3-Slot-Finality Protocol for Ethereum*

Francesco D'Amato, Ethereum Foundation
Roberto Saltini, Independent†
Thanh-Hai Tran, Independent†
Luca Zanolini, Ethereum Foundation

Abstract

Gasper, the consensus protocol currently employed by Ethereum, typically requires 64 to 95 slots – the units of time during which a new *chain* extending the previous one by one block is proposed and voted – to finalize. This means that under ideal conditions – where the network is synchronous, and all chain proposers, along with more than two-thirds of the validators, behave as dictated by the protocol – proposers construct blocks on a non-finalized chain that extends at least 64 blocks. This exposes a significant portion of the blockchain to potential reorganizations during changes in network conditions, such as periods of asynchrony. Specifically, this finalization delay heightens the network's exposure to Maximum Extractable Value (MEV) exploits, which could undermine the network's integrity. Furthermore, the extended finalization period forces users to balance the trade-off between economic security and transaction speed.

To address these issues and speed up finality, we introduce a partially synchronous finality gadget, which we combine with two dynamically available consensus protocols – synchronous protocols that ensure safety and liveness even with fluctuating validator participation levels. This integration results in secure ebb-and-flow protocols [SP 2021], achieving finality within three slots after a proposal and realizing *3-slot finality*.

1 Introduction and Related Work

Traditional Byzantine consensus protocols, such as PBFT [4] or HotStuff, are designed for distributed systems where participants are fixed, known in advance, and cannot go *offline* without being considered faulty.

Recently, dynamic participation has become a critical requirement for developing permissionless consensus protocols. This concept, initially formalized by Pass and Shi through their *sleepy model* [19], encapsulates the ability of a system to handle honest participants who may go offline and come back online. A consensus protocol that maintains safety and liveness while accommodating dynamic participation is called *dynamically-available*.

One problem of such dynamically-available protocols is that they do not tolerate network partitions [12]; no consensus protocols can satisfy both liveness (under dynamic participation) and safety (under network partitions). Simply put, a consensus protocol cannot produce a single chain[1] that concurrently offers Dynamic Availability and guarantees transaction finality in case of asynchronous periods or network partitions. Because of that, dynamically-available protocols studied so far are devised in synchronous settings [7, 13, 16, 10, 8].

Working around this impossibility result, Neu, Tas, and Tse [17] introduced a family of protocols referred to as *ebb-and-flow* protocols. An ebb-and-flow protocol comprises two sub-protocols, each with its own *confirmation rule*, and each outputting a chain, with one serving as a prefix of the other. The first confirmation rule defines what is known as the *available chain*, which provides liveness under dynamic participation (and synchrony). The second confirmation rule defines the *finalized chain*, and provides safety even under

*This work combines multiple preliminary publications which appeared at ESORICS-CBT 2023 [9], PODC 2024 [6], CSF 2024 [10], and *under submission* [8].

†Work done while at Consensys. Supported by Ethereum Foundation grant FY24-1433.

[1]In this context, we are extending the traditional notion of consensus, typically understood as a one-shot primitive. Technically, this should be referred to as total-order broadcast or atomic broadcast. However, for the sake of a general audience, we have adopted the term "consensus." Consequently, we consider the output of these protocols to be a sequence of transactions, batched in blocks, forming a chain. This will be formalized in Section 2.

network partitions, but loses liveness either under asynchrony or in case of fluctuation in the participation level. Interestingly, such family of protocols also captures the nature of the Ethereum consensus protocol, Gasper [3], in which the available chain is output by (the confirmation rule of) LMD-GHOST [22] protocol, and the finalized chain by the (confirmation rule of the) *finality gadget* Casper FFG [2]. However, the original version of LMD-GHOST is not secure even in a context of full participation and synchrony. Potential attack vectors have been identified [17, 20] that undermine the protocol's safety and liveness.

Motivated by finding a more secure alternative to LMD-GHOST, and following the ebb-and-flow approach, D'Amato *et al.* [7] devise a synchronous dynamically-available consensus protocol, Goldfish, that, combined with a generic (partially synchronous) finality gadget, implements a secure ebb-and-flow protocol. Moreover, Goldfish is *Reorg Resilient*: chains proposed by honest validators are guaranteed to not be reorganized. However, Goldfish is brittle to temporary asynchrony [10], in the sense that even a single violation of the bound of network delay can lead to a catastrophic failure, jeopardizing the safety of *any* previously confirmed chain, resulting in a protocol that is not practically viable to replace LMD-GHOST in Ethereum. In other words, Goldfish is not *Asynchrony Resilient*.

To cope with the limitation of Goldfish with asynchrony, D'Amato and Zanolini [10] propose RLMD-GHOST, a provably secure synchronous consensus protocol that does not lose safety during *bounded* periods of asynchrony and which tolerates a weaker form of dynamic participation, offering a trade-off between Dynamic Availability and Asynchrony Resilience. Their protocol results appealing for practical systems, where strict synchrony assumptions might not always hold, contrary to what is generally assumed with standard synchronous dynamically-available protocols.

The family of protocols to which Goldfish and RLMD-GHOST belong are consensus protocols that are *probabilistically* safe, guaranteeing safety with overwhelming probability. In contrast, Momose and Ren's research [16] laid the foundation for *deterministically* safe, dynamically-available consensus protocols, sparking a wave of subsequent research [13, 14, 11, 6]. Unlike Goldfish and RLMD-GHOST these protocols achieve deterministic safety by employing the notion of quorums. Traditional quorums, defined by a fixed number of actively engaging participants, are not suitable in a dynamic participation context. By leveraging *Graded Agreement*[2], Momose and Ren [16] redefined quorums dynamically, according to current participation levels, while maintaining critical properties of traditional quorums.

This development has led to the creation of various consensus protocols based on Graded Agreement, each with unique properties and varying levels of adversarial tolerance. The initial protocol by Momose and Ren [16] accommodates up to 1/2 adversarial participants but is limited by a high latency of 16Δ, with Δ being the message delay bound. The subsequent study by Malkhi, Momose, and Ren [13] introduced two protocols that reduce the latency to 3Δ and 2Δ, respectively, but at the expense of lower adversarial tolerance to 1/3 and 1/4. Later enhancements [14] managed to revert to tolerating minority corruption while maintaining a comparable latency of 4Δ. Another concurrent and independent work [11] achieves 1/2 adversarial resilience, with a latency of 6Δ.

D'Amato *et al.* [8], aiming to enhance the practicality of deterministically safe, dynamically-available consensus protocols, particularly in systems with many participants, introduced a consensus protocol, named TOB-SVD, that tolerates up to 1/2 adversarial participants and achieves latency comparable to [14] – slightly better in expectation and slightly worse in the best case. Crucially, it requires only a single vote round per decision in the best case, in contrast to the nine rounds required by [14]. Moreover, D'Amato, Losa, and Zanolini [6], explored mechanisms to make dynamically-available consensus protocols based on Graded Agreement resilient to *bounded* periods of asynchrony.

Previous work by D'Amato and Zanolini [9] proposed an ebb-and-flow protocol by combining the dynamically-available protocol RLMD-GHOST with a finality gadget similar to Casper FFG, hereafter referred to as the *SSF* protocol. In their work, as in ours, time is divided into *slots* and at the beginning of each slot a new chain is proposed by an elected proposer. Importantly, the SSF protocol ensures that, under synchrony and at least 2/3 of the participants being honest and online, any chain proposed by an honest participant is finalized within the same slot, *i.e.*, before the next proposer's turn. This single-slot finality is achieved through three vote rounds within a slot. Specifically, the SSF protocol operates in slots of duration 4Δ rounds each. During the first round, a proposal is made. In the second round, validators cast votes for

[2] In Graded Agreement, each decision is assigned a grade, which intuitively reflects the strength of the agreement. Notably, different formulations of graded agreement exist, each with slighly different properties. In this work, we focus on the properties defined by D'Amato *et al.* [8].

what they perceive as the tip of the chain, ideally the proposal just made. In the third round, if a quorum of votes for the same head block is observed, validators cast a *finality* vote for that block. In the final round, if a validator sees a quorum of finality votes for a block, it broadcasts an acknowledgment message for external observers. If an observer sees a quorum of acknowledgments for a block by the end of the slot, it can declare the block finalized.

While theoretically sound, the efficiency and usability of this protocol for large-scale blockchain networks are questionable due to the number of vote phases required for each slot. In fact, large-scale blockchain networks such as Ethereum, due to the large number of participants, to reduce the bandwidth requirements, employ an aggregation process by which votes are first sent to aggregators who then distribute the aggregated signatures. As a consequence of this, each vote phase requires double the normal network latency, increasing the slot time and decreasing the transaction throughput. This means that in practice, in the SSF protocol a slot is 6Δ as the second and third vote phase need to wait for the first and second ones to complete, respectively, but the third vote phase can proceed in parallel with the next slot. Moreover, it is unclear whether the messages cast during the third phase should be included on-chain or kept off-chain and potentially delaying the slot's finalization in practice.

In this work, we propose a finality gadget that can be composed with dynamically-available protocols to obtain an ebb-and-flow protocol with only one vote phase per slot and slot length of 5Δ if we consider a vote phase taking 2Δ. All things equals, this represents a 20% improvement in practical network throughout compared to the protocol by D'Amato and Zanolini [9][3].

The trade-off that we make is in delaying the finalization of a chain proposed by an honest validator to occur two slots later, *i.e.*, in our protocol a chain proposed by an honest validator in slot t is finalized[4] by the end of slot $t+2$, as long as synchrony holds, and at least $2/3$ of the participants are honest and active till the vote round of slot $t+2$. Importantly, this is ensured regardless of whether the proposers of slots $t+1$ and $t+2$ are honest. In practice, this means that, assuming that each vote phase takes 2Δ, we require the synchronous and participation assumptions to hold for 11Δ (in our protocol the vote round occurs Δ time after the beginning of a slot), rather than 5Δ as in the SSF protocol. However, like SSF, we offer a method by which, under these assumptions and considering that vote phases take 2Δ, a chain proposed by an honest proposer is *confirmed* by the dynamically-available protocol of any honest validator at time 3Δ of the same slot, which ensures that such chain will then be finalized in slot $t+2$. This offers an avenue for users to know in advance that, as long as the assumptions on synchrony and participation holds, such chain will be finalized.

Also, we can integrate the third round of voting from the SSF protocol into ours to obtain a protocol that has two vote phases per slot, but still retain slot length of 5Δ – when assuming that each vote phase takes 2Δ because, as explained above, the third vote phase from the SSF protocol can proceed in parallel with the rest of the protocol and therefore does not increase the practical slot length – but that reduces finalization from 11Δ down to 8Δ.

In practice, for networks where the periods of synchrony and at least $2/3$ of the participants being honest and online typically last much longer than 11Δ, our protocol presents no real drawback as confirmation still happens within the same slot which then leads to finalization, while attaining a higher transaction throughput and, arguably, a simpler protocol. Moreover, users are concerned with the time that a transaction takes to be either confirmed or finalized, which is the time that it takes for this transaction to be included in a proposed chain and for this chain to be either confirmed or finalized. This is not the same as the time taken to confirm or finalize a block proposed by an honest validator because transactions are not necessarily submitted just before an honest proposer proposes a chain. They are submitted whenever the user needs to interact with the blockchain. So, they could be submitted at any point of a slot. Also, they could be submitted when the proposer of the next slot is Byzantine who therefore might not include them in the chain that they proposer, if any. For this reason, from a user perspective, it makes sense to consider the expected confirmation and finalization times under the assumption that a transaction submission time is uniformly distributed. Then, the expected confirmation and finalization times correspond to the time taken to confirm

[3]This assuming the same block size for both the SSF protocol and our proposed protocol. In practice, throughput can be independent of slot duration, as desired per-second throughput can be set independently, with higher per-slot throughput achievable by using longer slots.

[4]Here, we consider the finalization time of a chain ch to be the time after which no chain conflicting with ch can ever by finalized, which can occur earlier than when some honest node finalizes chain ch in their view.

or finalize a chain proposed by an honest proposer, depending on which of the two measures we interested in, plus $\frac{(1+\beta)\cdot\text{slot-time}}{2(1-\beta)}$ where β represents the adversarial power in the network.

Let us compare the expected confirmation time of SSF and 3SF first. For $\beta = \frac{1}{3}$[5], the expected confirmation time for SSF is 9Δ whereas for 3SF is 8Δ meaning an $\approx 11\%$ improvement. For $\beta = 0$, the expected confirmation time for SSF is 6Δ whereas for 3SF is 5.5Δ meaning an $\approx 8\%$ improvement.

Moving to the expected finalization time, for $\beta = \frac{1}{3}$, for SSF it is 11Δ, for 3SF it is 16Δ and for the two-slot variant of 3SF it is 13Δ meaning that the expected finalization time for 3SF $\approx 46\%$ higher than the one of SSF, but this reduces to $\approx 18\%$ for the its two-slot variant. For $\beta = 0$, the expected finalization time for SSF is 8Δ, for 3SF is 13.5Δ and for the two-slot variant of 3SF is 10.5Δ meaning that the expected finalization time for 3SF $\approx 69\%$ higher than the one of SSF, but this reduces to $\approx 31\%$ for the its two-slot variant.

Finally, slot time has been shown[15] to be an important parameter in determining the *economic leakage* of on-chain automated market makers (AMMs) due to arbitrage. For instance, arbitrage profits (and equivalently liquidity providers (LP) losses) are proportional to the square root of slot time, so that a lower slot time is very desirable by financial applications built on top of a smart contract blockchain.

Overall, our protocol achieves a balance by trading a higher expected finalization time for a shorter expected confirmation time, which could be sufficient for most users. At the same time, it offers shorter slot time and improved throughput as discussed above.

Additionally, we show how to integrate our finality gadgetwith two dynamically-available protocols to obtain a secure ebb-and-flow. The first dynamically-available protocol that we consider is a probabilistically-safe and bounded-asynchrony-period-resilient variant of the deterministically-safe protocol TOB-SVD [8], the second is RLMD-GHOST [10]. The resulting protocols, in addition to the standard ebb-and-flow properties, ensure Safety and Reorg Resilience of the available chain even in the face of a subsequent bounded period of asynchrony. This makes our protocols particularly attractive for blockchain networks, such as Ethereum, where safety failures in the available chain can be exploited by dishonest participants to steal honest participant's Maximum Extractable Value (MEV) [5] without being punished for it. Moreover, critical to the practical application in large-scale blockchain networks, both of our resulting protocols ensure that the dynamically-available component can *heal* from any arbitrarily long period of asynchrony as long as at least 2/3 of participants are honest and online for a sufficient amount of time. This is not a property mentioned in the original work that introduced ebb-and-flow protocol [18] where the dynamically-available protocol is required to ensure safety only if synchrony holds from the beginning. However, any real system is bound to experience some period of asynchrony of arbitrary length at some point. Ensuring that the dynamically-available protocol can recover is then important from a practical point of view.

The remainder of this work is structured as follows. In Section 2, we present our system model along with all the necessary background notions and property definitions. Common notions for both the protocols we introduce in this work are detailed in Section 3. Section 4 introduces and proves the correctness of the finality gadget. Notably, the finality gadget, or *FFG-component*, is common to both presented protocols. Therefore, the properties and results discussed in Section 4 apply to both protocols. Our first faster finality protocol is presented in Section 5, where we provide the pseudo-code of the first protocol, the one based upon TOB-SVD [8], and prove its properties, demonstrating that it is a secure ebb-and-flow protocol (as defined in Section 2). The second protocol, the one based upon RLMD-GHOST [10], is introduced in Section 6, and we conduct a similar analysis to that in Section 5. In Section 7, we discuss how to resolve some of the main challenges presented in implementing either protocol, and examine their communication complexity. Then, in Section 8, we detail the conditions required for Dynamic Availability and Reorg Resilience to hold even in partially synchronous settings and provide the intuition underpinning this leaving, the detailed proof to Appendix A. Also, in Appendix B we explore a modification to our protocols that allow finalizing chains within two slots only by adding one vote round but without this affecting the practical length of slots when aggregation is used to reduce bandwidth requirements. Finally, conclusions are drawn in Section 9.

[5]Liveness can only be guaranteed if $\beta < \frac{1}{3}$

2 Model and Preliminary Notions

2.1 System model

Validators. We consider a set of n *validators* $v_1, \ldots, v_n$ that communicate with each other by exchanging messages. Every validator is identified by a unique cryptographic identity and the public keys are common knowledge. Validators are assigned a protocol to follow, consisting of a collection of programs with instructions for all validators. We assume the existence of a probabilistic poly-time adversary $\mathcal{A}$ that can choose up to f validators to corrupt over an entire protocol execution. Any validator is *honest* until, if ever, it is corrupted by the adversary at which point it stays corrupted for the remaining duration of the protocol execution, and is thereafter called *adversarial*. The adversary $\mathcal{A}$ knows the the internal state of adversarial validators. The adversary is *adaptive*: it chooses the corruption schedule dynamically, during the protocol execution. Honest validators faithfully follow the assigned protocol while adversarial validators may deviate from it arbitrarily. Each validator has a *stake*, which we assume to be the same for every validator. If a validator v_i misbehaves and a proof of this misbehavior is given, it loses a part of its stake (v_i gets *slashed*).

Time. Time is divided into discrete *rounds*. We define the notion of *slot* as a collection of 4Δ rounds. Given round r we define slot(r) as the slot of round r, *i.e.*, slot$(r) := \lfloor \frac{r}{4\Delta} \rfloor$.

View. A *view* (at a given round r), denoted by $\mathcal{V}^r$, is a subset of all the messages that a validator has received until r. The notion of view is *local* for the validators. For this reason, when we want to focus the attention on a specific view of a validator v_i, we denote with $\mathcal{V}_i^r$ the view of v_i (at round r).

Links. We assume that a best-effort gossip primitive that will reach all validators is available. In a protocol, this primitive is accessed through the events "sending a message through gossip" and "receiving a gossiped message." We also use the term *casts* to describe when a validator v_i sends a message through gossip. Moreover, we assume that messages from honest validator to honest validator are eventually received and cannot be forged. This includes messages sent by adversarial validators, once they have been received by some honest validator v_i and gossiped around by v_i.

Network Model. We consider a partially synchronous model in which validators have synchronized clocks but there is no a priori bound on message delays. However, there exists a time (not known by the validators), called *global stabilization time* (GST), after which message delays are bounded by Δ rounds with Δ known to validators.

We also allow for a *single short asynchronous period* after GST, starting at slot $t_a + 1$ and extending for at most π slots, *i.e.*, including at most rounds $[4\Delta(t_a + 1), 4\Delta(t_a + \pi + 1))$. The value of t_a is unbeknownst to the validators, but we assume that the validators know the value of π.

Sleepiness. The adversary $\mathcal{A}$ can decide for each round r which honest validator is *awake* or *asleep* at round r. Asleep validators do not execute the protocol and messages received in round r are queued and delivered in the first round in which the validator is awake again. Honest validators that become awake at round r, before starting to participate in the protocol, must first execute (and terminate) a *joining protocol* (see Section 3), after which they become *active* [7]. All adversarial validators are always awake, and are not prescribed to follow any protocol. Therefore, we always use active, awake, and asleep to refer to honest validators. As for corruptions, the adversary is adaptive also for sleepiness, *i.e.*, the sleepiness schedule is also chosen dynamically by the adversary. Note that awake and active validators coincide in the sleepy model [19]. Finally, there is a time (not known by the validators), called *global awake time* (GAT), after which all validators are always awake.

Transaction 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 a pair of elements, denoted as $B = (b, p)$. Here, b represents the *block body* – essentially, the main content of the block which contains a batch of transactions grouped together[6]. We assume that a block body can contain an unbounded number of transactions. For the sake of simplicity, we are not including details about the block's parent in this definition. In practice, each block body contains a reference pointing to another block. The second element of the pair, $p \geq 0$, indicates the *slot* where the block B is proposed. By definition, if B_p is the parent of B, then $B_p.p < B.p$. We denote with $B_{\text{genesis}} = (b_{-1}, -1)$ the *genesis block*, which is the only block that does not have a parent and has a negative slot. Given the definition above, each different block B implicitly identifies a different finite *chain* of blocks starting from block B, down to the genesis block, by recursively moving from a block to its parent. Hence, we make no real distinction between a block and the chain that it identifies. So, by chain ch, we mean the chain identified by the block ch. Let us consider two chains, ch_1 and ch_2. We write $\mathsf{ch}_1 \prec \mathsf{ch}_2$ if and only if ch_1 is a strict ancestor of ch_2. In this case, we also say that ch_1 is a strict prefix of ch_2 or, conversely, that ch_2 is a strict extension of ch_1. We say that ch_1 *conflicts* with ch_2 if and only if neither $\mathsf{ch}_1 \preceq \mathsf{ch}_2$ nor $\mathsf{ch}_2 \preceq \mathsf{ch}_1$ holds. Given a chain ch, we define the κ-deep prefix of ch at slot t (denoted by $\mathsf{ch}^{\lceil \kappa, t}$) as the longest prefix ch' of ch such that $\mathsf{ch}'.p \leq t - \kappa$. When slot t is clear from the context, we just write $\mathsf{ch}^{\lceil \kappa}$ to mean $\mathsf{ch}^{\lceil \kappa, t}$. We also define the following total pre-order between chains. We write $\mathsf{ch}_1 \leq \mathsf{ch}_2$ if and only if $\mathsf{ch}_1.p \leq \mathsf{ch}_2.p$. However, when we write $\mathsf{ch}_1 = \mathsf{ch}_2$, we mean that the two chains are the same, not that just $\mathsf{ch}_1.p = \mathsf{ch}_2.p$. Also, we sometimes also call $\mathsf{ch}.p$ the *height* or *length* of chain ch. Finally, we let $tx \in \mathsf{ch}$ to mean that transaction tx is included in a block of chain ch, *i.e.*, $\exists \mathsf{ch}' \preceq \mathsf{ch}$, $tx \in \mathsf{ch}'.b$. Naturally, if TX is a set of transactions, then we write $TX \subseteq \mathsf{ch}$ to mean $\forall tx \in TX, tx \in \mathsf{ch}$.

Blockchain Protocol. In this work, we only consider protocols that output one or more chains. If ch is the chain output by a protocol Π, then we use the notation ch_i^r to indicate the chain output by validator v_i at round r when executing protocol Π (we do not include Π in the notation as that will always be clear from the context). If v_i is asleep in round r and $r_a < r$ is the highest round during which v_i was awake, then we assume that $\mathsf{ch}_i^r = \mathsf{ch}_i^{r_a}$.

Proposer election mechanism. In each slot t, a validator v_p is selected to be a *proposer* for t, *i.e.*, to extend the chain with new blocks. Observe that, when we want to highlight the fact that v_p is a proposer for a specific slot t, we use the notation v_p^t. Otherwise, when it is clear from the context, we just drop the slot t, to make the notation simpler. As the specification of a proposal selection mechanism is not within the goals of this work, we assume the existence of a proposer selection mechanism satisfying the requirements of *uniqueness, unpredictability, and fairness*: v_p is unique, the identity of v_p is only known to other validators once v_p reveals itself, and any validator has probability $\frac{1}{n}$ of being elected to be a proposer at any slot.

Proposing. In this work, we only consider protocols that define the action of a validator *proposing* a chain. The specifics of this actions are protocol dependant. This generalizes the common notion of proposing a block. For any of the two protocols presented in this work, proposing happens in the first round of a slot. Hence, for readability purpose, we often use $\mathsf{propose}(t) := 4\Delta t$.

Voting. Either of the two protocols presented in this work include a vote round where validators vote on the proposed chain. Again for sake of readability, we often use $\mathsf{vote}(t) := 4\Delta t + \Delta$.

2.2 Adversary resiliency

Let H_r, A_r, and $H_{r,r'}$ be 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.*, $H_{r,r'} = \bigcup_{i=r}^{r'} H_i$ (if $i < 0$ then $H_i := \emptyset$), respectively. We let $A_\infty = \lim_{t \to \infty} A_{\mathsf{vote}(t)}$. Note that $f = |A_\infty|$. Also, we say that validator v_i is *always-honest* if and only if $v_i \notin A_\infty$, *i.e.*, it is never corrupted.

Except when explicitly mentioned, throughout this work we assume $f < \frac{n}{3}$. Each protocol presented in this work extends this assumption in slightly different ways.

[6]We often use the dot notation to refer to the elements of a tuple. For instance, $B.b$ represents the block body of B.

2.2.1 Assumptions for the protocol of Section 5

For the protocol in Section 5 we assume that for every slot t after GST:

$$|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}| > |A_{\mathsf{vote}(t+1)} \cup \left(H_{\mathsf{vote}(t-\eta+1),\mathsf{vote}(t-1)} \setminus H_{\mathsf{vote}(t)}\right)| \tag{1}$$

where $\eta := \begin{cases} 1, & \text{if } \pi = 0 \\ \pi + 2, & \text{otherwise} \end{cases}$.

In other words, we require the number of active validators at round $\mathsf{vote}(t-1)$ which are not corrupted by round $\mathsf{vote}(t)$ to be greater than the number of adversarial validators at round $\mathsf{vote}(t)$, together with the number of validators that were active at some point between rounds $\mathsf{vote}(t-\eta)$ and $\mathsf{vote}(t-2)$, but not at round $\mathsf{vote}(t-1)$, with $\eta = \pi + 2$ except if $\pi = 0$ (*i.e.*, we admit no asynchronous period after GST), in which case $\eta = 1$ and Constraint (1) simplifies to $|H_{\mathsf{vote}(t-1)} \setminus A_{\mathsf{vote}(t)}| > |A_{\mathsf{vote}(t)}|$. Intuitively, η corresponds to *the expiration period* applied to some messages that are discarded by the protocol if they were sent more than η slots ago. This will become clearer later in Section 5.

We also assume that all of the following constraints hold:

$$\forall t' \in \{t_a+1, \ldots, t_a+\pi+2\},\ |H_{\mathsf{vote}(t_a)} \setminus A_{\mathsf{vote}(t)}| > |A_{\mathsf{vote}(t')} \cup \left(H_{\mathsf{vote}(t'-\eta),\mathsf{vote}(t'-1)} \setminus H_{\mathsf{vote}(t_a)}\right)| \tag{2}$$

$$H_{\mathsf{vote}(t_a)} \setminus A_{\mathsf{vote}(t_a+1)} \subseteq H_{\mathsf{vote}(t_a)+\Delta} \tag{3}$$

$$|(H_{\mathsf{vote}(t_a+1,t_a+\pi+1)} \setminus H_{\mathsf{vote}(t_a)}) \cup A_\infty| < \frac{2}{3} \tag{4}$$

The three constraints above only deal with the short asynchronous period π. Constraint (2) limits how many honest validators that were not active in round $\mathsf{vote}(t_a)$ may awake during the short asynchronous period or the two following slots. This is because the adversary can take advantage of the short asynchronous period to manipulate them to send messages that would jeopardize the safety of the protocol. Constraint (3) just requires that all the validators active in round $\mathsf{vote}(t_a)$ and not corrupted by round $\mathsf{vote}(t_a+1)$ are also active Δ rounds after $\mathsf{vote}(t_a)$. Constraint (4) is required to prevent the adversary from leveraging the messages not subject to expiration to affect the safety of the protocol. We will provide a more detailed explanation of such a scenario in due course in Section 5.4.1.

2.2.2 Assumptions for the Protocol of Section 6

For the protocol in Section 6 we instead require that the following condition holds for any slot t after GST, in addition to Constraints (2) to (4) defined above:

$$|H_{\mathsf{vote}(t)}| > |A_{\mathsf{vote}(t+1)} \cup (H_{\mathsf{vote}(t-\eta+1),\mathsf{vote}(t-1)} \setminus H_{\mathsf{vote}(t)})| \tag{5}$$

Observe that Constraint (5) is less restrictive than Constraint (1), as it just imposes limits on $|H_{\mathsf{vote}(t-1)}|$ rather than on $|H_{\mathsf{vote}(t-1)} \setminus A_{\mathsf{vote}(t)}|$. The reason for this diminished restrictiveness will become clear when we present our protocol in Section 6.

We refer to the adversary model just described as the *generalized partially synchronous η-sleepy model* (or w.l.o.g., when the context is clear, as the *η-sleepy model* for short). Finally, we say that an execution in the generalized partially synchronous sleepy model is *η-compliant* if and only if it satisfies η-sleepiness, *i.e.*, if it satisfies either Constraints (1) to (4) or Constraints (2) to (5), depending on the protocol.

2.3 Security

Security Parameters. In this work we treat λ and κ[7] as the security parameters related to the cryptographic components utilized by the protocol and the protocol's own security parameter, respectively. We also account for a finite time horizon, represented as T_{conf}, which is polynomial in relation to κ. An event is said to occur with *overwhelming probability* if it happens except with probability which is $\text{negl}(\kappa) + \text{negl}(\lambda)$. The

[7] In this work, the value κ represents the number of slots that are required in order to be certain, except for a negligible probability, that at least one of the proposers in these slots is honest.

properties of cryptographic primitives hold true with a probability of $\text{negl}(\lambda)$, signifying an overwhelming probability, although we will not explicitly mention this in the subsequent sections of this work. We also assume that $\kappa > 1$.

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 $\mathsf{Ch}_j^{r'} \preceq \mathsf{Ch}_i^r$.

A protocol satisfies η *Safety* after time T_{sec} if it is safe after time T_{sec} in any η-compliant execution. We say that a protocol always satisfies η Safety if it satisfies η Safety after time 0. 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, $tx \in \mathsf{Ch}_i^{r_i}$.

A protocol satisfies η *Liveness* after time T_{sec} with confirmation time T_{conf} if it is live after time T_{sec} in any η-compliant execution and has confirmation time T_{conf}. We say that a protocol always satisfies η Liveness with confirmation time T_{conf} if it satisfies η Safety after time 0 and has confirmation time T_{conf}. 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 [7])**.** 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 (always) satisfies η Security with confirmation time T_{conf} if it (always) satisfies both η Safety and η Liveness with confirmation time T_{conf}. A protocol is always secure if it always both safe and live

We now recall the definitions of *Dynamic Availability* and *Reorg Resilience* from [10].

Definition 4 (Dynamic Availability)**.** We say that a protocol is *η-dynamically-available* after time T_{dyn} if and only if it satisfies η Security after time T_{dyn} with confirmation time $T_{\mathsf{conf}} = O(\kappa)$.

We simply say that a protocol is *η-dynamically-available* if and only if it always satisfies η Security when $\mathsf{GST} = 0$.

Moreover, we say that a protocol is *dynamically-available* if it is 1-dynamically-available, as this corresponds to the usual notion of Dynamic Availability.

Note that we define the concept of Dynamic Availability with the underlying assumption of a certain level of adversarial resilience. In the first protocol, as discussed in Section 5, we attain Dynamic Availability provided that Constraint (1) is satisfied. Following this, in Section 6, Dynamic Availability is achieved on the satisfaction of Constraint (5).

Definition 5 (Reorg Resilience)**.** We say that a protocol is *reorg-resilient* after slot t_{reorg} and time T_{reorg}[8] in executions $\mathcal{E}$ if and only if, for any execution in $\mathcal{E}$, round $r \geq T_{\mathsf{reorg}}$ and validator v_i honest in r, any chain proposed in any slot $t \geq t_{\mathsf{reorg}}$ by a validator honest in round $\mathsf{propose}(t)$ does not conflict with Ch_i^r.

We simply say that a protocol is *reorg-resilient* if and only if it is reorg-resilient after slot 0 and time 0.

A protocol is *η-reorg-resilient* if it is reorg-resilient in any η-compliant execution.

Together with Dynamic Availability, we want our protocol to be *accountably safe.*

Definition 6 (Accountable Safety)**.** We say that a protocol has *Accountable Safety* with resilience $f^{\text{acc}} > 0$, or that it is f^{acc}-accountable, if, upon a safety violation, by having access to all messages sent, it is possible to identify at least f responsible participants. In particular, by having access to all messages sent, it is possible to collect evidence from sufficiently many honest participants and generate a cryptographic proof that identifies f^{acc} adversarial participants as protocol violators. Such proof cannot falsely accuse any honest participant that followed the protocol correctly.

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

As a consequence of the CAP theorem [12], no consensus protocols can satisfy both liveness (under dynamic participation) and safety (under temporary network partitions). Simply put, a consensus protocol (for state-machine replication) cannot produce a single chain that concurrently offers Dynamic Availability and guarantees transaction finality in case of asynchronous periods or network partitions. To overcome this impossibility result, Neu, Tas, and Tse [17] introduce a family of protocols, referred to as *ebb-and-flow* protocols. Neu *et al.* [17] propose a protocol that outputs two chains, one that provides liveness under dynamic participation (and synchrony), and one that provides Accountable Safety even under network partitions. This protocol is called *ebb-and-flow* protocol. We present a generalization of it, in the η-sleepy model.

Definition 7 (η-secure ebb-and-flow protocol)**.** A η-secure *ebb-and-flow protocol* outputs an available chain chAva that is η-dynamically-available[9], and a finalized (and accountable) chain chFin that is always safe and is live after $\max(\mathsf{GST}, \mathsf{GAT})+O(\Delta)$ with $T_{\mathsf{conf}} = O(\kappa)$[10], therefore is secure after $\max(\mathsf{GST}, \mathsf{GAT})+O(\Delta)$ with $T_{\mathsf{conf}} = O(\kappa)$. Moreover, for each honest validator v_i and for every round r, chFin_i^r is a prefix of chAva_i^r.

Both our faster finality protocols adopt the ebb-and-flow methodology [18]: the available chain chAva is output by the η-dynamically-available protocol (or *component*), while the finalized chain chFin, is output by a PBFT-style protocol whose behavior is similar to that of FFG Casper [2].

Asynchrony Resilience. We define the set W_r of to be the set of active honest validator in round r, but restricted to also be in $H_{\mathsf{vote}(t_a)}$ if $\text{slot}(r)$ in $[t_a, t_a + \pi + 1]$, *i.e.*,

$$W_r := \begin{cases} H_r \cap H_{\mathsf{vote}(t_a)}, & \text{if slot}(r) \text{ in } [t_a, t_a + \pi + 1] \\ H_r, & \text{otherwise.} \end{cases}$$

Informally, we say that any validator in such a set is an *aware* validator. Then, we define η Asynchrony Reorg Resilience and η Asynchrony Safety Resilience as follows [10].

Definition 8 (η Asynchrony Reorg Resilience)**.** If $\pi > 0$, we say that a protocol is *η-asynchrony-reorg-resilient* after slot t_{reorg} and time T_{reorg} if and only if the following holds for any η-compliant execution. For any slot $r_i \geq T_{\mathsf{reorg}}$ and validator $v_i \in W_{r_i}$, a chain proposed in a slot $t \in [t_{\mathsf{reorg}}, t_a]$ by a validator honest in round $\mathsf{propose}(t)$ does not conflict with Ch_i^r.

We simply say that a protocol is *η-asynchrony-reorg-resilient* if and only if it is η-asynchrony-reorg-resilient after slot 0 and time 0.

Definition 9 (η Asynchrony Safety Resilience)**.** If $\pi > 0$, we say that a protocol is *η-asynchrony-safety-resilient* after time T_{sec} if and only if the following holds for any η-compliant execution. Let r_i be any round such that $r_i \in [T_{\mathsf{sec}}, 4\Delta t_a + \Delta]$ and v_i any validator honest in r_i. For any slot r_j and validator v_j in W_{r_j}, $\mathsf{Ch}_j^{r_j}$ does not conflict with $\mathsf{Ch}_i^{r_i}$.

We simply say that a protocol is *η-asynchrony-safety-resilient* if and only if it is η-asynchrony-safety-resilient after time 0.

3 Common Notions

Checkpoints. A *checkpoint* is a kind of tuple, represented as $\mathcal{C} = (\mathsf{ch}, c)$. In this tuple, ch is a chain, c signifies the slot where ch is proposed for justification (this concept is introduced and explained below). Observe that, as a consequence of our protocol design, the slot c for the checkpoint will always occur after the slot $\mathsf{ch}.p$ where the chain was proposed, i.e., $c \geq \mathsf{ch}.p$. We refer to c as the *checkpoint slot* of $\mathcal{C}$.

[9]When we say that a chain produced by a protocol Π satisfies property P (or *the chain is* P), we mean that the protocol Π itself satisfies property P (or *the protocol is* P).

[10]The reason for this is that we need κ slots to find a slot with an honest proposal, after which the proposal will be finalized in 12Δ rounds.

Votes. Validators cast two main types of votes: FFG-VOTEs and VOTEs. Each VOTE include an FFG-VOTE. Specifically, an FFG-VOTE is represented as a tuple $[\text{FFG-VOTE}, \mathcal{C}_1, \mathcal{C}_2, v_i]$, where v_i is the validator sending the FFG-VOTE, while $\mathcal{C}_1$ and $\mathcal{C}_2$ are checkpoints. These checkpoints are respectively referred to as the *source* ($\mathcal{C}_1$) and the *target* ($\mathcal{C}_2$) of the FFG-VOTE. Such an FFG-VOTE is *valid* (*i.e.*, valid$(\mathcal{C}_1 \rightarrow \mathcal{C}_2)$) if and only if $\mathcal{C}_1.c < \mathcal{C}_2.c$ and $\mathcal{C}_1.\mathsf{ch} \preceq \mathcal{C}_2.\mathsf{ch}$. FFG-VOTEs effectively act as *links* connecting the source and target checkpoints. We sometimes denote the whole FFG-VOTE simply as $\mathcal{C}_1 \rightarrow \mathcal{C}_2$. For instance, we might say that a validator v_i casts a $\mathcal{C}_1 \rightarrow \mathcal{C}_2$ vote. On the other hand, a VOTE cast by a validator v_i is a tuple $[\text{VOTE}, \mathsf{ch}, \mathcal{C}_1 \rightarrow \mathcal{C}_2, t, v_i]$, where ch represents a chain, $\mathcal{C}_1 \rightarrow \mathcal{C}_2$ encapsulates the associated FFG-VOTE $[\text{FFG-VOTE}, \mathcal{C}_1, \mathcal{C}_2, v_i]$, and t is the slot in which such vote is cast. In this case, we might say that v_i casts a VOTE message for chain ch. We say that a validator v_i *equivocates* if and only if it sends two VOTE messages $[\text{VOTE}, \mathsf{ch}, \cdot, t, v_i]$ and $[\text{VOTE}, \mathsf{ch}', \cdot, t, v_i]$ with $\mathsf{ch} \neq \mathsf{ch}'$, *i.e.*, it casts two VOTE messages for the same slot but different chains.

Proposals. This work contains protocols using two types of PROPOSAL messages. For the first protocol (Section 5), a proposal is a tuple $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \mathcal{C}, t, v_k]$ where ch_p is the PROPOSEd chain, Q^C is a *quorum certificate* for the *fast confirmed* ch^C, $\mathcal{C}$ is a checkpoint, and $t = \mathsf{ch}_p.p$[11] is the slot in which this proposal is cast. In this case, we say that validator v_k PROPOSEs chain ch_p in slot t. The notion of fast confirmed chain will be defined in Section 5.1.

Subsequently, in Section 6, a proposal is a tuple [PROPOSE, ch_p, $\mathcal{V}$, t,v_k] where ch_p is a chain (as above), $\mathcal{V}$ a view, and $t = B.p$. We refer to $\mathcal{V}$ as the *proposed view.*

Gossip behavior. Votes and blocks are gossiped at any time, regardless of whether they are received directly or as part of another message. For example, a validator receiving a vote also gossips the block that it contains, and a validator receiving a proposal also gossips the blocks and votes contained in the proposed view. Finally, a proposal from slot t is gossiped only during the first Δ rounds of slot t.

Joining protocol. Honest validators that become awake at round r, before starting to participate in the protocol, must first execute (and terminate) a *joining protocol*, after which they become *active*. Given slot t, when an honest validator v_i wakes up at some round $r \in (\mathsf{vote}(t-2)+\Delta, \mathsf{vote}(t-1)+\Delta]$, as per our system model, all the messages that it received while being asleep are immediately delivered to it. Then, validator v_i executes the protocol, but without sending any message, up until round $\mathsf{vote}(t)$ at which point it becomes active, until either corrupted or put to sleep by the adversary. This also implies that if a validator v_i is elected to be the leader in a given slot, but by the propose time of that slot it is not active yet, then v_i will not send any PROPOSE message in that slot.

Fork-choice function. A *fork-choice function* is a deterministic function, denoted as FC. This function, when given a set of views[12], a chain, and a slot t as inputs, produces a chain ch. In this work we will focus our attention on two types of fork-choice functions. For the first protocol (Section 5), we consider a *majority* fork-choice function, i.e., the outputted chain is the highest chain supported by a majority of the voting weight (Algorithm 2). Subsequently, in Section 6, we consider a fork-choice function based on GHOST [21] (Algorithm 5).

Confirmation rule. A confirmation rule allows validators to identify a *confirmed prefix* of the chain outputted by the fork-choice function, for which safety properties hold, and which is therefore used to define the output of the protocol. Since the protocol we are going to present outputs two chains, the available chain chAva and the finalized chain chFin, output by the finality component, we have two confirmation rules. One is *finality*, which we introduced in Section 4, and defines chFin. The other confirmation rule, defining chAva, is itself essentially split in two rules, a *slow κ-deep* confirmation rule, which is live also under dynamic participation, and a *fast (optimistic) confirmation rule*, requiring $\frac{2}{3}n$ honest validators to be awake, *i.e.*, a

[11]Note that in this context, the parameter t is redundant in the PROPOSE message since it is already incorporated within ch_p. However, for the sake of clarity, we have chosen to include it explicitly in the PROPOSE message.

[12]The number of views used as input varies depending on the protocol. This distinction will become evident when discussing the two different fork-choice functions in the following sections.

Algorithm 1 Justification and Finalization

```
1: function valid($\mathcal{C}_1 \to \mathcal{C}_2$)
2:     return
3:         $\land\ \mathcal{C}_1.\mathsf{ch} \preceq \mathcal{C}_2.\mathsf{ch}$
4:         $\land\ \mathcal{C}_1.c < \mathcal{C}_2.c$

5: function $\mathsf{J}(\mathcal{C}, \mathcal{V})$
6:     return
7:         $\lor\ \mathcal{C} = (B_{\text{genesis}}, 0)$
8:         $\lor\ \exists \mathcal{M} \subseteq \mathcal{V} : \land\ |\{v_k : [\text{VOTE}, \cdot, \cdot, \cdot, v_k] \in \mathcal{M}\}| \geq \frac{2}{3}n$
9:                   $\land\ \forall\, [\text{VOTE}, \cdot, \mathcal{S} \to \mathcal{T}, \cdot, \cdot] \in \mathcal{M} : \land\ \text{valid}(\mathcal{S} \to \mathcal{T})$
10:                                                    $\land\ \mathsf{J}(\mathcal{S}, \mathcal{V})$
11:                                                    $\land\ \mathcal{S}.\mathsf{ch} \preceq \mathcal{C}.\mathsf{ch} \preceq \mathcal{T}.\mathsf{ch}$
12:                                                    $\land\ \mathcal{T}.c = \mathcal{C}.c$

13: function $\mathsf{F}(\mathcal{C}, \mathcal{V})$
14:     return
15:         $\lor\ \mathcal{C} = (B_{\text{genesis}}, 0)$
16:         $\lor\ \land\ \mathsf{J}(\mathcal{C}, \mathcal{V})$
17:           $\land\ \exists \mathcal{M} \subseteq \mathcal{V} : \land\ |\{v_k : [\text{VOTE}, \cdot, \cdot, \cdot, v_k] \in \mathcal{M}\}| \geq \frac{2}{3}n$
18:                   $\land\ \forall\, [\text{VOTE}, \cdot, \mathcal{C} \to \mathcal{T}, \cdot, \cdot] \in \mathcal{M} : \land\ \text{valid}(\mathcal{C} \to \mathcal{T})$
19:                                                    $\land\ \mathcal{T}.c = \mathcal{C}.c + 1$
```

stronger assumption than just $\eta-$compliance. chAva is updated to the chain confirmed by either one, so that liveness of chAva only necessitates liveness of one of the two rules. In particular, η-compliance is sufficient for liveness. On the other end, safety of chAva requires both rules to be safe.

4 FFG component

In Section 3, we described two kind of votes that a validator v_i casts in the protocol. In particular, the FFG-VOTE, encapsulated within the generic VOTE, is used by the *FFG-component* of our protocol[13]. The FFG component of our protocol aims at finalizing, in each slot, a chain that extends the one finalized in the previous slot.

Justification. We say that a set of FFG-VOTEs is a *supermajority set* if it contains valid FFG-VOTEs from at least $\frac{2}{3}n$ distinct validators. A checkpoint $\mathcal{C}$ is considered *justified* if it either corresponds to the genesis block, i.e., $\mathcal{C} = (B_{\text{genesis}}, 0)$, or if there exists a supermajority set of links $\{\mathcal{C}_i \to \mathcal{C}_j\}$ satisfying the following conditions. First, for each link $\mathcal{C}_i \to \mathcal{C}_j$ in the set, $\mathcal{C}_i \to \mathcal{C}_j$ is valid and $\mathcal{C}_i.\mathsf{ch} \preceq \mathcal{C}.\mathsf{ch} \preceq \mathcal{C}_j.\mathsf{ch}$. Moreover, all source checkpoints $\mathcal{C}_i$ in these links need to be already justified, and the checkpoint slot of $\mathcal{C}_j$ needs to be the same as that of $\mathcal{C}$ ($\mathcal{C}_j.c = \mathcal{C}.c$), for every j. It is important to note that the source and target chain may vary across different votes. This justification rule is formalized by the binary function $\mathsf{J}(\mathcal{V}, \mathcal{C})$ in Algorithm 1 which outputs *true* if and only if checkpoint $\mathcal{C}$ is justified according to the set of messages $\mathcal{V}$. Lastly, we say that a chain ch *is justified* if and only if there exists a justified checkpoint $\mathcal{C}$ for which $\mathcal{C}.\mathsf{ch} = \mathsf{ch}$.

Ordering among checkpoints. We define a total pre-order among checkpoints $\mathcal{C}$ and $\mathcal{C}'$ by lexicographic ordering on checkpoint and proposal slots. Specifically, $\mathcal{C} \leq \mathcal{C}'$ if and only if either $\mathcal{C}.c < \mathcal{C}'.c$ or, in the case where $\mathcal{C}.c = \mathcal{C}'.c$, then $\mathcal{C}.\mathsf{ch}.p \leq \mathcal{C}'.\mathsf{ch}.p$. However, when we write $\mathcal{C} = \mathcal{C}'$ we mean that the two checkpoints are the same, not just that $\mathcal{C}.c = \mathcal{C}'.c \land \mathcal{C}.\mathsf{ch}.p = \mathcal{C}'.\mathsf{ch}.p$.

[13]The component of our protocol that outputs chFin is almost identical to Casper [2], the *friendly* finality gadget (FFG) adopted by the Ethereum consensus protocol Gasper [3]. This is the reason why we decided to use the *FFG* terminology already accepted within the Ethereum ecosystem.

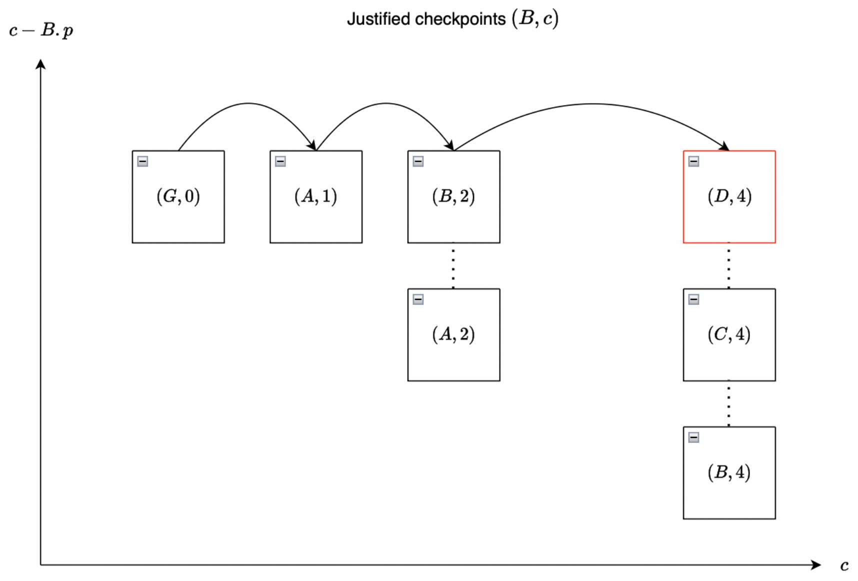


Figure 1: Example of justification. The figure illustrates the progression of justified checkpoints over four slots. The checkpoints are presented from left to right, indicating an increase in the checkpoint slot c. Within each checkpoint slot, the proposal slot p decreases from top to bottom. This visualization is based on the assumption that all justifications occur through votes with consistent source and target, as shown by the arrows. Chains labeled A, B, C, and D are PROPOSEd in slots $0, 1, 2$, and 3, respectively, with each chain extending the previous one. Specifically, at slot 2, votes originating from the source checkpoint $(A, 1)$ and targeting checkpoint $(B, 2)$ result in the justification of both checkpoints $(B, 2)$ and $(A, 2)$. Slot 3 does not see any checkpoint justification. However, in slot 4, checkpoints containing chains B, C, and D are justified. This justification is achieved through votes with the source $(B, 2)$ and the target $(D, 4)$.

Greatest justified checkpoint and chain. A checkpoint is considered justified in a view $\mathcal{V}$ if $\mathcal{V}$ contains a supermajority set of links justifying it. A *justified* checkpoint which is maximal in $\mathcal{V}$ with respect to the previously defined lexicographic ordering is referred to as the *greatest justified checkpoint* in $\mathcal{V}$, denoted as $\mathcal{GJ}(\mathcal{V})$. In the event of ties, they are broken arbitrarily. Chain $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$ is referred to as the *greatest justified chain*.

Finality. A checkpoint $\mathcal{C}$ is *finalized* if it is justified and there exists a supermajority link with source $\mathcal{C}$ and potentially different targets $\mathcal{C}_i$ where $\mathcal{C}_i.c = \mathcal{C}.c + 1$. A chain ch is finalized if there exists a finalized checkpoint $\mathcal{C}$ with $\mathsf{ch} = \mathcal{C}.\mathsf{ch}$. The checkpoint $\mathcal{C} = (B_{\text{genesis}}, 0)$ is finalized by definition. This finalization rule is formalized by the binary function $\mathsf{F}(\mathcal{V}, \mathcal{C})$ in Algorithm 1 which outputs *true* if and only if checkpoint $\mathcal{C}$ is finalized according to the set of messages $\mathcal{V}$. Given a view $\mathcal{V}$, a finalized checkpoint which is maximal in $\mathcal{V}$ with respect to the previously defined lexicographic ordering is referred to as the *greatest finalized checkpoint* in $\mathcal{V}$, denoted as $\mathcal{GF}(\mathcal{V})$. In the event of ties, they are broken arbitrarily. We say that a chain ch is finalized according to a view $\mathcal{V}$ if and only if $\mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}).\mathsf{ch}$.

Slashing. A validator v_i is subject to slashing (as introduced in Section 2) for two *distinct* FFG-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 cast distinct FFG-VOTEs for the same checkpoint slot; or $\mathbf{E_2}$ (Surround voting) if $\mathcal{C}_3 < \mathcal{C}_1$ according to the lexicographic ordering on checkpoint and proposal slots, and $\mathcal{C}_2.c < \mathcal{C}_4.c$, indicating that a validator must not vote using a lower checkpoint as source and must avoid voting within the span of its other votes[14].

[14] In the implementation, for efficiency purposes, FFG-VOTEs do not include a chain, but only its hash. Then, given that we want to be able to determine whether a validator committed a slashable offense just by looking at the FFG-VOTEs that it sent,

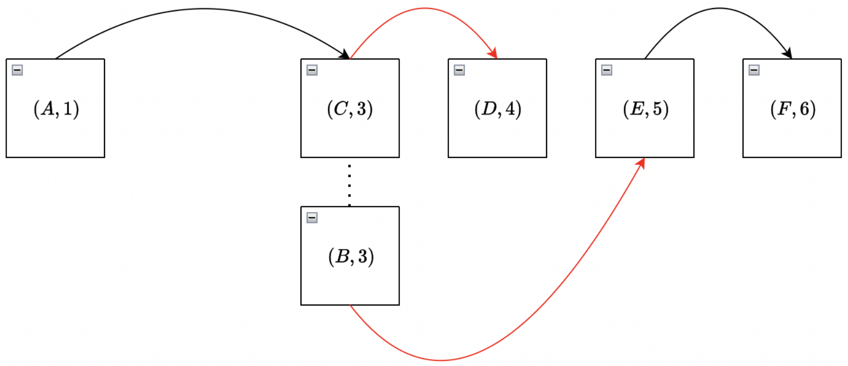


Figure 2: The figure illustrates an example of surround voting, where the slashable votes are highlighted in red. In this scenario, from slot 4 to slot 3, the source checkpoint shifts "backwards" from $(C, 3)$ to a lower checkpoint $(B, 3)$. This instance of surround voting is unique to this protocol and does not occur in Casper-FFG [2], due to the two source checkpoints sharing the same checkpoint slot.

4.1 Analysis

In this section, first we show that the finalized chain is accountably safe, exactly as done in Casper [2]. Then, we provide a set of conditions that, if satisfied, ensure that chains PROPOSEd by honest validators are finalized within two slots. In Sections 5 and 6, we will show that the algorithms presented in each of these sections ensure such properties after $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$.

Lemma 1. *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 ch' conflicting with ch_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 FFG-VOTEs with the chain of the target checkpoint being an extension of ch_f. Similarly, the justification of $(\mathsf{ch}', c_\mathsf{f})$ requires to have a supermajority of FFG-VOTEs with the chain of the target checkpoint being an extension of ch'. Given that any descendant of ch_f cannot be a descendant of ch', and vice versa, and that we need a supermajority set of links for justification, the intersection of the sets of voters contributing to the justification of $\mathcal{C}$ and $\mathcal{C}'$ include at least $\frac{n}{3}$ validators which have sent two FFG-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 $(A_i, c_i) \to (B_i, c'_\mathsf{j})$ and $(\mathsf{ch}_\mathsf{f}, c_\mathsf{f}) \to (C, c_\mathsf{f} + 1)$ be the FFG-VOTEs involved in the justification of $(\mathsf{ch}'_\mathsf{j}, c'_\mathsf{j})$ and in the finalization of ch_f, respectively, cast by a validator v_i. By definition of justification we have that $A_i \preceq \mathsf{ch}'_\mathsf{j} \preceq B_i$. 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:

in the implementation, checkpoints are represented by triples $\mathcal{C} = (H, c, p)$ where H is the hash of a chain ch and $p = \mathsf{ch}.p$. A checkpoint is valid only if $\mathsf{ch}.p = p$, and an FFG-VOTE is valid only if both checkpoints are valid.

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, there cannot exist any justified checkpoint $\mathcal{C}' = (\mathsf{ch}', c_\mathsf{f})$, with ch' conflicting with ch_f. Consequently, either $\mathsf{ch}_\mathsf{f} \preceq A_i$ holds, which implies either $\mathsf{ch}_\mathsf{f} \preceq \mathsf{ch}'_\mathsf{j}$, contradicting our assumption that $\mathsf{ch}_\mathsf{f} \not\preceq \mathsf{ch}'_\mathsf{j}$, or $A_i \prec \mathsf{ch}_\mathsf{f}$ which implies $A_i.p < \mathsf{ch}_\mathsf{f}.p$ from which we have that $(c_i, A_i.p) = (c_\mathsf{f}, A_i.p) < (c_\mathsf{f}, \mathsf{ch}_\mathsf{f}.p)$ and $c_\mathsf{f} + 1 < c'_\mathsf{j}$, which violates condition $\mathbf{E_2}$.

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 2. *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 1 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 $\mathbf{E_2}$. □

To complete the proof of Accountable Safety, we also need to show that up until a validator follows the protocol, it never gets slashed. This property cannot be concluded by the finalization and justifications rule alone as it depends on the algorithm employed to cast FFG-VOTEs. Then, to keep the treatment of Accountable Safety as decoupled as possible from this, below we detail a constraint to be guaranteed algorithm employed to cast FFG-VOTEs that then we prove to be sufficient to ensure that honest validators never get slashed.

Constraint A. *The algorithm employed to cast* FFG-VOTE*s must guarantee the following conditions.*

A.1. At most one FFG-VOTE *is sent in a slot.*

A.2. If $\mathcal{T}$ is the target checkpoint in the FFG-VOTE *sent in slot t, then $\mathcal{T}.c = t$.*

A.3. Let $\mathcal{S}$ and $\mathcal{S}'$ be the two source checkpoints in the FFG-VOTE*s sent in slot t and t'. If $t \leq t'$, then $\mathcal{S} \leq \mathcal{S}'$.*

Lemma 3. *If Constraint A holds, then honest validators are never slashed.*

Proof. Take any round r and any validator v_i honest in round r. Take also any two different FFG-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 FFG-VOTEs exists, then clearly v_i cannot be slashed. By Constraints A.1 and A.2, 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 $\mathcal{C}_2.c < \mathcal{C}_4.c$. By Constraint A.3, we know that $\mathcal{C}_1 \leq \mathcal{C}_3$. 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. □

Again with the aim to keep this analysis as general as possible, we define a second Constraint that is sufficient to ensure Accountable Safety.

Constraint B. *Let $\mathcal{V}^r_\mathsf{G}$ be the global view at time r,* i.e.*, the set of all messages sent up to time r, The chain chFin^r_i output by a validator v_i honest in round r is any finalized chain according to view $\mathcal{V}^r_\mathsf{G}$.*

Theorem 1 (Accountable Safety)**.** *Let $f \in [0, n]$. If Constraints A and B hold, then the finalized chain* chFin *is $\frac{n}{3}$-accountable.*

Proof. Follows from Lemmas 2 and 3. □

Let us now move to discuss Liveness. Given that also Liveness depends on the protocol employed to propose chains and cast FFG-VOTEs, like we did above, we provide a constraint to be guaranteed algorithm employed to cast FFG-VOTEs that then we prove to be sufficient to guarantee liveness of the FFG component.

Constraint C. *The algorithm employed to propose chains and cast* FFG-VOTE*s must guarantee the following conditions (assuming $f < \frac{n}{3}$).*

C.1. At any slot t, if an always honest validator v_i casts an FFG-VOTE *$\mathcal{S}_i \to \mathcal{T}_i$ during round r, then:*

C.1.1. the vote is valid;

C.1.2. There exists a set of messages $\mathcal{V}_i^{\mathsf{FFGvote},t}$ such that $\mathcal{S}_i = \mathcal{GJ}(\mathcal{V}_i^{\mathsf{FFGvote},t})$, i.e., the source checkpoint corresponds to the greatest justified checkpoint according to some set of messages $V_i^{\mathsf{vote},t} \subseteq \mathcal{V}_i^r$ in the view of validator v_i at round r; and

C.1.3. $\mathcal{T}_i.c = t$.

C.2. If a PROPOSE *message for chain ch_p is sent in a round $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta$ by a proposer honest in that round, then:*

C.2.1. $chain_p$ includes all the transactions in $txpool^{\mathsf{propose}(t)}$;

C.2.2. in slot t, any always-honest validator v_i casts a valid FFG-VOTE *$\mathcal{S} \to \mathcal{T}_i$ with $\mathcal{T}_i.\mathsf{ch} \preceq \mathsf{ch}_p$,* i.e., *all always-honest validators send* FFG-VOTE*s with the same source checkpoint and with a target checkpoint (potentially different for each validator) prefix of chain ch_p;*

C.2.3. in slot $t+1$:

C.2.3.1. for any always-honest validator v_i, $\mathcal{V}_i^{\mathsf{FFGvote},t+1}$ includes $\mathcal{V}_j^{\mathsf{FFGvote},t}$ of any always-honest validator v_j, and all the FFG-VOTE *messages sent by always-honest validators in slot t; and*

C.2.3.2. any always-honest validator v_i sends an FFG-VOTE *$\mathcal{S}_i \to \mathcal{T}_i$ with $\mathcal{T}_i.\mathsf{ch} = \mathsf{ch}_p$.*

C.2.4. in slot $t+2$,

C.2.4.1. for any always-honest validator v_i, $\mathcal{V}_i^{\mathsf{FFGvote},t+2}$ includes $\mathcal{V}_j^{\mathsf{FFGvote},t+1}$ of any always-honest validator v_j, and all the FFG-VOTE *messages sent by always-honest validators in slot $t+1$; and*

C.2.4.2. for any validator v_i honest in round $4\Delta(t+2)+2\Delta$, (i) $\mathcal{V}_i^{4\Delta(t+2)+2\Delta}$ includes all the FFG-VOTE*s cast by always-honest validators in slot $t+2$ and (ii) $\mathsf{chFin}_i^{4\Delta(t+2)+2\Delta} = \max(\{\mathsf{ch} \colon \mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}_i^{4\Delta(t+2)+2\Delta}).\mathsf{ch} \land \mathsf{ch}.p \preceq \mathsf{ch}_p.p\})$.*

Theorem 2 (Liveness)**.** *Assume that Constraint C holds (and $f < \frac{n}{3}$). Let v_p be any validator honest by the time it* PROPOSE*s chain ch_p in a slot t such that $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$. Then, chain ch_p is justified at slot $t+1$, and finalized at slot $t+2$. In particular, $\mathsf{ch}_p \preceq \mathsf{chFin}_i^{4\Delta(t+2)+2\Delta}$ and $txpool^{\mathsf{propose}(t)} \subseteq \mathsf{chFin}_i^{4\Delta(t+2)+2\Delta}$ for any validator $v_i \in H_{4\Delta(t+2)+2\Delta}$.*

Proof. Let $t := \text{slot}(r)$ and consider any honest validator v_i and the set of messages $\mathcal{V}_i^{\mathsf{FFGvote},t+1}$. By $f < \frac{n}{3}$, Constraint C.2.2 and Constraint C.2.3.1, we know that $\mathcal{V}_i^{\mathsf{FFGvote},t+1}$ includes a set of FFG-VOTEs from at least $\frac{2}{3}n$ of the validators such that each vote in this set has the same source checkpoint $\mathcal{S}$ and a target checkpoint prefix of chain ch_p. Also, by Constraint C.1.2 and Constraint C.2.3.1, we know that $\mathcal{S}$ is justified according to $\mathcal{V}_i^{\mathsf{FFGvote},t+1}$. Then, $f < \frac{n}{3}$ and Constraint C.1.3 imply that $\mathcal{GJ}(\mathcal{V}_i^{\mathsf{FFGvote},t+1}).c = t \land \mathcal{GJ}(\mathcal{V}_i^{\mathsf{FFGvote},t+1}).\mathsf{ch} \preceq \mathsf{ch}_p$. Also, by Constraint C.1.2, $\mathcal{GJ}(\mathcal{V}_i^{\mathsf{FFGvote},t+1})$ is the source checkpoint of the FFG-VOTE sent by v_i in slot $t+1$.

Then, given that by Constraint C.1.3 and Constraint C.2.3 all honest validators in slot $t+1$ send FFG-VOTEs with target $(\mathsf{ch}_p, t+1)$ and that, by Constraint C.2.4.1, all of these votes are included in $\mathcal{V}_i^{\mathsf{FFGvote},t+2}$, we have that $\mathcal{GJ}(\mathcal{V}_i^{\mathsf{FFGvote},t+2}) = (\mathsf{ch}_p, t+1)$.

Again by Constraint C.1.2, in slot t+2, all honest validators then send an FFG-VOTE with source $(\mathsf{ch}_p, t{+}1)$. By Constraint C.2.4.2, by round $4\Delta(t+2)+2\Delta$, all of these FFG-VOTEs are in the view of any validator v_i honest at such round. Because $f < \frac{n}{3}$, then $\mathsf{ch}_p \preceq \mathcal{GF}(\mathcal{V}_i^{4\Delta(t+2)+2\Delta}).\mathsf{ch}$. Again by Constraint C.2.4.2, $\mathsf{ch}_p \preceq \mathsf{chFin}_i^{4\Delta(t+2)+2\Delta}$. Constraint C.2.1 then implies that $txpool^{\mathsf{propose}(t)} \subseteq \mathsf{chFin}_i^{4\Delta(t+2)+2\Delta}$. □

By the Theorem above, we know that, as long as all the stated assumptions hold, then at time 2Δ into slot $t+2$ every honest validator has finalized the chain ch_p proposed by an honest proposer in slot t. However, already starting Δ time earlier, *i.e.*, from Δ time into slot $t+2$, no honest validator will ever finalize a chain conflicting with the chain ch_p proposed in slot t. This is formalized in the following Corollary.

Corollary 1. *Assume that Constraint C holds (and $f < \frac{n}{3}$). Let $\mathsf{chFin}^r_\mathsf{G}$ be the longest finalized chain according to view $\mathcal{V}^r_\mathsf{G}$ and v_p be any validator honest by the time it* PROPOSE*s chain ch_p in a slot t such that $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$. Then, $\mathsf{ch}_p \preceq \mathsf{chFin}^{4\Delta(t+2)+\Delta}_\mathsf{G}$.*

Proof. Given Constraint C.2.4.2, any honest validator send its FFG-VOTE for slot $t+2$ no later than time $4\Delta(t+2)+\Delta$. Then, the proof follows from the proof of Theorem 2. □

5 TOB-SVD-Based Faster Finality Protocol

In this section, our goal is to introduce a faster finality protocol for Ethereum that builds upon a dynamically-available protocol based on a modified version of the protocol TOB-SVD by D'Amato *et al.* [8]. We start by recalling the deterministically safe, dynamically-available consensus protocol TOB-SVD in Section 5.1. We refer the reader to the original paper for technical details. Our focus here is to briefly show how the protocol works, the guarantees it provides, and modifications to achieve a probabilistically safe, dynamically-available consensus protocol. Next, we show how to introduce fast confirmations in the modified consensus protocol and prove its η Dynamic Availability which enables its use in a majority-based faster finality protocol in Section 5.2. Finally, our majority-based faster finality protocol is presented in Section 5.3.

5.1 Revisiting TOB-SVD

TOB-SVD [8] proceeds as follows. During each slot t, a proposal (B) is made in the first round through a [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[15]. Let $\mathcal{V}$ be any view and t be the current slot. We define:

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

with $\mathsf{FIL}_{\eta\text{-exp}}$, $\mathsf{FIL}_{\text{lmd}}$, and FIL_{eq} defined as in Algorithm 2. Specifically, FIL_{eq} removes any VOTE message sent by a validator that has equivocated, $\mathsf{FIL}_{\eta\text{-exp}}$ retains only the unexpired VOTE messages, and $\mathsf{FIL}_{\text{lmd}}$ selects only the latest VOTE messages cast by each validator. So, intuitively, $\mathcal{V}^{\mathsf{ch},t}$ corresponds to the set of all latest VOTE messages in $\mathcal{V}$ that (i) are for a chain extending ch, (ii) are non-expired with respect to slot t, and (iii) are sent by validators that have never equivocated.

From here on, we adopt a *majority fork-choice function*, denoted as MFC and presented in Algorithm 2. The function $\text{MFC}(\mathcal{V}, \mathcal{V}', \mathsf{ch}^C, t)$ starts from a chain ch^C and returns the longest chain $\mathsf{ch} \succeq \mathsf{ch}^C$ such that $\left|\mathcal{V}^{\mathsf{ch},t} \cap (\mathcal{V}')^{\mathsf{ch},t}\right| > \frac{|\mathsf{S}(\mathcal{V}',t)|}{2}$, where $\mathsf{S}(\mathcal{V}',t)$ corresponds to the set of validators that, according to view $\mathcal{V}'$, have sent a non-expired VOTE message in slot t, *i.e.*, the majority of the validators in $\mathsf{S}(\mathcal{V}',t)$ are validators that have never equivocated according to either view $\mathcal{V}'$, and that, according to both views, their latest non-expired VOTE message is for a chain extending ch. If such chain does not exist, then it just returns ch^C.

The protocol executed by validator v_i works as follows:

1. **Propose,** $4\Delta t$**:** If v_p is the proposer for slot t, send message [PROPOSE, ch_p, t, v_p] through gossip, with $\mathsf{ch}_p \succeq \text{MFC}(\mathcal{V}_p, \mathcal{V}_p, B_{\text{genesis}}, t)$ and $\mathsf{ch}_p.p = t$[16].
2. **Vote,** $4\Delta t+\Delta$**:** Let $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, B_{\text{genesis}}, t)$ and send message [VOTE, ch', $\cdot$, t, v_i] through gossip, with ch' a chain PROPOSEd in slot t extending $\mathsf{ch}^{\mathsf{MFC}}$ and $\mathsf{ch}'.p = t$, or $\mathsf{ch}' = \mathsf{ch}^{\mathsf{MFC}}$ if no such proposal exists.

[15]Note that we use $\cdot$ for the third component of a VOTE message as this component is not necessary for this part of the paper. Specifically, this refers to the FFG component in Algorithm 4, similar to Algorithm 7. Since this section only describes a variant of the dynamically-available protocol TOB-SVD by D'Amato *et al.*, we omit the FFG component here and will reintroduce it later when presenting the final algorithm in Section 5.

[16]Note that, in practice, $\text{MFC}(\mathcal{V}_i, \mathcal{V}_i, B_{\text{genesis}}, t)$ is the parent of ch_p. However, in our treatment we consider the more general case where ch_p can be any extension of $\text{MFC}(\mathcal{V}_i, \mathcal{V}_i, B_{\text{genesis}}, t)$, as long as $\mathsf{ch}_p.p = t$.

Algorithm 2 MFC, the majority fork-choice function

```
1: function MFC($\mathcal{V}, \mathcal{V}', \mathsf{ch}^C, t$)
2:     return the longest chain $\mathsf{ch} \succeq \mathsf{ch}^C$ such that $\mathsf{ch} = \mathsf{ch}^C \vee \left|\mathcal{V}^{\mathsf{ch},t} \cap (\mathcal{V}')^{\mathsf{ch},t}\right| > \frac{\left|\mathsf{S}(\mathcal{V}',t)\right|}{2}$

3: function $\mathsf{S}(\mathcal{V}, t)$
4:     return $\{v_k : [\text{VOTE}, \cdot, \cdot, \cdot, v_k] \in \mathcal{V}' \wedge \mathcal{V}' = \mathsf{FIL}_{\eta\text{-exp}}(\mathcal{V}, t)\}$

5: function $\mathsf{FIL}_{\text{lmd}}(\mathcal{V})$
       let $\mathcal{V}' := \mathcal{V} \setminus \{[\text{VOTE}, \cdot, \cdot, t', v_k] \in \mathcal{V} : \exists [\text{VOTE}, \cdot, \cdot, t'', v_k] \in \mathcal{V} : t'' > t'\}$
       // $\mathcal{V}' := \mathcal{V}$ without all but the most recent (i.e., latest) votes of each validator
6:     return $\mathcal{V}'$

7: function $\mathsf{FIL}_{\eta\text{-exp}}(\mathcal{V}, t)$
8:     let $\mathcal{V}' := \mathcal{V} \setminus \{[\text{VOTE}, \cdot, \cdot, t', \cdot] \in \mathcal{V} : t' < t - \eta - 1\}$
       // $\mathcal{V}' := \mathcal{V}$ without all votes from slots $< t - \eta - 1$
9:     return $\mathcal{V}'$

10: function $\mathsf{FIL}_{\text{eq}}(\mathcal{V})$
11:    let $\mathcal{V}' := \mathcal{V} \setminus \{[\text{VOTE}, \cdot, \cdot, \cdot, v_k] \in \mathcal{V} : \exists t_{\text{eq}}, [\text{VOTE}, B', \cdot, t_{\text{eq}}, v_k] \in \mathcal{V}, [\text{VOTE}, B'', \cdot, t_{\text{eq}}, v_k] \in \mathcal{V} : B' \neq B''\}$
       // $\mathcal{V}' := \mathcal{V}$ without all votes by equivocators in $\mathcal{V}$
12:    return $\mathcal{V}'$
```

3. **Decide,** $4\Delta t + 2\Delta$**:** Set $\mathsf{Ch}_i = \text{MFC}(\mathcal{V}_i'', \mathcal{V}_i, B_{\text{genesis}}, t)$. Set $\mathcal{V}_i'' = \mathcal{V}_i$, and store $\mathcal{V}_i''$.

4. $4\Delta t + 3\Delta$**:** Set $\mathcal{V}_i^{\text{frozen}} = \mathcal{V}_i$, and store $\mathcal{V}_i^{\text{frozen}}$.

D'Amato *et al.* [8] have demonstrated that TOB-SVD satisfies both Safety and Liveness as defined in Definition 3 assuming $|H_{\mathsf{vote}(t-1)} \setminus A_{4\Delta t+2\Delta}| > |A_{4\Delta t+2\Delta}|$ [17]. While the original proofs by D'Amato *et al.* assume $\eta = 1$, the underlying arguments can readily be extended to scenarios where $\eta > 1$. This extension is feasible assuming a variant of Constraint (1) that considers the set $A_{\mathsf{vote}(t)}$ as the set of adversarial validators active at round $4\Delta t + 2\Delta$, and by considering only the latest messages cast by each validator. We do not directly prove this claim in the context of the current protocol. Instead, we demonstrate it in a modified version of the protocol, which we will present in the following.

The construction of a probabilistically safe total-order broadcast can be achieved by leveraging the protocol just presented, with a critical understanding that the validators' behavior is unaffected by decisions. Decisions primarily serve as a confirmation rule for the protocol and to ensure deterministic safety. In particular, the decide phase of slot t encapsulates both the decision-making and the storage of $\mathcal{V}''$, which is then used in the subsequent slot for further decisions. This indicates that all activities within the decide phase are pertinent only to making decisions, and nothing else. By omitting the decide phase, the protocol can be simplified, retaining only the necessary components to maintain probabilistic safety. We have then the following modified protocol.

1. **Propose,** $3\Delta t$**:** If v_i is the proposer for slot t, send message [PROPOSE, ch_p, t, v_i] through gossip, with $\mathsf{ch}_p \succeq \text{MFC}(\mathcal{V}_i, \mathcal{V}_i, B_{\text{genesis}}, t)$ and $\mathsf{ch}_p.p = t$.

2. **Vote,** $3\Delta t + \Delta$**:** Let $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, B_{\text{genesis}}, t)$ and send message [VOTE, ch', $\cdot$, t, v_i] through gossip, with ch' a PROPOSEd chain in slot t extending $\mathsf{ch}^{\mathsf{MFC}}$ and $\mathsf{ch}'.p = t$, or $\mathsf{ch}' = \mathsf{ch}^{\mathsf{MFC}}$ if no such proposal exists. Set $\mathsf{Ch}_i = \left(\mathsf{ch}^{\mathsf{MFC}}\right)^{\lceil\kappa}$.

3. $3\Delta t + 2\Delta$**:** Set $\mathcal{V}_i^{\text{frozen}} = \mathcal{V}$, and store $\mathcal{V}_i^{\text{frozen}}$. Set $t = t + 1$.

[17] Note that we consider the adversary at round $4\Delta t + 2\Delta$ due to the decision phase. This constraint differs from the one used in Constraint (1), which was intended for the modified version of the protocol that we will present shortly, which does not include a decision phase.

It is possible to show that when there is synchrony, Constraint (1) (with slots of duration 3Δ) holds, and assuming an honest proposer for slot t, it is guaranteed that the B will extend the lock (*i.e.*, $\mathsf{ch}_i^{3\Delta t+\Delta}$). As a result, all honest validators of slot t will vote for the honest proposal B of slot t. This property ensures that the validators' votes are consistently aligned with the honest proposal. This property allows us to obtain η Reorg Resilience. In turn, η Reorg Resilience gives us probabilistic safety for the κ-deep confirmation rule. Due to the similarity in the arguments presented in the proofs, we will demonstrate these statements on a final modification of the initially discussed protocol.

5.2 Probabilistically safe total-order broadcast with fast confirmations

This ultimate version of the protocol, presented in Algorithm 3, incorporates *fast confirmations.* The integration of these fast confirmations will serve as a building block for our majority-based faster finality protocol.

To this extent, Algorithm 3 defines the function $\texttt{fastconfirmsimple}(\mathcal{V}, t)$ to returns the tuple (ch^C, Q^C) where ch^C is a chain that has garnered support from more than $\frac{2}{3}n$ of the validators that have VOTEd in slot t according to the view $\mathcal{V}$, and Q^C, called *quorum certificate,* is a proof of this consisting of VOTE messages for chains extending ch^C. If such chain does not exist, then it just returns the tuple $(B_{\text{genesis}}, \emptyset)$.

Also, Algorithm 3 relies on the following external function.

- $\texttt{Extend}(\mathsf{ch}, t)$**:** If ch's length is less than t, then $\texttt{Extend}(\mathsf{ch}, t)$ produces a chain ch' of length t that extends ch and that includes all of the transactions in the transaction pool at time $\mathsf{propose}(t)$. The behavior is left unspecified for the case that ch's length is t or higher. This will not cause us any problem because, as we will show, this case can never happen. Formally,

  Property 1.

$$\begin{aligned}\mathsf{ch}.p < t \implies & \land\, \textbf{\textit{Extend}}(\mathsf{ch}, t) \succ \mathsf{ch} \\ & \land\, \forall tx \in \mathit{txpool}^{\mathsf{propose}(t)}, tx \in \textbf{\textit{Extend}}(\mathsf{ch}, t)\end{aligned}$$

Moreover, as per Algorithm 3, each honest validator v_i maintains the following state variables:

- **Message Set** $\mathcal{V}_i^{\text{frozen}}$: Each validator stores in $\mathcal{V}_i^{\text{frozen}}$ a snapshot of its view $\mathcal{V}_i$ at time 3Δ in each slot.
- **Chain variable** $\mathsf{ch}_i^{\text{frozen}}$: Each validator also stores in $\mathsf{ch}_i^{\text{frozen}}$ the greatest fast confirmed chain at time 3Δ in each slot t. The variable $\mathsf{ch}_i^{\text{frozen}}$ is then updated between time 0 and Δ of slot $t+1$ incorporating the information received via the PROPOSE message for slot $t+1$. This can be seen as effectively executing the view-merge logic (largely employed in the protocol presented in Section 6) on this variable.

Additionally, Algorithm 3 outputs the following chain.

- **Confirmed Chain** Ch_i**:** The confirmed chain Ch_i is updated at time Δ in each slot and also potentially at time 2Δ.

In the remainder of this work, when we want to refer to the value of a variable var *after* the execution of the code for round r, we write var^r. If the variable already has a superscript, like $\mathcal{V}_i^{\text{frozen}}$, we simply write $\mathcal{V}^{\text{frozen},r}$.

Then, Algorithm 3 proceeds as follows.

Propose: At round $4\Delta t$, there is the propose phase. Here, the proposer v_p of slot t broadcasts the proposal $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \cdot, t, v_p]$. Here, ch_p is the chain returned by the function $\texttt{Extend}(\mathsf{ch}^{\mathsf{MFC}}, t)$ where $\mathsf{ch}^{\mathsf{MFC}}$ corresponds to the output of the fork-choice $\text{MFC}(\mathcal{V}_i, \mathcal{V}_i, B_{\text{genesis}}, t)$. Whereas, ch^C and Q^C are the fast confirmed chain and relative quorum certificate, respectively, according to the current view of v_p as per output of $\texttt{fastconfirmsimple}$. Note that the proposer inputs in the fork-choice function MFC the current view $\mathcal{V}_p$ and fast confirmed chain ch^C.

Vote: During the interval $[4\Delta t, 4\Delta t + \Delta]$, a validator v_i, upon receiving a proposal $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \cdot, t', v_p]$ from the proposer v_p for slot $t' = t$, first verifies whether such proposal is *valid*. This corresponds to checking whether Q^C is a valid certificate for ch^C and $\mathsf{ch}_p.p = t$. If the PROPOSE message is valid, then validator v_i updates its local variable $\mathsf{ch}_i^{\text{frozen}}$ according to the received proposal.

During the voting phase, each validator v_i computes the fork-choice function MFC to continue building upon its output. The inputs to MFC are v_i's view at time 3Δ in the previous slot ($\mathcal{V}_i^{4\Delta(t-1)+3\Delta} = \mathcal{V}_i^{\text{frozen},\mathsf{vote}(t)}$), the current view ($\mathcal{V}_i^{\mathsf{vote}(t)}$), and the chain $\mathsf{ch}_i^{\text{frozen}}$ updated after receiving the PROPOSE message for slot t, if any.

Observe that by the definition of MFC (Algorithm 2), when computing the fork-choice during the voting phase, validator v_i only considers chains for which it received VOTE messages by round $4\Delta(t-1) + 3\Delta$, extending the chain from validators who have not equivocated up to the current round ($\mathsf{vote}(t)$), according to v_i's view. In contrast, when computing the fork-choice during the propose phase, the proposer considers any chain for which it received VOTE messages by the proposing time ($\mathsf{propose}(t)$), extending the chain from validators that have never equivocated according to the proposer's view at that time ($\mathsf{propose}(t)$). This approach ensures that the chain output by the fork-choice function in the voting phase is the prefix of the chain output by the fork-choice function in the propose phase of the same slot. This property is also known as Graded Delivery [8].

If validator v_i receives a valid PROPOSE message for a chain that extends the output of the fork-choice function, it casts a VOTE for that chain. Otherwise, it casts a VOTE for the current output of the fork-choice function.

In the voting phase, validator v_i also sets the confirmed chain Ch_i to the highest chain among the κ-deep prefix of the chain output by MFC and the previous confirmed chain, filtering out this last chain if it is not a prefix of the chain output by MFC.

Fast Confirm: Validator v_i checks if a chain ch can be fast confirmed, i.e., if at least $2/3n$ of the validators cast a VOTE message for a chain $\mathsf{ch}^{\text{cand}}$. If that is the case, it sets Ch_i to this chain.

Merge: At round $4\Delta t + 3\Delta$, validator v_i updates its variables $\mathcal{V}_i^{\text{frozen}}$ and $\mathsf{ch}_i^{\text{frozen}}$ to be used in the following slot.

For readability purpose, we often use $\mathsf{fconf}(t) := 4\Delta t + 2\Delta$ and $\mathsf{merge}(t) := 4\Delta t + 3\Delta$.

5.2.1 Analysis

Now, we proceed with proving that Algorithm 3 satisfied η Reorg Resilience (Theorem 3), η Dynamic Availability (Theorem 4), η Asynchronous Reorg Resilience (Theorem 5) and η Asynchronous Safety Resilience (Theorem 6). Remember that throughout this work we assume that less than one-third of the entire validator set is ever controlled by the adversary (*i.e.*, $f < \frac{n}{3}$). In all the following results, we also assume $\mathsf{GST} = 0$, , and that Constraint (1) holds.

In the remaining part of this section, given any slot t, we let $\text{MFC}_i^{\mathsf{propose}(t)} := \text{MFC}(\mathcal{V}_i^{\mathsf{propose}(t)}, \mathcal{V}_i^{\mathsf{propose}(t)}, \mathsf{ch}^C, t)$ with $(\mathsf{ch}^C, \cdot) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{\mathsf{propose}(t)}, t-1)$, and $\text{MFC}_i^{\mathsf{vote}(t)} := \text{MFC}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t)}, \mathcal{V}_i^{\mathsf{vote}(t)}, \mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)}, t)$.

At high level, our analysis proceeds as follows. Lemmas 4 to 6 set the base argument. Specifically, in Lemma 4, we show that if all active validators in a given slot t VOTE for a chain extending ch, then, for any validator active at the following slot $t+1$, the output of the fork-choice function MFC will be an extension of ch which, importantly, also implies that any validator active in slot $t+1$ VOTE for an extension of ch. Then, Lemma 5 leverages Lemma 4 to show that if any active validator fast confirms a chain ch in a given slot t, then for any validator active at any subsequent slot, the output of the fork-choice function MFC will be an extension of ch. Finally, in Lemma 6 we show similar results for any chain proposed by an honest validator. Thereafter, Lemma 7 makes use of these initial results to prove that the chain confirmed by any validator honest in a given round r_i, regardless of whether they are active in that round, is a prefix of the chain output by the fork-choice MFC by any validator active at subsequent rounds. This allows us to then

Algorithm 3 Probabilistically safe variant of the total-order broadcast protocol of D'Amato *et al.* [8] with fast confirmations - code for v_i

```
1: Output
2:     $\mathsf{Ch}_i \leftarrow B_{\text{genesis}}$: confirmed chain of validator $v_i$
3: State
4:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \{B_{\text{genesis}}\}$: snapshot of $\mathcal{V}$ at time $4\Delta t + 3\Delta$
5:     $\mathsf{ch}_i^{\text{frozen}} \leftarrow B_{\text{genesis}}$: snapshot of the fast confirmed chain at time $4\Delta t + 3\Delta$
6: function fastconfirmsimple($\mathcal{V}, t$)
7:     let $\mathsf{fast}^{\text{cands}} := \{\mathsf{ch} \colon |\{v_j \colon \exists \mathsf{ch}' \succeq \mathsf{ch} \colon\ [\text{VOTE}, \mathsf{ch}', \cdot, t, \cdot] \in \mathcal{V}\}| \geq \frac{2}{3}n\}$
8:     if $\mathsf{fast}^{\text{cands}} \neq \emptyset$ then
9:         let $\mathsf{fast}^{\text{cand}} := \max\left(\mathsf{fast}^{\text{cands}}\right)$
10:        let $Q := \{[\text{VOTE}, \mathsf{ch}', \cdot, t, \cdot] \in \mathcal{V} : \mathsf{ch}' \succeq \mathsf{fast}^{\text{cand}}\}$
11:        return $(\mathsf{fast}^{\text{cand}}, Q)$
12:    else
13:        return $(B_{\text{genesis}}, \emptyset)$
   PROPOSE
14: at round $4\Delta t$ do
15:    if $v_i = v_p^t$ then
16:        let $(\mathsf{ch}^C, Q^C) :=$ fastconfirmsimple($\mathcal{V}_i, t-1$)
17:        let $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i, \mathcal{V}_i, \mathsf{ch}^C, t)$
18:        let $\mathsf{ch}_p := \mathsf{Extend}(\mathsf{ch}^{\mathsf{MFC}}, t)$
19:        send message $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}^C, Q^C, \cdot, t, v_i]$ through gossip
   VOTE
20: at round $4\Delta t + \Delta$ do
21:    let $\mathsf{ch}^{\mathsf{MFC}} := \text{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, \mathsf{ch}_i^{\text{frozen}}, t)$
22:    $\mathsf{Ch}_i \leftarrow \max(\{\mathsf{ch} \in \{\mathsf{Ch}_i, (\mathsf{ch}^{\mathsf{MFC}})^{\lceil \kappa}\} : \mathsf{ch} \preceq \mathsf{ch}^{\mathsf{MFC}}\})$
23:    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
24:    send message $[\text{VOTE}, \mathsf{ch}, \cdot, t, v_i]$ through gossip
   FAST CONFIRM
25: at round $4\Delta t + 2\Delta$ do
26:    let $(\mathsf{fast}^{\text{cand}}, Q) :=$ fastconfirmsimple($\mathcal{V}_i, t$)
27:    if $Q \neq \emptyset$ then
28:        $\mathsf{Ch}_i \leftarrow \mathsf{fast}^{\text{cand}}$
   MERGE
29: at round $4\Delta t + 3\Delta$ do
30:    $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i$
31:    $(\mathsf{ch}_i^{\text{frozen}}, \cdot) \leftarrow$ fastconfirmsimple($\mathcal{V}_i, t$)

32: upon receiving a gossiped 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
33:    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
34:        $\mathsf{ch}_i^{\text{frozen}} \leftarrow \mathsf{ch}_p^C$
```

prove η Reorg Resilience in Theorem 3. Moving on, Lemma 8 shows that the premise of Property 1 is always satisfied which implies that honest proposers are always able to include all the transactions in the transaction pool in the chain that they propose. This, Lemma 6 and Lemma 7 are then put to use in Theorem 4 to show that Algorithm 3 is an η-dynamically-available protocol. Thereafter, Lemma 9 proves that chains can be fast confirmed which is not listed among the security properties but very relevant to Algorithm 3. Finally, the remaining Lemmas and Theorems deal with asynchrony resilience and we leave their high-level overview up to that point.

Lemma 4. *If, in slot t, all validators in $H_{\mathsf{vote}(t)}$ cast* VOTE *messages for chains extending chain* ch*, then, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $\text{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $\text{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ cast* 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 $\mathsf{propose}(t+1)$: $\mathsf{ch}_i^{\text{frozen},\mathsf{propose}(t+1)} = B_{\text{genesis}} \vee \mathsf{ch}_i^{\text{frozen},\mathsf{propose}(t+1)} \succeq \mathsf{ch}$. Now we consider each case separately and for each of them prove that $\text{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$.

Case 1: $\mathsf{ch}_i^{\text{frozen},\mathsf{propose}(t+1)} = B_{\text{genesis}}$**.** 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 $\mathcal{V}_i^{\mathsf{propose}(t+1)}$. 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, $\mathsf{S}(\mathcal{V}_i, t+1) \setminus \left(H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right) \subseteq A_{\mathsf{vote}(t+1)} \cup \left(H_{\mathsf{vote}(t-\eta+1),\mathsf{vote}(t-1)} \setminus H_{\mathsf{vote}(t)}\right)$ 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 $\mathsf{vote}(t-\eta+1)$ and $\mathsf{vote}(t-1)$ but not active at time $\mathsf{vote}(t)$. Then, we have $|\mathsf{S}(\mathcal{V}_i, t+1)| = \left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right| + \left|\left(\mathsf{S}(\mathcal{V}_i, t+1) \setminus \left(H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right)\right)\right| \leq \left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right| + \left|A_{\mathsf{vote}(t+1)} \cup \left(H_{\mathsf{vote}(t-\eta+1),\mathsf{vote}(t-1)} \setminus H_{\mathsf{vote}(t)}\right)\right| < 2\left|H_{\mathsf{vote}(t)} \setminus A_{\mathsf{vote}(t+1)}\right|$ where the last inequality comes from Constraint (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, $\text{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$.

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

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 34 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 34, 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 casts a VOTE message for a chain extending ch. □

Lemma 5. *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* $v_i \in H_{\mathsf{vote}(t')}$*,* $\text{MFC}_i^{\mathsf{propose}(t')} \succeq \mathsf{ch}$ *and* $\text{MFC}_i^{\mathsf{vote}(t')} \succeq \mathsf{ch}$*, which implies that, all validators in* $H_{\mathsf{vote}(t')}$ *cast 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 33 to 34 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 $H_{\mathsf{vote}(t+1)}$ cast VOTE messages extending chain ch.

Inductive Step: $t' > t+1$. Here we can just apply Lemma 4 to conclude the proof. □

Lemma 6. *Let t be a slot with an honest proposer v_p and assume that v_p casts a* $[\textsc{propose}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, t, v_p]$ *message. Then, for any slot $t' \geq t$, all validators in $H_{\mathsf{vote}(t')}$ cast a* VOTE *message for a chain extending* ch_p*. Additionally, for any slot $t'' > t$ and any validator $v_i \in H_{\mathsf{vote}(t'')}$, $\mathrm{MFC}_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ and $\mathrm{MFC}_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$.*

Proof. The proof is by induction on t'.

Base Case: $t' = t$. Suppose that in slot t, an honest proposer v_p sends a $[\textsc{propose}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C t, v_p]$ 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 33 to 34, $\mathsf{ch}_i^{\text{frozen},\mathsf{vote}(t)} = \mathsf{ch}_p^C$.

We know that either $\left|\left(\mathcal{V}_i^{\text{frozen}}\right)^{\mathrm{MFC}_i^{\mathsf{vote}(t)},t} \cap \mathcal{V}_i^{\mathrm{MFC}_i^{\mathsf{vote}(t)},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_i,t)|}{2}$ or $\mathrm{MFC}_i^{\mathsf{vote}(t)} = \mathsf{ch}_i^{\text{frozen}}$. 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 [8], this implies that at time $\mathsf{propose}(t)$, $\left|\mathcal{V}_p^{\mathrm{MFC}_i^{\mathsf{vote}(t)},t}\right| > \frac{|\mathsf{S}(\mathcal{V}_p,t)|}{2}$ meaning that, due to Line 18, $\mathsf{ch}_p \succeq \mathrm{MFC}_i^{\mathsf{vote}(t)}$ and hence, due to Lines 23 and 24, in slot t, v_i casts a VOTE message for ch_p.

Case 2: $\mathbf{MFC}_i^{\mathsf{vote}(t)} = \mathsf{ch}_i^{\textbf{frozen}}$. Due to Lines 23 and 24, v_i still casts a VOTE for ch_p as $\mathsf{ch}_p \succeq \mathsf{ch}_p^C = \mathsf{ch}_i^{\text{frozen}} = \mathrm{MFC}_i^{\mathsf{vote}(t)}$.

Inductive Step: $t' > t$. Here we can just apply Lemma 4 to conclude the proof. ☐

Lemma 7. *Let r_i be any round and r_j be any round such that $r_j \geq r_i$ and $r_j \in \{\mathsf{propose}(\mathrm{slot}(r_j)), \mathsf{vote}(\mathrm{slot}(r_j))\}$. Then, for any validator v_i honest in round r_i and any validator $v_j \in H_{r_j}$, $\mathsf{Ch}_i^{r_i} \preceq \mathrm{MFC}_j^{r_j}$.*

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 $r_j \in \{\mathsf{propose}(\mathrm{slot}(r_j)), \mathsf{vote}(\mathrm{slot}(r_j))\}$ and $\mathsf{Ch}_i^{r_i} \npreceq \mathrm{MFC}_j^{r_j}$, 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 $t_i = \mathrm{slot}(r_i)$ and proceed by analyzing each case separately.

Case 1: r_i is a vote round. Due to Line 22, $\mathsf{Ch}_i^{r_i} \succeq \left(\mathrm{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa}$. 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 2 [10]), 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 6 implies that $\mathrm{MFC}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p$. Then, because $t_p \geq t_i - \kappa$, we have that $\mathsf{ch}_p \succeq \left(\mathrm{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa} = \mathsf{Ch}_i^{r_i}$. Because $t_p < \mathrm{slot}(r_j)$, Lemma 6 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 $t_i < \mathrm{slot}(r_j)$. 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 5 to conclude that $\mathsf{ch}_j^{r_j} \succeq \mathsf{Ch}_i^{r_i}$ reaching a contradiction. ☐

Theorem 3 (Reorg Resilience)**.** *Algorithm 3 is η-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 $t_j > \max(t_p, \text{slot}(r_i))$. By Constraint (2) we know that $H_{\mathsf{vote}(t_j)}$ is not empty. So, pick any validator $v_j \in H_{\mathsf{vote}(t_j)}$. Lemma 6 implies that $\text{MFC}_j^{\mathsf{vote}(t_j)} \succeq \mathsf{ch}_p$. Lemma 7 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 8. *For any slot t and validator $v_i \in H_{\mathsf{propose}(t)}$, $\text{MFC}_i^{\mathsf{propose}(t)}.p < t$.*

Proof. Due to the Validity property of Graded Agreement [8] 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 23. □

Theorem 4. *Algorithm 3 is η-dynamically-available.*

Proof. We begin by proving the η 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 2 [10]). Let ch_p be the chain PROPOSEd by the honest proposer v_p in slot t_p. Due to Lemma 8 and Property 1, we know that ch_p includes all of the transaction in $\mathit{txpool}^{\mathsf{propose}(t_p)}$. Given that $\mathsf{propose}(t_p) \geq r$, $\mathit{txpool}^r \subseteq \mathit{txpool}^{\mathsf{propose}(t_p)}$, 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 6, $\text{MFC}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p$. 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 6, we know that any VOTE cast 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 $\eta-$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 7 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 9 (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 6 that every honest validator in $H_{\mathsf{vote}(t)}$ casts 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 27 and 28,$\mathsf{Ch}_i^{\mathsf{fconf}(t)} \succeq \mathsf{ch}_p$. □

Finally, we move to demonstrating that Algorithm 3 also provides η Asynchrony Reorg Resilience and η Asynchrony Safety Resilience, as defined in Section 2.3. Lemma 10 shows that if from slot t_a till a slot t' falling within the short period of asynchrony, all aware validators VOTE for chains extending ch, then so will do all aware validators at the next slot $t'+1$. Next, Lemma 12 leverages this result proving that, if in a slot before the short period of asynchrony starts, all active validators VOTE for chains extending ch, then the chain confirmed by any aware validator at any subsequent round does not conflict with ch. This and Theorem 3 allow then concluding in Theorem 5 that Algorithm 3 is η-asynchrony-reorg-resilient. Finally, Theorem 6 makes use of Lemmas 7 and 12, and Theorem 4 to prove that Algorithm 3 is also η-asynchrony-safety-resilient.

Lemma 10. *Let t be any slot in $[t_a, t_a+\pi]$ with $\pi > 0$. Assume that in any slot $t' \in [t_a, t]$, all validators in $W_{\mathsf{vote}(t')}$ cast* VOTE *messages for chains extending chain* ch. *Then, in slot $t+1$, all validators in $W_{\mathsf{vote}(t+1)}$ also cast* VOTE *messages for chains extending chain* ch.

Proof. The lemma makes use of two main ingredients: the expiration period is $\eta = \pi + 2$, and Constraint (2). Firstly, the votes which influence the voting phase in slot $t+1$ are those from slots $[t-\eta, t]$. These include votes from slot t_a since $t+1-\eta \leq t_a+\pi+1-\eta < t_a$. Since by Constraint (3) all validators in $H_{\mathsf{vote}(t_a)} \setminus A_{\mathsf{vote}(t_a+1)}$ were awake at $4\Delta t_a + 2\Delta$, they have all received each other's votes from slot t_a. Then, from the Lemma's assumption, for any validator $v_i \in W_{\mathsf{vote}(t+1)}$ and any validator $v_a \in H_{\mathsf{vote}(t_a)} \setminus A_{\mathsf{vote}(t+1)}$, the latest message from v_a in $\mathcal{V}_i^{\mathsf{vote}(t+1)}$ is for a chain extending ch. By Constraint (2), validators in $H_{\mathsf{vote}(t_a)} \setminus A_{\mathsf{vote}(t+1)}$ constitute a majority of all unexpired votes in slot $t+1$. This implies that any such validator v_i would again have $\left|\left(\mathcal{V}_i^{\text{frozen}}\right)^{\mathsf{ch},t+1} \cap \mathcal{V}_i^{\mathsf{ch},t+1}\right| > \frac{|\mathsf{S}(\mathcal{V}_i,t+1)|}{2}$. Note also that because we assume $f < \frac{n}{3}$, for any validator $v_i \in H_{\mathsf{fconf}(t)}$, if $(\mathsf{ch}', Q) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{\mathsf{fconf}(t)}, t)$ and $Q \neq \emptyset$, then $\mathsf{ch}' \succeq \mathsf{ch}$. Therefore, any such validator would in slot $t+1$ cast a VOTE for an extension of ch. □

Lemma 11. *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending* ch. *Then, for any slot $t_i \geq t$, any validator in $W_{\mathsf{vote}(t_i)}$ casts a* VOTE *message for a chain extending* ch.

Proof. The proof is by induction on t_i.

Base Case: $t_i \in [t, t_a]$. From Lemma 4.

Inductive Step: $t_i > t_a$. We assume that the Lemma holds for slot $t_i - 1$ and prove that it holds also for slot t_i. Let us proceed by cases.

Case 1: $t_i \in [t_a+1, t_a+\pi+1]$. From Lemma 10.

Case 2: $t_i = t_a+\pi+2$. From the induction hypothesis, we know that in slot $t_a+\pi+1$, any validator in $W_{\mathsf{vote}(t_a+\pi+1)}$ casts a VOTE message for a chain extending ch. Since slot $t_a+\pi+1$ is synchronous, such VOTE messages are all in the view $\mathcal{V}_i^{\mathsf{vote}(t_a+\pi+2)}$ of any validator $v_i \in W_{\mathsf{vote}(t_a+\pi+2)} = H_{\mathsf{vote}(t_a+\pi+2)}$. Also, given that $\eta = \pi+2$, such VOTE messages are not expired in slot $t_a+\pi+2$ as $t_a+\pi+2-\eta \leq t_a$. Given that Constraint (2) holds for $t_a+\pi+2$, we can use the same reasoning applied in Lemma 10 to conclude that validator $v_i \in H_{\mathsf{vote}(t_a+\pi+2)}$ casts a VOTE message for a chain extending ch.

Case 3: $t_a \geq t_a+\pi+3$. Given that for any slot $t_j \geq t_a+\pi+2$, $W_{\mathsf{vote}(t_j)} = H_{\mathsf{vote}(t_j)}$, here we can just apply Lemma 4. □

Lemma 12. *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending* ch. *Then, for any round $r_i \geq \mathsf{vote}(t)$ and validator $v_i \in W_{r_i}$, $\mathsf{Ch}_i^{r_i}$ does not conflict with* ch.

Proof. Take any round $r_i \geq \mathsf{vote}(t)$ and validator $v_i \in W_{r_i}$. Given that the confirmed chain is updated only either in a voting or fast confirmation round, let us consider the two following cases.

Case 1: $r_i = \mathsf{vote}(\text{slot}(r_i))$. From Lemma 11 we know that any validator $v_i \in W_{\mathsf{vote}(\text{slot}(r_i))}$ casts a VOTE messages for some chain $\mathsf{ch}' \succeq \mathsf{ch}$. Due to Lines 22 to 23, $\mathsf{ch}' \succeq \text{MFC}^{\mathsf{vote}(\text{slot}(r_i))} \succeq \mathsf{Ch}_i^{r_i}$. Hence, $\mathsf{Ch}_i^{r_i}$ does not conflict with ch.

Case 2: $r_i = \mathsf{fconf}(\text{slot}(r_i))$. Let us consider two sub cases.

Case 2.1: $\mathsf{Ch}_i^{r_i} = \mathsf{Ch}_i^{r_i-1}$. This implies that $\mathsf{Ch}_i^{r_i} = \mathsf{Ch}_i^{\mathsf{vote}(\text{slot}(r_i))}$. Due to the joining protocol, $v_i \in W_{\mathsf{fconf}(\text{slot}(r_i))}$ implies $v_i \in W_{\mathsf{vote}(\text{slot}(r_i))}$. Therefore, this case is proved by Case 1 above.

Case 2.2: $\mathsf{Ch}_i^{r_i} \neq \mathsf{Ch}_i^{r_i-1}$. This implies $(\mathsf{Ch}_i^{r_i}, Q) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{r_i}, \text{slot}(r_i)) \wedge Q \neq \emptyset$. Given that, as established above, all validators in $W_{\mathsf{vote}(\text{slot}(r_i))}$ cast VOTE messages for chains extending ch, Constraint (2) implies that the number of validators that might cast a VOTE message for a chain non extending ch is less than half the total number of validators. This implies that any chain non extending ch cannot receive a quorum of VOTE message for it. Therefore, $\mathsf{Ch}_i^{r_i} \succeq \mathsf{ch}$ meaning that $\mathsf{Ch}_i^{r_i}$ does not conflict with ch. □

Theorem 5 (Asynchrony Reorg Resilience)**.** *Algorithm 3 is η-asynchrony-reorg-resilient.*

Proof. Assume $\pi > 0$ and take a slot $t_p \leq t_a$ with an honest proposer, any round r_i and any validator $v_i \in W_{r_i}$. If $r_i < \mathsf{vote}(t_p)$, we can apply Theorem 3. Otherwise, we can apply Lemmas 6 and 12. ☐

Theorem 6 (Asynchrony Safety Resilience)**.** *Algorithm 3 η-asynchrony-safety-resilient.*

Proof. Assume $\pi > 0$ and pick any round $r_i \leq 4\Delta t_a + \Delta$, any round r_j, any validator v_i honest in r_i and any validator $v_j \in W_{r_j}$. Let us now proceed by cases.

Case 1: $r_j < \mathsf{vote}(t_a)$**.** The proof for this case follows from η Safety (Theorem 4).

Case 2: $r_j \geq \mathsf{vote}(t_a)$**.** Lemma 7 implies that for any validator $v_k \in H_{\mathsf{vote}(t_a)}$, $\mathrm{MFC}_k^{\mathsf{vote}(t_a)} \succeq \mathsf{Ch}_i^{r_i}$. This implies that, in slot t_a, v_k casts a VOTE message for a chain extending $\mathsf{Ch}_i^{r_i}$. Hence, we can apply Lemma 12 to conclude the proof for this case. ☐

We are now prepared to develop our faster-finality protocol, which integrates the dynamically-available protocol Algorithm 3 with the FFG components introduced in Section 4 to obtain an η-secure ebb-and-flow protocol.

5.3 Faster finality protocol execution

In this section, we present Algorithm 4 which integrates the η-dynamically-available and reorg-resilient protocol of Algorithm 3 with the FFG component introduced in Section 4 to obtain a η-secure ebb-and-flow protocol. We describe Algorithm 4 by discussing the main differences compared to Algorithm 3.

The first difference is in the logic employed by Algorithm 4 to determine fast confirmed chains. This is encoded in the function $\texttt{fastconfirm}(\mathcal{V}, t)$ which returns the same output of $\texttt{fastconfirmsimple}(\mathcal{V}, t)$ from Algorithm 4 as long as the returned fast confirmed chain extends $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$. Otherwise, it just returns $(\mathcal{GJ}(\mathcal{V}).\mathsf{ch}, \emptyset)$. The reason underpinning this change is to allow η-dynamical-availability and Reorg Resilience to be ensured again after long periods of asynchrony once the network becomes synchronous (see Section 8 for more details).

Second, compared to Algorithm 3, Algorithm 4 maintains the following additional state variable:

- **Checkpoint Variable** $\mathcal{GJ}_i^{\text{frozen}}$: Each validator stores in $\mathcal{GJ}_i^{\text{frozen}}$ the greatest justified checkpoint according to their view at time 3Δ in slot $t-1$. Like $\mathsf{ch}_i^{\text{frozen}}$, this variable is then updated between time 0 and Δ in slot t incorporating the information received via the PROPOSE message for slot t. Also like $\mathsf{ch}_i^{\text{frozen}}$, this can be seen as effectively executing a view-merge on this variable.

Next, Algorithm 4 outputs two chains.

- **Available Chain** chAva_i**:** This roughly corresponds to the confirmed chain Ch_i of Algorithm 3.
- **Finalized Chain** chFin_i**:** At any fast confirmation round, chFin is set to $\mathcal{GF}(\mathcal{V}_i).\mathsf{ch}$, *i.e.*, the chain of the greatest justified checkpoint.

  Also, at any vote round, the finalized chain chFin_i is set to the longest chain such that of $\mathsf{chFin}_i \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch} \land \mathsf{chFin}_i \preceq \mathsf{chAva}_i$[18][19] .

Finally, compared to Algorithm 3, Algorithm 4 proceeds as follows.

[18]This is to ensure that chFin_i is always a prefix of chAva_i, as required by η-secure ebb-and-flow protocols. Simply, setting chFin_i to $\mathcal{GF}(\mathcal{V}_i)$ in vote rounds would not ensure such properties during asynchrony.

[19]Setting chFin_i in vote rounds is not strictly required to ensure any of the theorems or lemmas of this work, even though, if one did remove setting chFin in vote rounds, then one would need to tweak some of the proofs. However, we decided to also set chFin in vote round as, there are scenarios under asynchrony whereby setting chFin in the vote round of slot t results in a chain longer than the chain chFin that was set in the fast confirmation round of slot $t-1$. Also, we believe, but have no formal proof for it yet, that updating chFin at vote and fast confirmation rounds provides the highest frequency at which chFin can be updated without breaking any of the theorems and lemmas of this work.

Algorithm 4 Faster finality protocol – code for validator v_i

```
1: Output
2:     $\mathsf{chAva}_i \leftarrow B_{\text{genesis}}$: available chain
3:     $\mathsf{chFin}_i \leftarrow B_{\text{genesis}}$: finalized chain
4: State
5:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \{B_{\text{genesis}}\}$: snapshot of $\mathcal{V}$ at time $4\Delta t + 3\Delta$
6:     $\mathsf{ch}_i^{\text{frozen}} \leftarrow B_{\text{genesis}}$: snapshot of the fast confirmed chain at time $4\Delta t + 3\Delta$
7:     $\mathcal{GJ}_i^{\text{frozen}} \leftarrow (B_{\text{genesis}}, 0)$: latest frozen greatest justified checkpoint
8: function fastconfirm($\mathcal{V}, t$)
9:     let $(\mathsf{ch}^C, Q) :=$ fastconfirmsimple($\mathcal{V}, t$)
10:    if $\mathsf{ch}^C \succeq \mathcal{GJ}(\mathcal{V}).\mathsf{ch}$ then
11:        return $(\mathsf{ch}^C, Q)$
12:    else
13:        return $(\mathcal{GJ}(\mathcal{V}).\mathsf{ch}, \emptyset)$
   PROPOSE
14: at round $4\Delta t$ do
15:    if $v_i = v_p^t$ then
16:        let $(\mathsf{ch}^C, Q^C) :=$ fastconfirm($\mathcal{V}_i, t-1$)
17:        let $\mathsf{ch}^{\mathsf{MFC}} := \mathrm{MFC}(\mathcal{V}_i, \mathcal{V}_i, \mathsf{ch}^C, t)$
18:        let $\mathsf{ch}_p := \mathsf{Extend}(\mathsf{ch}^{\mathsf{MFC}}, t)$
19:        send message [PROPOSE, $\mathsf{ch}_p$, $\mathsf{ch}^C$, $Q^C$, $\mathcal{GJ}(\mathcal{V}_i)$, $t$, $v_i$] through gossip
   VOTE
20: at round $4\Delta t + \Delta$ do
21:    let $\mathsf{ch}^{\mathsf{MFC}} := \mathrm{MFC}(\mathcal{V}_i^{\text{frozen}}, \mathcal{V}_i, \mathsf{ch}_i^{\text{frozen}}, t)$
22:    $\mathsf{chAva}_i \leftarrow \max(\{\mathsf{ch} \in \{\mathsf{chAva}_i, (\mathsf{ch}^{\mathsf{MFC}})^{\lceil \kappa}, \mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}\} : \mathsf{ch} \preceq \mathsf{ch}^{\mathsf{MFC}}\})$
23:    $\mathsf{chFin}_i \leftarrow \max(\{\mathsf{ch}\colon \mathsf{ch} \preceq \mathsf{chAva}_i \wedge \mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}\})$
24:    let $\mathcal{T} := (\mathsf{chAva}_i, t)$
25:    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
26:    send message [VOTE, $\mathsf{ch}$, $\mathcal{GJ}_i^{\text{frozen}} \to \mathcal{T}$, $t$, $v_i$] through gossip
   FAST CONFIRM
27: at round $4\Delta t + 2\Delta$ do
28:    let $(\mathsf{fast}^{\text{cand}}, \cdot) :=$ fastconfirm($\mathcal{V}_i, t$)
29:    if $\mathsf{chAva}_i \not\succeq \mathsf{fast}^{\text{cand}}$ then
30:        $\mathsf{chAva}_i \leftarrow \mathsf{fast}^{\text{cand}}$
31:    $\mathsf{chFin}_i \leftarrow \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}$
   MERGE
32: at round $4\Delta t + 3\Delta$ do
33:    $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i$
34:    $(\mathsf{ch}_i^{\text{frozen}}, \cdot) \leftarrow$ fastconfirm($\mathcal{V}_i, t$)
35:    $\mathcal{GJ}_i^{\text{frozen}} \leftarrow \mathcal{GJ}(\mathcal{V}_i)$

36: upon receiving a gossiped 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
37:    when round $4\Delta t + \Delta$ do
38:        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
39:            $\mathcal{GJ}_i^{\text{frozen}} \leftarrow \mathcal{GJ}_p$
40:            if $\mathsf{ch}_i^{\text{frozen}} \not\succeq \mathcal{GJ}_p.\mathsf{ch}$ then
41:                $\mathsf{ch}_i^{\text{frozen}} \leftarrow \mathcal{GJ}_p.\mathsf{ch}$
42:            if $\mathsf{ch}_i^{\text{frozen}} \preceq \mathsf{ch}_p^C$ then
43:                $\mathsf{ch}_i^{\text{frozen}} \leftarrow \mathsf{ch}_p^C$
```

Propose: In the propose phase, the proposer for slot t includes in the PROPOSE message that it sends also the greatest justified checkpoint according to its view at the time of proposing.

Vote: The validity condition of a [PROPOSE, $\mathsf{ch}_p, \mathsf{ch}^C, Q^C, \mathcal{GJ}_p, t, v_p$] message received by an honest validator v_i is extended to also check that $\mathcal{GJ}_p$ is indeed a justified checkpoint according to the v_i's current view. To give enough time for FFG-VOTEs that justify $\mathcal{GJ}_p$ to be received by v_i, v_i postpones to time $\mathsf{vote}(t)$ acting upon any PROPOSE messages received in the interval $[4\Delta t, 4\Delta t + \Delta]$. Then, only if at that time the validity condition passes and $\mathcal{GJ}_p$ is greater than v_i's frozen latest greatest justified checkpoint $\mathcal{GJ}_i^{\text{frozen}}$, then v_i updates its local variables $\mathcal{GJ}^{\text{frozen}}$ and $\mathsf{ch}^{\text{frozen}}$ according with the received proposal always ensuring that $\mathsf{ch}_i^{\text{frozen}} \succeq \mathcal{GJ}_i^{\text{frozen}}$.

In Algorithm 4, honest validators also include an FFG-VOTE in the VOTE messages that they send. Specifically, the source checkpoint of such FFG-VOTE sent corresponds to $\mathcal{GJ}_i^{\text{frozen}}$. The target checkpoint is determined by selecting the highest chain among the previous available chain, the κ-deep prefix of the chain output by the fork-choice function MFC, and $\mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}$, filtering out any chain if it is not a prefix of the chain output by the fork-choice function MFC.

Validator v_i also sets the available chain output, chAva_i to such chain. So, compared to the confirmed chain in Algorithm 3, when setting chAva_i, Algorithm 4 weighs the previous available chain also against $\mathcal{GJ}_i^{\text{frozen}}.\mathsf{ch}$[20].

Fast Confirm: Validator v_i checks whether chAva_i is a prefix of the chain $\text{fast}^{\text{cand}}$ output by $\mathtt{fastconfirm}(\mathcal{V}_i, t)$ or it conflicts with it. In either case, v_i updates chAva_i to $\text{fast}^{\text{cand}}$[21].

Merge: At round $4\Delta t + 3\Delta$, every validator v_i also updates $\mathcal{GJ}_i^{\text{frozen}}$ to the greatest justified checkpoint according to its view at that time.

5.4 Faster finality protocol analysis

Algorithm 4 works in the generalized partially synchronous sleepy model, and is in particular a η-secure ebb-and-flow protocol. In Section 4.1 we have already shown that the the finalized chain chFin is $\frac{n}{3}$-accountable, and thus always safe if $f < \frac{n}{3}$. For $\mathsf{GST} = 0$, we show in Section 5.4.1 that, if the execution is η-compliant in this stronger sense, then all the properties of Algorithm 3, i.e., η Dynamic Availability and η Reorg Resilience, keep holding. Finally, in Section 5.4.2 we show that, after $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$, Algorithm 4 guarantees the required conditions listed in Property C to ensure that the finalized chain is live.

5.4.1 Synchrony

In this section, we prove that chain chAva of Algorithm 4 is η-dynamically-available, η-reorg-resilient, η-asynchrony-reorg-resilient and η-asynchrony-safety-resilient. 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. At a high level, our proof strategy is to show that, under these assumptions, the integration of the FFG component into Algorithm 3 does not affect the protocol behavior in any way that could compromise any of the properties already proven for Algorithm 3.

To do so, we take any execution of Algorithm 4 and show that there exists an adversary that induces an execution of Algorithm 3 where (i) the messages sent by any honest validator in the two executions match except only for the FFG component of messages, (ii) the MFC fork-choice outputs of any honest validator in the two executions match and (iii) any available chain output by Algorithm 4 is also a confirmed chain of an honest validator in the execution of Algorithm 3. This then allows us to show that the result of the various Lemmas and Theorems presented in Section 5.2.1 for Algorithm 3 also hold for Algorithm 4.

We start by formalizing the concept of validators sending the same messages except only for their FFG component. Moreover, we define the equivalence between an execution in Algorithm 3 and one in Algorithm 4. These definitions will be leveraged upon in the subsequent Lemmas and Theorems.

Definition 10. We say that two messages are *dynamically-equivalent* if and only if they differ in at most their FFG-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 FFG-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.

[20]While, this change is not strictly required for correctness, under some conditions it ends up producing a longer available chain which is better from a responsiveness point of view. For example, this is the case if we have a slot t' where the proposer is honest, $\frac{2}{3}$ of the validators are active and honest up to the voting phase of slot $t' + 1$, but then some of these are corrupted immediately after, and v_i is asleep in both slots and it becomes active in slot $t' + 2$ only. In such a scenario, with the modification discussed in this paragraph, in the voting phase of slot $t' + 2$, v_i sets its available chain to to the chain PROPOSEd by the proposer of slot t'. Without such modification, v_i's would not update its available chain during the voting phase of slot $t' + 2$.

[21]It is possible to design an algorithm where, if chAva_i conflicts with $\mathsf{fast}^{\text{cand}}$, then it is left to the specific implementation decide whether to update chAva_i to $\mathsf{fast}^{\text{cand}}$ or leave it unchanged. However, to keep the algorithm straightforward, we chose not to incorporate this additional decision layer.

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 the difference in the fast confirmation function employed by the two algorithms, we do need to explicitly specify the meaning of $\overset{e}{\mathrm{MFC}}{}_i^{\mathsf{propose}(t)}$ with t being any slot. If e is an execution of Algorithm 3, then $\overset{e}{\mathrm{MFC}}{}_i^{\mathsf{propose}(t)}$ corresponds the definition provided in Section 5.2.1. If e is an execution of Algorithm 4, then $\overset{e}{\mathrm{MFC}}{}_i^{\mathsf{propose}(t)} := \overset{e}{\mathrm{MFC}}(\mathcal{V}_i^{\mathsf{propose}(t)}, \mathcal{V}_i^{\mathsf{propose}(t)}, \mathsf{ch}^C, t)$ with $(\mathsf{ch}^C, \cdot) = \mathtt{fastconfirm}(\mathcal{V}_i^{\mathsf{propose}(t)}, t-1)$.

Definition 11. Let e_{FFG} by an η-compliant execution of Algorithm 4 and $e_{\text{NO-FFG}}$ be an η-compliant execution of Algorithm 3. We say that e_{FFG} and $e_{\text{NO-FFG}}$ are MFC-*equivalent* if and only if the following constraints hold:

1. e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent up to any round
2. for any slot t_i,

   2.1 $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{propose}(t_i)} = \overset{e_{\text{FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{propose}(t_i)}$

   2.2 $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{vote}(t_i)} = \overset{e_{\text{FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{vote}(t_i)}$

   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_{FFG}, and chain Ch_i is in $e_{\text{NO-FFG}}$.

Lemma 13. *Let e_{FFG} by any η-compliant execution of Algorithm 4. There exists an η-compliant execution $e_{\text{NO-FFG}}$ of Algorithm 3 such that e_{FFG} and $e_{\text{NO-FFG}}$ are* MFC-*equivalent.*

Proof. Let e_{FFG} be any η-compliant execution of Algorithm 4 and let $A^{e_{\text{FFG}}}$ be the adversary in such an execution. Below, we specify the decisions made by an adversary $A^{e_{\text{NO-FFG}}}$ in an execution $e_{\text{NO-FFG}}$ of Algorithm 3. Later we show that this leads to $e_{\text{NO-FFG}}$ satisfying all the conditions in the Lemma.

(i) For any round r, the set of validators corrupted by $A^{e_{\text{NO-FFG}}}$ in round r corresponds to set of validators corrupted by $A^{e_{\text{FFG}}}$ in the same round r.

(ii) For any round r, the set of validators put to sleep by $A^{e_{\text{NO-FFG}}}$ in round r corresponds to set of validators put to sleep by $A^{e_{\text{FFG}}}$ in the same round r.

(iii) For any slot t, if an honest validator is the proposer for slot t in e_{FFG}, then it is also the proposer for slot t in $e_{\text{NO-FFG}}$.

(iv) For any round r, if e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent up to round r, then $A^{e_{\text{NO-FFG}}}$ 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_{FFG} is dynamically-equivalent to the set of messages received by v_i in execution $e_{\text{NO-FFG}}$.

Now, we prove by induction on t_i that the execution $e_{\text{NO-FFG}}$ induced by $A^{e_{\text{NO-FFG}}}$ satisfies all Definition 11's conditions. To do so, we add the following conditions to the inductively hypothesis.

2.5 For any $\mathcal{J}$ such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)})$, $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{propose}(t_i)} \succeq \mathcal{J}.\mathsf{ch}$.

2.6 For any $\mathcal{J}$ such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)})$, $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{vote}(t_i)} \succeq \mathcal{J}.\mathsf{ch}$.

and rephrase Condition 1 as follows

2.7 e_{FFG} and $e_{\text{NO-FFG}}$ 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.

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 $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)})$. Because the chain of the target checkpoint of an FFG-VOTE cast by an honest validator v_ℓ in round r_ℓ corresponds to $(\overset{e_{\text{FFG}}}{\mathsf{chAva}}{}_\ell^{r_\ell}.\mathsf{ch}, \cdot)$ 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 $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{J}.\mathsf{ch}$. 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 $\mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)}$. Given that $t_m \leq t_k < t_i$, from Lemma 7, we know that $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}^{\mathsf{propose}(t_i)} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{J}.\mathsf{ch}$ and that $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}^{\mathsf{vote}(t_i)} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathcal{J}.\mathsf{ch}$. Given that the set of justified checkpoints in $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}$ is a superset of the set of justified checkpoints in $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$, both Condition 2.5 and Condition 2.6 are proven.

Conditions 2.1 and 2.2. We know that e_{FFG} and $e_{\text{NO-FFG}}$ 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 $A^{e_{\text{NO-FFG}}}$'s set of decisions, $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$ and $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$ are dynamically-equivalent. From this follows that e_{FFG} and $e_{\text{NO-FFG}}$ are also honest-output-dynamically-equivalent up to round $\mathsf{vote}(t_i)$ and, therefore, $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}$ and $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}$ 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 22, $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} \in \{\mathsf{chAva}_i^{\mathsf{fconf}(t_i-1)}, \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch}, (\overset{e_{\text{FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{vote}(t_i)})^{\lceil \kappa}\}$. 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 34, 38 and 39, $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},t_i}, \mathcal{V}^{\mathsf{vote}(t_i)})$. 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 $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$. 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: $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = (\overset{e_{\text{FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{vote}(t_i)})^{\lceil \kappa}$**.** Line 22 of Algorithm 3 implies that $\mathsf{Ch}_i^{\mathsf{vote}(t_i)} \succeq \overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_i^{\mathsf{vote}(t_i)}$. 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 27 to 30, $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} \in \{\mathsf{chAva}_i^{\mathsf{vote}(t_i)}, \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).\mathsf{ch}, \mathsf{ch}^C\}$ with $(\mathsf{ch}^C, Q) = \texttt{fastconfirm}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}, t_i) \land Q \neq \emptyset$. 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: $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).\mathsf{ch}$. 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** $(\mathsf{ch}^C, Q) = \texttt{fastconfirm}(\overset{e_{\mathsf{ffg}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}, t_i) \land Q \neq \emptyset$**.** This implies that $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}$ includes a quorum of VOTE message for chain ch^C. Given Condition 2.2, $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}$ 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_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent up to round $\mathsf{propose}(t_i)$. This, Conditions 2.1 and 2.2 and item (iv) of $A^{e_{\text{NO-FFG}}}$'s set of decisions clearly imply Condition 2.7 up to round $\mathsf{propose}(t_i+1)$. $\square$

In the following Lemmas and Theorems, unless specified, we refer to executions of Algorithm 4.

Lemma 14 (Analogous of Lemma 4)**.** *If, in slot t, all validators in $H_{\mathsf{vote}(t)}$ cast* VOTE *messages for chains extending chain ch, then, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $\mathrm{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $\mathrm{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ cast* VOTE *messages for chains extending ch.*

Proof. Take any η-compliant execution e_{FFG} of Algorithm 4 where in slot t, all validators in $H_{\mathsf{vote}(t)}$ cast VOTE messages for chains extending chain ch. Let $e_{\text{NO-FFG}}$ be an MFC-equivalent execution of Algorithm 3 which Lemma 13 proves to exist. It is clear from the definition of MFC-equivalent honest-output-dynamically-equivalent executions that in slot t of execution $e_{\text{NO-FFG}}$ all validators in $H_{\mathsf{vote}(t)}$ cast VOTE messages for chains extending chain ch as well. Hence, by Lemma 4, in execution $e_{\text{NO-FFG}}$, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $\mathrm{MFC}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $\mathrm{MFC}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$. From Conditions 2.1 and 2.2 of Lemma 13 then it follows that this holds for execution e_{FFG} as well, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ cast VOTE messages for chains extending ch. $\square$

Lemma 15 (Analogous of Lemma 6)**.** *Let t be a slot with an honest proposer v_p and assume that v_p casts 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 $H_{\mathsf{vote}(t')}$ cast a* VOTE *message for a chain extending ch_p. Additionally, for any slot $t'' > t$ and any validator $v_i \in H_{\mathsf{vote}(t'')}$, $\mathrm{MFC}_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ and $\mathrm{MFC}_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$.*

Proof. Take any η-compliant execution e_{FFG} of Algorithm 4 where v_p, the proposer of slot t, is honest and casts a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C t, v_p]$ message. Let $e_{\text{NO-FFG}}$ be an MFC-equivalent execution of Algorithm 3 which Lemma 13 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 $e_{\text{NO-FFG}}$. Hence, by Lemma 6, in execution $e_{\text{NO-FFG}}$, for any slot $t' \geq t$, all validators in $H_{\mathsf{vote}(t')}$ cast a VOTE message for a chain extending ch_p, and, for any slot $t'' > t$ and any validator $v_i \in H_{\mathsf{vote}(t'')}$, $\mathrm{MFC}_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ and $\mathrm{MFC}_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$. From Conditions 1, 2.1 and 2.2 of Lemma 13 then it follows that this holds for execution e_{FFG} as well. $\square$

Lemma 16 (Analogous of Lemma 7)**.** *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 $v_j \in H_{r_j}$, $\mathsf{chAva}_i^{r_i} \preceq \mathsf{ch}_j^{r_j}$.*

Proof. Take any η-compliant execution e_{FFG} of Algorithm 4. Let $e_{\text{NO-FFG}}$ be an equivalent execution of Algorithm 3 which Lemma 13 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 $r'_i \in \{\mathsf{vote}(\mathrm{slot}(r'_i)), \mathsf{fconf}(\mathrm{slot}(r'_i))\}$. From Conditions 2.3 and 2.4 of Lemma 13, 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 7 implies that $\overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_j^{r_j} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)}$. From Lemma 13, we know that $\overset{e_{\text{FFG}}}{\mathrm{MFC}}{}_j^{r_j} = \overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_j^{r_j}$. Hence, $\overset{e_{\text{FFG}}}{\mathrm{MFC}}{}_j^{r_j} = \overset{e_{\text{NO-FFG}}}{\mathrm{MFC}}{}_j^{r_j} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_i^{r'_i} = \mathsf{chAva}_i^{r_i}$. $\square$

Theorem 7 (Reorg Resilience - Analogous of Theorem 3)**.** *Algorithm 4 is η-reorg-resilient.*

Proof. The proof follows the one for Theorem 3 with the following changes only.

1. Replace Ch with chAva,
2. Replace Lemma 6 with Lemma 15, and
3. Replace Lemma 7 with Lemma 16. $\square$

Theorem 8 (η-dynamic-availability - Analogous of Theorem 4)**.** *Algorithm 4 is η-dynamically-available.*

Proof. Note that Lemma 8 holds for Algorithm 4 as well. Then, the proof follows the one for Theorem 4 with the following changes only.

1. Replace Ch with chAva,
2. Replace the reference to Line 22 of Algorithm 3 with Line 22 of Algorithm 4,
3. Replace Lemma 6 with Lemma 15,
4. Replace Lemma 7 by Lemma 16, 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 30. Because the chain of the target checkpoint of an FFG-VOTE cast by an honest validator v_ℓ in round r_ℓ corresponds to $(\overset{e_{\text{FFG}}}{\mathsf{chAva}}{}_\ell^{r_\ell}.\mathsf{ch}, \cdot)$ 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 16 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 30 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 17 (Analogous of Lemma 9)**.** *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 9 with the only change being replacing Ch_i with chAva_i. □

Asynchrony Resilience. We now move to proving that Algorithm 4 also satisfies Asynchrony Resilience. However, compared to Algorithm 3, due to the FFG component, to prove Asynchrony Resilience for Algorithm 4 we also need to require that also Constraint (4) holds. The reason for this is that we want to avoid a situation where FFG-VOTEs sent by validators in $H_{\mathsf{vote}(t_a+1),\mathsf{vote}(t_a+\pi+1)} \setminus H_{\mathsf{vote}(t_a)}$ during the asynchronous period, together with FFG-VOTEs for a slot in $[t_a+1, t_a+\pi]$ but that are sent during a later slot $t_k > t_a+\pi+2$ by validators that are adversarial in t_k but that are honest in $t_a+\pi+2$ and are in $H_{\mathsf{vote}(t_a)} \setminus A_{\mathsf{vote}(t_a+\pi+2)}$ could justify a checkpoint conflicting with either the chain PROPOSEd or the chain confirmed by an honest validator at or before slot t_a.

Lemma 18 (Analogous of Lemma 11)**.** *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending ch. Then, for any slot $t_i \geq t$, any validator in $W_{\mathsf{vote}(t_i)}$ casts a* VOTE *message for a chain extending ch.*

Proof. We proceed by induction on t_i and add the following condition to the inductive hypothesis.

1. For any slot t', let $\mathcal{X}^{t',\mathsf{ch}}$ be the set of validators v_i in $H_{\mathsf{vote}(t')}$ such that $\mathsf{chAva}_i^{\mathsf{vote}(t')}$ conflicts with ch. Then, for any $t'' \leq t_i$, $\left|X^{t'',\mathsf{ch}} \cup A_\infty\right| < \frac{2}{3}n$.

Base Case: $t_i \in [t, t_a]$**.** From Lemma 14 which, given that we assume $f < \frac{n}{3}$, also implies Condition 1.

Inductive Step: $t_i > t_a$**.** We assume that the Lemma holds for any slot $t_i - 1$ and prove that it holds also for slot t_i. Note that Condition 1 holding for slot $t_i - 1$ means that for any validator $v_i \in H_{\mathsf{vote}(t_i)}$ and justified checkpoint $\mathcal{J}$ according to $\mathcal{V}_i^{\mathsf{vote}(t_i)}$, $\mathcal{J}$ does not conflict with ch. Let us proceed by cases and keep in mind that due to Line 22, if in slot t_i a validator $v_i \in H_{\mathsf{vote}(t_i)}$ casts a VOTE message for a chain extending ch, then $\mathsf{chAva}_i^{\mathsf{vote}(t_i)}$ does not conflict with ch.

Case 1: $t_i \in [t_a+1, t_a+\pi+1]$**.** It should be easy to see that, assuming that Condition 1 holds for slot $t_i - 1$, Lemma 10 can be applied to Algorithm 4 as well. This implies that all validators in $W_{\mathsf{vote}(t_i)}$ cast VOTE messages for chains extending ch. Given Constraint (4), then Condition 1 holds for slot t_i as well.

Case 2: $t_i = t_a + \pi + 2$. Given that we assume that Condition 1 holds for slot $t_i - 1$, it should be easy to see that we can apply here the same reasoning used for this same case in Lemma 11. Then, given that we assume $f < \frac{n}{3}$, Condition 1 holds for slot t_i as well.

Case 3: $t_a \geq t_a + \pi + 3$. It should be easy to see that, assuming that Condition 1 holds for slot $t_i - 1$, Lemma 4 can be applied to Algorithm 4 as well. This implies that all validators in $H_{\mathsf{vote}(t)}$ casts VOTE messages for chains extending ch. Then the proof proceeds as per the same case in Lemma 11. Also, given that we assume $f < \frac{n}{3}$, Condition 1 holds for slot t_i as well. □

Lemma 19 (Analogous of Lemma 12)**.** *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending* ch*. Then, for any round $r_i \geq \mathsf{vote}(t)$ and validator $v_i \in W_{r_i}$, $\mathsf{Ch}_i^{r_i}$ does not conflict with* ch*.*

Proof. We can follow the same reasoning used in the proof of Lemma 12 with the following changes only.

1. Replace Ch with chAva,
2. Replace Lemma 11 with Lemma 18,
3. In the proof of Case 1, refer to lines Lines 22, 25 and 26 of Algorithm 4 rather than Lines 22 to 23 of Algorithm 3.
4. Consider the following additional sub case for Case 2.

   Case 2.3 $\mathsf{chAva}_i^{r_i} = \mathcal{GJ}(\mathcal{V}_i^{\mathsf{fconf}(\mathrm{slot}(r_i))}).\mathsf{ch}$. Because the chain of the target checkpoint of an FFG-VOTE cast by an honest validator v_ℓ in round r_ℓ corresponds to $(\overset{e_{\text{FFG}}}{\mathsf{chAva}_\ell^{r_\ell}}.\mathsf{ch}, \cdot)$ and we assume $f < \frac{n}{3}$, that there exists a slot $t_k \in [0, \mathrm{slot}(r_i)]$ and validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathsf{chAva}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_i^{\mathsf{fconf}(\mathrm{slot}(r_i))}$. From Case 1 (see Lemma 12 for actual the proof of Case1) we have that $\mathsf{ch} \preceq \mathsf{chAva}_k^{\mathsf{vote}(t_k)} \vee \mathsf{ch} \succeq \mathsf{chAva}_k^{\mathsf{vote}(t_k)}$. In either case, ch does not conflict with $\mathsf{chAva}_i^{\mathsf{fconf}(\mathrm{slot}(r_i))}$. □

Theorem 9 (Asynchrony Reorg Resilience - Analogous of Theorem 5)**.** *Algorithm 4 is η-asynchrony-reorg-resilient.*

Proof. We can follow the same reasoning used in the proof of Theorem 5 with the following changes only.

1. Replace Theorem 3 with Theorem 7,
2. Replace Lemma 6 with Lemma 15, and
3. Replace Lemma 12 with Lemma 19. □

Theorem 10 (Asynchrony Safety Resilience - Analogous of Theorem 6)**.** *Algorithm 4 η-asynchrony-safety-resilient.*

Proof. We can follow the same reasoning used in the proof of Theorem 6 with the following changes only.

1. Replace Ch with chAva,
2. Replace Theorem 4 with Theorem 8, and
3. Replace Lemma 12 with Lemma 19. □

5.4.2 Partial synchrony

In this section, we show that Algorithm 4 ensures Constraints A to C and hence ensures that the chain chFin is always Accountably Safe and is live after time $\max(\mathsf{GST}, \mathsf{GAT}) + \Delta$, meaning that Algorithm 4 is an η-secure ebb-and-flow protocol under the assumption that $f < \frac{n}{3}$.

Lemma 20. *Algorithm 4 satisfies Constraint A.*

Proof. Let us prove that Algorithm 4 ensures all the conditions listed in Constraint A.

Constraint A.1. By Line 26, an honest active validator sends only one FFG-VOTE in any slot, and therefore, Constraint A.1 holds.

Constraint A.2. Direct consequence of Line 24.

Constraint A.3. Take any two slots t and t' with $t < t'$. Let $\mathcal{S}_k$ be the source checkpoint of the FFG-VOTE that validator v_i sent in any slot k. Now, we prove by induction that $\mathcal{S}_{\mathsf{vote}(t)} \leq \mathcal{S}_{t'}$. To do so, take any two slots r and r' such that $t \leq r < r' \leq t'$, $v_i \in H_r$ and r' is the first slot after r such that $v_i \in H_{r'}$. By Line 26, we know that $S_r = \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(r)}$ and by Lines 34 and 38, we have that $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(r)}, \mathcal{V}_i^{\mathsf{vote}(r)})$. Then, note that $r \leq r'-1$ and that because of the joining protocol, we know that v_i is awake at round $\mathsf{merge}(r'-1)$. Hence, we have that $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(r)} \leq \mathcal{GJ}(\mathcal{V}_i^{\mathsf{merge}(r'-1)}) = \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(r'-1)}$. Finally, by Lines 38 and 39, $\mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(r'-1)} \leq \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(r')} = \mathcal{S}_{r'}$ showing that $\mathcal{S}_r \leq \mathcal{S}_{r'}$. ☐

The proof of Constraint B is trivial so we skip it for now. Before being able to prove that Algorithm 4 satisfies Constraint C, we need to show that the base case of Lemma 6 holds even when $\mathsf{GST} > 0$.

Lemma 21. *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 casts a $[\text{PROPOSE}, \mathsf{ch}_p, \mathsf{ch}_p^C, Q_p^C, \mathcal{GJ}_p, t, v_p]$ message. Then, for any validator $v_i \in H_{\mathsf{vote}(t)}$, v_i casts 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 case.

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 42 and 43 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 40 to 43 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 38 to 43 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 [8], 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 casts a VOTE message for ch_p.
If no such a chain exists, v_i still casts 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 38 and 39, $\mathcal{GJ}^{\text{frozen},\mathsf{vote}(t)} = \mathcal{GJ}_p$. ☐

Lemma 22. *Algorithm 4 satisfies Constraint C.*

Proof. Let us prove that Algorithm 4 ensures each condition listed in Constraint C.

Constraint C.1.1. Let $[\text{VOTE}, \mathsf{ch}, \mathcal{S} \to \mathcal{T}, t, v_i]$ be the VOTE message cast by validator v_i in slot t. We need to show that $\mathcal{S}.\mathsf{ch} \preceq \mathcal{T}.\mathsf{ch} \preceq \mathsf{ch}$.

Lines 22 and 24 to 26 imply that $\mathcal{T}.\mathsf{ch} \preceq \mathrm{MFC}_i^{\mathsf{vote}(t)}$ and $\mathrm{MFC}_i^{\mathsf{vote}(t)} \preceq \mathsf{ch}$, hence $\mathcal{T}.\mathsf{ch} \preceq \mathsf{ch}$.

Then, we are left with showing that $\mathcal{S}.\mathsf{ch} \preceq \mathcal{T}.\mathsf{ch}$. Due to Lines 24 and 26, this amounts to proving that $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t)}.\mathsf{ch} \preceq \mathsf{chAva}_i^{\mathsf{vote}(t)}$. First, note that Algorithm 4 ensures that $\mathsf{ch}_i^{\text{frozen},4\Delta t} \succeq \mathcal{GJ}_i^{\text{frozen},4\Delta t}$. This can be proven by induction on slot t'. For $t' = 0$, the base case, this is obvious from the initialization code. For $t' > 0$, assume that the statement holds for t'. Given that $\texttt{fastconfirm}(\mathcal{V}, \cdot)$ always outputs a chain descendant of $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$, Lines 34 and 35 ensure that $\mathsf{ch}_i^{\text{frozen},4\Delta t'+3\Delta} \succeq \mathcal{GJ}_i^{\text{frozen},\mathsf{merge}(t')}.\mathsf{ch}$. Then, Lines 39 to 41 ensure that this relation is maintained for any round until $\mathsf{vote}(t'+1)$.

Hence, $\mathrm{MFC}(\cdot, \cdot, \mathsf{ch}^{\text{frozen},\mathsf{vote}(t)}, \cdot) \succeq \mathcal{GJ}_i^{\text{frozen},4\Delta t+\Delta}.\mathsf{ch}$. Therefore, Line 22 ensures that $\mathsf{chAva}_i^{\mathsf{vote}(t)} \succeq \mathcal{GJ}(\mathcal{V}_i^{\text{frozen},4\Delta t+\Delta}).\mathsf{ch}$.

Constraint C.1.2. Due to Lines 35 and 38, there exists a set $\mathcal{V}_i^{\mathsf{FFGvote},t} \supseteq \mathcal{V}_i^{\mathsf{merge}(t-1)}$ such that $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t)} = \mathcal{GJ}(\mathcal{V}_i^{\mathsf{FFGvote},t})$.

Constraint C.1.3. Line 26 implies that any FFG-VOTE $\mathcal{S} \to \mathcal{T}$ sent by an honest validator during slot t is such that $\mathcal{T}_i.c = t$.

Constraint C.2.1. Follows from Lemma 8.

Constraint C.2.2. Let the greatest justified checkpoint in the view of v_p at round $\mathsf{propose}(t)$ be $\mathcal{GJ}_p$ and v_i be any always-honest validator. By Lemma 21, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t)} = \mathcal{GJ}_p$. Hence, by Line 26, at time $\mathsf{vote}(t)$, validator v_i sends an FFG-VOTE $\mathcal{S} \to \mathcal{T}_i$ with the same source checkpoint, namely $\mathcal{S} = \mathcal{GJ}_p$. Finally, as argued when proving Constraint C.1.1, the chain that v_i VOTEs for is a descendant of $\mathcal{T}_i.\mathsf{ch}$. Then, from Lemma 21 and Line 22, we can conclude that $\mathcal{T}_i.\mathsf{ch} \preceq \mathsf{ch}_p$.

Constraint C.2.3.1. Given that $\mathsf{vote}(t) \geq \mathsf{GST}$, and that FFG-VOTEs for slot t are sent at round $\mathsf{vote}(t)$, they are received by any always-honest validator v_i at time $\mathsf{merge}(t)$ and, hence, included in $\mathcal{V}_i^{\text{frozen},\mathsf{merge}(t)}$. From Constraint C.1.2, we then have that $\mathcal{V}_i^{\text{frozen},\mathsf{merge}(t)} \subseteq \mathcal{V}_i^{\mathsf{FFGvote},t+1}$.

Constraint C.2.3.2. Let v_i be any always-honest validator. By Lemma 21, at time $\mathsf{vote}(t)$, every always-honest validator casts a VOTE message for chain ch_p. Hence, given that $\mathsf{fconf}(t) \geq \mathsf{GAT}$ and $f < \frac{n}{3}$, for any always-honest validator, $\texttt{fastconfirm}(\mathcal{V}_i^{\mathsf{fconf}(t)}, t) = (\mathsf{ch}_p, \cdot)$. Given that, as established in the proof of Constraint C.2.2, $\mathcal{T}_i.\mathsf{ch} = \mathsf{chAva}_i^{\mathsf{vote}(t)} \preceq \mathsf{ch}_p$, $\mathsf{chAva}_i^{\mathsf{fconf}(t)}$ is set to ch_p (Line 30). By Constraint C.2.2, at time $\mathsf{vote}(t)$, v_i sends an FFG-VOTE with the same source checkpoint and with a target checkpoint $\mathcal{T}_i$ such that $\mathcal{T}_i.\mathsf{ch} \preceq \mathsf{ch}_p$. Hence, given $f < \frac{n}{3}$ and Constraint C.1.3, at time $\mathsf{vote}(t+1)$, $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t+1)}).c = t \land \mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t+1)}).\mathsf{ch} \preceq \mathsf{ch}_p$. Given that $\kappa > 1$, $t + 1 - \kappa < t$. Therefore, given $\mathsf{chAva}_i^{\mathsf{fconf}(t)} = \mathsf{ch}_p$ and $\mathsf{ch}_p.p = t$, in round $\mathsf{vote}(t+1)$, due to Line 22, chAva_i is unchanged, *i.e.*, $\mathsf{chAva}_i = \mathsf{ch}_p$. Then, due to Lines 24 and 26, the FFG-VOTE sent by v_i in slot $t+1$ has a target $\mathcal{T}$ such that $\mathcal{T}.\mathsf{ch} = \mathsf{ch}_p$.

Constraint C.2.4.1. Same reasoning as per Constraint C.2.3.1.

Constraint C.2.4.2. Condition (i) follows from the synchrony assumption. As shown in the proof of Constraint C.2.3.2, for any always-honest validator v_i, $\mathsf{chAva}_i^{\mathsf{vote}(t+1)} = \mathsf{ch}_p$. By Lines 21, 22, 25 and 26, this implies that v_i VOTEs for a chain extending ch_p. Then, by following the same argument outlined in the proof of Constraint C.2.3.2 we can conclude the following. At time $\mathsf{fconf}(t+1)$, $\mathsf{chAva}_i^{\mathsf{fconf}(t+1)} \succeq \mathsf{ch}_p$. At time $H_{\mathsf{vote}(t+2)}$, $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t+2)}).c = t+1 \land \mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t+2)}).\mathsf{ch} = \mathsf{ch}_p$. Line 22 ensures then that $\mathsf{chAva}_i^{\mathsf{vote}(t+2)} \succeq \mathsf{ch}_p$ and, hence, by Line 22, any always-honest validator VOTEs for a chain extending ch_p. Hence, $\mathsf{chAva}_i^{\mathsf{fconf}(t+2)} \succeq \mathsf{ch}_p$. This and Line 31 prove condition (ii). □

Theorem 11. *Algorithm 4 is a η-secure ebb-and-flow protocol.*

Proof. Lines 3, 23 and 31 and the definition of $\mathcal{GF}$ prove that Algorithm 4 satisfies Constraint B. Safety of chAva and chFin, and Liveness of chAva follow from this, Theorems 1 and 8, and Lemma 20.

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 $t_p \in [t, t+\kappa)$ hosted by an honest proposer (Lemma 2 [10]). By Theorem 2 and Lemma 22, $\mathit{txpool}^{\mathsf{propose}(t)} \subseteq \mathit{txpool}^{\mathsf{propose}(t_p)} \subseteq \mathsf{chFin}_i^{\mathsf{fconf}(t_p+2)}$ for any validator honest in $\mathsf{fconf}(t_p+2)$ and, clearly, $\mathsf{fconf}(t_p+2) - \mathsf{propose}(t) \in O(\kappa)$.

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 23 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 28 to 30 ensure that $\mathcal{GJ}(\mathcal{V}_i^r).\mathsf{ch} \preceq \mathsf{chAva}_i^r$. By Algorithm 1, $\mathcal{GJ}(\mathcal{V}_i^r) \geq \mathcal{GF}(\mathcal{V}_i^r)$ which, due to Lemma 1 and the assumption that $f < \frac{n}{3}$, implies $\mathsf{chFin} \preceq \mathcal{GJ}(\mathcal{V}_i^r).\mathsf{ch} \preceq \mathsf{chAva}_i^r$. □

We conclude this section by showing that the finalized chain of an honest validator grows monotonically. Although this property is not explicitly defined in Section 2.3, it is frequently desired by Ethereum protocol implementers [1].

Lemma 23. *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 $\mathsf{chAva}_i^{r'} \succeq \mathcal{J}.\mathsf{ch} \land \mathsf{J}(\mathcal{J}, \mathcal{V}_i^{r'}) \land \mathcal{J} \geq \mathcal{GF}(\mathcal{V}_i^r)$. 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 34, 35, 40 and 41, $\mathsf{ch}_i^{\text{frozen},r'} \succeq \mathcal{GJ}_i^{\text{frozen},r'}.\mathsf{ch}$. Then, Line 22 implies that $\mathsf{chAva}_i^{r'} \succeq \mathcal{GJ}_i^{\text{frozen},r'}.\mathsf{ch}$. Also, Lines 35, 38 and 39 imply that $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},r'}, \mathcal{V}_i^{r'})$. Finally, Lines 35, 38 and 39, 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 28 to 30 ensure that $\mathsf{chAva}_i^{r'} \succeq \mathcal{GJ}(\mathcal{V}_i^{r'}).\mathsf{ch}$. Then, from Algorithm 1 and the fact that $\mathcal{V}_i^r \subseteq \mathcal{V}_i^{r'}$, we can conclude that $\mathcal{GJ}(\mathcal{V}_i^{r'}) \geq \mathcal{GF}(\mathcal{V}_i^{r'}) \geq \mathcal{GF}(\mathcal{V}_i^r)$.

From Algorithm 1 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 1 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. □

6 RLMD-GHOST-Based Faster Finality Protocol

In this section, we present a faster finality protocol that builds upon the dynamically-available consensus protocol of D'Amato and Zanolini [10], RLMD-GHOST. Specifically, RLMD-GHOST, which serves as the available component outputting chAva, is combined with the FFG component described in Section 4, which finalizes chains prefix of chAva and outputs chFin, achieving an η-secure ebb-and-flow protocol. We start by briefly recalling RLMD-GHOST [10] and state its most relevant properties.

6.1 Recalling RLMD-GHOST

Before introducing RLMD-GHOST with fast confirmation in Algorithm 6, we define a new fork-choice function in Algorithm 5, which will be used similarly to how the fork-choice function MFC is utilized in Algorithm 3.

The function RLMD-GHOST$(\mathcal{V}, B_{\text{start}}, t)$ in Algorithm 5 starts with refining the view $\mathcal{V}$ by calling the filter function $\mathsf{FIL}_{\text{rlmd}}$ that keeps only the latest, non-equivocating VOTE messages that fall within the expiry period $[t-\eta, t)$ for the current slot t. Formally, $\mathsf{FIL}_{\text{rlmd}}$ is defined as a composition of the filters already defined in Algorithm 2: $\mathsf{FIL}_{\text{rlmd}}(\mathcal{V}, t) = \mathsf{FIL}_{\text{lmd}}(\mathsf{FIL}_{\eta\text{-exp}}(\mathsf{FIL}_{eq}(\mathcal{V}), t))$. After this pruning, it proceeds as follows starting from B_{start}. From a given block, it selects the next block by identifying the child block that has the most weight, measured by the number of latest, unexpired, and non-equivocating VOTE messages for its descendants. One it reaches a block with no children, it returns the chain identified by this block. This is implemented as the function GHOST in Algorithm 5 . [22]

Algorithm 5 RLMD-GHOST fork-choice function

```
1: function RLMD-GHOST($\mathcal{V}, B_{\text{start}}, t$)
2:     return GHOST($\mathsf{FIL}_{\text{rlmd}}(\mathcal{V}, t), B_{\text{start}}, t$)

3: function GHOST($\mathcal{V}, B_{\text{start}}, t$)
4:     $B \leftarrow B_{\text{start}}$
5:     while $B$ has at least one child block $B'$ in $\mathcal{V}$ with $B'.p \leq t$ do
6:         $B \leftarrow \underset{B' \in \mathcal{V},\text{ child of } B}{\arg\max}\ w(B', \mathcal{V})$    // $w(B, \mathcal{V})$ outputs the number of votes in $\mathcal{V}$ for $B' \succeq B$ with $B'.p \leq t$
7:         // ties are broken according to a deterministic rule
8:     return $B$

9: function $\mathsf{FIL}_{\text{rlmd}}(\mathcal{V}, t)$
10:    return $\mathsf{FIL}_{\text{lmd}}(\mathsf{FIL}_{\eta\text{-exp}}(\mathsf{FIL}_{eq}(\mathcal{V}), t))$
```

Similarly to Algorithm 3 in Section 5, Algorithm 6 defines the function $\texttt{fastconfirmsimple}(\mathcal{V}, t)$ to returns a chain $\mathsf{fast}^{\text{cand}}$ that has garnered support from more than $\frac{2}{3}n$ of the validators that have VOTEd in slot t according to the view $\mathcal{V}$. Specifically, each validator v_i, via the function `fastconfirmsimple`, identifies the set $\mathsf{fast}^{\text{cands}}$ of all the chains ch from VOTE messages of slot t that meet a specific voting criterion, *i.e.*, when there are at least $\frac{2}{3}n$ validators sending VOTE messages of the form $[\text{VOTE}, \mathsf{ch}', \cdot, t, \cdot]$ in $\mathcal{V}_i$ with $\mathsf{ch} \preceq \mathsf{ch}'$, and then returns the maximum chain $\mathsf{fast}^{\text{cand}}$ in this set. If such chain does not exist, then it just returns B_{genesis}.

Also, similarly to Algorithm 3, each honest validator v_i maintains the following state variable:

- **Message Set $\mathcal{V}_i^{\text{frozen}}$**: Each validator stores in $\mathcal{V}_i^{\text{frozen}}$ a snapshot of its view $\mathcal{V}_i$ at time 3Δ in each slot, which is then updated between time 0 and time Δ of the next slot, if a PROPOSE message is received, by merging it with the view included in it.

However, compared to Algorithm 3, Algorithm 6 does not require to the state variable $\mathsf{ch}_i^{\text{frozen}}$.

Like Algorithm 3, each honest validator v_i outputs the following chain.

- **Confirmed Chain Ch_i:** The confirmed chain Ch_i is updated at time Δ in each slot and also potentially at time 2Δ.

Overall, Algorithm 6 proceeds following the structure of Algorithm 3.

Propose: At round $4\Delta t$, the propose phase occurs. During this phase, the proposer v_p for slot t broadcasts the message $[\text{PROPOSE}, \mathsf{ch}_p, \mathcal{V}_p \cup \{\mathsf{ch}_p\}, t, v_p]$. Here, ch_p is the chain returned by the function $\texttt{Extend}(\mathsf{ch}^{\mathsf{RLMD}}, t)$ where `Extend` is the same function utilized by Algorithm 3 and therefore ensures Property 1, and $\mathsf{ch}^{\mathsf{RLMD}}$ corresponds to the output of the RLMD-GHOST fork-choice RLMD-GHOST$(\mathcal{V}_i, B_{\text{genesis}}, t)$ after removing any block with slot higher or equal to t, *i.e.*, we take its 1-deep prefix[23]. Whereas, $\mathcal{V}_p \cup \{\mathsf{ch}_p\}$ denotes the view of v_p combined with the newly PROPOSEd chain ch_p.

[22]The RLMD-GHOST fork-choice function defined in [10] corresponds to the RLMD-GHOST fork-choice function of Algorithm 5 with $B_{\text{start}} = B_{\text{genesis}}$.

[23]The reason for taking the 1-deep prefix of the output of the RLMD-GHOST fork-choice is that, as per our protocol's definition, it is possible for a Byzantine validator to get the RLMD-GHOST fork-choice at propose time $\mathsf{propose}(t)$ to output a chain with height t by sending a block with slot t and relatives VOTEs for it. Alternatively, one could avoid taking the 1-deep prefix by requiring that blocks are signed by the proposer and we only add to the view blocks that are correctly singed by the expected proposer for their slot. We decided not to take this approach to keep the algorithms as compact as possible while maximizing the aspects of the protocols relevant to their properties that are captured by pseudo-code.

Vote: During the interval $[4\Delta t, 4\Delta t + \Delta]$, a validator v_i, upon receiving a proposal $[\textsc{propose}, \mathsf{ch}_p, \mathcal{V}_p, t', v_p]$ from the proposer v_p for slot $t' = t$, sets $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i^{\text{frozen}} \cup \mathcal{V}_p$.

During the voting phase, each validator v_i computes the fork-choice function RLMD-GHOST. The inputs to RLMD-GHOST are v_i's frozen view $\mathcal{V}_i^{\text{frozen}}$, the genesis block B_{genesis}, and the current slot t.

In the voting phase, validator v_i also sets the confirmed chain Ch_i to the highest chain among the κ-deep prefix of the chain output by RLMD-GHOST and the previous confirmed chain, filtering out this last chain if it is not a prefix of the chain output by RLMD-GHOST.

Finally, v_i casts a VOTE for the output of RLMD-GHOST.

Fast Confirm: Like in Algorithm 3, validator v_i checks if a chain ch can be fast confirmed, i.e., if at least $2/3n$ of the validators cast a VOTE message for a chain $\mathsf{ch}^{\text{cand}}$. If that is the case, it sets Ch_i to this chain.

Merge: Like in Algorithm 3, at round $4\Delta t + 3\Delta$, validator v_i updates its variable $\mathcal{V}_i^{\text{frozen}}$ to be used in the following slot.

Algorithm 6 RLMD-GHOST with fast confirmation – code for validator v_i

```
1: Output
2:     $\mathsf{Ch}_i \leftarrow B_{\text{genesis}}$: confirmed chain of validator $v_i$
3: State
4:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \{B_{\text{genesis}}\}$: frozen view of validator $v_i$
5: function fastconfirmsimple($\mathcal{V}, t$)
       let $\mathsf{fast}^{\text{cands}} := \{\mathsf{ch} \colon |\{v_j : \exists \mathsf{ch}' \succeq \mathsf{ch} : [\textsc{vote}, \mathsf{ch}', \cdot, t, v_j] \in \mathcal{V}_i\}| \geq \frac{2}{3}n\}$
6:     if $\mathsf{fast}^{\text{cands}} \neq \emptyset$ then
7:         return $\max(\mathsf{fast}^{\text{cands}})$
8:     else
9:         return $B_{\text{genesis}}$
   PROPOSE
10: at round $4\Delta t$ do
11:     if $v_i = v_p^t$ then
12:         let $\mathsf{ch}^{\mathsf{RLMD}} :=$ RLMD-GHOST$(\mathcal{V}_i, B_{\text{genesis}}, t)^{\lceil 1,t}$
13:         let $\mathsf{ch}_p := \mathsf{Extend}(\mathsf{ch}^{\mathsf{RLMD}}, t)$
14:         send message [PROPOSE, $\mathsf{ch}_p$, $\mathcal{V}_i \cup \{\mathsf{ch}_p\}$, $t$, $v_i$] through gossip
    VOTE
15: at round $4\Delta t + \Delta$ do
16:     let $\mathsf{ch}^{\mathsf{RLMD}} :=$ RLMD-GHOST$(\mathcal{V}_i^{\text{frozen}}, B_{\text{genesis}}, t)$
17:     $\mathsf{Ch}_i \leftarrow \max(\{\mathsf{ch} \in \{\mathsf{Ch}_i, (\mathsf{ch}^{\mathsf{RLMD}})^{\lceil \kappa}\} : \mathsf{ch} \preceq \mathsf{ch}^{\mathsf{RLMD}}\})$
18:     send message [VOTE, $\mathsf{ch}^{\mathsf{RLMD}}$, $\cdot$, $t$, $v_i$] through gossip
    FAST CONFIRM
19: at round $4\Delta t + 2\Delta$ do
20:     let $\mathsf{fast}^{\text{cand}} :=$ fastconfirmsimple$(\mathcal{V}_i, t)$
21:     if $\mathsf{fast}^{\text{cand}} \neq \bot$ then
22:         $\mathsf{Ch}_i \leftarrow \mathsf{fast}^{\text{cand}}$
    MERGE
23: at round $4\Delta t + 3\Delta$ do
24:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i$

25: upon receiving a gossiped message [PROPOSE, $\mathsf{ch}_p$, $\mathcal{V}_p$, $t'$, $v_p^t$] at any round in $[4\Delta t, 4\Delta t + \Delta]$ do
26:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i^{\text{frozen}} \cup \mathcal{V}_p$
```

6.1.1 Analysis

In this section, we show that Algorithm 6, like Algorithm 3, satisfied η Reorg Resilience (Theorem 12), η Dynamic Availability (Theorem 13), η Asynchronous Reorg Resilience (Theorem 14) and η Asynchronous

Safety Resilience (Theorem 15). The analysis presented in this section follows the same structure as the analysis of Algorithm 3 in Section 5.2.1. As we will see shortly, the analogous of Lemmas 4 to 6 are already proved in [10]. With these Lemmas, then we can prove that other Lemmas and Theorems by essentially recalling the proofs of the analogous Lemmas and Theorems in Section 5.2.1 with minor modifications. In the rest of this section, given any slot t, we let $\mathrm{RLMD}_i^{\mathsf{propose}(t)} := \text{RLMD-GHOST}(\mathcal{V}_i^{\mathsf{propose}(t)}, B_{\text{genesis}}, t)^{\lceil 1,t}$ and $\mathrm{RLMD}_i^{\mathsf{vote}(t)} := \text{RLMD-GHOST}(\mathcal{V}_i^{\mathsf{vote}(t)}, B_{\text{genesis}}, t)$.

Lemma 24. *If, in slot t, all validators in $H_{\mathsf{vote}(t)}$ cast* VOTE *messages for chains extending chain ch, then, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$, $\mathrm{RLMD}_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ and $\mathrm{RLMD}_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ cast* VOTE *messages for chains extending ch.*

Proof. See Lemma 4 of [10]. □

Lemma 25. *If an honest validator fast confirms a chain ch in slot t, then, for any slot $t' > t$ and validator $v_i \in H_{\mathsf{vote}(t')}$, $\mathrm{RLMD}_i^{\mathsf{propose}(t')} \succeq \mathsf{ch}$ and $\mathrm{RLMD}_i^{\mathsf{vote}(t')} \succeq \mathsf{ch}$, which implies that, all validators in $H_{\mathsf{vote}(t')}$ cast a* VOTE *message for a chain extending ch.*

Proof. The proof follows the one for Lemma 5 with the following changes only.

1. The base case follows from Lemma 5 of [10],
2. The inductive step follows from Lemma 24. □

Lemma 26. *Let t be a slot with an honest proposer v_p and assume that v_p casts a $[\text{PROPOSE}, \mathsf{ch}_p, \mathcal{V}_p, t, v_p]$ message. Then, for any slot $t' \geq t$, all validators in $H_{\mathsf{vote}(t')}$ cast a* VOTE *message for a chain extending ch_p. Additionally, for any slot $t'' > t$ and any validator $v_i \in H_{\mathsf{vote}(t'')}$, $\mathrm{RLMD}_i^{\mathsf{propose}(t'')} \succeq \mathsf{ch}_p$ and $\mathrm{RLMD}_i^{\mathsf{vote}(t'')} \succeq \mathsf{ch}_p$.*

Proof. The proof follows the one for Lemma 6 with the following changes only.

1. The base case follows from Lemma 1 of [10],
2. The inductive step follows from Lemma 24. □

Lemma 27. *Let r_i be any round and r_j be any round such that $r_j \geq r_i$ and $r_j \in \{\mathsf{propose}(\mathrm{slot}(r_j)), \mathsf{vote}(\mathrm{slot}(r_j))\}$. Then, for any validator v_i honest in round r_i and any validator $v_j \in H_{\mathrm{slot}(r_j)}$, $\mathsf{Ch}_i^{r_i} \preceq \mathsf{ch}_j^{r_j}$.*

Proof. The proof follows the one for Lemma 7 with the following changes only.

1. Replace MFC with RLMD,
2. Replace Line 22 of Algorithm 3 with Line 17 of Algorithm 6,
3. Replace Lemma 6 with Lemma 26, and
4. Replace Lemma 5 with Lemma 25. □

Theorem 12 (Reorg Resilience)**.** *Algorithm 6 is η-reorg-resilient.*

Proof. The proof follows the one for Theorem 3 with the following changes only.

1. Replace MFC with RLMD,
2. Replace Lemma 6 with Lemma 26, and
3. Replace Lemma 7 with Lemma 27. □

Lemma 28. *For any slot $t \geq 1$ and validator $v_i \in H_{\mathsf{propose}(t)}$, $\mathrm{RLMD}_i^{\mathsf{propose}(t)}.p < t$.*

Proof. Obvious by the definition of $\mathrm{RLMD}_i^{\mathsf{propose}(t)}$. □

Theorem 13. *Algorithm 6 is η-dynamically-available.*

Proof. The proof follows the one for Theorem 4 with the following changes only.

1. Replace MFC with RLMD,
2. Replace Algorithm 3 with Algorithm 6,
3. Replace Lemma 6 with Lemma 26,
4. Replace Lemma 7 with Lemma 27, and
5. Replace Lemma 8 with Lemma 28. □

Lemma 29. *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 9 with the only change being replacing Lemma 6 with Lemma 26. □

Asynchrony Resilience. Here, we demonstrate that Algorithm 6 also provides Asynchrony Reorg Resilience and Asynchrony Safety Resilience, as defined in Section 2.3.

Lemma 30. *Assume $\pi > 0$ and let t be any slot in $[t_a, t_a + \pi]$. Assume that in any slot $t' \in [t_a, t]$, all validators in $W_{\mathsf{vote}(t')}$ cast* VOTE *messages for chains extending chain ch. Then, in slot $t+1$, all validators in $W_{\mathsf{vote}(t+1)}$ also cast* VOTE *messages for chains extending chain ch.*

Proof. The proof follows the same reasoning of Lemma 10. Specifically, note that the condition $\left|\left(\mathcal{V}_i^{\text{frozen}}\right)^{\mathsf{ch},t+1} \cap \mathcal{V}_i^{\mathsf{ch},t+1}\right| > \frac{|\mathsf{S}(\mathcal{V}_i,t+1)|}{2}$ proved in the proof of Lemma 10 is stronger than the condition need for RLMD-GHOST$(\mathcal{V}_i, B_{\text{genesis}}, t+1)$ to output a chain extending ch. □

Lemma 31. *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending ch. Then, for any slot $t_i \geq t$, any validator in $W_{\mathsf{vote}(t_i)}$ casts a* VOTE *message for a chain extending ch.*

Proof. The proof follows the one for Lemma 11 with the following changes only.

1. Replace Lemma 4 with Lemma 24, and
2. Replace Lemma 10 with Lemma 30. □

Lemma 32. *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending ch. Then, for any round $r_i \geq \mathsf{vote}(t)$ and validator $v_i \in W_{r_i}$, $\mathsf{Ch}_i^{r_i}$ does not conflict with ch.*

Proof. The proof follows the one for Lemma 12 with the following changes only.

1. Replace Line 22 of Algorithm 3 with Line 17 of Algorithm 6,
2. Replace Lemma 11 with Lemma 31, and
3. Note that when considering Algorithm 6, Case 2.2 implies $\mathsf{Ch}_i = \max(\{\mathsf{ch}\colon |\{v_j\colon \exists \mathsf{ch}' \succeq \mathsf{ch} : [\text{VOTE}, \mathsf{ch}', \cdot, t, v_j] \in \mathcal{V}_i\}| \geq \frac{2}{3}n\})$, from which the rest of the proof for this case follows unaltered. □

Theorem 14 (Asynchrony Reorg Resilience)**.** *Algorithm 6 is η-asynchrony-reorg-resilient.*

Proof. Follows from the proof of Theorem 5 with the following modifications only.

1. Replace Theorem 3 with Theorem 12,
2. Replace Lemma 6 with Lemma 26, and

3. Replace Lemma 12 with Lemma 32. □

Theorem 15 (Asynchrony Safety Resilience)**.** *Algorithm 6 is η-asynchrony-safety-resilient.*

Proof. Follows from the proof of Theorem 6 with the following modifications only.

1. Replace MFC with RLMD,
2. Replace Theorem 4 with Theorem 13,
3. Replace Lemma 7 with Lemma 27, and
4. Replace Lemma 12 with Lemma 32. □

6.2 Faster finality protocol execution

In this section, we present Algorithm 7 which integrates the η-dynamically-available and reorg-resilient protocol of Algorithm 6 with the FFG component introduced in Section 4 to obtain a η-secure ebb-and-flow protocol.

We describe Algorithm 7 by discussing the main differences compared to Algorithm 6, which are quite similar to the differences between Algorithm 4 and Algorithm 3.

To start with, like in Algorithm 4, the logic employed by Algorithm 7 to determine fast confirmed chains is encoded in the function $\texttt{fastconfirm}(\mathcal{V}, t)$ which returns the same output of $\texttt{fastconfirmsimple}(\mathcal{V}, t)$ from Algorithm 6 as long as the returned fast confirmed chain extends $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$. Otherwise, it just returns $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$.

Subsequently, Algorithm 7 maintains the same state variables as those defined in Algorithm 6.

However, compared to Algorithm 6, like Algorithm 7, Algorithm 7outputs two chains.

- **Available Chain chAva_i:** This roughly corresponds to the confirmed chain Ch_i of Algorithm 3.
- **Finalized Chain chFin_i:** At any fast confirmation round, chFin is set to $\mathcal{GF}(\mathcal{V}_i).\mathsf{ch}$, *i.e.*, the chain of the greatest justified checkpoint.

  Also, at any vote round, the finalized chain chFin_i is set to the longest chain such that of $\mathsf{chFin}_i \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch} \land \mathsf{chFin}_i \preceq \mathsf{chAva}_i$.

Next, compared to Algorithm 6, Algorithm 7 proceeds as follows.

Propose: The propose phase remains identical to that in Algorithm 6, with the only difference being the input to the fork-choice function RLMD-GHOST. Specifically, instead of using the genesis block as input, validator v_i inputs $GJ(\mathcal{V}_i).\mathsf{ch}$[24].

Vote: First, similarly to the propose phase, instead of using the genesis block as input to the fork-choice function, validator v_i inputs $GJ(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}$.

Second, in Algorithm 7, honest validators also include an FFG-VOTE in the VOTE messages that they send. Specifically, the source checkpoint of such FFG-VOTE sent corresponds to $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen}})$. The target checkpoint is determined by selecting the highest chain among the previous available chain, the κ-deep prefix of the chain output by the fork-choice function RLMD-GHOST, and $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}$, filtering out any chain if it is not a prefix of the chain output by the fork-choice function RLMD-GHOST.

Validator v_i also sets the available chain output, chAva_i to such chain.

Fast Confirm: Exactly like in Algorithm 4, validator v_i checks whether chAva_i is a prefix of the chain $\text{fast}^{\text{cand}}$ output by $\texttt{fastconfirm}(\mathcal{V}_i, t)$ or it conflicts with it, and, in either case, v_i updates chAva_i to $\text{fast}^{\text{cand}}$.

Merge: The propose phase remains identical to that in Algorithm 6.

[24]The output of RLMD-GHOST$(\mathcal{V}_i, \mathcal{GJ}(V_i).\mathsf{ch}, t)$ corresponds to the output of HFC$(\mathcal{V}_i, t)$ with HFC being the fork-choice defined in [9].

Algorithm 7 Faster finality protocol – code for validator v_i

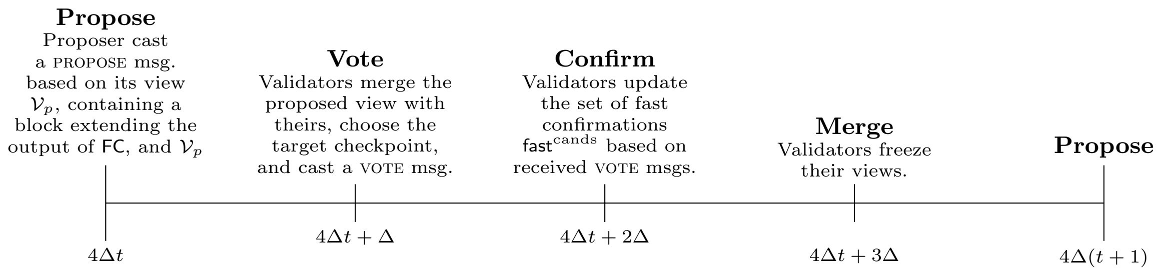
```
1: Output
2:     $\mathsf{chAva}_i \leftarrow B_{\text{genesis}}$: available chain of validator $v_i$
3:     $\mathsf{chFin}_i \leftarrow B_{\text{genesis}}$: finalized chain of validator $v_i$
4: State
5:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \{B_{\text{genesis}}\}$: frozen view of validator $v_i$
6: function $\texttt{fastconfirm}(\mathcal{V}, t)$
7:     let $\mathsf{fast}^{\text{cand}} := \texttt{fastconfirmsimple}(\mathcal{V}, t)$
8:     if $\mathsf{fast}^{\text{cand}} \neq \bot \wedge \mathsf{fast}^{\text{cand}} \succeq \mathcal{GJ}(\mathcal{V}).\mathsf{ch}$
9:         return $\mathsf{fast}^{\text{cand}}$
10:    else
11:        return $\mathcal{GJ}(\mathcal{V}).\mathsf{ch}$
   PROPOSE
12: at round $4\Delta t$ do
13:     if $v_i = v_p^t$ then
14:         let $\mathsf{ch}^{\mathsf{RLMD}} := \text{RLMD-GHOST}(\mathcal{V}_i, GJ(\mathcal{V}_i).\mathsf{ch}, t)^{\lceil 1,t}$
15:         let $\mathsf{ch}_p := \mathsf{Extend}(\mathsf{ch}^{\mathsf{RLMD}}, t)$
16:         send message [PROPOSE, $\mathsf{ch}_p$, $\mathcal{V}_i \cup \{\mathsf{ch}_p\}$, $t$, $v_i$] through gossip
   VOTE
17: at round $4\Delta t + \Delta$ do
18:     let $\mathsf{ch}^{\mathsf{RLMD}} := \text{RLMD-GHOST}(\mathcal{V}_i^{\text{frozen}}, \mathcal{GJ}(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}, t)$
19:     $\mathsf{chAva}_i \leftarrow \max(\{\mathsf{ch} \in \{\mathsf{chAva}_i, (\mathsf{ch}^{\mathsf{RLMD}})^{\lceil \kappa}, \mathcal{GJ}(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}\} : \mathsf{ch} \preceq \mathsf{ch}^{\mathsf{RLMD}}\})$
20:     $\mathsf{chFin}_i \leftarrow \max(\{\mathsf{ch} \colon \mathsf{ch} \preceq \mathsf{chAva}_i \wedge \mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}\})$
21:     let $\mathcal{T} := (\mathsf{chAva}_i, t)$
22:     send message [VOTE, $\mathsf{ch}^{\mathsf{RLMD}}$, $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen}}) \rightarrow \mathcal{T}$, $t$, $v_i$] through gossip
   FAST CONFIRM
23: at round $4\Delta t + 2\Delta$ do
24:     let $\mathsf{fast}^{\text{cand}} := \texttt{fastconfirm}(\mathcal{V}_i, t)$
25:     if $\mathsf{chAva}_i \not\succeq \mathsf{fast}^{\text{cand}}$ then
26:         $\mathsf{chAva}_i \leftarrow \mathsf{fast}^{\text{cand}}$
27:     $\mathsf{chFin}_i \leftarrow \max(\{\mathsf{ch} \colon \mathsf{ch} \preceq \mathsf{chAva}_i \wedge \mathsf{ch} \preceq \mathcal{GF}(\mathcal{V}_i).\mathsf{ch}\})$
   MERGE
28: at round $4\Delta t + 3\Delta$ do
29:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i$

30: upon receiving a gossiped message [PROPOSE, $\mathsf{ch}_p$, $\mathcal{V}_p$, $t$, $v_p^t$] at any round in $[4\Delta t, 4\Delta t + \Delta]$ do
31:     $\mathcal{V}_i^{\text{frozen}} \leftarrow \mathcal{V}_i^{\text{frozen}} \cup \mathcal{V}_p$
```

6.3 Faster finality protocol analysis

Algorithm 7, as Algorithm 4, works in the generalized partially synchronous sleepy model, and is in particular a η-secure ebb-and-flow protocol. In Section 4.1 we have already shown that the the finalized chain chFin is $\frac{n}{3}$-accountable, and thus always safe if $f < \frac{n}{3}$. Now, for $\mathsf{GST} = 0$, we show in Section 6.3.1 that, if the execution is η-compliant in this stronger sense, then all the properties of Algorithm 6, i.e., η Dynamic Availability and η Reorg Resilience, keep holding. Finally, in Section 6.3.2 we show that, after $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$, Algorithm 7 guarantees the required conditions listed in Property C to ensure that the finalized chain is live.

6.3.1 Synchrony

In this section, we prove that chain chAva of Algorithm 7 is η-dynamically-available, η-reorg-resilient, η-asynchrony-reorg-resilient and η-asynchrony-safety-resilient. 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. Observe that, in $\mathsf{RLMD\text{-}GHOST}$ with fast confirmations (Appendix B [10]), the assumption $f < \frac{n}{3}$ is strictly needed for safety (and only for clients which use fast confirmations), but for example not for Reorg Resilience or liveness, because fast confirmations do not affect the chain output



Propose
Proposer cast a PROPOSE msg. based on its view $\mathcal{V}_p$, containing a block extending the output of FC, and $\mathcal{V}_p$
Vote
Validators merge the proposed view with theirs, choose the target checkpoint, and cast a VOTE msg.
Confirm
Validators update the set of fast confirmations $\mathsf{fast}^{\text{cands}}$ based on received VOTE msgs.
Merge
Validators freeze their views.
Propose
$4\Delta t$
$4\Delta t+\Delta$
$4\Delta t+2\Delta$
$4\Delta t+3\Delta$
$4\Delta(t+1)$

Figure 3: Slot t of the protocol, with its four phases.

by the fork-choice function. On the other hand, Algorithm 7 utilizes confirmations as a prerequisite for justification, and justification does affect the chain output by the fork-choice function. This is because in Algorithm 7, RLMD-GHOST takes in input the chain of the greatest justified checkpoint and filters out any branch conflicting with it. Therefore, we require that less than $\frac{n}{3}$ validators are ever controlled by the adversary, and hence slashable, for all of the properties which we are going to prove.

Our faster finality protocol implemented in Algorithm 7 uses the RLMD-GHOST fork-choice function, dealing with checkpoints and justifications. However, one could implement it using also different fork-choice functions.

We will now adopt a similar approach to the one used in Section 5.4 to demonstrate that the integration of the FFG component into Algorithm 6 does not alter the protocol's behavior in any way that could compromise the properties previously established for Algorithm 6 in Section 6.1. To do so, as we did in Section 5.4, we take any execution of Algorithm 7 and show that there exists an adversary that induce an execution of Algorithm 6 where (i) the messages sent by any honest validator in the two executions match except only for the FFG component, (ii) the fork-choice outputs of any honest validator in the two executions match and (iii) any available chain output by Algorithm 7 is also a confirmed chain of an honest validator in the execution of Algorithm 6. This then allows us to show that the result of the various Lemmas and Theorems presented in Section 6.1 for Algorithm 6 also hold for Algorithm 7. In the following, if e is an execution of Algorithm 6, then $\overset{e}{\text{RLMD}}^{\mathsf{propose}(t)}$ corresponds to the definition provided in Section 6.1.1. If e is an execution of Algorithm 4, then we define $\overset{e}{\text{RLMD}}_i^{\mathsf{propose}(t)} := \text{RLMD-}\overset{e}{\text{GHOST}}(\mathcal{V}_i^{\mathsf{propose}(t)}, \mathcal{GJ}(\mathcal{V}_i^{\mathsf{propose}(t)}).\mathsf{ch}, t)^{\lceil 1,t}$ and $\overset{e}{\text{RLMD}}_i^{\mathsf{vote}(t)} := \text{RLMD-}\overset{e}{\text{GHOST}}(\mathcal{V}_i^{\mathsf{vote}(t)}, \mathcal{GJ}(\mathcal{V}_i^{\mathsf{vote}(t)}).\mathsf{ch}, t)$.

Definition 12. Let e_{FFG} be any η-compliant execution of Algorithm 7 and $e_{\text{NO-FFG}}$ be an η-compliant execution of Algorithm 6. We say that e_{FFG} and $e_{\text{NO-FFG}}$ are RLMD-GHOST-*equivalent* if and only if the following constraints hold:

1. e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent up to any round
2. for any slot t_i,

   2.1 $\text{RLMD-}\overset{e_{\text{NO-FFG}}}{\text{GHOST}}_i^{\mathsf{propose},t_i} = \text{RLMD-}\overset{e_{\text{FFG}}}{\text{GHOST}}_i^{\mathsf{propose},t_i}$

   2.2 $\text{RLMD-}\overset{e_{\text{NO-FFG}}}{\text{GHOST}}_i^{\mathsf{vote},t_i} = \text{RLMD-}\overset{e_{\text{FFG}}}{\text{GHOST}}_i^{\mathsf{vote},t_i}$

   2.3 for any slot $t_j \le t_i$ and validator $v_j \in H_{\mathsf{vote}(t_j)}$, there exists a slot $t_k \le t_j$ 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 $t_j \le t_i$ and validator $v_j \in H_{\mathsf{vote}(t_j)}$, there exists a slot $t_k \le t_j$ and a validator $v_k \in H_{\mathsf{vote}(t_k)}$ such that $\mathsf{Ch}_k^{\mathsf{vote}(t_k)} \succeq \mathsf{chAva}_j^{\mathsf{fconf}(t_j)}$.

Lemma 33. *Let e_{FFG} be any η-compliant execution of Algorithm 7. There exists an η-compliant execution $e_{\text{NO-FFG}}$ of Algorithm 6 such that e_{FFG} and $e_{\text{NO-FFG}}$* RLMD-GHOST-*equivalent.*

Proof. Follows from the proof of Lemma 13 with the following modifications only.

1. Replace MFC with RLMD,
2. Replace Lemma 7 with Lemma 27,
3. Replace Line 22 of Algorithm 4 with Line 19 of Algorithm 7,
4. Replace $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}$ with $\mathcal{GJ}(\mathcal{V}_i^{\mathsf{vote}(t_i)})$,
5. In the treatment of Case 2 of Condition 2.3, note that, due to the change above, we do not need to refer to any line to show that $\mathsf{J}(\mathcal{GJ}(\mathcal{V}_i^{\mathsf{vote}(t_i)}), \mathcal{V}_i^{\mathsf{vote}(t_i)})$,
6. Replace Line 22 of Algorithm 3 with Line 17 of Algorithm 6,
7. Replace the reference to Lines 27 to 30 of Algorithm 4 with reference to Lines 23 to 26 of Algorithm 7, and
8. In the treatment of Case 3 of Condition 2.4, replace $(\mathsf{ch}^C, Q) = \mathtt{fastconfirm}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}, t_i) \land Q \neq \emptyset$ with $\mathsf{ch}^C = \mathtt{fastconfirm}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}, t_i) \land \mathsf{ch}^C \neq \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{merge}(t_i)}.\mathsf{ch})$ and let `fastconfirm` be as defined in Algorithm 7. □

As a result of Lemma 33, we can derive analogous lemmas to Lemma 24, Lemma 25, and Lemma 26, similar to the process followed with Algorithm 4.

In the following Lemmas and Theorems, unless specified, we refer to executions of Algorithm 7.

Lemma 34 (Analogous of Lemma 24)**.** *If, in slot t, all validators in $H_{\mathsf{vote}(t)}$ cast* VOTE *messages for chains extending chain* ch*, then, for any validator $v_i \in H_{\mathsf{vote}(t+1)}$,* RLMD-GHOST$_i^{\mathsf{propose}(t+1)} \succeq \mathsf{ch}$ *and* RLMD-GHOST$_i^{\mathsf{vote}(t+1)} \succeq \mathsf{ch}$*, which implies that, in slot $t+1$, all validators in $H_{\mathsf{vote}(t+1)}$ cast* VOTE *messages for chains extending* ch*.*

Proof. Follows from the proof of Lemma 14 with the following modifications only.

1. Replace MFC with RLMD-GHOST,
2. Replace Algorithm 3 with Algorithm 6,
3. Replace Algorithm 4 with Algorithm 7, and
4. Replace Lemma 13 with Lemma 33 □

Lemma 35 (Analogous of Lemma 26)**.** *Let t be a slot with an honest proposer v_p and assume that v_p casts a* $[\text{PROPOSE}, \mathsf{ch}_p, \mathcal{V}_p, t, v_p]$ *message. Then, for any slot $t' \geq t$, all validators in $H_{\mathsf{vote}(t')}$ cast a* VOTE *message for a chain extending* ch_p*.*

Proof. Follows from the proof of Lemma 15 with the following modifications only.

1. Replace MFC with RLMD-GHOST,
2. Replace Algorithm 3 with Algorithm 6,
3. Replace Algorithm 4 with Algorithm 7,
4. Replace Lemma 6 with Lemma 26, and
5. Replace Lemma 13 with Lemma 33. □

We now prove η Reorg Resilience. Observe that D'Amato and Zanolini define η-reorg-resilient in a slightly different, albeit related, manner. The authors in [10] state that chAva is η-reorg-resilient if any honest proposal B from a slot t is always included in the chain outputted by the fork-choice function of all active validators at any propose and vote rounds equal or higher than $\mathsf{vote}(t)$ in all η compliant executions. This definition differs from ours, as we define Reorg Resilience with respect to the confirmed chain, rather than the chain outputted by the fork-choice function. However, we will now demonstrate how the definition provided by D'Amato and Zanolini [10] implies our definition, obtaining the following result.

We begin by demonstrating that the confirmed chain at round r, denoted as chAva^r, is always a prefix of the chain outputted by the fork-choice function of any active validator in any round $r' \geq r$. Specifically, we show that if r is a propose or vote round, then the confirmed chain is always a prefix of the chain outputted by the fork-choice function in every round of slot slot(r). Conversely, if r is a confirm or merge round, then the confirmed chain is always a prefix of the chain outputted by the fork-choice function in every round of slot $t' > \text{slot}(r)$. This result will then be used in the proof of η Reorg Resilience.

Lemma 36 (Analogous of Lemma 27)**.** *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 $v_j \in H_{\text{slot}(r_j)}$, $\mathsf{chAva}_i^{r_i} \preceq \text{RLMD}_j^{r_j}$.*

Proof. The proof follows the one for Lemma 16 with the following changes only.

1. Replace Algorithm 3 with Algorithm 6,
2. Replace Algorithm 4 with Algorithm 7,
3. Replace Lemma 13 with Lemma 33, and
4. Replace Lemma 7 with Lemma 27. □

Theorem 16 (Reorg Resilience)**.** *The chain* chAva *output by Algorithm 7 is η-reorg-resilient.*

Proof. As in the proof of Theorem 7, we can follow the proof of Theorem 3 with the following changes only.

1. Replace Ch with chAva,
2. Replace Lemma 6 with Lemma 35, and
3. Replace Lemma 7 with Lemma 36. □

Theorem 17 (η-dynamic-availability)**.** *Algorithm 7 is η-dynamically-available.*

Proof. Note that Lemma 28 holds for Algorithm 7 as well. Then, as in the proof of Theorem 8, we can follow the proof of Theorem 4 with the following changes only.

1. Replace Ch with chAva,
2. Replace the reference to Line 22 of Algorithm 3 with Line 19 of Algorithm 7,
3. Replace Lemma 6 with Lemma 35,
4. Replace Lemma 7 with Lemma 36,
5. Replace Lemma 8 with Lemma 28, and
6. In the proof of liveness, consider the additional case mentioned ad point 5 of the proof of Theorem 8 with the following changes only.
   - 6.1 Replace MFC with RLMD-GHOST and
   - 6.2 Replace Lemma 16 with Lemma 36. □

Asynchrony Resilience. We now proceed to demonstrate that Algorithm 7 also satisfies Asynchrony Resilience, similar to the proof provided for Algorithm 4 in Section 5.

Due to the FFG component, establishing Asynchrony Resilience for Algorithm 7 necessitates that Constraint (4) holds, just as it was required for Algorithm 4.

Lemma 37 (Analogous of Lemma 31)**.** *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending* ch*. Then, for any slot $t_i \geq t$, any validator in $W_{\mathsf{vote}(t_i)}$ casts a* VOTE *message for a chain extending* ch*.*

Proof. The proof follows the one for Lemma 18 with the following changes only.

1. Replace Algorithm 4 with Algorithm 7,
2. Replace the reference to Line 22 of Algorithm 4 with the reference to Line 19 of Algorithm 7,
3. Replace Lemma 14 with Lemma 34,
4. Replace Lemma 10 with Lemma 30,
5. Replace Lemma 11 with Lemma 31, and
6. Replace Lemma 4 with Lemma 24. □

Lemma 38 (Analogous of Lemma 32)**.** *Assume $\pi > 0$ and take a slot $t \leq t_a$ such that, in slot t, any validator in $H_{\mathsf{vote}(t)}$ casts a* VOTE *message for a chain extending* ch*. Then, for any round $r_i \geq \mathsf{vote}(t)$ and validator $v_i \in W_{r_i}$, $\mathsf{Ch}_i^{r_i}$ does not conflict with* ch*.*

Proof. As in the proof of Lemma 19, we can follow the same reasoning used in the proof of Lemma 12 with the following changes only.

1. Replace Ch with chAva,
2. Replace Lemma 11 with Lemma 37,
3. In the proof of Case 1, refer to lines Lines 19, 21 and 22 of Algorithm 7 rather than Lines 22 to 23 of Algorithm 3.
4. For Case 2, consider the additional sub case and related proof as per point 4 of Lemma 19. □

Theorem 18 (Asynchrony Reorg Resilience - Analogous of Theorem 14)**.** *Algorithm 7 is η-asynchrony-reorg-resilient.*

Proof. We can follow the same reasoning used in the proof of Theorem 9. As in the proof of Theorem 9, we can follow the same reasoning used in the proof of Theorem 6 with the following changes only.

1. Replace Ch with chAva,
2. Replace Theorem 4 with Theorem 17, and
3. Replace Lemma 12 with Lemma 38. □

Theorem 19 (Asynchrony Safety Resilience - Analogous of Theorem 15)**.** *Algorithm 7 is η-asynchrony-safety-resilient.*

Proof. As in the proof of Theorem 10, we can follow the same reasoning used in the proof of Theorem 6 with the following changes only.

1. Replace Ch with chAva,
2. Replace Theorem 4 with Theorem 17, and
3. Replace Lemma 12 with Lemma 38. □

6.3.2 Partial synchrony

In this section, we show that Algorithm 7, like Algorithm 4, ensures Constraints A to C and hence ensures that the chain chFin is always Accountably Safe and is live after time $\max(\mathsf{GST}, \mathsf{GAT}) + \Delta$, meaning that Algorithm 7 is an η-secure ebb-and-flow protocol.

Lemma 39. *Algorithm 7 satisfies Constraint A.*

Proof. Let us prove that Algorithm 7 ensures all the conditions listed in Constraint A.

Constraint A.1. By Line 18, an honest active validator sends only one FFG-VOTE in any slot.

Constraint A.2. Direct consequence of Line 21.

Constraint A.3. Take any two slots t and t' with $t < t'$. Due to Line 18,the source checkpoints of FFG-VOTE sent by an honest validator v_i in slots t and t' are $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t)})$ and $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t')})$, respectively. Since $\mathcal{V}_i^{\text{frozen}}$ only keeps growing from t to t', we have $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t)}) \leq \mathcal{GJ}(\mathcal{V}_i^{\text{frozen},\mathsf{vote}(t')})$. □

Lemma 40. *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 casts a [PROPOSE, $\mathsf{ch}_p, \mathcal{V}_p, \mathcal{GJ}_p, t, v_p$] message. Then, for any validator $v_i \in H_{\mathsf{vote}(t)}$, v_i casts a VOTE message for chain ch_p and $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t)} = \mathcal{GJ}_p$.*

Proof. Follows from Lemma 2 of [9]. □

Lemma 41. *Algorithm 7 satisfies Constraint C.*

Proof. Let us prove that Algorithm 7 ensures each condition listed in Constraint C.

Constraint C.1.1. Let [VOTE, $\mathsf{ch}, \mathcal{S} \to \mathcal{T}, t, v_i$] be the VOTE message cast by validator v_i in slot t. We need to show that $\mathcal{S}.\mathsf{ch} \preceq \mathcal{T}.\mathsf{ch} \preceq \mathsf{ch}$.

Lines 19, 21 and 22 imply that $\mathcal{T}.\mathsf{ch} \preceq \text{RLMD-GHOST}_i^{\mathsf{vote}(t)} = \mathsf{ch}$.

Then, we are left with showing that $\mathcal{S}.\mathsf{ch} \preceq \mathcal{T}.\mathsf{ch}$. Due to Lines 19, 21 and 22, this amounts to showing that $\mathsf{chAva}_i \succeq \mathcal{GJ}(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}$ which is implied by the fact that $\text{RLMD-GHOST}(\mathcal{V}_i^{\text{frozen}}, GJ(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}, t) \succeq \mathcal{GJ}(\mathcal{V}_i^{\text{frozen}}).\mathsf{ch}$

Constraint C.1.2. Obvious from Line 22.

Constraint C.1.3. Lines 21 and 22 imply that any FFG-VOTE $\mathcal{S}_i \to \mathcal{T}_i$ sent by an honest validator during a slot t' is such that $\mathcal{T}_i.c = t'$.

Constraint C.2.1. Follows from Lemma 28.

Constraint C.2.2. Follows the same reasoning used in the proof of Constraint C.2.2 in Lemma 22 with the following changes only.

1. Replace Lemma 21 with Lemma 40,
2. Replace Line 22 of Algorithm 4 with Line 19 of Algorithm 7, and
3. Replace Line 26 of Algorithm 4 with Line 22 of Algorithm 7.

Constraint C.2.3.1. As per proof of Constraint C.2.3.1 in Lemma 22.

Constraint C.2.3.2. Follows the same reasoning used in the proof of Constraint C.2.3.2 in Lemma 22 with the following changes only.

1. Replace $\texttt{fastconfirm}(\mathcal{V}_i^{\mathsf{fconf}(t)}, t) = (\mathsf{ch}_p, \cdot)$ with $\texttt{fastconfirm}(\mathcal{V}_i^{\mathsf{fconf}(t)}, t) = \mathsf{ch}_p$,
2. Replace Lemma 21 with Lemma 40,
3. Replace Line 22 of Algorithm 4 with Line 19 of Algorithm 7,
4. Replace Line 24 of Algorithm 4 with Line 21 of Algorithm 7,

5. Replace Line 26 of Algorithm 4 with Line 22 of Algorithm 7, and
6. Replace Line 30 of Algorithm 4 with Line 26 of Algorithm 7.

Constraint C.2.4.1. Same reasoning as per proof of Constraint C.2.3.1.

Constraint C.2.4.2 Follows the same reasoning used in the proof of Constraint C.2.4.2 in Lemma 22 with the following changes only.

1. Replace Line 21 of Algorithm 4 with Line 18 of Algorithm 7,
2. Replace Line 22 of Algorithm 4 with Line 19 of Algorithm 7,
3. Remove Line 25,
4. Replace Line 26 of Algorithm 4 with Line 22 of Algorithm 7, and
5. Replace Line 31 of Algorithm 4 with Line 27 of Algorithm 7. □

Theorem 20. *Algorithm 7 is a η-secure ebb-and-flow protocol.*

Proof. Follows from the proof of Theorem 11 with the following modifications only.

1. Replace Algorithm 4 with Algorithm 7,
2. Replace Lines 3, 23 and 31 of Algorithm 4 with Lines 3, 20 and 27 of Algorithm 7 respectively,
3. Replace Lines 28 to 30 of Algorithm 4 with Lines 24 to 26 of Algorithm 7,
4. Replace Theorem 8 with Theorem 17,
5. Replace Lemma 20 with Lemma 39, and
6. Replace Lemma 22 with Lemma 41. □

We conclude this section showing that Algorithm 7 also ensures that the finalized chain of an honest validator grows monotonically.

Lemma 42. *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. Follows the same reasoning used in the proof of Lemma 23 with the following changes only.

1. Replace $\mathcal{GJ}_i^{\text{frozen},r'}$ with $\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},r'})$,
2. In the treatment of Case 1, $\mathsf{chAva}_i^{r'} \succeq \mathcal{GJ}(\mathcal{V}_i^{\text{frozen},r'}).\mathsf{ch}$ is derived from Lines 18 and 19 and $\mathsf{J}(\mathcal{GJ}(\mathcal{V}_i^{\text{frozen},r'}, \mathcal{V}_i^{r'}))$ is derived from $\mathcal{V}_i^{\text{frozen},r'} \subseteq \mathcal{V}_i^{r'}$, and
3. Replace $(\mathsf{chAva}_i^{r'}, \cdot) = \mathtt{fastconfirm}(\mathcal{V}_i^{r'}, \text{slot}(r'))$ with $\mathsf{chAva}_i^{r'} = \mathtt{fastconfirm}(\mathcal{V}_i^{r'}, \text{slot}(r'))$. □

7 Practical considerations

The protocols we just presented in Sections 5 and 6, although theoretically sound as demonstrated by the theorems, presents practical challenges. In this section, we discuss implementation techniques that can be employed to mitigate such challenges. We proceed by discussing peculiarities of Algorithm 4 first, then we move to addressing aspects unique to Algorithm 7, before discussing mitigations that apply to both protocols. We conclude by analyzing message complexity of both protocols.

7.1 Optimizations

1. **Optimizations only for Algorithm 4:** Every validator participating in Algorithm 4 tracks all the received VOTE messages, including those that equivocate. This approach is necessary due to the way the fork-choice function MFC operates.

   The fork-choice function MFC takes two views, $\mathcal{V}$ and $\mathcal{V}'$ (with the possibility that $\mathcal{V} = \mathcal{V}'$ for the proposer), as input. It then considers the latest and unexpired VOTE messages sent by validators that never equivocated within each of these views. The function checks if the intersection of these filtered views contains more than half of all the senders within $\mathcal{V}'$ as Consequently, a generic view $\mathcal{V}$ must contain information regarding VOTE messages, equivocations, and the senders of these messages. This information is crucial for the fork-choice function to determine its output chain.

   However, there is no need to retain unexpired messages within $\mathcal{V}$, as these will be filtered out during the evaluation of the fork-choice function. Therefore, at each slot, every validator v_i can prune their local view $\mathcal{V}_i$ and $\mathcal{V}_i^{\text{frozen}}$ by removing all expired messages and retaining only the unexpired ones. This approach streamlines the process by reducing the amount of data that validators need to store and process, enhancing the overall efficiency of the protocol. However, we need to note that for a given chain ch, this optimization leads to the set $\mathcal{V}^{\mathsf{ch},t}$ having potentially more elements than in the original protocol. This is because the original definition of $\mathcal{V}^{\mathsf{ch},t}$ removes first the VOTEs of any validator that has ever equivocated in $\mathcal{V}$, regardless of the age of their VOTE messages. However, it should be easy to see that this change does not invalidate any of the proofs conducted with respect to the original protocol.

2. **Optimizations only for Algorithm 7:** The way in which views are used in Algorithm 7 is not optimally implementable. Recall that every validator executing Algorithm 7 keeps track of a view that is constantly updated with each message the validator receives. Moreover, the proposer for slot t broadcasts the entire view, containing all the messages ever received, to all the other validators (Line 14, Algorithm 7). This is done to achieve key properties such as ensuring that under synchrony with an honest proposer, the view of validators at the the vote round is a subset of the proposer's view at the propose round, leading the output of the fork-choice function RLMD-GHOST at the vote round to match the chain PROPOSEd by the proposer in that slot.

   Clearly, from a practical standpoint, having a constantly growing set of messages that is forwarded at each slot poses significant issues. This approach leads to inefficiencies and increased overhead.

   However, these issues can be greatly mitigated with some considerations. To start, observe that the fork choice function RLMD-GHOST outputs the chain with the accumulated most weight from unexpired latest messages. This means that a proposer could potentially send only the messages that contribute to this chain as far as the fork choice function is concerned, along with all the VOTE messages carrying FFG-VOTEs that justify the greatest justified checkpoint used in the fork choice function. This way, a large number of messages that do not contribute to the output of the fork-choice function are excluded, reducing the message overhead.

   The intuition behind this is as follows. Take a view $\mathcal{V}$ and a slot t, and remove from $\mathcal{V}$ all the VOTE messages that are not in support of prefixes of the chain RLMD-GHOST$(\mathcal{V}, \mathcal{GJ}(\mathcal{V}).\mathsf{ch}, t)$. Let $\mathcal{V}'$ be the resulting view. Clearly, RLMD-GHOST$(\mathcal{V}, \mathcal{GJ}(\mathcal{V}).\mathsf{ch}, t) =$ RLMD-GHOST$(\mathcal{V}', GJ(\mathcal{V}').\mathsf{ch}, t)$. Therefore, by implementing the optimization above, we still have the property that, under synchrony, the RLMD-GHOSToutput by honest validators in a given slot matches the chain PROPOSEd by an honest validator is that same slot, which is all that we effectively need to ensure all the various properties holding under synchrony. Additionally, in protocols like the Ethereum's consensus protocols, votes are included in blocks. This means that validators can derive the proposer's view – or more precisely, the useful messages used by the proposer to output its chain from the fork-choice function – from the chain it proposes.

   Thus, Algorithm 7 can be made more practical by removing the overall message overhead of sending all messages all the time. While we used the notion of view to clarify the properties and behavior of our protocol, in practice, this can be avoided.

3. **Optimizations for both Algorithms 4 and 7:** In both Algorithms 4 and 7, proposers send chains. However, this does not translate with the proposer having to send all the blocks in that chain in practice. In fact, in both protocols, the proposer extends the chain output by the respective fork-choice function at the time of proposing. Hence, due to gossiping and the synchrony assumptions leveraged upon, it is sufficient for the proposer to send only the blocks added to the chain output by the fork-choice function. While theoretically this can be more than one block (in the case that the output of the fork-choice function has length strictly lower than the previous slot number), in practice, given that this is a quite unlikely scenario, most probably the proposing action will be coded to just add one block to the chain output by the fork-choice function. In addition, some local optimizations in the use of views can still be made in Algorithms 4 and 7. For instance, there is no need to retain unexpired messages within $\mathcal{V}$, as these will be filtered out during the evaluation of the fork-choice function. Therefore, at each slot, every validator v_i can prune their local view $\mathcal{V}_i$ and $\mathcal{V}^{\text{frozen}}$ by removing all expired messages and retaining only the unexpired ones. This approach streamlines the process by reducing the amount of data that validators need to store and process, enhancing the overall efficiency of the protocol.

7.2 Message complexity

Regarding message complexity, both Algorithm 4 and Algorithm 7 exhibit the same expected communication complexity, namely $O(Ln^3)$, where L denotes the block size. This matches the complexity of the underlying dynamically available protocols, TOB-SVD and RLMD-GHOST. This result arises because, similar to TOB-SVD and RLMD-GHOST, our protocols require honest validators to forward all received messages in every round. Consequently, each validator's forwarding of messages they receive contributes to the $O(Ln^3)$ complexity. The finality gadget layered atop the dynamically available protocol does not alter this complexity, as it is encapsulated within the VOTE message, cast at round $4\Delta t + \Delta$. Therefore, the communication complexity of both protocols remains equivalent to that of their underlying dynamically available protocols, with the finality gadget introducing no additional complexity.

8 Healing process

So far, we have proved that, if $\mathsf{GST} = 0$, then both Algorithm 4 and Algorithm 7 are η-dynamically-available, η-reorg-resilient, η-asynchrony-reorg-resilient and η-asynchrony-safety-resilient. In the reminder of this section, we use the term *synchronous properties* to refer to these properties. We chose to study the synchronous properties assuming $\mathsf{GST} = 0$ because this is the usual setting employed to analyze dynamically-available protocols.

However, the assumption that $\mathsf{GST} = 0$ is too strong in practice. In fact, this would mean that as soon as we have some period of asynchrony, none of the synchronous properties is guaranteed to hold anymore[25]. This is quite undesirable in practice as it is reasonable to expect that a protocol meant to run uninterruptedly for decades will experience periods of asynchrony at different points in time.

In this section, we show that, even if $\mathsf{GST} > 0$, once the network becomes synchronous and a set of additional conditions is met, then both protocols start to guarantee all the synchronous properties from that point onward.

We formalize this list of additional conditions below by letting t_{heal} be the first slot satisfying them.

Definition 13 (Slot t_{heal})**.** Let t_{heal} be the first slot such that

1. $\mathsf{propose}(t_{\mathsf{heal}}) \geq \mathsf{GST} + \Delta$,
2. all validators that are honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$ are also active in round $\mathsf{fconf}(t_{\mathsf{heal}})$ and
3. the proposer of slot t_{heal} is honest.

Provided that such slot t_{heal} exists, then we have the following results.

[25]Note that the asynchrony-reorg and safety-resiliency properties only guarantee limited resiliency to periods of asynchrony, even for those lasting less than η. Specifically, they only offer resiliency towards periods of asynchrony for events, either a chain being PROPOSEd by an honest validator or an honest validator confirming a chain, that happened before the period of asynchrony.

Theorem 21 (Reorg Resilience)**.** *Both Algorithms 4 and 7 are η-reorg-resilient after slot t_{heal} and time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Theorem 22 (η Dynamic Availability)**.** *Both Algorithms 4 and 7 are η-dynamically-available after time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Theorem 23 (Asynchrony Reorg Resilience)**.** *Both Algorithms 4 and 7 are asynchrony-reorg-resilient after slot t_{heal} and time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Theorem 24. *Both Algorithms 4 and 7 are asynchrony-safety-resilient after time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

In the reminder of this section, we provide the intuition underpinning these results. We refer the reader to Appendix A for detailed proofs.

Let us start with considering the dynamically-available protocols Algorithms 3 and 6 that the faster-finality protocols Algorithms 4 and 7 are based on. In those protocols, intuitively, all that it needs to happen for the synchronous properties to start holding is that there exists a slot t_p with an honest proposer such that $\mathsf{merge}(t_p - 1) \geq \mathsf{GST}$. This is because the only synchrony requirement that Lemma 6 relies on is to have synchrony starting from the merge round preceding a slot with an honest proposer. However, under these conditions only, η Safety after time t_{heal}, as we have stated it, would not quite hold. In fact, it would hold only if we limit it to be satisfied just for those validators that have been active in at least one round since the confirm round of slot t_{heal}. Nevertheless, with the additional requirements that all validators honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$ are also active in that round, η Safety after time t_{heal} is guaranteed by either Algorithm 3 or Algorithm 6. This is because, thanks to this additional requirement, in round $\mathsf{fconf}(t_{\mathsf{heal}})$, all validators honest in that round set their confirmed chain to the chain PROPOSEd in slot t_{heal}.

Now, let us move to Algorithms 4 and 7 and consider the implications of the added logic to handle the FFG component in either protocol. What can happen during asynchrony is that enough honest validators send FFG-VOTEs targeting a checkpoint $\mathcal{C}$ conflicting with the chain ch_p PROPOSEd by an honest validator after GST such that these votes alone do not justify such checkpoint, but they can justify it together with the FFG-VOTEs that the adversary can send via the validators that it has corrupted. Such FFG-VOTEs are then released by the adversary after ch_p is PROPOSEd. If no checkpoint higher than $\mathcal{C}$ has been justified yet, which can very well happen due to low honest participation, then, after such FFG-VOTEs are released, honest validators would switch to VOTing and confirming a chain extending $\mathcal{C}$ and, therefore, they would violate η Dynamic Availability and Reorg Resilience.

However, what happens in slot t_{heal} is that all honest validators cast FFG-VOTEs with the same source checkpoint (the greatest justified checkpoint in the view of the honest proposer at the proposing time) and with target a checkpoint $\mathcal{C}'$, possibly different for each validator, with $\mathcal{C}'.\mathsf{ch} \preceq \mathsf{ch}_p$ and $\mathcal{C}'.c = t_{\mathsf{heal}}$. Given that all those validators are also active in the confirm round of slot t_{heal} and that we assume $f < \frac{n}{3}$, this leads to all validators honest in the confirm round of slot t_{heal} to see a checkpoint $\mathcal{J}$, possibly different for each validator, with $\mathcal{J}.\mathsf{ch} \preceq \mathsf{ch}_p$ and $\mathcal{J}.c = t_{\mathsf{heal}}$ as the greatest justified checkpoint. Like in the case of Algorithms 3 and 6, this also leads to all validators honest in the confirm round of slot t_{heal} to set chAva to ch_p. As a consequence of this, by the vote round of slot $t_{\mathsf{heal}} + 1$, for any honest validator, it is impossible that a greatest justified checkpoint conflicting with chain ch_p can emerge. Hence, all honest validators in $H_{t_{\mathsf{heal}}+1}$ cast VOTE messages extending chain ch_p and FFG-VOTEs for targets that are prefixes of ch_p. Therefore, given that we assume $f < \frac{n}{3}$, no checkpoint for slot $t_{\mathsf{heal}} + 1$ conflicting with ch_p can ever be justified. Clearly this will keep being the case for any slot onward. In summary, the key intuition about Algorithm 4 and Algorithm 7 guaranteeing the synchronous properties after slot t_{heal} or after round $\mathsf{fconf}(t_{\mathsf{heal}})$, depending on the property, is that after the confirm round of slot t_{heal}, all honest validators always see greatest justified checkpoints for a slot t_{heal} or higher and no checkpoint for slot t_{heal} or higher conflicting with the chain PROPOSEd in slot t_{heal} can ever be justified.

9 Conclusion

In this work, we present a novel finality gadget that, when integrated with dynamically-available consensus protocols, enables a secure ebb-and-flow protocol with just a single voting phase per proposed chain. This streamlined design enhances the practical throughput of large-scale blockchain networks. By limiting the process to a single voting phase for each chain proposal, our protocols reduce the time interval between proposals. Consequently, this optimization leads to a 20% increase in transaction throughput compared to the SSF protocol of D'Amato and Zanolini (assuming same block size), all while maintaining essential security guarantees.

Moreover, assuming a uniformly distributed transaction submission time, our protocols achieve a shorter expected confirmation time compared to the SSF protocol of D'Amato and Zanolini. Specifically, for an adversary with power $\beta = \frac{1}{3}$, the expected confirmation time for 3SF is improved by approximately 11% over the SSF protocol.

A key trade-off in our approach is a slight delay in chain finalization, which extends to the time required to propose three additional chains, rather than finalizing before the next chain proposal. In practice, this translates to an expected finalization time of 16Δ (which can be reduced to 13Δ) when $\beta = \frac{1}{3}$, compared to 11Δ in the SSF protocol. However, this delay is offset by providing early confirmation of the proposed chain before the subsequent chain is introduced. Users can be confident that, under conditions of network synchrony and at least 2/3 honest node participation, the chain will be finalized within the time required to propose three additional chains. This feature is particularly beneficial in practical scenarios, where periods of synchrony and robust honest participation often extend well beyond the duration required for finalization in our protocol. Finally, our protocols enhance the practicality of large-scale blockchain networks by enabling the dynamically-available component to recover from extended asynchrony, provided at least two-thirds of validators remain honest and online for sufficient time. This improvement addresses a limitation in the original ebb-and-flow protocol, which assumed constant synchrony to ensure safety, highlighting the importance of resilience in real-world, asynchronous conditions.

References

[1] Aditya Asgaonkar, Francesco D'Amato, Roberto Saltini, and Luca Zanolini. Ethereum consensus property list. URL: https://docs.google.com/document/d/1Q_iDOODIq-glLRPSMnSf3HAKDQGHXsENHzRG9iwok7g.

[2] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017. URL: http://arxiv.org/abs/1710.09437, arXiv:1710.09437.

[3] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and Casper. 2020.

[4] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.

[5] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 910–927. IEEE, 2020. doi:10.1109/SP40000.2020.00040.

[6] Francesco D'Amato, Giuliano Losa, and Luca Zanolini. Asynchrony-resilient sleepy total-order broadcast protocols. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 247–256. ACM, 2024. doi:10.1145/3662158.3662779.

[7] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. No more attacks on proof-of-stake ethereum? *CoRR*, abs/2209.03255, 2022. URL: https://doi.org/10.48550/arXiv.2209.03255.

[8] Francesco D'Amato, Roberto Saltini, Thanh-Hai Tran, and Luca Zanolini. TOB-SVD: Total-Order Broadcast with Single-Vote Decisions in the Sleepy Model, 2024. URL: https://arxiv.org/abs/2310.11331, arXiv:2310.11331.

[9] Francesco D'Amato and Luca Zanolini. A simple single slot finality protocol for ethereum. In Sokratis K. Katsikas, Frédéric Cuppens, Nora Cuppens-Boulahia, Costas Lambrinoudakis, Joaquín García-Alfaro, Guillermo Navarro-Arribas, Pantaleone Nespoli, Christos Kalloniatis, John Mylopoulos, Annie I. Antón, and Stefanos Gritzalis, editors, *Computer Security. ESORICS 2023 International Workshops - CyberICS, DPM, CBT, and SECPRE, The Hague, The Netherlands, September 25-29, 2023, Revised Selected Papers, Part I*, volume 14398 of *Lecture Notes in Computer Science*, pages 376–393. Springer, 2023. `doi:10.1007/978-3-031-54204-6\_23`.

[10] Francesco D'Amato and Luca Zanolini. 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*, pages 127–142. IEEE, 2024. `doi:10.1109/CSF61375.2024.00001`.

[11] Eli Gafni and Giuliano Losa. Brief announcement: Byzantine consensus under dynamic participation with a well-behaved majority. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 41:1–41:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.DISC.2023.41`, `doi:10.4230/LIPICS.DISC.2023.41`.

[12] Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *CoRR*, abs/2304.14701, 2023. URL: `https://doi.org/10.48550/arXiv.2304.14701`, `arXiv:2304.14701`, `doi:10.48550/ARXIV.2304.14701`.

[13] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Byzantine Consensus under Fully Fluctuating Participation. 2022. URL: `https://eprint.iacr.org/archive/2022/1448/20221024:011919`.

[14] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy BFT. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 490–503. ACM, 2023. `doi:10.1145/3576915.3623073`.

[15] Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing, 2024. URL: `https://arxiv.org/abs/2208.06046`, `arXiv:2208.06046`.

[16] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2295–2308. ACM, 2022. `doi:10.1145/3548606.3559347`.

[17] Joachim Neu, Ertem Nusret Tas, and David Tse. 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*, pages 446–465. IEEE, 2021. `doi:10.1109/SP40001.2021.00045`.

[18] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *42nd IEEE Symposium on Security and Privacy*, 2021. Forthcoming. URL: `https://arxiv.org/abs/2009.04987`.

[19] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017.

[20] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 560–576. Springer, 2022.

[21] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[22] Vlad Zamfir. Casper the friendly ghost. a correct-by-construction blockchain consensus protocol. URL: `https://github.com/ethereum/research/blob/master/papers/cbc-consensus/AbstractCBC.pdf`.

A Healing process - Detailed Proofs

In this section, we provide detailed proofs of the results presented in Section 8. In the interest of space and time, we will only show the proofs for Algorithm 4 as the proofs for Algorithm 7 are very similar. The reader should be convinced of this by considering the similarities of the proofs of the various Lemmas and Theorems for Algorithm 7 to the proof of the analogous Lemmas and Theorems for Algorithm 4.

First, we prove that all the synchronous properties hold for Algorithm 3, the dynamically-available protocol that Algorithm 4 is based on, from slot t_{heal} onward. Then, in line with the strategy followed so far, we show that given an execution of Algorithm 4 there exists an execution of Algorithm 3 that is a dynamically-equivalent to the execution of Algorithm 4 from t_{heal} onward. This will then enable us to then show that the synchronous properties hold for Algorithm 4 from slot t_{heal} onward.

A.1 Analysis for Algorithm 3

Lemma 43. *If* $\mathsf{merge}(t) \geq \mathsf{GST}$*, then Lemma 4 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. The only synchrony requirement that the proof of Lemma 4 relies on is that VOTE messages sent by validators in $H_{\mathsf{vote}(t)}$ are received by round $\mathsf{merge}(t)$ by all honest validator awake at that round. □

Lemma 44. *If* $\mathsf{merge}(t) \geq \mathsf{GST}$*, then Lemma 5 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. Given Lemma 43, the only synchrony requirement that the proof Lemma 5 relies on is that synchrony holds since $\mathsf{merge}(t)$. □

Lemma 45. *If* $\mathsf{merge}(t-1) \geq \mathsf{GST}$*, then Lemma 6 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. Given Lemma 43, the only synchrony requirement that the proof of Lemma 6 relies on is that synchrony holds since $\mathsf{merge}(t-1) = 4\Delta t - \Delta \geq \mathsf{GST}$. □

Lemma 46. *If* $r_i \geq \mathsf{fconf}(t_{\mathsf{heal}})$*, then Lemma 7 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. The proof of this Lemma is very similar to the proof of Lemma 7. However, the changes are intertwined enough with the proof that, for the sake of clarity, we are better off writing the entire proof.

Like in the proof of Lemma 7, we proceed by contradiction. Let $r_i \geq \mathsf{fconf}(t_{\mathsf{heal}})$ 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 $r_j \in \{\mathsf{propose}(\mathrm{slot}(r_j)), \mathsf{vote}(\mathrm{slot}(r_j))\}$ and $\mathsf{Ch}_i^{r_i} \not\preceq \mathrm{MFC}_j^{r_j}$. Due to the joining protocol, $H_{t_{\mathsf{heal}}}$ includes all the validators honest in round $\mathsf{vote}(t_{\mathsf{heal}})$. Given this and the minimality of r_i, $v_i \in H_{r_i}$ and $\mathsf{Ch}_i^{r_i-1} \neq \mathsf{Ch}_i^{r_i}$. This can only happen if r_i is either a voting or a fast confirmation round. Let $t_i = \mathrm{slot}(r_i)$ and proceed by analyzing each case separately.

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

- **Case 1.1: $\mathsf{Ch}_i^{r_i} = \left(\mathbf{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa}$.** We know that with overwhelming probability (Lemma 2 [10]), there exists at least one slot t_p in the interval $[\max(t_i-\kappa, t_{\mathsf{heal}}), t_i)$ with an honest proposer v_p. Note that this interval is not empty as, due to the Lemma's conditions, this case implies $\mathrm{slot}(r_i) > t_{\mathsf{heal}}$. Let ch_p be the chain PROPOSEd by v_p in slot t_p. Given that $t_{\mathsf{heal}} \leq t_p < t_i$, Lemma 45 implies that $\mathrm{MFC}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p$. Then, because $t_p \geq t_i - \kappa$, we have that $\mathsf{ch}_p \succeq \left(\mathrm{MFC}_i^{\mathsf{vote}(t_i)}\right)^{\lceil\kappa} = \mathsf{Ch}_i^{r_i}$. Because $t_{\mathsf{heal}} \leq t_p < \mathrm{slot}(r_j)$, Lemma 45 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 $t_i < \text{slot}(r_j)$. Let us consider three sub cases.

Case 2.1: $r_i = \mathsf{fconf}(t_{\mathsf{heal}})$**.** Given that the the proposer v_p of t_{heal} is honest, we can apply Lemma 45 to conclude that all validators in $H_{t_{\mathsf{heal}}}$ cast a VOTE message for the chain ch PROPOSEd by v_p. Then, because we assume that $f < \frac{n}{3}$, $H_{t_{\mathsf{heal}}}$ includes all the validators honest in round $\mathsf{vote}(t_{\mathsf{heal}})$ and any validator honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$ is also awake in round $\mathsf{fconf}(t_{\mathsf{heal}})$, then, for any validator v_i honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$, $(\mathsf{ch}_p, Q) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{\mathsf{fconf}(t_{\mathsf{heal}})}, t_i) \wedge Q \neq \emptyset$ meaning that v_i sets $\mathsf{Ch}_i^{r_i} = \mathsf{ch}_p$. Then, we can apply Lemma 44 to conclude that $\mathsf{ch}_j^{r_j} \succeq \mathsf{Ch}_i^{r_i}$ reaching a contradiction.

Case 2.2: $r_i > \mathsf{fconf}(t_{\mathsf{heal}}) \wedge \mathsf{Ch}_i^{r_i - 1} \neq \mathsf{Ch}_i^{r_i}$**.** Due to Lines 25 to 28, this case implies $(\mathsf{Ch}_i^{r_i}, Q) = \texttt{fastconfirmsimple}(\mathcal{V}_i^{r_i}, t_i) \wedge Q \neq \emptyset$. Therefore, we can apply Lemma 44 to conclude that $\mathsf{ch}_j^{r_j} \succeq \mathsf{Ch}_i^{r_i}$ reaching a contradiction.

Case 2.3: $r_i > \mathsf{fconf}(t_{\mathsf{heal}}) \wedge \mathsf{Ch}_i^{r_i - 1} = \mathsf{Ch}_i^{r_i}$**.** This case implies that $r_i - 1 \geq \mathsf{fconf}(t_{\mathsf{heal}})$. Hence, due to the minimality of r_i we reach a contradiction. ☐

Theorem 25 (Reorg Resilience)**.** *Algorithm 3 is η-reorg-resilient after slot t_{heal} and time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. Follows from Lemmas 45 and 46 and the proof of Theorem 3. ☐

Theorem 26. *Algorithm 3 is η-dynamically-available after time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. Follows from Lemmas 45 and 46 and the proof of Theorem 4. ☐

Lemma 47 (Liveness of fast confirmations)**.** *If $\mathsf{merge}(t-1) \geq \mathsf{GST}$, then Lemma 9 holds for $\mathsf{GST} > 0$ as well.*

Proof. From Lemma 45 and the proof of Lemma 9. ☐

Lemma 48. *If $\mathsf{vote}(t_a) \geq \mathsf{GST}$, then Lemma 10 holds for $\mathsf{GST} > 0$ as well.*

Proof. The only synchrony requirement that the proof of Lemma 10 relies on is that synchrony holds since $\mathsf{vote}(t_a)$. ☐

Lemma 49. *If $\mathsf{vote}(t) \geq \mathsf{GST}$, then Lemma 11 holds for $\mathsf{GST} > 0$ as well.*

Proof. Given Lemmas 43 and 48 the only synchrony requirements that Lemma 12 relies on is that synchrony holds from $\min(\mathsf{vote}(t_a), \mathsf{merge}(t))$ which, given that we assume $t_a \geq t$, is implied by $\mathsf{vote}(t) \geq \mathsf{GST}$. ☐

Lemma 50. *If $\mathsf{vote}(t) \geq \mathsf{GST}$, then Lemma 12 holds for $\mathsf{GST} > 0$ as well.*

Proof. Given Lemma 49, follow the proof of Lemma 12. ☐

Theorem 27 (Asynchrony Reorg Resilience)**.** *Algorithm 3 is η-asynchrony-reorg-resilient after slot t_{heal} and time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. From Theorem 25, Lemmas 45 and 50, and the proof of Theorem 5. ☐

Theorem 28 (Asynchrony Safety Resilience)**.** *Algorithm 3 is η-asynchrony-safety-resilient after time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. From Theorem 26, Lemmas 46 and 50, and the proof of Theorem 6. ☐

A.2 Analysis for Algorithm 4

Before proceeding, we need to tweak the definition of dynamical-equivalence to capture the fact that we interested in it from round $\mathsf{vote}(t_{\mathsf{heal}})$ only.

Definition 14 (dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round r)**.** We say that two sets of messages are dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round r if and only if, after removing from both sets all the PROPOSE messages with slot field lower or equal to t_{heal} and all the VOTE message with field lower than t_{heal}, the resulting sets are dynamically-equivalent up to round r. The definition of two executions being honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round r follow naturally from this.

Consequently, we also need to tweak the definition of MFC-equivalent executions accordingly.

Definition 15. Let e_{FFG} by an η-compliant execution of Algorithm 4 and $e_{\text{NO-FFG}}$ be an η-compliant execution of Algorithm 3. We say that e_{FFG} and $e_{\text{NO-FFG}}$ are t_{heal}-MFC-*equivalent* if and only if the following constraints hold:

1. e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to any round
2. for any slot $t_i \geq t_{\mathsf{heal}}$,
   2.1 $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{propose}(t_i)} = \overset{e_{\text{FFG}}}{\text{MFC}}{}_i^{\mathsf{propose}(t_i)}$.
   2.2 for any slot $t_j \in [t_{\mathsf{heal}}, t_i]$ and validator $v_j \in H_{\mathsf{vote}(t_j)}$, there exists round $r_k \in [\mathsf{fconf}(t_{\mathsf{heal}}), \mathsf{fconf}(t_j)]$ and a validator v_k honest in round r_k such that $\mathsf{Ch}_k^{r_k} \succeq \mathsf{chAva}_j^{\mathsf{fconf}(t_j)}$.
3. for any slot $t_i > t_{\mathsf{heal}}$ and validator $v_i \in H_{\mathsf{vote}(t_i)}$,
   3.1 $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)} = \overset{e_{\text{FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)}$
   3.2 for any slot $t_j \in [t_{\mathsf{heal}} + 1, t_i]$ and validator $v_j \in H_{\mathsf{vote}(t_j)}$, there exists round $r_k \in [\mathsf{fconf}(t_{\mathsf{heal}}), \mathsf{vote}(t_j)]$ and a validator v_k honest in round r_k such that $\mathsf{Ch}_k^{r_k} \succeq \mathsf{chAva}_j^{\mathsf{vote}(t_j)}$.

Lemma 51. *Let e_{FFG} by any η-compliant execution of Algorithm 4. There exists an η-compliant execution $e_{\text{NO-FFG}}$ of Algorithm 3 that is t_{heal}-MFC-equivalent to e_{FFG}.*

Proof. Let e_{FFG} by any η-compliant execution of Algorithm 4 and let $A^{e_{\text{FFG}}}$ be the adversary in such an execution. Let v_p be the honest proposer in slot t_{heal} and let $[\text{PROPOSE}, \mathsf{ch}_p, \cdot, \cdot, \mathcal{GJ}_p, \cdot, \cdot]$ the PROPOSE message sent by v_p in slot t_{heal} in execution e_{FFG}..

Let the adversary $A^{e_{\text{NO-FFG}}}$ behave as follows.

(i) As per item (i) in the proof of Lemma 13.

(ii) As per item (ii) in the proof of Lemma 13.

(iii) As per item (iii) in the proof of Lemma 13.

(iv) Up to round $\mathsf{merge}(t_{\mathsf{heal}} - 1)$,

  (iv.i) For any slot $t_r < t_{\mathsf{heal}}$, if there exists a chain $\mathsf{ch}_r \preceq \overset{e_{\text{FFG}}}{\text{MFC}}{}_p^{\mathsf{propose}(t_{\mathsf{heal}})}$ such that $\mathsf{ch}_r.p = t_r$ and the proposer of t_r is adversarial in round $\mathsf{propose}(t_r)$, then v_r PROPOSEs ch_r in slot $\mathsf{ch}_r.p$.

  (iv.ii) Other than the above, the adversary does not send any message.

  (iv.iii) All messages are delivered within Δ rounds.

(v) From round $\mathsf{merge}(t_{\mathsf{heal}} - 1)$ onward, $A^{e_{\text{NO-FFG}}}$ behaves as per item (iv) in the proof of Lemma 13.

Now, we move to proving by induction on t_i that the execution $e_{\text{NO-FFG}}$ induced by $A^{e_{\text{NO-FFG}}}$ satisfies all Definition 15's conditions. To do so, we add the following conditions to the inductively hypothesis,

2. for any slot $t_i \geq t_{\mathsf{heal}}$,

2.3 $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).c \geq t_{\mathsf{heal}}$

3. for any slot $t_i > t_{\mathsf{heal}}$ and validator $v_i \in H_{\mathsf{vote}(t_i)}$,

   3.4 For any $\mathcal{J}$ such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}) \wedge \mathcal{J}.c \geq t_{\mathsf{heal}}$, $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{propose}(t_i)} \succeq \mathcal{J}.\mathsf{ch}$.

   3.5 For any $\mathcal{J}$ such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}) \wedge \mathcal{J}.c \geq t_{\mathsf{heal}}$, $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)} \succeq \mathcal{J}.\mathsf{ch}$.

   3.6 $\mathcal{GJ}_i^{\text{frozen},t_i}.c \geq t_{\mathsf{heal}}$.

and rephrase Condition 1 as follows

2. for any slot $t_i \geq t_{\mathsf{heal}}$,

   2.4 e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round $4\Delta(t_i+1)$

Base Case: $t_i = t_{\mathsf{heal}}$. The only non-vacuously true conditions for this case are Conditions 2.1 to 2.4. Let us now prove that each holds.

Condition 2.1. It should be quite easy to see that, due to item (iv) of $A^{e_{\text{NO-FFG}}}$'s set of decisions, execution $e_{\text{NO-FFG}}$ of Algorithm 3 is such that $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_p^{\mathsf{propose}(t_{\mathsf{heal}})} = \overset{e_{\text{FFG}}}{\text{MFC}}{}_p^{\mathsf{propose}(t_{\mathsf{heal}})}$.

Condition 2.4. Given the above, in either execution v_p sends a PROPOSE message for the same chain ch_p. Then, Lemmas 15 and 21 show that all honest validators in round $\mathsf{vote}(t_{\mathsf{heal}})$ send the same VOTE messages for chain ch_p. Given that no other messages are sent by honest validators before round $\mathsf{propose}(t_{\mathsf{heal}}+1)$, this proves Condition 2.4 for slot t_{heal}.

Condition 2.3. Lemma 21 and Lines 22, 24 and 26 and the definition of t_{heal} imply that al validators honest in round $\mathsf{vote}(t_{\mathsf{heal}})$ send FFG-VOTEs that have source $\mathcal{GJ}_p$ and target checkpoint $\mathcal{T}$ such that $\mathcal{T}.c = t$ and $\mathcal{T}.\mathsf{ch} \preceq \mathsf{ch}_p$. All such votes are in the view of any validator honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$. Given that we assume $f < \frac{n}{3}$, this implies that, for any validator v_i honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$, $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).\mathsf{ch} \preceq \mathsf{ch}_p$ and $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).c = t_{\mathsf{heal}}$ proving Condition 2.3.

Condition 2.2. Then, given that in execution e_{FFG}, for any validator v_i honest in round $\mathsf{vote}(t_{\mathsf{heal}})$, v_i cast VOTE messages for ch_p, $\mathsf{ch}_p \succeq \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).\mathsf{ch}$, then, for any validator v_j honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$, $\mathsf{chAva}_j^{\mathsf{fconf}(t_{\mathsf{heal}})} = \mathsf{ch}_p$. Similarly, for any validator v_j honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$ of execution $e_{\text{NO-FFG}}$, $\mathsf{Ch}_j^{\mathsf{fconf}(t_{\mathsf{heal}})} = \mathsf{ch}_p$. Hence, Condition 2.2 is proved.

Inductive Step: $t_i > t_{\mathsf{heal}}$. Assume that the Lemma and the additional conditions Conditions 2.3, 2.4 and 3.4 to 3.6 hold for slot $t_i - 1$. Below we prove that they also hold for slot t_i.

Conditions 3.4 and 3.5. Let $\mathcal{J}$ be any checkpoint such that $\mathsf{J}(\mathcal{J}, \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}) \wedge \mathcal{J}.c \geq t_{\mathsf{heal}}$. Because the chain of the target checkpoint of an FFG-VOTE cast by an honest validator v_ℓ in round r_ℓ corresponds to $(\overset{e_{\text{FFG}}}{\mathsf{chAva}}{}_\ell^{r_\ell}.\mathsf{ch}, \text{slot}(r_\ell))$ and we assume $f < \frac{n}{3}$, we have that $\mathcal{J}.c \in [t_{\mathsf{heal}}, t_i)$ and there exists a validator $v_k \in H_{\mathcal{J}.c}$ such that $\mathsf{chAva}_k^{\mathsf{vote}(\mathcal{J}.c)} \succeq \mathcal{J}.\mathsf{ch}$. Let us consider two cases.

Case 1: $\mathcal{J}.c = t_{\mathsf{heal}}$. Lemmas 15 and 21 show that all honest validators in round $\mathsf{vote}(t_{\mathsf{heal}})$ send the same VOTE messages for chain ch_p. Then, because we assume that $f < \frac{n}{3}$, due to Lines 22 to 25, in execution e_{FFG}, we have that $\mathsf{ch}_p \succeq \mathcal{J}.\mathsf{ch}$. Given that $t_{\mathsf{heal}} < t_i$, Lemma 45 also implies that $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{propose}(t_i)} \succeq \mathsf{ch}_p$ and $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p$. Therefore, $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{propose}(t_i)} \succeq \mathsf{ch}_p \succeq \mathcal{J}.\mathsf{ch}$ and $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{ch}_p \succeq \mathcal{J}.\mathsf{ch}$.

Case 2: $\mathcal{J}.c > t_{\mathsf{heal}}$. By the inductive hypothesis, Condition 3.2 implies that there exists a slot $t_m \in [t_{\mathsf{heal}}+1, \mathcal{J}.c]$ and validator $v_m \in H_{\mathsf{vote}(t_m)}$ such that $\mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(\mathcal{J}.c)}$. Given that $t_m \leq \mathcal{J}.c < t_i$, from Lemma 7, we know that $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}^{\mathsf{propose}(t_i)} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(\mathcal{J}.c)} \succeq \mathcal{J}.\mathsf{ch}$ and that $\overset{e_{\text{NO-FFG}}}{\text{MFC}}{}^{\mathsf{vote}(t_i)} \succeq \mathsf{Ch}_m^{\mathsf{vote}(t_m)} \succeq \mathsf{chAva}_k^{\mathsf{vote}(\mathcal{J}.c)} \succeq \mathcal{J}.\mathsf{ch}$.

Given that the set of justified checkpoints in $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}$ is a superset of the set of justified checkpoints in $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$, both Condition 3.4 and Condition 3.5 are proven.

Conditions 2.1 and 3.1. We know that e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round $\mathsf{merge}(t_i)$. Hence, due to item (v) of $A^{e_{\text{NO-FFG}}}$'s set of decisions, for any $v_i \in H_{\mathsf{vote}(t)}$, $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$ and $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$ are dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round $\mathsf{merge}(t_i)$. By Lemma 45, we know that, in execution $e_{\text{NO-FFG}}$, any VOTE cast by a validator honest in a round in $[\mathsf{vote}(t_{\mathsf{heal}}), \mathsf{vote}(t_i - 1)]$ is for a chain extending ch_p. Then, given that e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round $\mathsf{merge}(t_i)$, this is true for execution e_{FFG} as well. From the fact that e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent from round $\mathsf{propose}(t_{\mathsf{heal}})$ up to round $\mathsf{merge}(t_i)$, we also know that, for any validator v_i honest in round $\mathsf{propose}(t_i)$, any VOTE message $[\text{VOTE}, \mathsf{ch}_v, \cdot, t_v, \cdot]$ that is in in one of the two views $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$ and $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$, for which there does not exists a dynamically-equivalent VOTE message in the other view, then $t_v < t_{\mathsf{heal}}$. Hence, given that in both executions, any VOTE cast by a validator honest in a round in $[\mathsf{vote}(t_{\mathsf{heal}}), \mathsf{vote}(t_i - 1)]$ is for a chain extending ch_p, this implies that $\mathsf{ch}_v \not\succeq \mathsf{ch}_p$. Hence, in $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$, for any VOTE message $[\text{VOTE}, \mathsf{ch}', \cdot, \cdot, \cdot]$ with $\mathsf{ch}' \succeq \mathsf{ch}_p$, there exists a dynamically-equivalent message in $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{propose}(t_i)}$, and vice-versa. From this and Condition 3.4 follows that Condition 2.1 holds.

Therefore, $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}$ and $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{vote}(t_i)}$ are t_{heal}-MFC-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round $\mathsf{vote}(t_i)$. Hence, we can apply the same reasoning outlined above and, due to Condition 3.5, conclude that Condition 3.1 holds as well.

Condition 3.6. From the inductive hypothesis, by Condition 2.3, $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i-1)}).c \geq t_{\mathsf{heal}}$. From Lines 38 and 39 then we can conclude that $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c \geq t_{\mathsf{heal}}$.

Condition 2.3. From the inductive hypothesis, by Condition 2.3, $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i-1)}).c \geq t_{\mathsf{heal}}$. Given that $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i-1)} \subseteq \overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}$, clearly $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}).c \geq t_{\mathsf{heal}}.$.

Condition 3.2. By Line 22, $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} \in \{\mathsf{chAva}_i^{\mathsf{fconf}(t_i-1)}, \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch}, (\overset{e_{\text{FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)})^{\lceil\kappa}\}$. Let us consider each case.

Case 1: $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = \mathsf{chAva}_i^{\mathsf{fconf}(t_i-1)}$**.** Condition 2.2 for slot $t_i - 1$ implies Condition 3.2 for slot t_i.

Case 2: $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch}$ Due to Lines 34, 38 and 39, $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}, \mathcal{V}^{\mathsf{vote}(t_i)})$. And from Condition 3.6, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c \geq t_{\mathsf{heal}}$. Hence, by following the reasoning applied when discussing Conditions 3.4 and 3.5, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c \in [t_{\mathsf{heal}}, t_i)$ and there exists a validator $v_k \in H_{\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c}$ such that $\mathsf{chAva}_k^{\mathsf{vote}(\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c)} \succeq \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch} = \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$. Consider two sub cases.

Case 2.1: $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c = t_{\mathsf{heal}}$**.** Given that $\mathsf{J}(\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}, \mathcal{V}^{\mathsf{vote}(t_i)})$, from the proof of Condition 2.2 for slot t_{heal} and the proof of Case 1 of Conditions 3.4 and 3.5, we know that, for any validator v_j honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$, $\mathsf{Ch}_j^{\mathsf{fconf}(t_{\mathsf{heal}})} = \mathsf{ch}_p \succeq \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch}$. Hence, $\mathsf{Ch}_j^{\mathsf{fconf}(t_{\mathsf{heal}})} \succeq \mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.\mathsf{ch} = \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$.

Case 2.2: $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c > t_{\mathsf{heal}}$**.** In this case, we can apply the inductive hypothesis. Specifically, Condition 3.2 for slot $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}.c < t_i$ implies Condition 3.2 for slot t_i.

Case 3: $\mathsf{chAva}_i^{\mathsf{vote}(t_i)} = (\overset{e_{\text{FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)})^{\lceil\kappa}$**.** Line 22 of Algorithm 3 implies that $\mathsf{Ch}_i^{\mathsf{vote}(t_i)} \succeq \overset{e_{\text{NO-FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)}$. Then, Condition 3.1 implies that $\mathsf{Ch}_i^{\mathsf{vote}(t_i)} \succeq \overset{e_{\text{FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)}$. From Line 22 of Algorithm 4, we can conclude that $\mathsf{Ch}_i^{\mathsf{vote}(t_i)} \succeq \overset{e_{\text{FFG}}}{\text{MFC}}{}_i^{\mathsf{vote}(t_i)} \succeq \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$.

Condition 2.2. By Lines 27 to 30, $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} \in \{\mathsf{chAva}_i^{\mathsf{vote}(t_i)}, \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).\mathsf{ch}, \mathsf{ch}^C\}$ with $(\mathsf{ch}^C, Q) = \texttt{fastconfirm}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}, t_i) \land Q \neq \emptyset$. Let us consider each case.

Case 1: $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathsf{chAva}_i^{\mathsf{vote}(t_i)}$**.** In this case, given that $t_i > t_{\mathsf{heal}}$, Condition 3.2 implies Condition 2.2.

Case 2: $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).\mathsf{ch}$. From Condition 2.3, we know that $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).c \geq t_{\mathsf{heal}}$. By following the reasoning applied when discussing Conditions 3.4 and 3.5, this means that $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}) \in [t_{\mathsf{heal}}, t_i)$ and there exists a validator $v_k \in H_{\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).c}$ such that $\mathsf{chAva}_k^{\mathsf{vote}(\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).c)} \succeq \mathsf{chAva}_i^{\mathsf{fconf}(t_i)}$. Consider two sub cases.

Case 2.1: $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).c = t_{\mathsf{heal}}$. Similar to the proof of Case 2.1 of Condition 3.2. From the proof of Condition 2.2 for slot t_{heal} and the proof of Case 1 of Conditions 3.4 and 3.5, we know that, for any validator v_j honest in round $\mathsf{fconf}(t_{\mathsf{heal}})$, $\mathsf{Ch}_j^{\mathsf{fconf}(t_{\mathsf{heal}})} = \mathsf{ch}_p \succeq \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).\mathsf{ch}$. Hence, $\mathsf{Ch}_j^{\mathsf{fconf}(t_{\mathsf{heal}})} \succeq \mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).\mathsf{ch} = \mathsf{chAva}_i^{\mathsf{fconf}(t_i)}$.

Case 2.2: $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).c > t_{\mathsf{heal}}$. In this case, Condition 3.2 for slot $\mathcal{GJ}(\overset{e_{\text{FFG}}}{\mathcal{V}}{}^{\mathsf{fconf}(t_i)}).c < t_i$ implies Condition 2.2 for slot t_i.

Case 3: $\mathsf{chAva}_i^{\mathsf{fconf}(t_i)} = \mathsf{ch}^C$ **with** $(\mathsf{ch}^C, Q) = \texttt{fastconfirm}(\overset{e_{\mathbf{ffg}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}, t_i) \land Q \neq \emptyset$**.** This implies that $\overset{e_{\text{FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}$ includes a quorum of VOTE message for chain ch^C and slot t_i. Given Condition 3.1, $\overset{e_{\text{NO-FFG}}}{\mathcal{V}}{}_i^{\mathsf{fconf}(t_i)}$ also includes a quorum of VOTE message for chain ch^C and slot t_i. Hence, $\mathsf{Ch}_i^{\mathsf{fconf}(t_i)} = \mathsf{chAva}_i^{\mathsf{fconf}(t_i)}$.

Condition 2.4. As argued in the proof of Conditions 2.1 and 3.1, we know that e_{FFG} and $e_{\text{NO-FFG}}$ are honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$ up to round $\mathsf{merge}(t_i)$. This, Conditions 2.1 and 3.1 and item (v) of $A^{e_{\text{NO-FFG}}}$'s set of decisions clearly imply Condition 2.4 up to round $\mathsf{propose}(t_i + 1)$. □

Lemma 52. *If $t \geq t_{\mathsf{heal}}$, then Lemma 14 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. Due to Lemma 45, we can use the proof of Lemma 14, with the following changes.

1. Replace Lemma 13 with Lemma 51,
2. Replace "dynamically-equivalent" with "t_{heal}-dynamically-equivalent",
3. Replace "honest-output-dynamically-equivalent" with "honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$", and
4. Replace Conditions 2.1 and 2.2 of Lemma 13 with Conditions 2.1 and 3.1 of Lemma 51. □

Lemma 53. *If $t \geq t_{\mathsf{heal}}$, then Lemma 15 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. Given Lemma 52, follow the proof of Lemma 15 with the following changes.

1. Replace Lemma 13 with Lemma 51,
2. Replace "dynamically-equivalent" with "t_{heal}-dynamically-equivalent",
3. Replace "honest-output-dynamically-equivalent" with "honest-output-dynamically-equivalent from round $\mathsf{vote}(t_{\mathsf{heal}})$", and
4. Replace Conditions 2.3 and 2.4 of Lemma 13 with Conditions 2.2 and 3.2 of Lemma 51. □

Lemma 54. *If $r_i \geq \mathsf{fconf}(t_{\mathsf{heal}})$, then Lemma 16 holds for* $\mathsf{GST} > 0$ *as well.*

Proof. Given Lemma 46, follow the proof of Lemma 16 with the following changes.

1. Replace Lemma 13 with Lemma 51,
2. Replace "dynamically-equivalent" with "t_{heal}-dynamically-equivalent", and
3. Replace Conditions 2.3 and 2.4 of Lemma 13 with Conditions 2.2 and 3.2 of Lemma 51. □

Theorem 29 (Reorg Resilience)**.** *Algorithm 4 is η-reorg-resilient after slot t_{heal} and time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. From Lemmas 53 and 54 and the proof of Theorem 7. □

Theorem 30 (η-dynamic-availability)**.** *Algorithm 4 is η-dynamically-available from time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. Given Lemmas 53 and 54, it follows from the proof of Theorem 4. □

Lemma 55. *If $t \geq t_{\mathsf{heal}}$, then Lemma 18 holds for $\mathsf{GST} > 0$ as well.*

Proof. As in the proof of Lemma 18, we proceed by induction on t_i and and the following conditions to the inductive hypothesis.

1. For any slot t', let $\mathcal{X}^{t',\mathsf{ch}}$ be the set of validators v_i in $H_{\mathsf{vote}(t')}$ such that $\mathsf{chAva}_i^{\mathsf{vote}(t')}$ conflicts with ch. Then, for any $t'' \in (t, t_i]$, $\left|X^{t'',\mathsf{ch}} \cup A_\infty\right| < \frac{2}{3}n$.

However, in this proof, we also need to add the following condition to the inductive hypothesis.

2. For any slot $t_j \in [t, t_i]$ and validator $v_j \in W_{\mathsf{vote}(t_j)}$,
   2.1. $\mathcal{GJ}(\mathcal{V}_j^{\mathsf{fconf}(t_j)}).c \geq t$ and
   2.2. $\mathcal{GJ}(\mathcal{V}_j^{\mathsf{fconf}(t_j)})$ does not conflict with ch.

Base Case: $t_i \in [t, t_a]$**.** Condition 2.1 is proved in the proof of Lemma 51.

From Lemma 52 which, given that we assume $f < \frac{n}{3}$, also implies Condition 1. This and Lines 22, 25 and 26 imply Condition 2.2.

Inductive Step: $t_i > t_a$**.** We assume that the Lemma and the additional conditions hold for any slot $t_i - 1$ and prove that it holds also for slot t_i. It should be easy to see that if Condition 2.1 holds for a given slot, it will keep holding for any future slot. Given that we have already proved that it holds in the base case and that $[t, t_a]$ is not any empty set, then Condition 2.1 is clearly true for slot t_i as well. Now, note that due to Line 38, for any honest validator $v_i \in H_{\mathsf{vote}(t_i)}$, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)} \geq \mathcal{GJ}(\mathcal{V}_i^{\mathsf{fconf}(t_i-1)})$. Hence, given that $t_i - 1 \geq t$, Conditions 1, 2.1 and 2.2 holding for slot $t_i - 1$ means that for any validator $v_i \in H_{\mathsf{vote}(t_i)}$, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}$ does not conflict with ch. Then, observe that if Lemma's statement holds for slot t_i, then Condition 2.2 clearly holds as well. Hence, we are left only with proving the Lemma's statement and Condition 1. Let us now proceed by cases and keep in mind that due to Line 22, if in slot t_i a validator $v_i \in H_{\mathsf{vote}(t_i)}$ casts a VOTE message for a chain extending ch, then $\mathsf{chAva}_i^{\mathsf{vote}(t_i)}$ does not conflict with ch.

Case 1: $t_i \in [t_a + 1, t_a + \pi + 1]$. Given that for any honest validator $v_i \in H_{\mathsf{vote}(t_i)}$, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}$ does not conflict with ch, due to Lemma 48, it should be easy to see that, Lemma 10 can be applied to Algorithm 4 as well. This implies that all validators in $W_{\mathsf{vote}(t_i)}$ cast VOTE messages for chains extending ch. Given Constraint (4), then Condition 1 holds for slot t_i as well.

Case 2: $t_i = t_a + \pi + 2$. Given that for any honest validator $v_i \in H_{\mathsf{vote}(t_i)}$, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}$ does not conflict with ch, due to Lemma 49, it should be easy to see that we can apply here the same reasoning used for this same case in Lemma 11. Then, given that we assume $f < \frac{n}{3}$, Condition 1 holds for slot t_i as well.

Case 3: $t_a \geq t_a + \pi + 3$. Given that for any honest validator $v_i \in H_{\mathsf{vote}(t_i)}$, $\mathcal{GJ}_i^{\text{frozen},\mathsf{vote}(t_i)}$ does not conflict with ch and due to Lemma 43, it should be easy to see that Lemma 4 can be applied to Algorithm 4 as well. This implies that all validators in $H_{\mathsf{vote}(t)}$ casts VOTE messages for chains extending ch. Then, due to Lemma 49, the proof proceeds as per the same case in Lemma 11. Also, given that we assume $f < \frac{n}{3}$, Condition 1 holds for slot t_i as well. □

Lemma 56. *If $t \geq t_{\mathsf{heal}}$, then Lemma 19 holds for $\mathsf{GST} > 0$ as well.*

Proof. Given, Lemmas 50 and 55, we can just follow the proof of Lemma 19. ☐

Theorem 31 (Asynchrony Reorg Resilience)**.** *Algorithm 4 is asynchrony resilient after slot t_{heal} and time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. Given Theorem 21, and Lemmas 19 and 53 we can just follow the proof of Theorem 9. ☐

Theorem 32. *Algorithm 4 ensures Asynchrony Safety Resilience after time $\mathsf{fconf}(t_{\mathsf{heal}})$.*

Proof. Given Theorem 22 and Lemma 56, we can just follow the proof of Theorem 10. ☐

B Two-Slot Finality

In this section, we explore a further trade-off between the SSF protocol [9] and our protocol as presented in the main text. Specifically, by introducing a second vote round corresponding to the third vote round from the SSF protocol, we obtain a protocol that can finalize chains proposed by honest proposers one slot earlier, *i.e.*, by the end of the next slot without increasing the slot length even if vote aggregation is employed.

Acknowledgment Messages. As per the SSF protocol, we introduce *acknowledgment* messages that have the form $[\textsc{ack}, \mathcal{C}, t, v_i]$ where $\mathcal{C}$ is a checkpoint with $\mathcal{C}.c = t$[26]. A *supermajority acknowledgment of* $\mathcal{C}$ is a set of at least $\frac{2}{3}n$ distinct ACK messages for $\mathcal{C}$. At round $\mathsf{fconf}(t)$, for any honest validator v_i, if $\mathcal{GJ}(\mathcal{V}_i^{\mathsf{fconf}(t)}).c = t$, then it broadcasts $[\textsc{ack}, \mathcal{GJ}(\mathcal{V}_i^{\mathsf{fconf}(t)}), t, v_i]$. Any observer that receives a supermajority acknowledgment for a *justified* checkpoint $\mathcal{C}$ considers $\mathcal{C}$ to be finalized.

Slashing Rule. The acknowledgment vote $[\textsc{ack}, \mathcal{C}, t, v_i]$ can be interpreted as an FFG-VOTE $\mathcal{C} \to \mathcal{C}$. Then, as in the SSF protocol, we introduce the third slashing $\mathbf{E_3}$: If a validator v_i has sent an FFG-VOTE $\mathcal{C}_1 \to \mathcal{C}_2$, an ACK vote for $\mathcal{C}_a$, and $\mathcal{C}_1 < \mathcal{C}_a \wedge \mathcal{C}_a.c < \mathcal{C}_2.c$, then v_i is slashable. We do not need to introduce any slashing rule for the case that $\mathcal{C}_2 \neq \mathcal{C}_a \wedge \mathcal{C}_2.c = \mathcal{C}_a.c$ as, to finalize $\mathcal{C}_a$, $\mathcal{C}_a$ must first be justified and, hence, such a condition is already covered by the $\mathbf{E_1}$ slashing condition.

Lemma 57. *Let $f \in [0, n]$ and allow for finalization of checkpoints via* ACK *message as well. 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 $\mathbf{E_1}$, $\mathbf{E_2}$, or $\mathbf{E_3}$.*

Proof. Assume that there exists two conflicting finalized checkpoints $\mathcal{C}_1$ and $\mathcal{C}_2$. This implies that both checkpoints are justified as well. Without loss of generality, assume $\mathcal{C}_2.c \geq \mathcal{C}_1.c$. Let us consider two cases.

Case 1: $\mathcal{C}_1$ is finalized via supermajority links. We can apply Lemma 1 to reach a contradiction.

Case 2: $\mathcal{C}_1$ is finalized via supermajority acknowledgment. Let $\mathcal{C}_j$ be the smallest checkpoint such that $\mathcal{C}_j.c > \mathcal{C}_1.c$ and $\mathcal{C}_j.\mathsf{ch} \not\succeq \mathcal{C}_1.\mathsf{ch}$. By our assumptions, we know that such a checkpoint exists. We also know that there exits a validator v_i that has cast both an ACK vote for $\mathcal{C}_1$ and a valid FFG-VOTE $\mathcal{C}'_j \to \mathcal{C}_j$. Given that $\mathcal{C}_j.\mathsf{ch}$ conflicts with $\mathcal{C}_1.\mathsf{ch}$, this implies that $\mathcal{C}'_j.\mathsf{ch} \not\succeq \mathcal{C}_1.\mathsf{ch}$ which, by the minimality of $\mathcal{C}_j$ implies that $\mathcal{C}'_j.c \leq \mathcal{C}_1.c$. If $\mathcal{C}'_j.c = \mathcal{C}_1.c$, then given that as proven in Lemma 1, $\mathcal{C}'_j.\mathsf{ch}$ cannot conflict with $\mathcal{C}_1.\mathit{chain}$ and thath $\mathcal{C}'_j.\mathsf{ch} \not\succeq \mathcal{C}_1.\mathsf{ch}$, $\mathcal{C}'_j.\mathsf{ch} \preceq \mathcal{C}_1.\mathsf{ch}$. Hence, $\mathcal{C}'_j.c \leq \mathcal{C}_1.c$, implies $\mathcal{C}'_j < \mathcal{C}_1$. Given that $\mathcal{C}'_j.c > \mathcal{C}_1.c$, v_i violates condition $\mathbf{E_3}$. ☐

Lemma 58. *Allow for finalization of checkpoints via* ACK *message as well If Constraint A holds, then honest validators are never slashed.*

Proof. Honest validators only ever send ACK messages for the greatest justified checkpoint in their view at time $\mathsf{fconf}(t)$. This implies that they will never send an FFG-VOTE in a slot strictly higher than t with a source checkpoint lower than the checkpoint that they send an ACK message for. ☐

[26]Note that in this context, the parameter t is redundant in the ACK message since it is already incorporated within $\mathcal{C}$. However, for the sake of clarity, we have chosen to include it explicitly in the ACK message.

Theorem 33 (Accountable Safety with Acknowledgments)**.** *Let $f \in [0, n]$ and allow for finalization of checkpoints via* ACK *message as well. If Constraints A and B hold, then the finalized chain* chFin *is* $\frac{n}{3}$*-accountable.*

Proof. Follows from Lemmas 57 and 58. □

Theorem 34 (Liveness with Acknowledgments)**.** *Allow for finalization of checkpoints via* ACK *message as well and assume that Constraint C holds (and $f < \frac{n}{3}$). Let v_p be any validator honest by the time it* PROPOSE*s chain* ch_p *in a slot t such that* $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$*. Then, chain* ch_p *is justified and finalized during slot $t+1$. In particular,* $\mathsf{ch}_p \preceq \mathsf{chFin}_i^{4\Delta(t+1)+3\Delta}$ *for any validator $v_i \in H_{4\Delta(t+1)+3\Delta}$.*

Proof. Follows from the proof of Theorem 2 and the fact that ACK messages for checkpoint $(\mathsf{ch}, t+1)$ are sent at time $\mathsf{fconf}(t+1)$ and therefore received by time $\mathsf{merge}(t+1) = 4\Delta(t+1) + 3\Delta$. □

The following Corollary shows that it takes 6Δ for a chain proposed by an honest validator to become finalized in the global view, meaning that no honest validator can ever finalize any conflicting chain. If we assume that vote rounds take 2Δ, then this latency becomes 8Δ as Fast Confirmation must wait to receive the FFG-VOTEs sent during the vote round, but there is no need at any point to wait for ACK messages.

Corollary 2. *Allow for finalization of checkpoints via* ACK *message as well and and assume that Constraint C holds (and $f < \frac{n}{3}$). Let* $\mathsf{chFin}_\mathsf{G}^r$ *be the longest finalized chain according to view* $\mathcal{V}_\mathsf{G}^r$ *and v_p be any validator honest by the time it* PROPOSE*s chain* ch_p *in a slot t such that* $\mathsf{propose}(t) \geq \max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$*. Then,* $\mathsf{ch}_p \preceq \mathsf{chFin}_\mathsf{G}^{4\Delta(t+1)+2\Delta}$*.*

Proof. Follows from the proof of Theorem 34. □