partition or extended asynchrony, a *finalized* prefix should remain forever safe, while only the non-finalized suffix may be reverted. Achieving both goals calls for a dynamically available protocol with an efficient voting structure that can be paired cleanly with a partially synchronous finality gadget.

For these reasons, we take TOB-SVD as our starting point. TOB-SVD is a dynamically available consensus protocol based on majority quorums; in the synchronous model, it achieves optimal Byzantine resilience within this design space and, in the best case, requires only a single voting phase per decision. In the remainder of this overview, we describe the modifications we apply to TOB-SVD and how we pair the resulting dynamically available protocol with a partially synchronous finality layer whose purpose is to *finalize* – that is, to provide asynchronous safety to – a prefix of the produced chain, yielding Majorum.

The appeal of Majorum, and of ebb-and-flow protocols more generally, is that they can remain live under synchrony even in low participation regimes – for instance, when a large fraction of validators are temporarily offline. Ethereum is a noteworthy deployed example of this design pattern: block production can

continue under synchrony even when participation drops (although finality may stall). Moreover, Ethereum's incentive structure encourages validators to stay online and participate promptly; nevertheless, dynamic participation remains practically relevant, for example due to crash failures caused by consensus-client bugs [17].

TOB-SVD proceeds in *slots*: at the start of slot t, an elected proposer broadcasts a block (propose phase); after Δ, validators cast votes (vote phase); and after another Δ, the protocol decides according to the majority of observed votes (decide phase). In the original protocol, proposals made in slot t are decided during the decide phase of slot $t+1$. All safety guarantees are fully deterministic; no probabilistic arguments are used. Moreover, in TOB-SVD, proposals made by honest proposers will always be voted on by all awake honest validators.

A key insight is that the deterministic decision phase does not influence validator behavior; rather, it acts as an explicit confirmation mechanism. We therefore remove the decision phase and confirm chains directly during the vote phase via a *κ-deep confirmation rule*, where κ is chosen so that, within κ slots, an honest proposer is elected with overwhelming probability, obtaining a variant that provides *probabilistic* rather than deterministic safety (Section 3.2). However, this modification alone confirms only those chains whose proposals lie at least κ slots in the past. To address this limitation, we introduce an optimistic mechanism for *fast confirmations* (Section 3.3). Under favorable conditions, a chain can be confirmed in the very slot in which it is proposed: concretely, if all validators are awake and more than $\frac{2}{3}n$ of them vote for the same chain, that chain is *fast-confirmed* and may be immediately adopted in place of the κ-deep prefix.

This new protocol constitutes the dynamically available component of the ebb-and-flow construction Majorum: once a block is confirmed in the dynamically available layer, this confirmation can be fed into the partially synchronous layer as a candidate for finalization.

Integrating these two layers is straightforward. We extend the voting message format of TOB-SVD by adding an additional component that closely mirrors the mechanism of Casper FFG – the finality gadget used in Ethereum's consensus protocol. Concretely, the voting phase of slot t serves two purposes: (1) voting for the proposal made in slot t, and (2) contributing to the finalization of a previously confirmed chain within TOB-SVD.

Under this design, a proposal made by an honest proposer in slot t is optimistically fast-confirmed during the same slot. It then becomes the *target* of the first (of two) voting rounds in slot $t+1$. If it gathers more than $\frac{2}{3}n$ votes in this first round, we say that it is *justified*. Finally, it becomes finalized in slot $t+2$, during the second round of voting. Figure 1 illustrates the progression of the combined protocol. Although this represents roughly a $3\times$ increase over same-slot finality, it avoids the need for multiple voting rounds within a single slot – an approach that, at scale, would necessitate longer slot durations – while retaining a single voting round per slot and thereby simplifying the protocol's timing assumptions and overall design.

The remainder of this paper is organized as follows. Section 2 presents the model and background. Section 3 introduces the dynamically available component, reviewing TOB-SVD (Section 3.1) and extending it to probabilistic safety and fast confirmations (Section 3.2; proofs in Appendix A). Section 4 presents the finality gadget (proofs in Appendix B). Section 5 gives the full ebb-and-flow protocol and its security analysis (Appendix C). We conclude in Section 6.

2 Model and Preliminary Notions

2.1 System Model

Validators. We consider a set of n *validators* $v_1, \ldots, v_n$, each with a unique cryptographic identity and known public key. Validators execute a common protocol. An adaptive probabilistic poly-time adversary $\mathcal{A}$ may corrupt up to f validators, gaining access to their state. Once corrupted, a validator is *adversarial*; otherwise it is *honest*. Honest validators follow the protocol, while adversarial ones behave arbitrarily. All validators have equal *stake*, and a proven misbehaving validator is *slashed* (loses part of its stake).
Network. We assume partial synchrony: validators have synchronized clocks and message delays are initially unbounded, but after an unknown *global stabilization time* (GST) delays are bounded by Δ.
Time. Time is divided into discrete *rounds*. A *slot* consists of 4Δ rounds. For round r, its slot is $\text{slot}(r) := \lfloor r/4\Delta \rfloor$.
Proposer Election Mechanism. In each slot t, a validator v_p is chosen as *proposer*, denoted v_p^t. If t is clear, we omit the superscript. The proposer election mechanism must satisfy: *Uniqueness* (exactly one proposer per slot), *Unpredictability* (identity hidden until revealed), and *Fairness* (each validator elected with probability $1/n$).
Transactions Pool. We assume the existence of an ever growing set of transaction, called *transaction pool* and denoted by *txpool*, that any every validator has read access to. Consistently with previous notation, we use $txpool^r$ to refer to the content of *txpool* at time r.
Blocks and Chains. A *block* is $B = (b, p)$, where b is the block body (a batch of transactions) and $p \geq 0$ is the slot when B is proposed. The *genesis block* is $B_{\text{genesis}} = (b_{-1}, -1)$ and has no parent. Each block references its parent, and a child must have a larger slot. A block induces a *chain*, the path to genesis. Chains are identified with their tip blocks. We write $\mathsf{ch}_1 \prec \mathsf{ch}_2$ if ch_1 is a strict prefix of ch_2, and say that two chains *conflict* if neither extends the other.

The κ-deep prefix of chain ch at slot t, denoted $\mathsf{ch}^{\lceil \kappa, t}$, is the longest prefix ch' with $\mathsf{ch}'.p \leq t - \kappa$. When t is clear, we write $\mathsf{ch}^{\lceil \kappa}$. We define $\mathsf{ch}_1 \leq \mathsf{ch}_2$ if $\mathsf{ch}_1.p \leq \mathsf{ch}_2.p$, with equality only if the chains are identical. We call $\mathsf{ch}.p$ the *height* (or *length*) of ch. A transaction tx belongs to ch if it appears in a block of ch, written $tx \in \mathsf{ch}$. Similarly, $TX \subseteq \mathsf{ch}$ means all transactions in TX are included in ch.
Proposing and Voting. Validators may *propose* chains, generalizing block proposals. Proposals occur in the first round of each slot. For readability, define

$\mathsf{propose}(t) := 4\Delta t$. Each slot also has a voting round, where validators *vote* on proposed chains. Define $\mathsf{vote}(t) := 4\Delta t + \Delta$.

Gossip. We assume a best-effort gossip primitive that eventually delivers messages to all validators. Honest-to-honest messages are always delivered and cannot be forged. Once an adversarial message is received by an honest validator v_i, it is also gossiped by v_i. Votes and blocks are always gossiped, whether received directly or inside another message. A validator receiving a vote also gossips the corresponding block, and receiving a proposal gossips its blocks and votes. A proposal for slot t is gossiped only during the first Δ rounds of slot t.

Views. A *view* at round r, denoted $\mathcal{V}^r$, is the set of all messages a validator has received by round r. We write $\mathcal{V}_i^r$ for the view of validator v_i.

Sleepiness. In each round r, the adversary may declare which honest validators are *awake* or *asleep*. Asleep validators pause protocol execution; messages are queued until they wake. Upon waking at round r, a validator must complete a *joining protocol* before becoming *active* [6] (see the paragraph below). Adversarial validators are always awake. Thus, *awake*, *asleep*, and *active* apply only to honest validators. The sleep schedule is adversarially adaptive. In the sleepy model [18], awake and active coincide. Eventually, at an unknown *global awake time* (GAT), all validators stay awake forever.

Let H_r, A_r, and $H_{r,r'}$ denote, respectively, the set of active validators at round r, the set of adversarial validators at round r, and the set of validators that are active *at some point* in rounds $[r, r']$, i.e., $H_{r,r'} = \bigcup_{i=r}^{r'} H_i$, with $H_i := \emptyset$ if $i < 0$. We let $A_\infty = \lim_{t\to\infty} A_{\mathsf{vote}(t)}$ and note that $f = |A_\infty|$. Unless otherwise stated, we assume throughout that $f < \frac{n}{3}$. For every slot t after GST, we require

$$|H_{\mathsf{vote}(t-1)} \setminus A_{\mathsf{vote}(t)}| > |A_{\mathsf{vote}(t)}| \tag{1}$$

In words, the number of validators active at round $\mathsf{vote}(t-1)$ and not corrupted by $\mathsf{vote}(t)$ must exceed the sum of adversarial validators at $\mathsf{vote}(t)$.

We impose this restriction to limit the number of honest validators the adversary can corrupt after the voting phase of slot $t-1$ but before those votes are tallied in later slots. Without such a bound, the adversary could corrupt *too many* honest validators immediately after the voting phase and force them to equivocate.

The main difference from the original sleepy model is that our constraints apply only to *active* validators, i.e., those that have completed a joining protocol and therefore have been awake for at least Δ rounds before $\mathsf{vote}(t)$. This reflects the 2Δ stabilization period assumed by $\mathsf{TOB\text{-}SVD}$ for security; we refer to the original paper for a formal treatment [7]. In practice, a validator is unlikely to enjoy buffered and instantaneous delivery of messages as soon as it wakes up: it must request recovery and wait for peers to return any decision-relevant messages it missed. We treat a validator as active only after these replies arrive, which takes at least 2Δ (one Δ to send, one Δ to receive) and may be much longer when substantial state must be fetched. Hence, an explicit 2Δ stabilization window is typically negligible relative to practical recovery times.

We call the adversary model defined above the *generalized partially synchronous sleepy model*. When the context is clear, we refer to it simply as the *sleepy model*. An execution in this model is said to be *compliant* if and only if it satisfies the sleepiness Condition (1).

Joining Protocol. An honest validator waking at round r must first run the *joining protocol*. If v_i wakes in round $r \in (\mathsf{vote}(t-2) + \Delta, \mathsf{vote}(t-1) + \Delta]$, it receives all queued messages and resumes execution, but sends nothing until round $\mathsf{vote}(t)$, when it becomes active (unless corrupted or put back to sleep). If v_i is elected leader for slot t but is inactive at proposal time, it does not propose in that slot.

2.2 Security

We treat λ and κ as security parameters for cryptographic primitives and the protocol, respectively, assuming $\kappa > 1$. We consider a finite time horizon T_{conf}, polynomial in κ. An event occurs *with overwhelming probability* (w.o.p.) if it fails with probability $\mathrm{negl}(\kappa) + \mathrm{negl}(\lambda)$. Cryptographic guarantees hold with probability $\mathrm{negl}(\lambda)$, though we omit this explicitly in later sections.

Definition 1 (Safe protocol). *We say that a protocol outputting a confirmed chain* Ch *is* safe *after time* T_{sec} *in executions* $\mathcal{E}$*, if and only if for any execution in* $\mathcal{E}$*, two rounds* $r, r' \geq T_{\mathsf{sec}}$*, and any two honest validators* v_i *and* v_j *(possibly* $i = j$*) at rounds* r *and* r' *respectively, either* $\mathsf{Ch}_i^r \preceq \mathsf{Ch}_j^{r'}$ *or* $\mathsf{Ch}_j^{r'} \preceq \mathsf{Ch}_i^r$*. If* $T_{\mathsf{sec}} = 0$ *and* $\mathcal{E}$ *includes all partially synchronous executions, then we say that a protocol is* always safe*.*

Definition 2 (Live protocol). *We say that a protocol outputting a confirmed chain* Ch *is* live *after time* T_{sec} *in executions* $\mathcal{E}$*, and has confirmation time* T_{conf}*, if and only if for any execution in* $\mathcal{E}$*, any rounds* $r \geq T_{\mathsf{sec}}$ *and* $r_i \geq r + T_{\mathsf{conf}}$*, any transaction* tx *in the transaction pool at time* r*, and any validator* v_i *active in round* r_i*,* $tx \in \mathsf{Ch}_i^{r_i}$*. If* $T_{\mathsf{sec}} = 0$ *and* $\mathcal{E}$ *includes all partially synchronous executions, then we say that a protocol is* always live*.*

Definition 3 (Secure protocol [6]). *We say that a protocol outputting a confirmed chain* Ch *is* secure *after time* T_{sec}*, and has confirmation time* T_{conf}*, if it is safe after time* T_{sec} *and live after time* T_{sec} *with confirmation time* T_{conf}*. A protocol is always secure if it always both safe and live*

We now recall the notions of *Dynamic Availability* and *Reorg Resilience* from [8].

Definition 4 (Dynamic Availability). *A protocol is* dynamically available *after time* T_{dyn} *if and only if it is secure after* T_{dyn} *with confirmation time* $T_{\mathsf{conf}} = O(\kappa)$*. We say a protocol is* dynamically available *if it always secure when* $\mathsf{GST} = 0$*.*

Note that Dynamic Availability presupposes a certain level of adversarial resilience. In the protocol of Section 5, Dynamic Availability is attained provided Constraint (1) holds.

Definition 5 (Reorg Resilience). *A protocol is* reorg-resilient *after slot* t_{reorg} *and time* T_{reorg}[3] *in executions* $\mathcal{E}$ *if, for any* $\mathcal{E}$*-execution, round* $r \geq T_{\mathsf{reorg}}$*, and validator* v_i *honest in* r*, any chain proposed in slot* $t \geq t_{\mathsf{reorg}}$ *by a validator honest in* $\mathsf{propose}(t)$ *does not conflict with* Ch_i^r*. We say a protocol is* reorg-resilient *if this holds after slot* 0 *and time* 0*.*

Accountable Safety. In addition to Dynamic Availability, we require accountability in the event of safety violations.

Definition 6 (Accountable Safety). *A protocol has* Accountable Safety *with resilience* $f^{\mathrm{acc}} > 0$ *if any safety violation admits a cryptographic proof – constructed from the message transcript – that identifies at least* f^{acc} *misbehaving validators, while never falsely accusing an honest protocol-following participant.*

By the CAP theorem [10], no consensus protocol can ensure both liveness with dynamic participation and safety under partitions. Thus, Neu, Tas, and Tse [15] proposed *ebb-and-flow* protocols, which output two chains: one live under synchrony and dynamic participation, and one safe under asynchrony.

Definition 7 (Secure ebb-and-flow protocol). *A secure* ebb-and-flow protocol *outputs an available chain* chAva *that is dynamically available*[4]*, and a finalized chain* chFin *that is always safe and live after* $\max(\mathsf{GST}, \mathsf{GAT}) + O(\Delta)$ *with* $T_{\mathsf{conf}} = O(\kappa)$*. Hence,* chFin *is secure after* $\max(\mathsf{GST}, \mathsf{GAT}) + O(\Delta)$ *with* $T_{\mathsf{conf}} = O(\kappa)$*. Moreover, for every honest validator* v_i *and round* r*,* chFin_i^r *is a prefix of* chAva_i^r*.*

Our protocol follows the ebb-and-flow approach [16]: chAva comes from the dynamically available component (Sec.3), while chFin is finalized by a PBFT-style protocol akin to Casper FFG[2] (Sec. 4).

Fork-Choice Function. A *fork-choice function* FC is deterministic: given views, a chain, and a slot t, it outputs a chain ch. We use the *majority fork-choice function* (Algorithm 1), which selects the longest chain supported by a majority of the voting weight.

3 The Dynamically Available Component

We begin by presenting the dynamically available component of our protocol. In Section 3.1, we recall $\mathsf{TOB\text{-}SVD}$, summarizing its operation and guarantees. In Section 3.2, we describe our modifications that yield a probabilistically safe, dynamically available consensus protocol. We then introduce the notion of fast confirmations and prove its Dynamic Availability (Appendix A), which enables integration into the ebb-and-flow protocol presented in Section 5 and fully detailed in Appendix C.

[3] In this work we always have $T_{\mathsf{reorg}} \geq 4\Delta t_{\mathsf{reorg}}$.

[4] When we say that a chain produced by protocol Π satisfies property P, we mean that protocol Π itself satisfies P.

3.1 Recalling TOB-SVD

We recall the deterministically safe, dynamically available consensus protocol of D'Amato *et al.* [7]. We refer the reader to the original work for full technical details, and summarize here its operation, guarantees, and the aspects relevant to our modifications toward a probabilistically safe variant. After recalling the original protocol, we show how to introduce fast confirmations and prove Dynamic Availability, which enables its use in our ebb-and-flow protocol.

TOB-SVD [7] proceeds as follows. During each slot t, a proposal (B) is made in the first round through a $[\text{PROPOSE}, B, t, v_i]$ message, and a decision is taken in the third round. During the second round, every active validator v_i casts a $[\text{VOTE}, \mathsf{ch}, \cdot, t, v_i]$ message for a chain ch (The third component in the vote message is omitted here; it will be introduced later when describing the ebb-and-flow protocol.) Importantly, proposals made in slot t are decided during the decide phase of slot $t+1$. All safety guarantees are fully deterministic.

Let $\mathcal{V}$ be a validator's view at slot t. We define:

$$\mathcal{V}^{\mathsf{ch},t} := \left\{[\text{VOTE}, \mathsf{ch}', \cdot, \cdot, v_k] \in \mathcal{V}' : \mathcal{V}' = \mathsf{FIL}_{\text{lmd}}(\mathsf{FIL}_{\text{1-exp}}(\mathsf{FIL}_{\text{eq}}(\mathcal{V}), t)) \wedge \mathsf{ch} \preceq \mathsf{ch}'\right\},$$

where FIL_{eq} removes equivocations, $\mathsf{FIL}_{\text{1-exp}}$ discards *expired* votes (i.e., votes older than one slot, which are no longer considered), and $\mathsf{FIL}_{\text{lmd}}$ retains only the latest vote from each validator.

Thus, $\mathcal{V}^{\mathsf{ch},t}$ is the set of latest, non-expired, non-equivocating votes supporting chains that extend ch.

The protocol employs a *majority fork-choice function* MFC (defined formally in Algorithm 1): Given views $\mathcal{V}$ and $\mathcal{V}'$, a base chain ch^C, and slot t, it returns the longest $\mathsf{ch} \succeq \mathsf{ch}^C$ such that $|\mathcal{V}^{\mathsf{ch},t} \cap (\mathcal{V}')^{\mathsf{ch},t}| > \frac{1}{2}|\mathsf{S}(\mathcal{V}', t)|$, where $\mathsf{S}(\mathcal{V}', t)$ is the set of validators appearing with non-expired votes in $\mathcal{V}'$ at slot t. If no such chain exists, MFC returns ch^C.

Algorithm 1 MFC, the majority fork-choice function

```
function MFC(V, V', ch^C, t)
    return the longest chain ch ⪰ ch^C such that
        ch = ch^C ∨ |V^{ch,t} ∩ (V')^{ch,t}| > |S(V', t)| / 2
function S(V, t)
    return {v_k : [VOTE, ·, ·, ·, v_k] ∈ V' ∧ V' = FIL_{1-exp}(V, t)}
```

The protocol executed by validator v_i proceeds in four phases:

1. **Propose ($4\Delta t$):** If v_p is the proposer for slot t, it broadcasts $[\text{PROPOSE}, \mathsf{ch}_p, t, v_p]$ through gossip, where $\mathsf{ch}_p \succeq \text{MFC}(\mathcal{V}_p, \mathcal{V}_p, B_{\text{genesis}}, t)$ and $\mathsf{ch}_p.p = t$. In practice, $\text{MFC}(\mathcal{V}_i, \mathcal{V}_i, B_{\text{genesis}}, t)$ is the parent of ch_p, but we allow the more general case where ch_p is any extension of this parent with $\mathsf{ch}_p.p = t$.
2. **Vote ($4\Delta t+\Delta$):** Each validator computes $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, B_{\text{genesis}}, t)$, then gossips a vote message $[\text{VOTE}, \mathsf{ch}', \cdot, t, v_i]$, where ch' is a chain proposed

in slot t extending $\mathsf{ch}^{\mathsf{MFC}}$ with $\mathsf{ch}'.p = t$, or $\mathsf{ch}' = \mathsf{ch}^{\mathsf{MFC}}$ if no valid proposal exists.

3. **Decide** $(4\Delta t + 2\Delta)$**:** Set $\mathsf{Ch}_i = \mathrm{MFC}(\mathcal{V}_i'', \mathcal{V}_i, B_{\text{genesis}}, t)$, update $\mathcal{V}_i'' = \mathcal{V}_i$, and store $\mathcal{V}_i''$.
4. **Freeze** $(4\Delta t + 3\Delta)$**:** Set $\mathcal{V}_i^{\text{frozen}} = \mathcal{V}_i$ and store it for the next slot.

3.2 Probabilistically-Safe Variant

A probabilistically safe variant of the protocol just presented is obtained by observing that decisions do not influence validator behavior: they serve only as a confirmation mechanism for deterministic safety. In slot t, the decision phase stores $\mathcal{V}''$ for use in the next slot but otherwise has no effect. Removing this phase simplifies the protocol while preserving probabilistic safety. The main difference – aside from removing the decide phase and thus reducing the slot duration from 4Δ to 3Δ – is the introduction of a new probabilistic confirmation rule. During the vote phase, the protocol designates as *confirmed* the κ-deep prefix of the chain output by MFC. The parameter κ is chosen so that, within κ slots, an honest proposer is elected with overwhelming probability, ensuring that consensus reliably progresses and that this prefix will not later be reverted.

1. **Propose** $(3\Delta t)$**:** If v_i is proposer for slot t, it gossips $[\textsc{propose}, \mathsf{ch}_p, t, v_i]$, with $\mathsf{ch}_p \succeq \mathrm{MFC}(\mathcal{V}_i, \mathcal{V}_i, B_{\text{genesis}}, t)$ and $\mathsf{ch}_p.p = t$.
2. **Vote** $(3\Delta t{+}\Delta)$**:** Compute $\mathsf{ch}^{\mathsf{MFC}} := \mathrm{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, B_{\text{genesis}}, t)$, then gossip a vote message $[\textsc{vote}, \mathsf{ch}', \cdot, t, v_i]$, where ch' is a chain proposed in slot t extending $\mathsf{ch}^{\mathsf{MFC}}$ with $\mathsf{ch}'.p = t$, or $\mathsf{ch}' = \mathsf{ch}^{\mathsf{MFC}}$ if no valid proposal exists. Set the confirmed chain as the κ-deep prefix $\mathsf{Ch}_i = \left(\mathsf{ch}^{\mathsf{MFC}}\right)^{\lceil \kappa}$.
3. **Freeze** $(3\Delta t + 2\Delta)$**:** Set $\mathcal{V}_i^{\text{frozen}} = \mathcal{V}_i$, store it, and advance to slot $t + 1$.

Under synchrony, Constraint (1) (adapted to slots of duration 3Δ) holds. With an honest proposer in slot t, the proposed chain ch_p extends the lock (*i.e.*, $\mathsf{ch}_i^{3\Delta t+\Delta}$), ensuring all honest validators vote for ch_p. This alignment of votes yields Reorg Resilience, which in turn provides probabilistic safety under the κ-deep rule. The details are shown in Appendix, in the proof of the final protocol variant.

3.3 Probabilistically-Safe Variant with Fast Confirmations

The protocol described above confirms only chains proposed at least κ slots in the past. We now extend it with a mechanism for *fast confirmations*: under certain favorable conditions, a chain can be confirmed in the very slot in which it was proposed. Concretely, if all validators are awake and more than $\frac{2}{3}n$ of them vote for the same chain, that chain is *fast-confirmed* and may be adopted immediately in place of the κ-deep prefix.

The advantage of this confirmation rule, compared to the original one, is that chains proposed in slot t can now be confirmed already in slot t, rather

than waiting until slot $t+1$ as in the original design. The drawback is that, unlike the original rule, this fast confirmation requires full participation in the vote phase. However, since the protocol still retains the κ-deep confirmation mechanism, lack of full participation simply causes the protocol to fall back to confirming the κ-deep prefix of the chain output by MFC.

This capability (formalized in Algorithm 2) serves as a building block for our ebb-and-flow protocol.

Observe that κ-deep confirmation would suffice, but it yields poor latency, whereas fast confirmation would suffice but makes no progress under poor participation.

The Protocol. Algorithm 2 defines $\texttt{fastconfirmsimple}(\mathcal{V}, t) \mapsto (\mathsf{ch}^C, Q^C)$, where ch^C is a chain supported by more than $\frac{2}{3}n$ validators voting in slot t according to view $\mathcal{V}$, and Q^C is a *quorum certificate* consisting of the corresponding VOTE messages. If no such chain exists, it returns $(B_{\text{genesis}}, \emptyset)$.

The protocol also relies on the function $\texttt{Extend}(\mathsf{ch}, t)$: If $\mathsf{ch}.p < t$, then $\texttt{Extend}(\mathsf{ch}, t)$ produces a chain ch' of length t that extends ch and includes all transactions in $\mathit{txpool}^{\mathsf{propose}(t)}$. If instead $\mathsf{ch}.p \geq t$, the behavior of the function is left unspecified.

Each honest validator v_i maintains three local objects. First, the *message set* $\mathcal{V}_i^{\text{frozen}}$, which is the snapshot of $\mathcal{V}_i$ taken at time 3Δ in each slot. Second, the *chain variable* $\mathsf{ch}_i^{\text{frozen}}$, defined as the greatest fast-confirmed chain at time 3Δ in slot t; this variable is updated between 0 and Δ of slot $t+1$ using the PROPOSE message for slot $t+1$. Finally, the *confirmed chain* Ch_i, which is updated at time Δ in each slot and may be updated again at 2Δ.

For notation, if var is a state variable, we write var^r for its value after round r, or $\mathcal{V}_i^{\text{frozen},r}$ if it already has a superscript. For readability, we abbreviate $\mathsf{fconf}(t) := 4\Delta t + 2\Delta$ and $\mathsf{merge}(t) := 4\Delta t + 3\Delta$.

Algorithm 2 proceeds as follows:

1. **Propose ($4\Delta t$):** The proposer v_p broadcasts $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \cdot, t, v_p]$, where $\mathsf{ch}_p = \texttt{Extend}(\mathsf{ch}^{\mathsf{MFC}}, t)$ and $\mathsf{ch}^{\mathsf{MFC}} = \text{MFC}(\mathcal{V}_p, \mathcal{V}_p, B_{\text{genesis}}, t)$. The tuple (ch^C, Q^C) is computed via $\texttt{fastconfirmsimple}(\mathcal{V}_p, t)$.
2. **Vote ($4\Delta t + \Delta$):** Upon receiving a valid proposal (with Q^C certifying ch^C and $\mathsf{ch}_p.p = t$), validator v_i updates $\mathsf{ch}_i^{\text{frozen}}$. It then computes MFC using: (i) $\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t)}$, (ii) $\mathcal{V}_i^{\mathsf{vote}(t)}$, and (iii) $\mathsf{ch}_i^{\text{frozen}}$. By construction, the voting-phase output is always a prefix of the proposer's chain (the *Graded Delivery* property [7]). If the proposal extends this chain, v_i votes for it; otherwise it votes for its local fork-choice. It also updates Ch_i to the longer of: (i) the κ-deep prefix of its MFC output, and (ii) the previous Ch_i, unless the latter is not a prefix of the former.
3. **Fast Confirm ($\mathsf{fconf}(t)$):** If at least $2/3n$ validators vote for a chain $\mathsf{ch}^{\text{cand}}$, then v_i fast confirms it by setting $\mathsf{Ch}_i = \mathsf{ch}^{\text{cand}}$.
4. **Merge ($\mathsf{merge}(t)$):** At round $4\Delta t + 3\Delta$, v_i updates $\mathcal{V}_i^{\text{frozen}}$ and $\mathsf{ch}_i^{\text{frozen}}$ for use in slot $t+1$.

Algorithm 2 Probabilistically-safe variant of TOB-SVD with fast confirmations - code for v_i

Output
  $\mathsf{Ch}_i \leftarrow B_{\text{genesis}}$: confirmed chain of validator v_i
State
  $\mathcal{V}_i^{\text{frozen}} \leftarrow \{B_{\text{genesis}}\}$: snapshot of $\mathcal{V}$ at time $4\Delta t + 3\Delta$
  $\mathsf{ch}_i^{\text{frozen}} \leftarrow B_{\text{genesis}}$: snapshot of the fast-confirmed chain at time $4\Delta t + 3\Delta$
function `fastconfirmsimple`$(\mathcal{V}, t)$
  let $\mathsf{fast}^{\text{cands}} := \{\mathsf{ch} : |\{v_j : \exists \mathsf{ch}' \succeq \mathsf{ch} : \ [\text{VOTE}, \mathsf{ch}', \cdot, t, \cdot] \in \mathcal{V}\}| \geq \frac{2}{3}n\}$
  if $\mathsf{fast}^{\text{cands}} \neq \emptyset$ **then**
    let $\mathsf{fast}^{\text{cand}} := \max\left(\mathsf{fast}^{\text{cands}}\right)$
    let $Q := \{[\text{VOTE}, \mathsf{ch}', \cdot, t, \cdot] \in \mathcal{V} : \mathsf{ch}' \succeq \mathsf{fast}^{\text{cand}}\}$
    return $(\mathsf{fast}^{\text{cand}}, Q)$
  else
    return $(B_{\text{genesis}}, \emptyset)$
PROPOSE
at round $4\Delta t$ **do**
  if $v_i = v_p^t$ **then**
    let $(\mathsf{ch}^C, Q^C) := \texttt{fastconfirmsimple}(\mathcal{V}_i, t-1)$
    let $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i, \mathcal{V}_i, \mathsf{ch}^C, t)$
    let $\mathsf{ch}_p := \mathsf{Extend}(\mathsf{ch}^{\mathsf{MFC}}, t)$
    send message $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \cdot, t, v_i]$ through gossip
VOTE
at round $4\Delta t + \Delta$ **do**
  let $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, \mathsf{ch}_i^{\text{frozen}}, t)$
  $\mathsf{Ch}_i \leftarrow \max(\{\mathsf{ch} \in \{\mathsf{Ch}_i, (\mathsf{ch}^{\mathsf{MFC}})^{\lceil \kappa}\} : \mathsf{ch} \preceq \mathsf{ch}^{\mathsf{MFC}}\})$
  let $\mathsf{ch} :=$ PROPOSEd chain from slot t extending $\mathsf{ch}^{\mathsf{MFC}}$ and
    with $\mathsf{ch}.p = t$, if there is one, or $\mathsf{ch}^{\mathsf{MFC}}$ otherwise
  send message $[\text{VOTE}, \mathsf{ch}, \cdot, t, v_i]$ through gossip
FAST CONFIRM
at round $4\Delta t + 2\Delta$ **do**
  let $(\mathsf{fast}^{\text{cand}}, Q) := \texttt{fastconfirmsimple}(\mathcal{V}_i, t)$
  if $Q \neq \emptyset$ **then**
    $\mathsf{Ch}_i \leftarrow \mathsf{fast}^{\text{cand}}$
MERGE
at round $4\Delta t + 3\Delta$ **do**
  $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i$
  $(\mathsf{ch}_i^{\text{frozen}}, \cdot) \leftarrow \texttt{fastconfirmsimple}(\mathcal{V}_i, t)$

upon receiving a message $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \cdot, t, v_p]$
  at any round in $[4\Delta t, 4\Delta t + \Delta]$ **do**
  if valid$([\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \cdot, t, v_p]) \wedge \mathsf{ch}_i^{\text{frozen}} \preceq \mathsf{ch}_p^C$ **then**
    $\mathsf{ch}_i^{\text{frozen}} \leftarrow \mathsf{ch}_p^C$

In Appendix A, we prove that Algorithm 2 satisfies Reorg Resilience (Theorem 1) and Dynamic Availability (Theorem 2).

We first establish that if all active validators in slot t send a VOTE for a chain extending ch, then every validator active in slot $t{+}1$ will also vote for a chain extending ch, since MFC necessarily outputs such an extension (Lemma 2). It follows that if any validator fast confirms a chain ch in slot t, then all validators active in later slots will output an extension of ch via MFC (Lemma 3). A similar guarantee holds for chains proposed by honest validators (Lemma 4).

These results imply that the confirmed chain of any honest validator at round r_i – even if the validator is inactive at r_i – is always a prefix of the MFC output of any validator active in later rounds (Lemma 5). This property is the key ingredient in proving Reorg Resilience.

We further show that an honest proposer can include all transactions from the pool in its proposal (Lemma 6). Combined with the above lemmas, this yields Dynamic Availability w.o.p. (Theorem 2).

We prove that fast confirmations are achievable (Lemma 7). Although not a formal security property, this result is central to the practical responsiveness of the protocol.

4 Finality Component

The finality component of our protocol finalizes, in each slot, a chain that extends the one finalized in the previous slot. We introduce the following terminology.

Checkpoints. A *checkpoint* is a tuple $\mathcal{C} = (\mathsf{ch}, c)$, where ch is a chain and c is the slot in which ch is proposed for justification. By construction $c \geq \mathsf{ch}.p$. We use c to order checkpoints: $\mathcal{C} \leq \mathcal{C}'$ iff $\mathcal{C}.c \leq \mathcal{C}'.c$, with equality only for identical checkpoints.

Justification. A set of FIN-VOTEs forms a *supermajority link* $\mathcal{C} \to \mathcal{C}'$ if at least $2n/3$ validators issue FIN-VOTEs from $\mathcal{C}$ to $\mathcal{C}'$. A checkpoint is *justified* if either (i) $\mathcal{C} = (B_{\text{genesis}}, 0)$ or (ii) it is reachable from $(B_{\text{genesis}}, 0)$ by a chain of supermajority links. A chain ch is justified iff some justified checkpoint $\mathcal{C}$ satisfies $\mathcal{C}.\mathsf{ch} = \mathsf{ch}$. Given a view $\mathcal{V}$, $\mathcal{C}$ is justified in $\mathcal{V}$ if $\mathcal{V}$ contains the corresponding supermajority links. The justified checkpoint with largest slot in $\mathcal{V}$ is the *greatest justified checkpoint* $\mathcal{GJ}(\mathcal{V})$, whose chain is the *greatest justified chain*. Ties break arbitrarily.

Finalization. A checkpoint $\mathcal{C}$ is *finalized* if it is justified and there is a supermajority link $\mathcal{C} \to \mathcal{C}'$ with $\mathcal{C}'.c = \mathcal{C}.c{+}1$. A chain is finalized if it belongs to some finalized checkpoint. By definition $(B_{\text{genesis}}, 0)$ is finalized. Function $\mathsf{F}(\mathcal{V}, \mathcal{C})$ (Algorithm 3) returns true iff $\mathcal{C}$ is finalized in $\mathcal{V}$. The finalized checkpoint of highest slot is $\mathcal{GF}(\mathcal{V})$. A chain ch is finalized in $\mathcal{V}$ iff $\mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}).\mathsf{ch}$.

Finality Votes. Validators issue two types of votes. The standard VOTE message embeds a FIN-VOTE, a tuple $[\text{FIN-VOTE}, \mathcal{C}_1, \mathcal{C}_2, v_i]$ linking a source $\mathcal{C}_1$ to a target $\mathcal{C}_2$. A FIN-VOTE is *valid* if $\mathcal{C}_1.c < \mathcal{C}_2.c$ and $\mathcal{C}_1.\mathsf{ch} \preceq \mathcal{C}_2.\mathsf{ch}$. We write such a vote simply as $\mathcal{C}_1 \to \mathcal{C}_2$.

Equivocations. A validator v_i *equivocates* if it sends two VOTE messages in the same slot t for different chains $\mathsf{ch} \neq \mathsf{ch}'$.

Algorithm 3 Justification and Finalization

function valid($\mathcal{C}_1 \to \mathcal{C}_2$)
  return $\wedge\ \mathcal{C}_1.\mathsf{ch} \preceq \mathcal{C}_2.\mathsf{ch}$
    $\wedge\ \mathcal{C}_1.c < \mathcal{C}_2.c$

function $\mathsf{J}(\mathcal{C}, \mathcal{V})$
  return $\vee\ \mathcal{C} = (B_{\text{genesis}}, 0)$
    $\vee\ \exists \mathcal{S}, \mathcal{M} \subseteq \mathcal{V} : \wedge\ \mathsf{J}(\mathcal{S}, \mathcal{V})$
      $\wedge\ \text{valid}(\mathcal{S} \to \mathcal{C})$
      $\wedge\ |\{v_k : [\textsc{vote}, \cdot, \mathcal{S} \to \mathcal{C}, \cdot, v_k] \in \mathcal{M}\}| \geq \frac{2}{3}n$

function $\mathsf{F}(\mathcal{C}, \mathcal{V})$
  return $\vee\ \mathcal{C} = (B_{\text{genesis}}, 0)$
    $\vee\ \wedge\ \mathsf{J}(\mathcal{C}, \mathcal{V})$
      $\wedge\ \exists \mathcal{T}, \mathcal{M} \subseteq \mathcal{V} : \wedge\ \mathcal{T}.c = \mathcal{C}.c + 1$
        $\wedge\ \text{valid}(\mathcal{C} \to \mathcal{T})$
        $\wedge\ |\{v_k : [\textsc{vote}, \cdot, \mathcal{C} \to \mathcal{T}, \cdot, v_k] \in \mathcal{M}\}| \geq \frac{2}{3}n$

A validator v_i is subject to slashing (as introduced in Section 2) for two *distinct* FIN-VOTEs $\mathcal{C}_1 \to \mathcal{C}_2$ and $\mathcal{C}_3 \to \mathcal{C}_4$ if either of the following conditions holds: $\mathbf{E_1}$ (Double voting) if $\mathcal{C}_2.c = \mathcal{C}_4.c$, implying that a validator must not send distinct FIN-VOTEs for the same checkpoint slot; or $\mathbf{E_2}$ (Surround voting) if $\mathcal{C}_3.c < \mathcal{C}_1.c < \mathcal{C}_2.c < \mathcal{C}_4.c$.

In Appendix B, we prove that if a checkpoint (ch_j, c_j) is justified while another checkpoint (ch_f, c_f) is already finalized with $c_j \geq c_f$, then either ch_j extends ch_f, or at least $n/3$ validators must have violated one of the two slashing conditions (Lemma 8). In particular, we prove that if two conflicting chains are ever finalized, then at least $n/3$ validators must have misbehaved (Lemma 9).

5 Majorum

Algorithm 4 State and output variables of Majorum (Algorithm 5)

Output
  $\mathsf{chAva}_i \leftarrow B_{\text{genesis}}$: available chain
  $\mathsf{chFin}_i \leftarrow B_{\text{genesis}}$: finalized chain
State
  $\mathcal{V}_i^{\text{frozen}} \leftarrow \{B_{\text{genesis}}\}$: snapshot of $\mathcal{V}$ at time $4\Delta t + 3\Delta$
  $\mathsf{ch}_i^{\text{frozen}} \leftarrow B_{\text{genesis}}$: snapshot of the fast-confirmed chain at time $4\Delta t + 3\Delta$
  $\mathcal{GJ}_i^{\text{frozen}} \leftarrow (B_{\text{genesis}}, 0)$: latest frozen greatest justified checkpoint

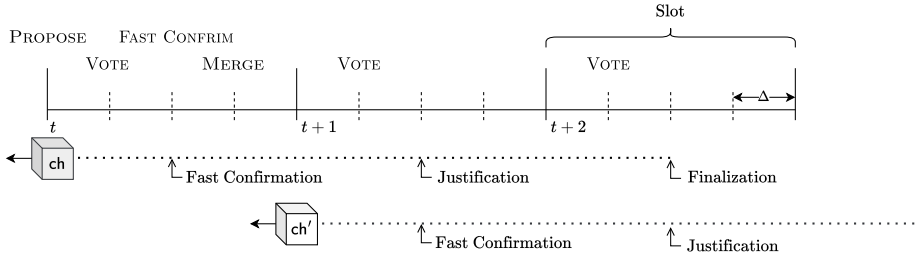

Fig. 1. An overview of the slot structure is shown, highlighting phases relevant to the fast confirmation and finalization of a proposal. A proposal ch is made in the first slot t (top left) and becomes fast-confirmed within the same slot. It is then justified in slot $t+1$ and finalized in slot $t+2$. Similarly, the next proposal ch' made in slot $t+1$ is fast-confirmed once ch is justified, and it is justified once ch is finalized. This illustrates how the confirmation and finalization process is pipelined across consecutive slots.

We now present Majorum (Algorithm 5), which integrates the dynamically available, reorg-resilient protocol of Algorithm 2 with the finality component from Section 4, yielding a secure ebb-and-flow protocol. We describe Majorum by highlighting the main differences from Algorithm 2. Figure 1 presents an overview of the happy path execution and corresponding slot structure.

The function $\texttt{fastconfirm}(\mathcal{V}, t)$ extends $\texttt{fastconfirmsimple}(\mathcal{V}, t)$ from Algorithm 2: it returns the same output if the fast-confirmed chain extends $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$, and otherwise returns $(\mathcal{GJ}(\mathcal{V}).\mathsf{ch}, \emptyset)$.

Compared to Algorithm 2, validators maintain one additional variable: the *checkpoint variable* $\mathcal{GJ}_i^{\text{frozen}}$, defined as the greatest justified checkpoint in v_i's view at time 3Δ of slot $t-1$. This variable is updated during the interval $[0, \Delta]$ of slot t based on the PROPOSE message.

Majorum outputs two chains:

1. **Available chain** chAva_i**:** corresponds to the confirmed chain Ch_i of Algorithm 2.
2. **Finalized chain** chFin_i**:** set at fast confirmation rounds to $\mathcal{GF}(\mathcal{V}_i).\mathsf{ch}$, and at vote rounds to the longest chain ch such that $\mathsf{chFin}_i \preceq \mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}$ and $\mathsf{chFin}_i \preceq \mathsf{chAva}_i$.

Relative to Algorithm 2, the main changes are:

1. **Propose:** The proposer for slot t includes in its PROPOSE message the greatest justified checkpoint in its current view.
2. **Vote:** A proposal $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \mathcal{GJ}_p, t, v_p]$ is valid only if $\mathcal{GJ}_p$ is justified in v_i's view. To allow FIN-VOTEs to accumulate, validators delay processing such proposals until $\mathsf{vote}(t)$. If valid and $\mathcal{GJ}_p \succ \mathcal{GJ}_i^{\text{frozen}}$, then v_i updates both $\mathcal{GJ}_i^{\text{frozen}}$ and $\mathsf{ch}_i^{\text{frozen}}$ (ensuring $\mathsf{ch}_i^{\text{frozen}} \succeq \mathcal{GJ}_i^{\text{frozen}}$). The available chain chAva_i is then set to the maximum among: (i) its previous value, (ii) the κ-deep prefix of the current fork-choice, and (iii) $\mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}$,

Algorithm 5 Majorum – code for validator v_i

function $\texttt{fastconfirm}(\mathcal{V}, t)$
  let $(\mathsf{ch}^C, Q) := \texttt{fastconfirmsimple}(\mathcal{V}, t)$
  if $\mathsf{ch}^C \succeq \mathcal{GJ}(\mathcal{V}).\mathsf{ch}$ **then return** (ch^C, Q)
  else return $(\mathcal{GJ}(\mathcal{V}).\mathsf{ch}, \emptyset)$
PROPOSE
at round $4\Delta t$ **do**
  if $v_i = v_p^t$ **then**
    let $(\mathsf{ch}^C, Q^C) := \texttt{fastconfirm}(\mathcal{V}_i, t-1)$
    let $\mathsf{ch}^{\mathsf{MFC}} := \mathrm{MFC}(\mathcal{V}_i, \mathcal{V}_i, \mathsf{ch}^C, t)$ and $\mathsf{ch}_p := \mathsf{Extend}(\mathsf{ch}^{\mathsf{MFC}}, t)$
    send message [PROPOSE, ch_p, ch^C, Q^C, $\mathcal{GJ}(\mathcal{V}_i)$, t, v_i] through gossip
VOTE
at round $4\Delta t + \Delta$ **do**
  let $\mathsf{ch}^{\mathsf{MFC}} := \mathrm{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, \mathsf{ch}_i^{\text{frozen}}, t)$
  $\mathsf{chAva}_i \leftarrow \max(\{\mathsf{ch} \in \{\mathsf{chAva}_i, (\mathsf{ch}^{\mathsf{MFC}})^{\lceil \kappa}, \mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}\} : \mathsf{ch} \preceq \mathsf{ch}^{\mathsf{MFC}}\})$
  $\mathsf{chFin}_i \leftarrow \max(\{\mathsf{ch} \colon \mathsf{ch} \preceq \mathsf{chAva}_i \wedge \mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}\})$
  if $\mathcal{GJ}_i^{\text{frozen}}.c = t-1$
    let $\mathcal{T} := (\mathsf{chAva}_i, t)$
  else
    let $\mathcal{T} := (\mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}, t)$
  let ch := PROPOSEd chain from slot t extending $\mathsf{ch}^{\mathsf{MFC}}$
    and with $\mathsf{ch}.p = t$, if there is one, or $\mathsf{ch}^{\mathsf{MFC}}$ otherwise
  send message [VOTE, ch, $\mathcal{GJ}_i^{\text{frozen}} \to \mathcal{T}$, t, v_i] through gossip
FAST CONFIRM
at round $4\Delta t + 2\Delta$ **do**
  let $(\mathsf{fast}^{\text{cand}}, \cdot) := \texttt{fastconfirm}(\mathcal{V}_i, t)$
  if $\mathsf{chAva}_i \not\succeq \mathsf{fast}^{\text{cand}}$ **then** $\mathsf{chAva}_i \leftarrow \mathsf{fast}^{\text{cand}}$
  $\mathsf{chFin}_i \leftarrow \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}$
MERGE
at round $4\Delta t + 3\Delta$ **do**
  $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i$
  $(\mathsf{ch}_i^{\text{frozen}}, \cdot) \leftarrow \texttt{fastconfirm}(\mathcal{V}_i, t)$
  $\mathcal{GJ}_i^{\text{frozen}} \leftarrow \mathcal{GJ}(\mathcal{V}_i)$

upon receiving a message [PROPOSE, $\mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \mathcal{GJ}_p, t, v_p$]
  at any round in $[4\Delta t, 4\Delta t + \Delta]$ **do**
  when round $4\Delta t + \Delta$ **do**
    if valid([PROPOSE, $\mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \mathcal{GJ}_p, t, v_p$]) $\wedge$ $\mathsf{J}(\mathcal{GJ}_p, \mathcal{V}_i) \wedge$
        $\mathcal{GJ}_p \geq \mathcal{GJ}_i^{\text{frozen}}$ **then**
      $\mathcal{GJ}_i^{\text{frozen}} \leftarrow \mathcal{GJ}_p$
      if $\mathsf{ch}_i^{\text{frozen}} \not\succeq \mathcal{GJ}_p.\mathsf{ch}$ **then**
        $\mathsf{ch}_i^{\text{frozen}} \leftarrow \mathcal{GJ}_p.\mathsf{ch}$
      if $\mathsf{ch}_i^{\text{frozen}} \preceq \mathsf{ch}_p^C$ **then**
        $\mathsf{ch}_i^{\text{frozen}} \leftarrow \mathsf{ch}_p^C$

discarding any option not extending the fork-choice. Each validator also includes a FIN-VOTE in its VOTE message, with source $\mathcal{GJ}_i^{\text{frozen}}$ and target

depending on its slot: if $\mathcal{GJ}_i^{\text{frozen}}$ is from slot $t-1$, the target is chAva_i; otherwise, the target is $\mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}$ lifted to slot t.

3. **Fast Confirm:** If $\texttt{fastconfirm}(\mathcal{V}_i, t)$ returns a candidate $\mathsf{ch}^{\text{cand}}$, then v_i sets $\mathsf{chAva}_i := \mathsf{ch}^{\text{cand}}$.
4. **Merge:** At round $4\Delta t + 3\Delta$, v_i updates $\mathcal{GJ}_i^{\text{frozen}}$ to the greatest justified checkpoint in its current view.

For $\mathsf{GST} = 0$, Appendix C shows that chAva in Majorum is dynamically available w.o.p. (Theorem 2) and reorg-resilient (Theorem 1).

At a high level, assuming $\mathsf{GST} = 0$, $f < n/3$, and Constraint (1), Appendix C.1 shows that integrating finality does not alter the core behavior of Algorithm 2. Concretely, we prove that: (i) honest validators send identical messages in both protocols except for the finality component, (ii) their fork-choice outputs coincide, and (iii) every chAva in Majorum is a confirmed chain of Algorithm 2. Thus, all lemmas and theorems proved in Appendix A for Algorithm 2 also apply to Majorum.

Finally, in Appendix C.2, we show that after time $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$, Majorum guarantees both accountable safety and liveness. In particular, it satisfies three key properties. First, *accountable safety*: honest validators are never slashed (Lemma 15); hence, the existence of two conflicting finalized chains implies that at least $\frac{n}{3}$ validators violated a slashing condition (Theorem 5). Second, *monotonicity*: the finalized chain chFin_i of any honest validator grows monotonically (Lemma 16). Third, *liveness*: after $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$, if two honest proposers are elected in consecutive slots, then the first proposer's chain is finalized within 2–3 slots (Theorem 6), depending on whether rejustification is needed. We conclude in Theorem 7 that Majorum satisfies the properties of a secure ebb-and-flow protocol w.o.p.

Communication Complexity Majorum has expected communication complexity $\mathcal{O}(Ln^3)$, where L is the block size. This matches the bound of the underlying dynamically available protocol TOB-SVD, since honest validators forward all received messages in each round, contributing to the $\mathcal{O}(Ln^3)$ cost. The finality gadget introduces no additional overhead, as it is embedded in the VOTE message at round $4\Delta t + \Delta$. Thus, the overall complexity remains identical to that of TOB-SVD.

6 Conclusion

We presented Majorum, a secure ebb-and-flow consensus protocol that achieves security and efficiency with only a single voting phase per slot. Building on Momose and Ren [13] and subsequent work, we adapted TOB-SVD as a dynamically available foundation and simplified it to probabilistic safety, yielding a protocol that is both simpler and more practical while retaining its guarantees. Combined with a finality protocol, Majorum finalizes chains within two or three slots under optimistic conditions, while preserving the communication complexity of its underlying dynamically available protocol.

References

1. Asgaonkar, A., D'Amato, F., Saltini, R., Zanolini, L.: Ethereum consensus property list, https://docs.google.com/document/d/1Q_iDOODIq-glLRPSMnSf3HAKDQGHXsENHzRG9iwok7g
2. Buterin, V., Griffith, V.: Casper the friendly finality gadget. CoRR **abs/1710.09437** (2017), http://arxiv.org/abs/1710.09437
3. Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., Zhang, Y.X.: Combining GHOST and Casper (2020)
4. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Seltzer, M.I., Leach, P.J. (eds.) Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999. pp. 173–186. USENIX Association (1999)
5. D'Amato, F., Losa, G., Zanolini, L.: Asynchrony-resilient sleepy total-order broadcast protocols. In: Gelles, R., Olivetti, D., Kuznetsov, P. (eds.) Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024. pp. 247–256. ACM (2024). https://doi.org/10.1145/3662158.3662779, https://doi.org/10.1145/3662158.3662779
6. D'Amato, F., Neu, J., Tas, E.N., Tse, D.: Goldfish: No more attacks on ethereum?! In: Clark, J., Shi, E. (eds.) Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part I. Lecture Notes in Computer Science, vol. 14744, pp. 3–23. Springer (2024). https://doi.org/10.1007/978-3-031-78676-1_1, https://doi.org/10.1007/978-3-031-78676-1_1
7. D'Amato, F., Saltini, R., Tran, T., Zanolini, L.: TOB-SVD: total-order broadcast with single-vote decisions in the sleepy model. In: 45th IEEE International Conference on Distributed Computing Systems, ICDCS 2025, Glasgow, United Kingdom, July 21-23, 2025. pp. 1033–1043. IEEE (2025). https://doi.org/10.1109/ICDCS63083.2025.00104, https://doi.org/10.1109/ICDCS63083.2025.00104
8. D'Amato, F., Zanolini, L.: Recent latest message driven GHOST: balancing dynamic availability with asynchrony resilience. In: 37th IEEE Computer Security Foundations Symposium, CSF 2024, Enschede, Netherlands, July 8-12, 2024. pp. 127–142. IEEE (2024). https://doi.org/10.1109/CSF61375.2024.00001, https://doi.org/10.1109/CSF61375.2024.00001
9. Gafni, E., Losa, G.: Brief announcement: Byzantine consensus under dynamic participation with a well-behaved majority. In: Oshman, R. (ed.) 37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy. LIPIcs, vol. 281, pp. 41:1–41:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.DISC.2023.41, https://doi.org/10.4230/LIPIcs.DISC.2023.41
10. Lewis-Pye, A., Roughgarden, T.: Permissionless consensus. CoRR **abs/2304.14701** (2023). https://doi.org/10.48550/ARXIV.2304.14701, https://doi.org/10.48550/arXiv.2304.14701
11. Malkhi, D., Momose, A., Ren, L.: Byzantine Consensus under Fully Fluctuating Participation (2022), https://eprint.iacr.org/archive/2022/1448/20221024:011919
12. Malkhi, D., Momose, A., Ren, L.: Towards practical sleepy BFT. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023. pp. 490–503. ACM (2023). https://doi.org/10.1145/3576915.3623073, https://doi.org/10.1145/3576915.3623073

13. Momose, A., Ren, L.: Constant latency in sleepy consensus. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 2295–2308. ACM (2022). https://doi.org/10.1145/3548606.3559347, https://doi.org/10.1145/3548606.3559347
14. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
15. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 446–465. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00045, https://doi.org/10.1109/SP40001.2021.00045
16. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: 42nd IEEE Symposium on Security and Privacy (2021), https://arxiv.org/abs/2009.04987, forthcoming
17. Nijkerk, M.: Ethereum briefly stopped finalizing transactions. what happened? https://www.coindesk.com/tech/2023/05/17/ethereums-loss-of-finality-what-happened (May 2023)
18. Pass, R., Shi, E.: The sleepy model of consensus. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 10625, pp. 380–409. Springer (2017)
19. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: Hotstuff: BFT consensus with linearity and responsiveness. In: Robinson, P., Ellen, F. (eds.) Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019. pp. 347–356. ACM (2019)
20. Zamfir, V.: Casper the friendly ghost. a correct-by-construction blockchain consensus protocol, https://github.com/ethereum/research/blob/master/papers/cbc-consensus/AbstractCBC.pdf

A Analysis of Algorithm 2

Lemma 1. *Consider a sequence of κ independent slots, where in each slot an honest validator is elected as proposer with probability at least $p = \frac{h_0}{n}$, where h_0 is the number of active honest validators and n is the total number of validators. For a finite time horizon T_{conf}, polynomial in κ, if p is non-negligible, then with overwhelming probability, i.e., with probability at least $1-\mathrm{negl}(\kappa)-\mathrm{negl}(\lambda)$, there exists at least one pair of consecutive slots in which honest proposers are elected.*

Proof. Let X_i denote the event that the i-th slot has an honest proposer. By assumption, each X_i occurs independently with probability at least p. Let E be the event that no two consecutive slots have honest proposers. We conservatively upper bound $P(E)$ by partitioning the sequence into $\lfloor \kappa/2 \rfloor$ disjoint groups of two consecutive slots. In each such pair, the probability that neither of the two adjacent pairs $(i, i+1)$ both contain honest proposers is at most $1 - p^2$.

Therefore, the probability of event E is at most:

$$P(E) \leq (1 - p^2)^{\lfloor \kappa/2 \rfloor}.$$

If p is non-negligible and κ is treated as a security parameter, then this upper bound becomes negligible in κ; i.e., $P(E) \leq \mathrm{negl}(\kappa)$.

Since cryptographic failures occur with probability $\mathrm{negl}(\lambda)$ (Section 2.2), it follows that the probability of failing to observe two consecutive honest proposers is at most $\mathrm{negl}(\kappa) + \mathrm{negl}(\lambda)$.

Hence, with overwhelming probability, there exists at least one pair of consecutive slots in which honest proposers are elected.

For any slot t, we define:

- $\mathrm{MFC}_i^{\mathsf{propose}(t)} := \mathrm{MFC}(\mathcal{V}_i^{\mathsf{propose}(t)}, \mathcal{V}_i^{\mathsf{propose}(t)}, \mathsf{ch}^C, t)$, where $(\mathsf{ch}^C, \cdot) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{\mathsf{propose}(t)}, t-1)$;
- $\mathrm{MFC}_i^{\mathsf{vote}(t)} := \mathrm{MFC}(\mathcal{V}_i^{\mathsf{frozen},\mathsf{vote}(t)}, \mathcal{V}_i^{\mathsf{vote}(t)}, \mathsf{ch}_i^{\mathsf{frozen},\mathsf{vote}(t)}, t)$.

Lemma 2. *If, in slot t, all validators in $H_{\mathsf{vote}(t)}$ send VOTE messages for chains extending chain ch, then, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $MFC_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $MFC_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ send VOTE messages for chains extending ch.*

Proof. Let v_i be any honest validator in $H_{\mathsf{vote}(t+1)}$. Due to the joining protocol, this implies that v_i is awake at round $\mathsf{merge}(t) = 4\Delta(t+1) - \Delta$. Clearly the same holds at time $\mathsf{propose}(t+1)$: $\mathsf{ch}_i^{\mathrm{frozen},\mathsf{propose}(t+1)} = B_{\mathrm{genesis}} \vee \mathsf{ch}_i^{\mathrm{frozen},\mathsf{propose}(t+1)} \succeq \mathsf{ch}$. Now we consider each case separately and for each of them prove that $\mathrm{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$.

Case 1: $\mathsf{ch}_i^{\mathrm{frozen},\mathsf{propose}(t+1)} = B_{\mathrm{genesis}}$**.** Note that due to the Lemma's and synchrony assumptions, all the VOTE messages sent by validators in $H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}$ are included in $\mathcal{V}_i^{\mathsf{propose}(t+1)}$. Hence, at time $\mathsf{propose}(t+1)$, $\left|\mathcal{V}_i^{\mathsf{ch},t+1}\right| \geq \left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right|$. Similarly, $H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)} \subseteq \mathsf{S}(\mathcal{V}_i, t+1)$. Also, $\mathsf{S}(\mathcal{V}_i, t+1) \setminus \left(H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right) \subseteq A_{\mathsf{vote}(t+1)} \cup \left(H_{\mathsf{vote}(t-1+1),\mathsf{vote}(t-1)} \setminus H_{\mathsf{vote}(t)}\right)$ as any VOTE in $\mathsf{S}(\mathcal{V}_i, t+1)$ that is not from a validator in $H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}$ must be either from a validator Byzantine at time $\mathsf{vote}(t+1)$ or from a validator active at some point between $\mathsf{vote}(t-1+1)$ and $\mathsf{vote}(t-1)$ but not active at time $\mathsf{vote}(t)$. Then, we have

$$\begin{aligned}
|\mathsf{S}(\mathcal{V}_i, t+1)| &= \left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right| + \left|\mathsf{S}(\mathcal{V}_i, t+1) \setminus \left(H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right)\right| \\
&\leq \left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right| + \left|A_{\mathsf{vote}(t+1)} \cup \left(H_{\mathsf{vote}(t-1+1),\mathsf{vote}(t-1)} \setminus H_{\mathsf{vote}(t)}\right)\right| \\
&< 2\left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right|
\end{aligned}$$

where the last inequality comes from Equation (1).

Hence, at time $\mathsf{propose}(t+1)$, $\left|\mathcal{V}_i^{\mathsf{ch},t+1}\right| \geq \left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right| > \frac{|\mathsf{S}(\mathcal{V}_i, t+1)|}{2}$.

Therefore, $\mathrm{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$.

Case 2: $\mathsf{ch}_i^{\mathrm{frozen},\mathsf{propose}(t+1)} \succeq \mathsf{ch}$**.** Given that by definition, $\mathrm{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}_i^{\mathrm{frozen},\mathsf{propose}(t+1)}$, we have that $\mathrm{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$.

Now let us move to proving that $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$. Note that if a $[\text{PROPOSE}, B, \mathsf{ch}_p^C, Q_p^C, t+1, v_p]$ message is valid, then $\mathsf{ch}_p^C = B_{\text{genesis}} \vee \mathsf{ch}_p^C \succeq \mathsf{ch}$. Line 36 implies that $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} = \mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t)} \vee \mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} = \mathsf{ch}_p^C$. Then, from $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t)} = B_{\text{genesis}} \vee \mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t)} \succeq \mathsf{ch}$ and Line 36, it follows that, at time $\mathsf{vote}(t+1)$, $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} = B_{\text{genesis}} \vee \mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} \succeq \mathsf{ch}$. Now we consider each case separately.

Case 1: $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} = B_{\text{genesis}}$**.** Note that due to the Lemma's and synchrony assumptions, all the VOTE messages sent by validators in $H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}$ are included in $\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t+1)} \cap \mathcal{V}_i^{\mathsf{vote}(t+1)}$. From here the proof is effectively the same as the one for Case 1 for $\text{MFC}_i^{\mathsf{propose}(t+1)}$, just with $\mathcal{V}_i^{\mathsf{propose}(t+1)}$ replaced by $\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t+1)} \cap \mathcal{V}_i^{\mathsf{vote}(t+1)}$.

Case 2: $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} \succeq \mathsf{ch}$**.** Similarly to proof of Case 2 for $\text{MFC}_i^{\mathsf{propose}(t+1)}$, given that by definition, $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)}$, we have that $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$.l

From $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$ follows that, in slot $t+1$, v_i sends a VOTE message for a chain extending ch.

Lemma 3. *If an honest validator fast confirms a chain* ch *in slot* t *(*i.e.*, there exists* Q *such that* $(\mathsf{ch}, Q) = \mathit{fastconfirmsimple}(\mathcal{V}_i^{\mathsf{fconf}(t)}, t) \wedge Q \neq \emptyset$*), then, for any slot* $t' > t$ *and validator* $v_i \in H_{\mathsf{vote}(t')}$*,* $\mathit{MFC}_i^{\mathsf{propose}(t')} \succeq \mathsf{ch}$ *and* $\mathit{MFC}_i^{\mathsf{vote}(t')} \succeq \mathsf{ch}$*, which implies that, all validators in* $H_{\mathsf{vote}(t')}$ *send a* VOTE *message for a chain extending* ch*.*

Proof. The proof is by induction on t'.

Base Case: $t' = t+1$**.** Assume that at round $\mathsf{fconf}(t)$, an honest validator in $H_{\mathsf{vote}(t)}$ fast confirms a chain ch. Given that we assume $f < \frac{n}{3}$, conflicting quorum certificates cannot form in the same slot. Then, due to the joining protocol and the synchrony assumption, at time $\mathsf{merge}(t) = 4\Delta(t+1) - \Delta$, $\mathsf{ch}_i^{\text{frozen, }\mathsf{merge}(t)} = \mathsf{ch}$ for any validator $v_i \in H_{\mathsf{vote}(t+1)}$.

Similarly, if a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, t+1, v_p]$ message is valid, then $\mathsf{ch}_p^C = B_{\text{genesis}} \vee \mathsf{ch}_p^C = \mathsf{ch}$. This and Lines 35 to 36 imply that, at time $\mathsf{vote}(t+1)$, $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)} = \mathsf{ch}$.

Hence, because $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t+1)}$, $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$, implying that all validators in $H_{\mathsf{vote}(t+1)}$ send VOTE messages extending chain ch.

Inductive Step: $t' > t+1$. Here we can just apply Lemma 2 to conclude the proof. □

Lemma 4. *Let* t *be a slot with an honest proposer* v_p *and assume that* v_p *sends a* $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, t, v_p]$ *message. Then, for any slot* $t' \geq t$*, all validators in* $H_{\mathsf{vote}(t')}$ *send a* VOTE *message for a chain extending* ch_p*. Additionally, for any slot* $t'' > t$ *and any validator* $v_i \in H_{\mathsf{vote}(t'')}$*,* $\mathit{MFC}_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ *and* $\mathit{MFC}_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$*.*

Proof. The proof is by induction on t'.

Base Case: $t' = t$**.** Suppose that in slot t, an honest proposer v_p sends a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C t, v_p]$ message. Consider an honest validator $v_i \in H_{\mathsf{vote}(t)}$. Note that due to the synchrony assumption, $\mathcal{V}_i^{\mathsf{merge}(t-1)} \subseteq \mathcal{V}_p^{\mathsf{propose}(t)}$. Given that we assume $f < \frac{n}{3}$, this further implies that $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} \neq B_{\text{genesis}} \implies \mathsf{ch}_p^C = \mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)}$. Hence, clearly $\mathsf{ch}_p^C \succeq \mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)}$. Therefore, due to Lines 35 to 36, $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$.

We know that either $\left|\left(\mathcal{V}_i^{\text{frozen}}\right)^{\text{MFC}_i^{\mathsf{vote}(t)},t} \cap \mathcal{V}_i^{\text{MFC}_i^{\mathsf{vote}(t)},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_i,t)|}{2}$ or $\text{MFC}_i^{\mathsf{vote}(t)} = \mathsf{ch}_i^{\text{frozen}}$. Let us consider each case separately.

Case 1: $\left|\left(\mathcal{V}^{\text{frozen}}\right)^{\mathbf{MFC}_i^{\mathsf{vote}(t)},t} \cap \mathcal{V}_i^{\mathbf{MFC}_i^{\mathsf{vote}(t)},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_i,t)|}{2}$**.** By the Graded Delivery property [7], this implies that at time $\mathsf{propose}(t)$, $\left|\mathcal{V}_p^{\text{MFC}_i^{\mathsf{vote}(t)},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_p,t)|}{2}$ meaning that, due to Line 18, $\mathsf{ch}_p \succeq \text{MFC}_i^{\mathsf{vote}(t)}$ and hence, due to Lines 24 and 25, in slot t, v_i sends a VOTE message for ch_p.

Case 2: $\mathbf{MFC}_i^{\mathsf{vote}(t)} = \mathsf{ch}_i^{\mathbf{frozen}}$**.** Due to Lines 24 and 25, v_i still sends a VOTE for ch_p as $\mathsf{ch}_p \succeq \mathsf{ch}_p^C = \mathsf{ch}_i^{\text{frozen}} = \text{MFC}_i^{\mathsf{vote}(t)}$.

Inductive Step: $t' > t$**.** Here we can just apply Lemma 2 to conclude the proof. □

Lemma 5. *Let r_i be any round and r_j be any round such that $r_j \geq r_i$ and $r_j \in \{\mathsf{propose}(\text{slot}(r_j)), \mathsf{vote}(\text{slot}(r_j))\}$. Then, for any validator v_i honest in round r_i and any validator $v_j \in H_{r_j}$, $\mathsf{Ch}_i^{r_i} \preceq MFC_j^{r_j}$.*

Proof. We proceed by contradiction. Let r_i be the smallest round such that there exist two honest validators v_i and v_j, and round r_j such that $r_j \geq r_i$ and $r_j \in \{\mathsf{propose}(\text{slot}(r_j)), \mathsf{vote}(\text{slot}(r_j))\}$ and $\mathsf{Ch}_i^{r_i} \not\preceq \text{MFC}_j^{r_j}$, that is, r_i is the first round where the chain confirmed by an honest validator conflicts with the output of MFC of (another) honest validator at a propose or vote round $r_j \geq r_i$. Given the minimality of r_i, $\mathsf{Ch}_i^{r_i-1} \neq \mathsf{Ch}_i^{r_i}$ which then implies that $v_i \in H_{r_i}$. This can only happen if r_i is either a voting or a fast confirmation round. Let $t_i = \text{slot}(r_i)$ and proceed by analyzing each case separately.

Case 1: r_i **is a vote round.** Due to Line 22, $\mathsf{Ch}_i^{r_i} \succeq \left(\text{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa}$. Let us now consider two sub cases.

Case 1.1: $\mathsf{Ch}_i^{r_i} = \left(\mathbf{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa}$**.** With overwhelming probability (Lemma 1), there exists at least one slot t_p in the interval $[t_i - \kappa, t_i)$ with an honest proposer v_p. Let ch_p be the chain PROPOSEd by v_p in slot t_p. Given that $t_p < t_i$, Lemma 4 implies that $\text{MFC}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p$. Then, because $t_p \geq t_i - \kappa$, we have that $\mathsf{ch}_p \succeq \left(\text{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa} = \mathsf{Ch}_i^{r_i}$.

Because $t_p < \text{slot}(r_j)$, Lemma 4 also implies that $\mathsf{ch}_j^{r_j} \succeq \mathsf{ch}_p \succeq \mathsf{Ch}_i^{r_i}$ leading to a contradiction.

Case 1.2: $\mathsf{Ch}_i^{r_i} \succ \left(\mathbf{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil \kappa}$**.** This case implies that $\mathsf{Ch}_i^{r_i} = \mathsf{Ch}_i^{r_i - 1}$. From the minimality of r_i we reach a contradiction.

Case 2: r_i **is a fast confirmation round.** Note that this implies that $t_i < \text{slot}(r_j)$. Because of the minimality of r_i, we only need to consider the case that $(\mathsf{Ch}_i^{r_i}, Q) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{r_i}, t_i) \wedge Q \neq \emptyset$. Therefore, we can apply Lemma 3 to conclude that $\mathsf{ch}_j^{r_j} \succeq \mathsf{Ch}_i^{r_i}$ reaching a contradiction. □

Theorem 1 (Reorg Resilience). *Algorithm 2 is reorg-resilient.*

Proof. Take a slot t_p with an honest proposer v_p who sends a PROPOSE message for chain ch_p. Take also any round r_i and validator v_i honest in round r_i. Now let t_j be any slot such that $t_j > \max(t_p, \text{slot}(r_i))$. Pick any validator $v_j \in H_{\mathsf{vote}(t_j)}$. Lemma 4 implies that $\text{MFC}_j^{\mathsf{vote}(t_j)} \succeq \mathsf{ch}_p$. Lemma 5 implies that $\text{MFC}_j^{\mathsf{vote}(t_j)} \succeq \mathsf{Ch}_i^{r_i}$. Hence, ch_p does not conflict with $\mathsf{Ch}_i^{r_i}$.

Lemma 6. *For any slot t and validator $v_i \in H_{\mathsf{propose}(t)}$, $MFC_i^{\mathsf{propose}(t)}.p < t$.*

Proof. Due to the Validity property of Graded Agreement [7] and the fact that $B_{\text{genesis}}.p < 0$, it is sufficient to prove that no validator honest in round $\mathsf{propose}(t)$ has ever sent a VOTE message for a chain ch with $\mathsf{ch}.p \geq t$. The proof is by induction.

Base Case: $t = 0$**.** Obvious as no validator in $H_{\mathsf{propose}(t)}$ has ever sent any message.

Inductive Step: $t > 0$**.** Due to the inductive hypothesis and the fact that honest nodes do not send any VOTE message in round $\mathsf{propose}(t-1)$, by the Validity property of Graded Agreement, we know that $\text{MFC}^{\mathsf{vote}(t-1)}.p < t - 1$. Then, the proof follows from Line 24. □

Theorem 2. *Algorithm 2 is dynamically available w.o.p..*

Proof. We begin by proving the Liveness of the protocol with a confirmation time of $T_{\text{conf}} = 8\kappa\Delta + \Delta$. We prove liveness by first considering the κ-deep rule only. Take a round r at slot $t = \text{slot}(r)$, another round $r_i \geq r + 8\kappa\Delta + \Delta \geq 4\Delta(t + 2\kappa) + \Delta = \mathsf{vote}(t + 2\kappa)$, and an honest validator $v_i \in H_{r_i}$. Let $t_i = \text{slot}(r_i)$. Due to the joining protocol, we know that the first active round for v_i, at or after $\mathsf{vote}(t + 2\kappa)$, is a vote round. There is a high probability of finding a slot $t_p \in [t + 1, t + \kappa]$ hosted by an honest proposer (Lemma 1). Let ch_p be the chain PROPOSEd by the honest proposer v_p in slot t_p. Due to Lemma 6 and Property 1, we know that ch_p includes all of the transaction in $txpool^{\mathsf{propose}(t_p)}$. Given that $\mathsf{propose}(t_p) \geq r$, $txpool^r \subseteq txpool^{\mathsf{propose}(t_p)}$, which implies that ch_p includes all of the transactions in $txpool^r$. Given that $\text{slot}(r_i) > t_p$, as a consequence of Lemma 4, $\text{MFC}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p$. Note that Line 22 implies that $\mathsf{Ch}_i^{r_i} \succeq \left(\text{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil \kappa, t_i}$. Then, because $t_p \leq t + \kappa \leq t_i - \kappa$, $\mathsf{ch}_p \preceq$

$\left(\text{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil \kappa, t_i}$ and hence $\mathsf{ch}_p \preceq \mathsf{Ch}_i^{r_i}$, which implies that $\mathsf{Ch}_i^{r_i}$ includes any transaction in $txpool^r$.

We now want to show that fast confirmation does not interfere. Given that $t_i \geq t_j$, from Lemma 4, we know that any VOTE sent in slot t_i by honest validators are for chains extending ch_p, given that we assume $f < \frac{n}{3}$, if r_i is a fast confirmation round and v_i sets $\mathsf{Ch}_i^{r_i}$, then $\mathsf{Ch}_i^{r_i} \succeq \mathsf{ch}_p$ still holds.

We now show safety. Take any two rounds r_i and r_j and validators v_i and v_j honest in round r_i and r_j respectively. Now let t_k be any slot such that $t_k > \max(\text{slot}(r_j), \text{slot}(r_i))$. By Constraint (1) we know that $H_{\mathsf{vote}(t_k)}$ is not empty. So, pick any validator $v_k \in H_{\mathsf{vote}(t_k)}$. Lemma 5 implies that $\text{MFC}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{Ch}_i^{r_i}$ and $\text{MFC}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{Ch}_j^{r_j}$. Hence, $\mathsf{Ch}_i^{r_i}$ does not conflict with $\mathsf{Ch}_j^{r_j}$.

Lemma 7 (Liveness of fast confirmations). *Take a slot t in which $|H_{\mathsf{vote}(t)}| \geq \frac{2}{3}n$. If in slot t an honest validator sends a PROPOSE message for chain ch_p, then, for any validator $v_i \in H_{\mathsf{fconf}(t)}$, $\mathsf{Ch}_i^{\mathsf{fconf}(t)} \succeq \mathsf{ch}_p$.*

Proof. By assumption we have that $|H_{\mathsf{vote}(t)}| \geq \frac{2}{3}n$, implying from Lemma 4 that every honest validator in $H_{\mathsf{vote}(t)}$ sends a VOTE message for ch_p. It follows that any validator $v_i \in H_{\mathsf{fconf}(t)}$ receives a quorum of VOTE messages for ch_p. Hence, due to Lines 28 and 29,$\mathsf{Ch}_i^{\mathsf{fconf}(t)} \succeq \mathsf{ch}_p$.

B Analysis of the Finality-Component

Lemma 8. *Assume $f \in [0, n]$ and let $\mathcal{C}_\mathsf{f} = (\mathsf{ch}_\mathsf{f}, c_\mathsf{f})$ be a finalized checkpoint and $\mathcal{C}_\mathsf{j} = (\mathsf{ch}_\mathsf{j}, c_\mathsf{j})$ be a justified checkpoint with $c_\mathsf{j} \geq c_\mathsf{f}$ according to any two views. Either $\mathsf{ch}_\mathsf{j} \succeq \mathsf{ch}_\mathsf{f}$ or at least $\frac{n}{3}$ validators can be detected to have violated either* $\mathbf{E_1}$ *or* $\mathbf{E_2}$.

Proof. First, we show that no checkpoint $\mathcal{C}' = (\mathsf{ch}', c_\mathsf{f})$, with $\mathsf{ch}' \neq \mathsf{ch}_\mathsf{f}$, can ever be justified. If that is not the case, clearly $\geq \frac{n}{3}$ validators are slashable for violating $\mathbf{E_1}$: For the justification of $(\mathsf{ch}_\mathsf{f}, c_\mathsf{f})$, it is required to have a supermajority of FIN-VOTEs with the chain of the target checkpoint being ch_f. Similarly, the justification of $(\mathsf{ch}', c_\mathsf{f})$ requires to have a supermajority of FIN-VOTEs with the chain of the target checkpoint being ch'. Given that we need a supermajority set of links for justification, the intersection of the sets of voters contributing to the justification of $\mathcal{C}_\mathsf{f}$ and $\mathcal{C}'$ include at least $\frac{n}{3}$ validators which have sent two FIN-VOTEs $\mathcal{S}_\mathsf{f} \to \mathcal{T}_\mathsf{f}$ and $\mathcal{S}' \to \mathcal{T}'$ with $\mathcal{T}_\mathsf{f}.c = \mathcal{T}'.c = c_\mathsf{f}$ and $\mathcal{T}_\mathsf{f}.\mathsf{ch} \neq \mathcal{T}'.\mathsf{ch}$ thereby violating condition $\mathbf{E_1}$.

Now, by contradiction, assume that $\mathsf{ch}_\mathsf{j} \not\succeq \mathsf{ch}_\mathsf{f}$ and that there does not exist a set of at least $\frac{n}{3}$ validators that can be detected to have violated either $\mathbf{E_1}$ or $\mathbf{E_2}$. Let $c'_\mathsf{j} > c_\mathsf{f}$ be the smallest slot for which a checkpoint $\mathcal{C}'_\mathsf{j} = (\mathsf{ch}'_\mathsf{j}, c'_\mathsf{j})$ is justified with $\mathsf{ch}'_\mathsf{j} \not\succeq \mathsf{ch}_\mathsf{f}$, *i.e.*, either ch'_j conflicts with ch_f or ch'_j is a strict prefix of ch_f. Given our assumptions we know that one such a checkpoint exists.

Let $(A_i, c_i) \to (\mathsf{ch}'_\mathsf{j}, c'_\mathsf{j})$ and $(\mathsf{ch}_\mathsf{f}, c_\mathsf{f}) \to (C, c_\mathsf{f}+1)$ be the FIN-VOTEs involved in the justification of $(\mathsf{ch}'_\mathsf{j}, c'_\mathsf{j})$ and in the finalization of ch_f, respectively, sent by a validator v_i. We observe two cases:

Case 1: $c'_\mathsf{j} = c_\mathsf{f} + 1$**.** If ch'_j conflicts with ch_f, then $A_i \preceq \mathsf{ch}'_\mathsf{j}$ implies $A_i \neq \mathsf{ch}_\mathsf{f}$, and thus the two votes are different. Conversely, if ch'_j is a strict prefix of ch_f, then $A_i \preceq \mathsf{ch}'_\mathsf{j} \prec \mathsf{ch}_\mathsf{f}$, and thus $A_i \neq \mathsf{ch}_\mathsf{f}$, which implies that the two votes are different in this case as well. Hence, in both cases, validator v_i violates condition $\mathbf{E_1}$ and therefore is slashable.

Case 2: $c'_\mathsf{j} > c_\mathsf{f} + 1$**.** We have to consider three cases:

- **Case 2.1:** $c_i > c_\mathsf{f}$**.** By the definition of justification, if the checkpoint $(\mathsf{ch}'_\mathsf{j}, c'_\mathsf{j})$ is justified, then the checkpoint (A_i, c_i) must also be justified. However, given that $c_i < c'_\mathsf{j}$, under the assumption of the minimality of c'_j, we have $\mathsf{ch}_\mathsf{f} \preceq A_i$. Given that $A_i \preceq \mathsf{ch}'_\mathsf{j}$, this leads to the contradiction $\mathsf{ch}_\mathsf{f} \preceq \mathsf{ch}'_\mathsf{j}$.
- **Case 2.2:** $c_i = c_\mathsf{f}$**.** As argued in the proof of the case above, the checkpoint (A_i, c_i) is justified. As proved at the beginning of this proof, either $A_i = \mathsf{ch}_\mathsf{f}$, or at least $\frac{n}{3}$ validators have violated condition $\mathbf{E_1}$. Consequently $\mathsf{ch}_\mathsf{f} = A_i \preceq \mathsf{ch}'_\mathsf{j}$ leading to a contradiction.
- **Case 2.3:** $c_i < c_\mathsf{f}$**.** Considering $c_i < c_\mathsf{f} < c_\mathsf{f} + 1 < c'_\mathsf{j}$, this situation constitutes a violation of $\mathbf{E_2}$ due to the existence of surrounding voting.

Given that justifications require supermajority link, the intersection of the set of voters involved in the finalization of $\mathcal{C}_\mathsf{f}$ and justification of $\mathcal{C}'_\mathsf{j}$ includes at least $\frac{n}{3}$ validators which, by the reasoning outlined above, would violated either $\mathbf{E_1}$ or $\mathbf{E_2}$. This leads to contradicting our assumptions which concludes the proof.

Lemma 9. *Assume $f \in [0, n]$. If two conflicting chains are finalized according to any two respective views, then at least $\frac{n}{3}$ validators can be detected to have violated either $\mathbf{E_1}$ or $\mathbf{E_2}$.*

Proof. Let ch_1 and ch_2 be two conflicting chains according to view $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively. By the definition of finalized chains, this implies that there exists two checkpoints $\mathcal{C}_1 = (\mathsf{ch}'_1, c_1)$ and $\mathcal{C}_2 = (\mathsf{ch}'_2, c_2)$ that are finalized according to view $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively, such that $\mathsf{ch}'_1 \succeq \mathsf{ch}_1$ conflicts with $\mathsf{ch}'_2 \succeq \mathsf{ch}_2$. Assume without loss of generality that $c_2 \geq c_1$. Given that finalization implies justification, we can apply Lemma 8 to conclude that if $\mathsf{ch}'_2 \not\succeq \mathsf{ch}'_1$, then at least $\frac{n}{3}$ validators can be detected to have violated either $\mathbf{E_1}$ or $\mathbf{E_2}$.

C Analysis of Majorum

C.1 Synchrony

Throughout this part of the analysis, we assume $\mathsf{GST} = 0$, that less than one-third of the entire validator set is ever controlled by the adversary (*i.e.*, $f < \frac{n}{3}$), and that Constraint (1) holds.

We start by formalizing the concept of validators sending the same messages except only for their finality component. Moreover, we define the equivalence between an execution in Algorithm 2 and one in Algorithm 5. These definitions will be leveraged upon in the subsequent Lemmas and Theorems.

Definition 8. *We say that two messages are* dynamically-equivalent *if and only if they differ in at most their finality-related components,* i.e.*, they either are* PROPOSE *messages and differ at most in the greatest justified checkpoint component or they are* VOTE *messages and differ at most in the* FIN-VOTE *component.*

We say that two sets of messages are dynamically-equivalent if and only if they have the same set of equivalence classes under type-equivalence, i.e.*, for any message m in any of the two sets, there exists a message m' in the other set such that m and m' are dynamically-equivalent.*

Given two executions e and e' and round r, we say that the two executions are honest-output-dynamically-equivalent up to round r *if and only if for any round $r_j < r$ and validator v_j honest in round r_j in both executions, the set of messages sent by v_j in round r_j in execution e is dynamically-equivalent to the set of messages sent by v_j in round r_j in execution e'.*

In the remainder of this section, we use the notation $\overset{e}{\mathcal{X}}$, where $\mathcal{X}$ is any variable or definition, to explicitly indicate the value of $\mathcal{X}$ in execution e. Due to the difference in the fast confirmation function employed by the two algorithms, we do need to explicitly specify the meaning of $\overset{e}{\text{MFC}}_i^{\mathsf{propose}(t)}$ with t being any slot. If e is an execution of Algorithm 2, then $\overset{e}{\text{MFC}}_i^{\mathsf{propose}(t)}$ corresponds the definition provided in Appendix A. If e is an execution of Algorithm 5, then $\overset{e}{\text{MFC}}_i^{\mathsf{propose}(t)} := \overset{e}{\text{MFC}}(\mathcal{V}_i^{\mathsf{propose}(t)}, \mathcal{V}_i^{\mathsf{propose}(t)}, \mathsf{ch}^C, t)$ with $(\mathsf{ch}^C, \cdot) = \texttt{fastconfirm}(\mathcal{V}_i^{\mathsf{propose}(t)}, t-1)$.

Definition 9. *Let e_{FIN} by a compliant execution of Algorithm 5 and e_{FIN} be a compliant execution of Algorithm 2. We say that e_{FIN} and e_{FIN} are* MFC-equivalent *if and only if the following constraints hold:*

1. *e_{FIN} and e_{FIN} are honest-output-dynamically-equivalent up to any round*
2. *for any slot t_i,*
   - 2.1 $\overset{e_{FIN}}{MFC}_i^{\mathsf{propose}(t_i)} = \overset{e_{FIN}}{MFC}_i^{\mathsf{propose}(t_i)}$
   - 2.2 $\overset{e_{FIN}}{MFC}_i^{\mathsf{vote}(t_i)} = \overset{e_{FIN}}{MFC}_i^{\mathsf{vote}(t_i)}$
   - 2.3 *for any slot $t_j \leq t_i$ and validator $v_j \in H_{\mathsf{vote}(t_j)}$, there exists a slot $t_k \leq t_j$ and a validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathsf{Ch}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_j^{\mathsf{vote}(t_j)}$.*
   - 2.4 *for any slot $t_j \leq t_i$ and validator $v_j \in H_{\mathsf{vote}(t_j)}$, there exists a slot $t_k \leq t_j$ and a validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathsf{Ch}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_j^{\mathsf{fconf}(t_j)}$.*

   where chain chAva_j is in e_{FIN}, and chain Ch_i is in e_{FIN}.

Lemma 10. *Let e_{FIN} by any compliant execution of Algorithm 5. There exists a compliant execution e_{FIN} of Algorithm 2 such that e_{FIN} and e_{FIN} are MFC-equivalent.*

Proof. Let e_{FIN} be any compliant execution of Algorithm 5 and let $A^{e_{\text{FIN}}}$ be the adversary in such an execution. Below, we specify the decisions made by an adversary $A^{e_{\text{FIN}}}$ in an execution e_{FIN} of Algorithm 2. Later we show that this leads to e_{FIN} satisfying all the conditions in the Lemma.

(i) For any round r, the set of validators corrupted by $A^{e_{\text{FIN}}}$ in round r corresponds to set of validators corrupted by $A^{e_{\text{FIN}}}$ in the same round r.
(ii) For any round r, the set of validators put to sleep by $A^{e_{\text{FIN}}}$ in round r corresponds to set of validators put to sleep by $A^{e_{\text{FIN}}}$ in the same round r.
(iii) For any slot t, if an honest validator is the proposer for slot t in e_{FIN}, then it is also the proposer for slot t in e_{FIN}.
(iv) For any round r, if e_{FIN} and e_{FIN} are honest-output-dynamically-equivalent up to round r, then $A^{e_{\text{FIN}}}$ schedules the delivery of messages to the validators honest in round r such that, for any validator v_i honest in round r_i, the set of messages received by v_i in execution e_{FIN} is dynamically-equivalent to the set of messages received by v_i in execution e_{FIN}.

Now, we prove by induction on t_i that the execution e_{FIN} induced by $A^{e_{\text{FIN}}}$ satisfies all Definition 9's conditions. To do so, we add the following conditions to the inductive hypothesis.

2.5 For any $\mathcal{J}$ such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FIN}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)})$, $\overset{e_{\text{FIN}}}{\text{MFC}}{}_i^{\mathsf{propose}(t_i)} \succeq \mathcal{J}.\mathsf{ch}$.
2.6 For any $\mathcal{J}$ such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FIN}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)})$, $\overset{e_{\text{FIN}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)} \succeq \mathcal{J}.\mathsf{ch}$.

and rephrase Condition 1 as follows

2.7 e_{FIN} and e_{FIN} are honest-output-dynamically-equivalent up to round $4\Delta(t_i+1)$.

Note that Condition 2.7 holding for any t_i implies Condition 1.

To reduce repetitions in the proof, we treat the base case of the induction within the inductive step. Also, we let $\mathcal{T}_j^{\mathsf{vote}(t_\ell)}$ be the target checkpoint used by validator v_j when sending an FIN-VOTE in slot t_ℓ.

Then, take any $t_i \geq 0$ and assume that, if $t_i > 0$, then the Lemma and the additional conditions Conditions 2.5 to 2.7 hold for slot $t_i - 1$. We prove that they also hold for slot t_i. We start by proving Conditions 2.5 and 2.6 for slot t_i, then we move to Conditions 2.1 to 2.4 in this order and conclude with proving Condition 2.7.

Conditions 2.5 and 2.6. Let $\mathcal{J}$ be any checkpoint such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FIN}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)})$. Because the target checkpoint of an FIN-VOTE sent by an honest validator v_ℓ in round r_ℓ corresponds to $(\overset{e_{\text{FIN}}}{\mathcal{T}}{}_\ell^{r_\ell}.\mathsf{ch}, \cdot)$ and we assume $f < \frac{n}{3}$, there exists a slot $t_k \in [0, t_i)$ and a validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathcal{T}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{J}.\mathsf{ch}$. By the inductive hypothesis, Condition 2.3 implies that there exists a slot $t_m \in [0, t_k]$ and validator $v_m \in H_{\mathsf{vote}(t_m)}$ such that $\mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)}$. Also, due to Lines 12, 15 and 17, celarly $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{T}_k^{\mathsf{vote}(t_k)}$. Given that

$t_m \leq t_k < t_i$, from Lemma 5, we know that $\overset{e_{\text{FIN}}}{\text{MFC}}^{\mathsf{propose}(t_i)} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{T}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{J}.\mathsf{ch}$ and that $\overset{e_{\text{FIN}}}{\text{MFC}}^{\mathsf{vote}(t_i)} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{T}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{J}.\mathsf{ch}$. Given that the set of justified checkpoints in $\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{vote}(t_i)}$ is a superset of the set of justified checkpoints in $\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{propose}(t_i)}$, both Condition 2.5 and Condition 2.6 are proven.

Conditions 2.1 and 2.2. We know that e_{FIN} and e_{FIN} are honest-output-dynamically-equivalent up to round $\mathsf{propose}(t_i)$. If $t_i > 0$, then, by the inductive hypothesis, Condition 2.7 implies this, if $t = 0$, then this is vacuously true. Hence, due to item (iv) of $A^{e_{\text{FIN}}}$'s set of decisions, $\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{propose}(t_i)}$ and $\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{propose}(t_i)}$ are dynamically-equivalent. From this follows that e_{FIN} and e_{FIN} are also honest-output-dynamically-equivalent up to round $\mathsf{vote}(t_i)$ and, therefore, $\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{vote}(t_i)}$ and $\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{vote}(t_i)}$ are dynamically-equivalent as well. This and Conditions 2.5 and 2.6 imply Conditions 2.1 and 2.2.

Condition 2.3. By Line 12, $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} \in \{\mathsf{chAva}_i^{\mathsf{fconf}(t_i-1)}, \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch}, (\overset{e_{\text{FIN}}}{\text{MFC}}_i^{\mathsf{vote}(t_i)})^{\lceil\kappa}\}$. Let us consider each case.

- **Case 1:** $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = \mathsf{chAva}_i^{\mathsf{fconf}(t_i-1)}$**.** This case implies $t_i > 0$. Hence, we can apply the inductive hypothesis. Specifically, Condition 2.4 for slot $t_i - 1$ implies Condition 2.3 for slot t_i.
- **Case 2:** $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = \mathcal{GJ}_i^{\text{frozen},t_i}.\mathsf{ch}$ Due to Lines 27, 33 and 34, $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},t_i}, \mathcal{V}^{\mathsf{vote}(t_i)})$. Hence, by following the reasoning applied when discussing Conditions 2.5 and 2.6, there exists a slot $t_k \in [0, t_i)$ and validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathcal{T}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$ from which we can conclude that $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$.
  Therefore, we can apply the inductive hypothesis. Specifically, Condition 2.3 for slot t_k implies Condition 2.3 for slot t_i.
- **Case 3:** $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = (\overset{e_{\text{FIN}}}{\text{MFC}}_i^{\mathsf{vote}(t_i)})^{\lceil\kappa}$**.** Line 22 of Algorithm 2 implies that $\mathsf{Ch}_i^{\mathsf{vote}(t_i)} \succeq \overset{e_{\text{FIN}}}{\text{MFC}}_i^{\mathsf{vote}(t_i)}$. Then, Condition 2.2 implies that $\mathsf{Ch}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$.

Condition 2.4. By Lines 21 to 23, $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} \in \{\mathsf{chAva}_i^{\mathsf{vote}(t_i)}, \mathcal{GJ}(\overset{e_{\text{FIN}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).\mathsf{ch}, \mathsf{ch}^C\}$ with $(\mathsf{ch}^C, Q) = \texttt{fastconfirm}(\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{fconf}(t_i)}, t_i) \wedge Q \neq \emptyset$. Let us consider each case.

- **Case 1:** $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$**.** In this case, Condition 2.3 implies Condition 2.4.
- **Case 2:** $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathcal{GJ}(\overset{e_{\text{FIN}}}{\mathcal{V}}_i^{\mathsf{fconf}(t_i)}).\mathsf{ch}$. By following the reasoning applied when discussing Conditions 2.5 and 2.6, this means that there exists a slot $t_k \in [0, t_i]$ and validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_i^{\mathsf{fconf}(t_i)}$. Then, Condition 2.3 for slot t_k implies Condition 2.4 for slot t_i.

Case 3: $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathsf{ch}^C$ **with** $(\mathsf{ch}^C, Q) = \texttt{fastconfirm}(\mathcal{V}_i^{e_{\text{FIN}}\mathsf{fconf}(t_i)}, t_i) \wedge Q \neq \emptyset$**.** This implies that $\mathcal{V}_i^{e_{\text{FIN}}\mathsf{fconf}(t_i)}$ includes a quorum of VOTE message for chain ch^C. Given Condition 2.2, $\mathcal{V}_i^{e_{\text{FIN}}\mathsf{fconf}(t_i)}$ also includes a quorum of VOTE message for chain ch^C. Hence, $\mathsf{Ch}_i^{\mathsf{fconf}(t_i)} = \mathsf{chAva}_i^{\mathsf{fconf}(t_i)}$.

Condition 2.7. As argued in the proof of Conditions 2.1 and 2.2, we know that e_{FIN} and e_{FIN} are honest-output-dynamically-equivalent up to round $\mathsf{propose}(t_i)$. This, Conditions 2.1 and 2.2 and item (iv) of $A^{e_{\text{FIN}}}$'s set of decisions clearly imply Condition 2.7 up to round $\mathsf{propose}(t_i + 1)$. □

In the following Lemmas and Theorems, unless specified, we refer to executions of Algorithm 5.

Lemma 11 (Analogous of Lemma 2). *If, in slot t, all validators in $H_{\mathsf{vote}(t)}$ send VOTE messages for chains extending chain ch, then, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $MFC_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $MFC_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ send VOTE messages for chains extending ch.*

Proof. Take any compliant execution e_{FIN} of Algorithm 5 where in slot t, all validators in $H_{\mathsf{vote}(t)}$ send VOTE messages for chains extending chain ch. Let e_{FIN} be an MFC-equivalent execution of Algorithm 2 which Lemma 10 proves to exist. It is clear from the definition of MFC-equivalent honest-output-dynamically-equivalent executions that in slot t of execution e_{FIN} all validators in $H_{\mathsf{vote}(t)}$ send VOTE messages for chains extending chain ch as well. Hence, by Lemma 2, in execution e_{FIN}, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $\text{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$. From Conditions 2.1 and 2.2 of Lemma 10 then it follows that this holds for execution e_{FIN} as well, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ send VOTE messages for chains extending ch.

Lemma 12 (Analogous of Lemma 4). *Let t be a slot with an honest proposer v_p and assume that v_p sends a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, t, v_p]$ message. Then, for any slot $t' \geq t$, all validators in $H_{\mathsf{vote}(t')}$ send a VOTE message for a chain extending ch_p. Additionally, for any slot $t'' > t$ and any validator $v_i \in H_{\mathsf{vote}(t'')}$, $MFC_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ and $MFC_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$.*

Proof. Take any compliant execution e_{FIN} of Algorithm 5 where v_p, the proposer of slot t, is honest and sends a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C t, v_p]$ message. Let e_{FIN} be an MFC-equivalent execution of Algorithm 2 which Lemma 10 proves to exist. It is clear from the definition of MFC-equivalent honest-output-dynamically-equivalent executions that v_p sends the same PROPOSE message in slot t of execution e_{FIN}. Hence, by Lemma 4, in execution e_{FIN}, for any slot $t' \geq t$, all validators in $H_{\mathsf{vote}(t')}$ send a VOTE message for a chain extending ch_p, and, for any slot $t'' > t$ and any validator $v_i \in H_{\mathsf{vote}(t'')}$, $\text{MFC}_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ and $\text{MFC}_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$. From Conditions 1, 2.1 and 2.2 of Lemma 10 then it follows that this holds for execution e_{FIN} as well.

Lemma 13 (Analogous of Lemma 5). *Let r_i be any round and r_j be any round such that $r_j \geq r_i$ and $r_j \in \{\mathsf{propose}(r_j), \mathsf{vote}(r_j)\}$. Then, for any validator v_i honest in round r_i and any validator $v_j \in H_{r_j}$, $\mathsf{chAva}_i^{r_i} \preceq MFC_j^{r_j}$.*

Proof. Take any compliant execution e_{FIN} of Algorithm 5. Let e_{FIN} be an equivalent execution of Algorithm 2 which Lemma 10 proves to exist. Let r_i be any round and r_j be any round such that $r_j \geq r_i$ and $r_j \in \{\mathsf{propose}(r_j), \mathsf{vote}(r_j)\}$. We know that $\mathsf{chAva}_i^{r_i}$ is first set by v_i in round $r_i' \leq r_i$ such that $r_i' \in \{\mathsf{vote}(\text{slot}(r_i')), \mathsf{fconf}(\text{slot}(r_i'))\}$. From Conditions 2.3 and 2.4 of Lemma 10, we know that there exits a slot $t_m \leq r_k$ and a validator $v_m \in H_{\mathsf{vote}(t_m)}$ such that $\mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_i^{r_i'} = \mathsf{chAva}_i^{r_i}$. Lemma 5 implies that $\overset{e_{\text{FIN}}}{\text{MFC}}{}_j^{r_j} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)}$. From Lemma 10, we know that $\overset{e_{\text{FIN}}}{\text{MFC}}{}_j^{r_j} = \overset{e_{\text{FIN}}}{\text{MFC}}{}_j^{r_j}$. Hence, $\overset{e_{\text{FIN}}}{\text{MFC}}{}_j^{r_j} = \overset{e_{\text{FIN}}}{\text{MFC}}{}_j^{r_j} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_i^{r_i'} = \mathsf{chAva}_i^{r_i}$.

Theorem 3 (Reorg Resilience - Analogous of Theorem 1). *Algorithm 5 is reorg-resilient.*

Proof. The proof follows the one for Theorem 1 with the following changes only.

1. Replace Ch with chAva,
2. Replace Lemma 4 with Lemma 12, and
3. Replace Lemma 5 with Lemma 13. □

Theorem 4 (dynamic-availability - Analogous of Theorem 2). *Algorithm 5 is dynamically available w.o.p..*

Proof. Note that Lemma 6 holds for Algorithm 5 as well. Then, the proof follows the one for Theorem 2 with the following changes only.

1. Replace Ch with chAva,
2. Replace the reference to Line 22 of Algorithm 2 with Line 12 of Algorithm 5,
3. Replace Lemma 4 with Lemma 12,
4. Replace Lemma 5 by Lemma 13, and
5. In the proof of liveness, consider the following additional case. Assume that $r_i \geq \mathsf{vote}(t + 2\kappa)$ is a fast confirmation round and that $\mathsf{chAva}_i^{r_i}$ is set to $\mathcal{GJ}(\mathcal{V}_i^{r_i}).\mathsf{ch}$ via Line 23. Because the chain of the target checkpoint of an FINVOTE sent by an honest validator v_ℓ in round r_ℓ corresponds to $(\overset{e_{\text{FIN}}}{\mathcal{T}}{}_\ell^{r_\ell}.\mathsf{ch}, \cdot)$, with $\overset{e_{\text{FIN}}}{\mathsf{chAva}}{}_\ell^{r_\ell}.\mathsf{ch} \succeq \overset{e_{\text{FIN}}}{\mathcal{T}}{}_\ell^{r_\ell}.\mathsf{ch}$, and we assume $f < \frac{n}{3}$, there exists a slot $t_k \in [0, \text{slot}(r_i)]$ and a validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{GJ}(\mathcal{V}_i^{r_i}).\mathsf{ch} = \mathsf{chAva}_i^{r_i}$. We have already established that $\mathsf{chAva}_i^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathsf{ch}_p$. Then, Lemma 13 implies that $\text{MFC}_i^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathcal{GJ}(\mathcal{V}_i^{r_i}).\mathsf{ch} = \mathsf{chAva}_i^{r_i}$ and $\text{MFC}_i^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathsf{chAva}_i^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathsf{ch}_p$.
This implies that $\mathsf{chAva}_i^{\mathsf{vote}(\text{slot}(r_i))}$ and $\mathcal{GJ}(\mathcal{V}_i^{r_i}).\mathsf{ch}$ do not conflict. Hence, given that we assume that Line 23 is executed, $\mathsf{chAva}_i^{\mathsf{vote}(\text{slot}(r_i))} \prec \mathcal{GJ}(\mathcal{V}_i^{r_i}).\mathsf{ch}$. Because, $\mathsf{chAva}_i^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathsf{ch}_p$, we can conclude that $\mathsf{chAva}_i^{r_i} = \mathcal{GJ}(\mathcal{V}_i^{r_i}).\mathsf{ch} \succ \mathsf{chAva}_i^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathsf{ch}_p$. □

Lemma 14 (Analogous of Lemma 7). *Take a slot t in which $|H_{\mathsf{vote}(t)}| \geq \frac{2}{3}n$. If in slot t an honest validator sends a* PROPOSE *message for chain ch_p, then, for any validator $v_i \in H_{\mathsf{fconf}(t)}$, $\mathsf{chAva}_i^{\mathsf{fconf}(t)} \succeq \mathsf{ch}_p$.*

Proof. The proof follows the one for Lemma 7 with the only change being replacing Ch_i with chAva_i.

C.2 Partial synchrony

In this section, we show that Algorithm 5 ensures that the chain chFin is always Accountably Safe and is live after time $\max(\mathsf{GST}, \mathsf{GAT}) + \Delta$, meaning that Algorithm 5 is a secure ebb-and-flow protocol under the assumption that $f < \frac{n}{3}$.

We start by showing that honest validators following Algorithm 5 are never slashed.

Lemma 15. *Honest validators following Algorithm 5 are never slashed.*

Proof. First of all, observe that

1. By Line 20, an honest active validator sends only one FIN-VOTE in any slot.
2. As a consequence of Line 15, if $\mathcal{T}$ is the target checkpoint in the FIN-VOTE sent in slot t, then $\mathcal{T}.c = t$.
3. Take any two slots t and t' with $t < t'$ such that $v_i \in H_{\mathsf{vote}(t)}$ and $v_i \in H_{\mathsf{vote}(t')}$. Let $\mathcal{S}^\ell$ be the source checkpoint of the FIN-VOTE that validator v_i sent in any slot ℓ. Then $\mathcal{S}^t.c \leq \mathcal{S}^{t'}.c$. Specifically, take any two slots k and k' such that $t \leq k < k' \leq t'$, $v_i \in H_{\mathsf{vote}(k)}$ and k' is the first slot after k such that $v_i \in H_{\mathsf{vote}(k')}$. By Line 20, we know that $\mathcal{S}^k = \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(k)}$ and by Lines 27 and 33, we have that $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(k)}, \mathcal{V}_i^{\mathsf{vote}(k)})$. Then, note that $k \leq k' - 1$ and that because of the joining protocol, we know that v_i is awake at round $\mathsf{merge}(k'-1)$. Hence, we have that $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(k)}.c \leq \mathcal{GJ}(\mathcal{V}_i^{\mathsf{merge}(k'-1)}).c = \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(k'-1)}.c$. Finally, by Lines 33 and 34, $\mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(k'-1)}.c \leq \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(k')}.c = \mathcal{S}^{k'}.c$ and $\mathcal{S}^k.c \leq \mathcal{S}^{k'}.c$.

Now, take any round r and any validator v_i honest in round r. Take also any two different FIN-VOTEs $\mathcal{C}_1 \to \mathcal{C}_2$ and $\mathcal{C}_3 \to \mathcal{C}_4$ that v_i has sent by round r. If no such FIN-VOTEs exists, then clearly v_i cannot be slashed. By 1 and 2 above we have that $\mathcal{C}_2.c \neq \mathcal{C}_4.c$. Hence, $\mathbf{E_1}$ is not violated. Then, without loss of generality, assume $\mathcal{C}_2.c < \mathcal{C}_4.c$. This means that FIN-VOTE $\mathcal{C}_1 \to \mathcal{C}_2$ has been sent before FIN-VOTE $\mathcal{C}_3 \to \mathcal{C}_4$. Then, by 3 above, we know that $\mathcal{C}_1.c \leq \mathcal{C}_3.c$. Given that $\mathcal{C}_2.c < \mathcal{C}_4.c$, $\mathbf{E_2}$ cannot be violated. Therefore, until v_i is honest, it cannot be slashed. □

Let $\mathcal{V}_{\mathsf{G}}^r$ denote the global view at time r, that is, the set of all messages sent up to and including time r. The chain chFin_i^r output by an honest validator v_i in round r of Algorithm 5 is any chain that is finalized according to the view $\mathcal{V}_{\mathsf{G}}^r$. This directly follows from Line 3 and lines 13 and 24 and the definition of $\mathcal{GF}$.

Theorem 5 (Accountable Safety). *Let $f \in [0, n]$. The finalized chain* chFin *is $\frac{n}{3}$-accountable.*

Proof. Follows from Lemmas 9 and 15.

We now show that the finalized chain of an honest validator grows monotonically. Although this property is not explicitly defined in Section 2.2, it is frequently desired by Ethereum protocol implementers [1].

Lemma 16. *For any two round $r' \geq r$ and validator v_i honest in round r', $\mathsf{chFin}_i^{r'} \succeq \mathsf{chFin}_i^{r}$.*

Proof. Let us prove this Lemma by contradiction. Let r be the smallest round such that there exist round $r' \geq r$ and a validator v_i honest in round r' such that $\mathsf{chFin}_i^{r'} \not\succeq \mathsf{chFin}_i^{r}$. Clearly $r' > r$. Assume r' to be the smallest such round. First, we want to show that there exists a checkpoint $\mathcal{J}$ such that $\mathsf{chAva}_i^{r'} \succeq \mathcal{J}.\mathsf{ch} \wedge \mathsf{J}(\mathcal{J}, \mathcal{V}_i^{r'}) \wedge\ \mathcal{J} \geq \mathcal{GF}(\mathcal{V}_i^{r})$. Note that both r are r' are either a vote or a fast confirmation round. This is due to the minimality of r and r' and the fact that chFin_i is only set in these types of rounds. Then, let us proceed by cases.

Case 1: r' is a vote round. In this case $\mathcal{J} = \mathcal{GJ}_i^{\text{frozen},r'}$. Note that due to Lines 27, 28, 35 and 36, $\mathsf{ch}_i^{\text{frozen},r'} \succeq \mathcal{GJ}_i^{\text{frozen},r'}.\mathsf{ch}$. Then, Line 12 implies that $\mathsf{chAva}_i^{r'} \succeq \mathcal{GJ}_i^{\text{frozen},r'}.\mathsf{ch}$. Also, Lines 28, 33 and 34 imply that $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},r'}, \mathcal{V}_i^{r'})$. Finally, Lines 28, 33 and 34, and $r' > r$ imply that $\mathcal{GJ}_i^{\text{frozen},r'} \geq \mathcal{GJ}(\mathcal{V}_i^{r})$.

Case 2: r' is a fast confirmation round. In this case, $\mathcal{J} = \mathcal{GJ}(\mathcal{V}_i^{r'})$. Clearly, $\mathsf{J}(\mathcal{GJ}(\mathcal{V}_i^{r'}), \mathcal{V}_i^{r'})$. Lines 22 to 23 ensure that $\mathsf{chAva}_i^{r'} \succeq \mathcal{GJ}(\mathcal{V}_i^{r'}).\mathsf{ch}$. Then, from Algorithm 3 and the fact that $\mathcal{V}_i^{r} \subseteq \mathcal{V}_i^{r'}$, we can conclude that $\mathcal{GJ}(\mathcal{V}_i^{r'}) \geq \mathcal{GF}(\mathcal{V}_i^{r'}) \geq \mathcal{GF}(\mathcal{V}_i^{r}).c$.

From Algorithm 3 and $\mathcal{V}_i^{r} \subseteq \mathcal{V}_i^{r'}$, we also have that $\mathcal{GF}(\mathcal{V}_i^{r'}) \geq \mathcal{GF}(\mathcal{V}_i^{r})$. Then, Lemma 8 and $f < \frac{n}{3}$ imply $\mathsf{chAva}_i^{r'} \succeq \mathcal{J}.\mathsf{ch} \succeq \mathcal{GF}(\mathcal{V}_i^{r}).\mathsf{ch} \succeq \mathsf{chFin}_i^{r}$ and $\mathcal{GF}(\mathcal{V}_i^{r'}).\mathsf{ch} \succeq \mathcal{GF}(\mathcal{V}_i^{r}).\mathsf{ch}$, which further imply $\mathsf{chFin}_i^{r'} \succeq \mathcal{GF}(\mathcal{V}_i^{r}).\mathsf{ch}$. Hence, $\mathsf{chFin}_i^{r'} \succeq \mathcal{GF}(\mathcal{V}_i^{r}).\mathsf{ch} \succeq \mathsf{chFin}_i^{r}$ reaching a contradiction.

Now let us move to the liveness property.

Lemma 17. *Let t be a slot with an honest proposer v_p such that $\mathsf{propose}(t) \geq \mathsf{GST}+\Delta$ and assume that v_p sends a $[\textsc{propose}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \mathcal{GJ}_p, t, v_p]$ message. Then, for any validator $v_i \in H_{\mathsf{vote}(t)}$, v_i sends a VOTE message for chain ch_p and $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t)} = \mathcal{GJ}_p$.*

Proof. Suppose that in slot t, an honest proposer sends a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \mathcal{GJ}_p, t, v_p]$ message. Consider an honest validator v_i in $H_{\mathsf{vote}(t)}$. Note that due to the synchrony assumption, $\mathcal{V}_i^{\mathsf{merge}(t-1)} \subseteq \mathcal{V}_p^{\mathsf{propose}(t)}$. Hence, $\mathcal{GJ}_p \geq \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(t-1)}$. Now, we want to show that $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$. Let us consider two cases.

Case 1: $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} \succ \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(t-1)}.\mathsf{ch}$**.** This implies that at time $\mathsf{merge}(t-1)$, v_i has received a quorum of VOTE messages for $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)}$. Let us consider three sub cases.

- **Case 1.1:** $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} = \mathcal{GJ}_p.\mathsf{ch}$**.** Given that $\mathsf{ch}_p^C \succeq \mathcal{GJ}_p.\mathsf{ch}$, Lines 37 and 38 set $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$.
- **Case 1.2:** $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} \succ \mathcal{GJ}_p.\mathsf{ch}$**.** This implies that $\mathcal{V}_p^{\mathsf{propose}(t)}$ include a quorum of VOTE messages for ch_p^C. Due to the synchrony assumption, the quorum of VOTE messages for $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)}$ are in the view of validator v_p at time $\mathsf{propose}(t)$. Given that $f < \frac{n}{3}$, this case implies that $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} = \mathsf{ch}_p^C$, and therefore $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$.
- **Case 1.3:** $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} \not\succeq \mathcal{GJ}_p.\mathsf{ch}$**.** In this case, Lines 35 to 38 set $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$.

Case 2: $\mathsf{ch}_i^{\text{frozen},\mathsf{merge}(t-1)} = \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(t-1)}.\mathsf{ch}$**.** Given that $\mathsf{ch}_p^C \succeq \mathcal{GJ}_p$ and $\mathcal{GJ}_p \geq \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(t-1)}$, Lines 33 to 38 imply that $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$.

Now, first suppose that at round $\mathsf{vote}(t)$ there exists a chain $\mathsf{ch} \succeq \mathsf{ch}_i^{\text{frozen}} = \mathsf{ch}_p^C$ such that $\left|\left(\mathcal{V}^{\text{frozen}}\right)^{\mathsf{ch},t} \cap \mathcal{V}_i^{\mathsf{ch},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_i,t)|}{2}$. By the Graded Delivery property [7], this implies that at time $\mathsf{propose}(t)$, $\left|\mathcal{V}_p^{\mathsf{ch},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_p,t)|}{2}$ meaning that $\mathsf{ch}_p \succeq \mathsf{ch}$ and hence, in slot t, v_i sends a VOTE message for ch_p.

If no such a chain exists, v_i still sends a VOTE for ch_p as $\mathsf{ch}_p \succeq \mathsf{ch}_p^C = \mathsf{ch}_i^{\text{frozen}}$.

Also, given that $\mathcal{GJ}_p \geq \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(t-1)}$, due to Lines 33 and 34, $\mathcal{GJ}^{\text{frozen},\mathsf{vote}(t)} = \mathcal{GJ}_p$.

Lemma 18. *Assume that $f < \frac{n}{3}$. Let v_p be any validator honest by the time it* PROPOSEs *chain ch_p in a slot t such that $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$ and assume that $\mathcal{GJ}_p^{\mathsf{propose}(t)}.c < t-1$. Then, chain ch_p is justified at slot $t+1$, and finalized at slot $t+2$.*

Proof. Let ch_p be the chain proposed by the honest proposer at slot t, with $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$, extending both the output of the fork-choice function MFC and the greatest justified checkpoint $\mathcal{GJ}_p$ with $\mathcal{GJ}_p^{\text{frozen}}.c < t-1$.

By Line 17 and Lemma 17, each honest validator sets the target checkpoint to the same checkpoint $\mathcal{T} := (\mathcal{GJ}_p^{\mathsf{propose}(t)}.\mathsf{ch}, t)$.

Then, again due to Lemma 17, in slot t, all honest validators v_i send VOTE messages of the form [VOTE, ch_p, $\mathcal{GJ}_p^{\mathsf{propose}(t)} \to \mathcal{T}$, t, v_i], which justify $\mathcal{T}$ (since $f < n/3$ ensures sufficient honest quorum).

Consequently, in slot $t+1$, $\mathcal{T}$ is the greatest justified checkpoint in the view of all honest validators and is used as the source of the FIN-VOTEs. Lemma 17 also implies that $\mathsf{chAva}_i^{\mathsf{vote}(t+1)} = \mathsf{ch}_p$.

Therefore, due to Lines 14 and 15, in slot $t+1$, any honest validator sends FIN-VOTEs with source $\mathcal{T}$ and target $(\mathsf{ch}_p, t+1)$, thereby justifying $(\mathsf{ch}_p, t+1)$. In slot $t+2$, all honest validators vote with source $(\mathsf{ch}_p, t+1)$, and by round

$4\Delta(t+2)+2\Delta$, these votes are visible to all honest validators, finalizing ch_p at slot $t+2$.

Theorem 6 (Liveness). *Assume $f < \frac{n}{3}$, and let t and $t+1$ be slots after $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$. Then, the chain ch_p proposed by the proposer of slot t is finalized at slot $t+2$ or slot $t+3$.*

In particular, for any validator $v_i \in H_r$, where $r \geq 4\Delta(t+3)+2\Delta$, it holds that $\mathsf{ch}_p \preceq \mathsf{chFin}_i^r$ and $txpool^{\mathsf{propose}(t)} \subseteq \mathsf{chFin}_i^r$.

Proof. Let v_{p_1} and v_{p_2} be the honest proposers in slot t and $t+1$, respectively. Let us first assume that $\mathcal{GJ}_p^{\mathsf{propose}(t)}.c = t-1$. We distinguish two cases:

1. **Case 1: The slot of the greatest justified checkpoint in the view of proposer v_{p_2} at propose time in slot $t+1$ is t, i.e., $\mathcal{GJ}_{p_2}^{\mathsf{propose}(t+1)} = \mathcal{C} = (\mathsf{ch}', t)$ for some chain ch'.** By Lemma 17, and Lines 14 and 15, all honest validators v_i in slot $t+1$ have $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t+1)} = \mathcal{GJ}_{p_2}^{\mathsf{propose}(t+1)} = \mathcal{C}$, and send FIN-VOTEs with source $\mathcal{C}$ and target $(\mathsf{ch}_p, t+1)$. Since $f < \frac{n}{3}$, these votes justify $(\mathsf{ch}_p, t+1)$ in slot $t+1$. In slot $t+2$, honest validators vote with source $(\mathsf{ch}_p, t+1)$. By round $4\Delta(t+2)+2\Delta$, these votes are included in every honest validator's view, finalizing ch_p at slot $t+2$. In particular, $\mathsf{ch}_p \preceq \mathcal{GF}(\mathcal{V}^{4\Delta(t+2)+2\Delta}).\mathsf{ch}$ and due to Line 24, $\mathsf{ch}_p \preceq \mathsf{chFin}_i^{4\Delta(t+2)+2\Delta}$. Due to Lemma 6, it follows that $txpool^{\mathsf{propose}(t)} \subseteq \mathsf{chFin}_i^{4\Delta(t+2)+2\Delta}$. As a consequence of Lemma 16, $\mathsf{chFin}_i^{4\Delta(t+2)+2\Delta} \preceq \mathsf{chFin}_i^{4\Delta(t+3)+2\Delta} \preceq \mathsf{chFin}_i^r$, for every $r \geq 4\Delta(t+3)+2\Delta$, and the result follows.
2. **Case 2: No justification for slot t is observed by v_{p_2} at propose time in slot $t+1$.** In this case, we apply Lemma 18 to slot $t+1$, which implies that the chain PROPOSEd by v_{p_2} at slot $t+1$ is justified in slot $t+2$ and finalized in slot $t+3$. Since this chain extends ch_p proposed by v_{p_1} (due to Lemma 12), ch_p is finalized as a prefix of the finalized chain in slot $t+3$. In particular, $\mathsf{ch}_p \preceq \mathcal{GF}(\mathcal{V}^{4\Delta(t+3)+2\Delta}).\mathsf{ch}$ and due to Line 24, $\mathsf{ch}_p \preceq \mathsf{chFin}_i^{4\Delta(t+3)+2\Delta}$. Due to Lemma 6, it follows that $txpool^{\mathsf{propose}(t)} \subseteq \mathsf{chFin}_i^{4\Delta(t+3)+2\Delta}$

Observe that if $\mathcal{GJ}_p^{\mathsf{propose}(t)}.c < t-1$, then we can immediately apply Lemma 18 finalizing the chain PROPOSEd by v_p^t in slot $t+2$.

Theorem 7. *Algorithm 5 is a secure ebb-and-flow protocol w.o.p..*

Proof. Safety of chAva and chFin, and Liveness of chAva follow from this, Theorems 4 and 5, and Lemma 15.

Then, we need to show that chFin is live after $\max(\mathsf{GST}, \mathsf{GAT}) + O(\Delta)$ with confirmation time $O(\kappa)$. Let t be any slot such that $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta$. There is a high probability of finding a slot $t_p \in [t, t+\kappa)$ such that both t_p and t_p+1 have honest proposers (Lemma 1). By Theorem 6, $txpool^{\mathsf{propose}(t)} \subseteq txpool^{\mathsf{propose}(t_p)} \subseteq \mathsf{chFin}_i^{\mathsf{fconf}(t_p+3)}$ for any validator honest in $\mathsf{fconf}(t_p+3)$ and, clearly, $\mathsf{fconf}(t_p+3) - \mathsf{propose}(t) \in O(\kappa)$.

Finally, we need to show that, for any round r and validator $v_i \in H_r$, $\mathsf{chFin}_i^r \preceq \mathsf{chAva}_i^r$. Line 13 clearly ensures that this is the case when r is a vote round. Note that the only other times at which either chAva or chFin are potentially modified are fast confirmation rounds. Then, let r be any fast confirmation round. Lines 22 to 23 ensure that $\mathcal{GJ}(\mathcal{V}_i^r).\mathsf{ch} \preceq \mathsf{chAva}_i^r$. By Algorithm 3, $\mathcal{GJ}(\mathcal{V}_i^r) \geq \mathcal{GF}(\mathcal{V}_i^r)$ which, due to Lemma 8 and the assumption that $f < \frac{n}{3}$, implies $\mathsf{chFin} \preceq \mathcal{GJ}(\mathcal{V}_i^r).\mathsf{ch} \preceq \mathsf{chAva}_i^r$.