A Simple Single Slot Finality Protocol For Ethereum

Francesco D'Amato
Ethereum Foundation
francesco.damato@ethereum.org

Luca Zanolini
Ethereum Foundation
luca.zanolini@ethereum.org

Abstract

The implemented consensus protocol of Ethereum, Gasper, has an hybrid design: it combines a protocol that allows dynamic participation among validators, called LMD-GHOST, and a finality gadget, called Casper. This design has been motivated and formalized by Neu, Tas, and Tse (S&P 2021) through the introduction of the ebb-and-flow class of protocols, which are protocols with two confirmation rules that output two ledgers, one that provides liveness under dynamic participation (and synchrony), LMD-GHOST, and one that provides safety even under network partitions, Casper.

Currently, Gasper takes between 64 and 95 slots to finalize blocks. Because of that, a significant portion of the chain is susceptible to reorgs. The possibility to capture MEV (Maximum Extractable Value) through such reorgs can then disincentivize honestly following the protocol, breaking the desired correspondence of honest and rational behavior. Moreover, the relatively long time to finality forces users to choose between economic security and faster transaction confirmation. This motivates the study of the so-called single slot finality protocols: consensus protocols that finalize a block in each slot and, more importantly, that finalize the block proposed at a given slot within such slot.

In this work we propose a *simple, non-blackbox* protocol that combines a synchronous dynamically available protocol with a finality gadget, resulting in a secure ebb-and-flow protocol that can finalize one block per slot, paving the way to *single slot finality* within Ethereum. Importantly, the protocol we present can finalize the block proposed in a slot, within such slot.

1 Introduction

Traditional Byzantine consensus protocols, such as PBFT [5] or HotStuff [17], are devised in a partial synchronous network model [9], in the sense that they always guarantee safety, but they guarantee liveness only after GST. In this setting, however, participants in the protocol are fixed, known in advance, and without possibility to go *offline*, unless being counted as adversarial.

Recently, dynamic participation became a key requirement to devise Byzantine consensus protocols, as it adds a more robustness to systems that allow participants to go offline (or to become *sleepy* [14]), while preserving safety and liveness of such *dynamically available* protocols. Generally speaking, a protocol is dynamically available if, in a context of dynamic participation, safety and liveness can be ensured. One problem of such protocols is that they do not tolerate network partitions; no consensus protocols can both satisfy liveness (under dynamic participation) and safety (under temporary network partitions). For this reason, dynamically available protocols studied so far are focused on a synchronous model [10, 11]. Neu *et al.* [13] formally prove this result by presenting the *availability-finality dilemma*, which states that there cannot be a consensus protocol for state-machine replication, one that outputs a single ledger, that is both dynamically available and that can finalize, *i.e.*, that can always provide safety, even during asynchronous periods or network partitions, and liveness during synchrony. Neu *et al.* [13] propose then a protocol with two confirmation rules that outputs two ledgers, one that provides liveness under dynamic participation (and synchrony), and one that provides safety even under network partitions. This protocol is called *ebb-and-flow* protocol, and it works in the *partially synchronous model* in the sleepy model [14]. In particular, in such a timing model (i) before a global stabilization time (GST) message delays are adversarially chosen, and after GST the network becomes synchronous, with delay upper-bound Δ, and (ii) before a global awake time (GAT) the adversary can set any sleeping schedule for the participants, and after that all honest participants become awake. Precisely, an ebb-and-flow protocol outputs *both* a dynamically available ledger *and* a finalized ledger that is guaranteed to be safe at all times, and live after $\max\{\mathsf{GST}, \mathsf{GAT}\}$.

Interestingly, such a protocol also captures the nature of the Ethereum consensus protocol, Gasper [4], in which the available ledger is output by LMD-GHOST [18], and the finalized ledger by the *finality gadget* Casper FFG [3]. However, the (original version of) LMD-GHOST is actually not secure [13] even in a context of full-participation [13, 15].

Motivated by finding a (more secure) alternative to Gasper, and following the ebb-and-flow approach, D'Amato *et al.* [7] devise a synchronous dynamically available protocol, Goldfish, in the partially synchronous sleepy model [13] that, composed with a generic finality gadget, implements an ebb-and-flow protocol. However, also Goldfish is not completely secure, as it is brittle to temporary asynchrony, 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 block, resulting in a protocol that is not practically viable to replace LMD-GHOST in Ethereum.

To cope with this problem, D'Amato and Zanolini [8] propose RLMD-GHOST, a synchronous dynamically available protocol that does not lose safety during *bounded* periods of asynchrony, 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 protocols.

In this work, we build upon the work of D'Amato and Zanolini [8], and we devise a protocol that combines RLMD-GHOST with a finality gadget. We implement such protocol in the *generalized sleepy model* [8], an extension of the original sleepy model originally presented by Pass and Shi [14], with more generalized and stronger constraints in the corruption and sleepiness power of the adversary.

Our protocol results in a secure ebb-and-flow protocol that can finalize (at most) one block each slot, paving the way to *single slot finality* [2] protocols for practical use within Ethereum. Importantly, the protocol we present can finalize the block proposed in the current slot, within such slot.

The remainder of this work is structured as it follows. In Section 2 we discuss related works. We present our system model in Section 3. Prerequisites for this work are presented in Section 4; we recall RLMD-GHOST as originally presented by D'Amato and Zanolini [8], state its properties, and show a class of protocols, called *propose-vote-merge* protocols, that groups together LMD-GHOST, Goldfish, and RLMD-GHOST under an unique framework.

Protocol specification are described in Section 5. In particular, we show how to slightly modify RLMD-GHOST to interact with a finality gadget, and then present the full protocol. In Section 6 we formally prove the properties that our protocol satisfy. Finally, in Section 7 we enable our protocol to finalize the block proposed in the current slot through *acknowledgments*, messages sent by participants in the consensus protocol, but only relevant to external observers.

2 Related works

Pass and Shi [14] introduced the *sleepy model of consensus*, which models a distributed system where the participants can be either online or offline, meaning their participation is dynamic. This differs from the standard models in the literature that assume honest participants are always online and execute the assigned protocol. Dynamic participation became a key requirement to devise consensus protocols, as it adds a more robustness to systems that allow participants to go offline, while preserving safety and liveness of such *dynamically available* protocols.

Neu et al [13] introduce the *partially synchronous sleepy model* and define the objectives of the Ethereum consensus protocol, Gasper [4], through the concept of an *ebb-and-flow protocol*. A secure ebb-and-flow protocol produces both a dynamically available ledger and a finalized ledger, that is always safe and live after $\max\{\mathsf{GST}, \mathsf{GAT}\}$. In the context of Gasper, the dynamically available ledger is defined by LMD-GHOST [18] and the finalized ledger by Casper [3].

However, under a deeper analysis, Neu *et al* [13] show that LMD-GHOST is not dynamically available, by presenting an attack to its liveness. Neu *et al.* [7] introduce Goldfish, a simplified variant of LMD-GHOST, aiming at solving some problems related to LMD-GHOST [13, 12], that results in a synchronous dynamically available protocol in the partially synchronous sleepy model that, composed with a generic finality gadget, implements an ebb-and-flow protocol. Goldfish however is brittle to temporary asynchrony, 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 block.

D'Amato and Zanolini [8] introduce the *generalized sleepy model*. This model takes up from the original sleepy model presented by Pass and Shi [14] and extends it with more generalized and stronger constraints in the corruption and sleepiness power of the adversary. This allow to explore a broad space of dynamic participation regimes which fall between complete dynamic participation and no dynamic participation. Moreover, they introduce RLMD-GHOST, a generalization of Goldfish and LMD-GHOST, that offers a trade-off between resilience to temporary asynchrony and dynamic availability. RLMD-GHOST represents a middle ground between LMD-GHOST, an asynchrony resilient but not dynamically available protocol, and Goldfish, a dynamically available but not asynchrony resilient protocol. RLMD-GHOST is resilient to bounded asynchrony *up to a vote expiry period*, and satisfies an appropriate notion of dynamic availability.

3 Model and Preliminary Notions

3.1 System model

We consider a set of validators $v_1, \ldots, v_n$ that communicate with each other through 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. A validator that follows its protocol during an execution is called *honest*. On the other hand, a faulty process may crash or even deviate arbitrarily from its specification, e.g., when corrupted by an adversary. We consider Byzantine faults here and 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. Corrupted validators stay corrupted for the remaining duration of the protocol execution, and are 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.

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." Moreover, we assume that messages from honest validator to honest validator are eventually received and cannot be forged. This includes messages sent by Byzantine validators, once they have been received by some honest validator v_i and gossiped around by v_i.

Time is divided into discrete *rounds*. We consider a partially synchronous model in which validators have synchronized clocks but there is no a priori bound on message delays. However, there is a time (not known by the validators), called *global stabilization time* (GST), after which message delays are bounded by Δ rounds. Moreover, we define the notion of *slot* as a collection of 4Δ rounds. The adversary $\mathcal{A}$ can decide for each round which honest validator is *awake* or *asleep* at that round [14]. Asleep validators do not execute the protocol and messages for that round 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* (Section 4.1), after which they become *active*. 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. Moreover, there is a time (not known by the validators), called *global awake time* (GAT), after which all validators are always awake.

Finally, we require that, for some fixed parameter $1 \leq \tau \leq \infty$, the following condition, referred as *τ-sleepiness at slot t* [8], holds for any slot t after GST:

$$|H_{t-1}| > |A_t \cup (H_{t-\tau,t-2} \setminus H_{t-1})| \tag{1}$$

with H_t, A_t, and $H_{s,t}$ are the set of active validators at round $4\Delta t + \Delta$, the set of adversarial validators at round $4\Delta t + \Delta$, and the set of validators that are active *at some point* in slots $[s, t]$, *i.e.*, $H_{s,t} = \bigcup_{i=s}^{t} H_i$ (if $i < 0$ then $H_i := \emptyset$), respectively. An execution in the partially synchronous network model is *τ-compliant* if it satisfies τ-sleepiness. In other terms, we require the number of active validators at round $4\Delta(t-1) + \Delta$ to be greater than the number of adversarial validators at round $4\Delta t + \Delta$, together with the number of validators that were active at some point between rounds $4\Delta(t-\tau) + \Delta$ and $4\Delta(t-2) + \Delta$, but not at round $4\Delta(t-1) + \Delta$.

Observe that, $\tau = 1$ corresponds to the sleepy model from $\mathsf{Goldfish}$ [7], which constraints the adversary in the minimum way that can allow for a secure protocol under dynamic participation. For $\tau = \infty$, τ-sleepiness requires that $|H_{t-1}| > |A_t \cup (H_{0,t-2} \setminus H_{t-1})|$, *i.e.*, all honest validators which are not active at round $4\Delta(t-1)+\Delta$, and which have voted at least once in the past, are counted together with the adversarial ones. If all validators have voted at least once in slots $[0, s-1]$, this requires that $|H_t| > \frac{n}{2}$ for all slots $t > s$, *i.e.*, dynamic participation is allowed only in an extremely narrow sense.

3.2 Validator internals

View Due to adversarial validators and message delays, validators may have different set of received messages. A *view* (at a given round r), denoted by $\mathcal{V}$, is a subset of all the messages that a process has received until r. Observe that 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 $\mathcal{V}_i$ the view of v_i (at a round r).

Blocks and chains For two chains ch_1 and ch_2, we say $\mathsf{ch}_1 \prec \mathsf{ch}_2$ if ch_1 is a prefix of ch_2. If block B is the tip of chain ch, we say that it is the *head of* ch, and we identify the whole chain with B. Accordingly, if $\mathsf{ch}' \prec \mathsf{ch}$ and A is the head of ch', we also say $\mathsf{ch}' \prec B$ and $A \prec B$.

Fork-choice functions A fork-choice function is a deterministic function FC, which takes as input a view $\mathcal{V}$ and a slot t and outputs a block B, satisfying the following *consistency property*: if B is a block extending $\mathsf{FC}(\mathcal{V}, t)$, then $\mathsf{FC}(\mathcal{V} \cup \{B\}, t) = B$. We refer to the output of FC as the *head of the canonical chain in* $\mathcal{V}$, and to the chain whose head is B as the *canonical chain in* $\mathcal{V}$. Each validator keeps track of its canonical chain, which it updates using FC, based on its local view. We refer to the canonical chain of validator v_i at round r as ch_i^r.

3.3 Security

Security Parameters We consider λ and κ be the security parameter associated with the cryptographic components used by the protocol and the security parameter of the protocol itself, respectively. We consider a finite time horizon T_{hor}, which is polynomial in κ. An event happens with *overwhelming probability* if it happens except with probability which is negl(κ)+negl(λ). Properties of cryptographic primitives hold except with probability negl(λ), *i.e.*, with overwhelming probability, but we leave this implicit in the remainder of this work.

Definition 1 (Secure protocol [7]). We say that a protocol outputting a chain Ch is *secure* after time T_{sec}, and has confirmation time T_{conf} [1], if Ch satisfies:

- **Safety:** For any 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 $\mathsf{Ch}_i^r \prec \mathsf{Ch}_j^{r'}$ or $\mathsf{Ch}_j^{r'} \prec \mathsf{Ch}_i^r$.
- **Liveness:** For any rounds $r \geq T_{\mathsf{sec}}$ and $r' \geq r + T_{\mathsf{conf}}$, and any honest validator v_i active at round r', $\mathsf{Ch}_i^{r'}$ contains a block proposed by an honest validator at a round $> r$.

A protocol satisfies τ*-safety* and τ*-liveness* if it satisfies safety and liveness, respectively, *in the* τ*-sleepy model*, *i.e.*, in τ-compliant executions. A protocol satisfies τ-security if it satisfies τ-safety and τ-liveness.

We recall the definition of *dynamic availability* and *reorg resilience* as in [8]

Definition 2 (Dynamic availability). We say that a protocol is τ*-dynamically-available* if and only if it satisfies τ-security after time $T_{\mathsf{sec}} = \mathsf{GST} + O(\kappa)$, with confirmation time $T_{\mathsf{conf}} = O(\kappa)$. 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.

[1] If the protocol satisfies liveness, then at least one honest proposal is added to the confirmed chain of all active validators every T_{conf} slots. Since honest validators include all transactions they see, this ensures that transactions are confirmed within time $T_{\mathsf{conf}} + \Delta$ (assuming infinite block sizes or manageable transaction volume)

Definition 3 (Reorg resilience)**.** An execution in the partially synchronous network model satisfies *reorg resilience* if any honest proposal B from a slot t after $\mathsf{GST} + \Delta$ is always in the canonical chain of all active validators at rounds $\geq 4\Delta t + \Delta$. A protocol is *τ-reorg-resilient* if all τ-compliant executions satisfy reorg resilience.

Definition 4 (Accountable safety)**.** We say that a protocol has *accountable safety* with resilience $f > 0$ if, upon a safety violation, it is possible to identify at least f responsible participants. In particular, it is possible to collect evidence from sufficiently many honest participants and generate a cryptographic proof that identifies f adversarial participants as protocol violators. Such proof cannot falsely accuse any honest participant that followed the protocol correctly. Finally, we also say that a chain is *accountable* if the protocol outputting it has accountable safety. If a protocol Π outputs multiple chains $\mathsf{Ch}_1, \ldots, \mathsf{Ch}_k$, we say that Ch_i is accountable if Π_i, the protocol which runs Π and outputs only Ch_i, is accountable.

Ebb-and-flow protocols Neu *et al.* [13] proved an *availability-finality dilemma*, which states that there cannot be a consensus protocol for state-machine replication, one that outputs a single chain, that provides both dynamic availability and finality (as described below). They propose a protocol with two confirmation rules that outputs two ledgers, 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 τ-sleepy model.

Definition 5 (τ-secure ebb-and-flow protocol)**.** A τ-secure *ebb-and-flow protocol* outputs an available chain chAva that is τ-dynamically-available if $\mathsf{GST} = 0$, and a finalized (and accountable) chain chFin that, if $f < \frac{n}{3}$, is always safe and is live after $\max\{\mathsf{GST}, \mathsf{GAT}\}$. Moreover, for each honest validator v_i and for every round r, chFin_i^r is a prefix of chAva_i^r.

4 Prerequisites

4.1 Propose-vote-merge protocols

The aim of this work is to present an ebb-and-flow [13] protocol that can finalize (at most) one block per slot and, in particular, that can finalize within slot t the block proposed in t. This is achieved by revisiting the *propose-vote-merge* protocol, first introduced by D'Amato and Zanolini [8], as the basis for our protocol implementation. Propose-vote-merge protocols proceed in *slots* consisting of k rounds, each having a proposer v_p, chosen through a proposer selection mechanism among the set of validators. In particular, at the beginning of each slot t, the proposer v_p proposes a block B. Then, all active validators (also referred as *voters*) vote after Δ rounds. The last Δ rounds of the slot are needed for the *view-merge* synchronization technique [6], a technique used to synchronize the views of all the honest validators, *before* they broadcast a vote in a slot, with the view $\mathcal{V}_p$ of the proposer of that slot. Every validator v_i has a buffer $\mathcal{B}_i$, a collection of messages received from other validators, and a view $\mathcal{V}_i$, used to make consensus decisions, which admits messages from the buffer only at specific points in time. Propose-vote-merge protocols are defined through a deterministic fork-choice function FC, which is used by honest proposers and voters to decide how to propose and vote, respectively, based on their view at the round in which they are performing those actions. It is moreover used as the basis of a *confirmation rule* (Section 5.2), which defines the output of the protocol, and thus with respect to which the security of the protocol is defined. A propose-vote-merge protocol proceeds in three phases:

PROPOSE: In this phase, which starts at the beginning of a slot, the proposer v_p merges its view $\mathcal{V}_p$ with its buffer $\mathcal{B}_p$, *i.e.*, $\mathcal{V}_p \leftarrow \mathcal{V}_p \cup \mathcal{B}_p$, and sets $\mathcal{B}_p \leftarrow \emptyset$. Then, v_p runs the fork-choice function FC with inputs its view $\mathcal{V}_p$ and slot t, obtaining the head of the chain $B' = \mathsf{FC}(\mathcal{V}_p, t)$. Proposer v_p extends B' with a new block B, and updates its canonical chain accordingly, setting $\mathsf{ch}_p \leftarrow B$. Finally, it broadcasts the proposal message [PROPOSE, B, $\mathcal{V}_p \cup \{B\}$, t, v_p].

VOTE: Here, every validator v_i that receives a proposal message [PROPOSE, B, $\mathcal{V}$, t, v_p] from v_p merges its view with the proposed view $\mathcal{V}$, by setting $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$. Then, it broadcasts votes for some blocks based on its view. We omit, for the moment, for which blocks a validator v_i votes: it will become clear once we present the full protocol.

MERGE: In this phase, every validator v_i merges its view with its buffer, *i.e.*, $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$, and sets $\mathcal{B}_i \leftarrow \emptyset$.

D'Amato and Zanolini [8] implement a generic propose-vote-merge protocol following the above phases, with $k = 3\Delta$. However, we will consider $k = 4\Delta$, following the approach taken by Neu *et al.* [7] when presenting Goldfish with fast confirmation, where the PROPOSE phase begins at round $4\Delta t$, the VOTE phase happens between round $4\Delta t + \Delta$ and $4\Delta t + 2\Delta$, and the MERGE phase begins at round $4\Delta t + 3\Delta$.

Neu *et al.* [7] introduce the notion of *active* validators[2]: awake validators that have terminated a *joining protocol* at a round r, described as it follows. When an honest validator v_i wakes up at some round $r \in (4\Delta(t-1)+3\Delta, 4\Delta t+3\Delta]$, it immediately receives all the messages that were sent while it was asleep, and it adds them into its buffer $\mathcal{B}_i$, without actively participating in the protocol yet. All new messages which are received are added to the buffer $\mathcal{B}_i$, as usual. Validator v_i then waits for the *next view-merge opportunity*, at round $4\Delta t + 3\Delta$, in order to merge its buffer B_i into its view $\mathcal{V}_i$. At this point, v_i starts executing the protocol. From this point on, validator v_i becomes *active*, until either corrupted or put to sleep by the adversary. We consider such a joining protocol when describing our propose-vote-merge protocol.

4.2 RLMD-GHOST protocol

RLMD-GHOST is a propose-vote-merge protocol first introduced by D'Amato and Zanolini [8]. It is characterized by the GHOST-based [16] fork-choice FC = RLMD-GHOST (Algorithm 1), which combines vote expiry [7] with the latest message driven (LMD) fork-choice [4]. Its filter function $\mathsf{FIL}_{\text{rlmd}}(\mathcal{V}, t)$ removes *all but the latest messages within the expiry period* $[t-\eta, t)$ *for slot* t*, from non-equivocating validators*, *i.e.*, $\mathsf{FIL}_{\text{rlmd}} = \mathsf{FIL}_{\text{lmd}} \circ \mathsf{FIL}_{\eta\text{-exp}} \circ \mathsf{FIL}_{eq}$.

Algorithm 1 RLMD-GHOST Fork-Choice function

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

3: function GHOST($\mathcal{V}, t$)
4:     $B \leftarrow B_{\text{genesis}}$
5:     $M \leftarrow$ all votes in $\mathcal{V}$
6:     while $B$ has descendant blocks in $\mathcal{V}$ do
7:         $B \leftarrow \arg\max_{B' \in \mathcal{V}, \text{ child of } B} w(B', M)$
8:         // ties are broken according to a deterministic rule
9:     return $B$

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

12: function $\mathsf{FIL}_{\text{lmd}}(\mathcal{V}, t)$
13:     $\mathcal{V}' \leftarrow \mathcal{V}$ without all but the most recent (latest) votes of each validator
14:     return $(\mathcal{V}', t)$

15: function $\mathsf{FIL}_{\eta\text{-exp}}(\mathcal{V}, t)$
16:     $\mathcal{V}' \leftarrow \mathcal{V}$ without all votes from slots $< t - \eta$
17:     return $(\mathcal{V}', t)$

18: function $\mathsf{FIL}_{\text{eq}}(\mathcal{V}, t)$
19:     $\mathcal{V}' \leftarrow \mathcal{V}$ without all votes by equivocators in $\mathcal{V}$
20:     return $(\mathcal{V}', t)$
```

In particular, $\mathsf{FIL}_{\text{lmd}}(\mathcal{V}, t)$ removes all but the latest votes of every validator (possibly more than one) from $\mathcal{V}$ and outputs the resulting view, *i.e.*, it implements the *latest message* (LMD) rule, $\mathsf{FIL}_{\eta\text{-exp}}(\mathcal{V}, t)$ removes all votes from slots $< t - \eta$ from $\mathcal{V}$ and outputs the resulting view, and $\mathsf{FIL}_{eq}(\mathcal{V}, t)$ removes all votes

[2]Observe that Neu *et al.* [7] actually refer to *awake* validators to indicate what we call active, and to *dreamy* validators to indicate what we call awake (but not active)

by *equivocating validators in* $\mathcal{V}$ [1], *i.e.*, validators for which $\mathcal{V}$ contains multiple, equivocating, votes for some slot t. In other terms, the fork-choice function RLMD-GHOST considers the last (non equivocating) messages sent by validators that are not older than $t - \eta$ slots, in order to make protocol's decisions.

D'Amato and Zanolini [8] proved that a propose-vote-merge protocol implementing RLMD-GHOST fork-choice function, called RLMD-GHOST protocol, is η-reorg-resilient and η-dynamically-available. Finally, RLMD-GHOST can support *fast confirmations* of honest proposals optimistically (Appendix B [8]), *i.e.*, when honest participation is high.

5 Protocol specification

5.1 Data structures

We consider five data types: PROPOSE, BLOCK, CHECKPOINT, HEAD-VOTE, and FFG-VOTE. We make no distinctions between the network-level representation of objects and their representation in a validator's view, *e.g.*, a BLOCK is both the object that is contained in a validator's view and the one which is gossiped. We describe the objects as tuples (DATA-TYPE, ...) with their data type as a tag, but in practice mostly refer to them without the tag. We use dot notation to refer to the fields. For the tag, we do so simply with .tag, for the other fields we use the generic names specified in the object descriptions below, to access the different fields, *e.g.*, $B.t$ is the slot of block B. In the following, t is a slot and v a validator.

Blocks and checkpoints A block is a tuple $B = (\text{BLOCK}, b, t, v)$, where b is a *block body*, *i.e.*, the protocol-specific content of the block [3]. A checkpoint is a tuple $\mathcal{C} = (\text{CHECKPOINT}, B, t)$, where B is a block and $\mathcal{C}.t \geq B.t$.

Votes A head vote is a tuple [HEAD-VOTE, B, t, v], where B is a block. An FFG vote is a tuple [FFG-VOTE, $\mathcal{C}_1$, $\mathcal{C}_2$, v], where $\mathcal{C}_1, \mathcal{C}_2$ are checkpoints, $\mathcal{C}_1.t < \mathcal{C}_2.t$, and $\mathcal{C}_1.B \prec \mathcal{C}_2.B$. We refer to the two checkpoints as *source* and *target*, respectively, and to FFG votes as *links* between source and target. When v is clear from context, we also write $\mathcal{C}_1 \to \mathcal{C}_2$ for the whole vote, *e.g.*, we say that v casts a $\mathcal{C}_1 \to \mathcal{C}_2$ vote.

Proposals A proposal is a tuple [PROPOSE, B, $\mathcal{V}$, t, v] where B is a block and $\mathcal{V}$ a view. We refer to $\mathcal{V}$ as a *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.

5.2 Confirmation

A confirmation rule allows validators to identify a *confirmed prefix* of the canonical chain, for which safety properties hold, and which is therefore used to define the output of the protocol. Since our protocol outputs two chain, the available chain chAva and the finalized chain chFin, we have two confirmation rules. One is finality, which we introduce in Section 5.3, and defines chFin. The other confirmation rule, defining chAva, is the one adopted by RLMD-GHOST, in its variant supporting fast confirmation[4]. It is itself essentially split in two rules, a *slow* κ-deep confirmation rule, which is live also under dynamic participation, and a *fast optimistic rule*, requiring $\frac{2}{3}n$ honest validators to be awake. Both rules are employed at round $4\Delta t + 2\Delta$, and chAva is updated to the highest block confirmed by either one, so that liveness of chAva only necessitates liveness of one of the two rules.

[3]For simplicity, we omit a reference to the parent block. As mentioned in Section 3, we leave questions of validity implicit

[4]With some minor changes, as RLMD-GHOST still has 3Δ rounds per slots, by requiring an optimistic assumption on network latency in order for fast confirmations to be live

5.3 FFG component

As mentioned above, a propose-vote-merge protocol is characterized by a fork-choice function, that identifies for every slot the current head of the canonical chain for a given validator. Moreover, we described two kind of votes that a validator v_i executes in the VOTE phase: a HEAD-VOTE, used to vote for the head of the canonical chain, i.e., the output of the fork-choice function evaluated at the current slot, and an FFG-VOTE.

The FFG component of our protocol takes inspiration from Casper [3], the finality gadget adopted by the Ethereum's consensus protocol Gasper [4], and aims at finalizing one block per slot by counting FFG-VOTES cast at a given slot.

Justification We say that a set of $\frac{2}{3}n$ distinct FFG votes $\mathcal{C}_1 \to \mathcal{C}_2$ is a *supermajority link* between $\mathcal{C}_1$ and $\mathcal{C}_2$. We say that a checkpoint C is *justified* if there is a chain of $k \geq 0$ supermajority links $(B_{\text{genesis}}, 0) \to \mathcal{C}_1 \cdots \to \mathcal{C}_{k-1} \to C$. In particular, $(B_{\text{genesis}}, 0)$ is justified. Finally, we say that a block B is justified if there exists a justified checkpoint $\mathcal{C}$ with $\mathcal{C}.B = B$.

Slashing The slashing rules are the same as in Casper FFG. Validator v_i is slashable for two *distinct* FFG votes $(\mathcal{C}_1, \mathcal{C}_2, v_i), (\mathcal{C}_3, \mathcal{C}_4, v_i)$ if either:

1. **$\mathbf{E_1}$ (Equivocation)** $\mathcal{C}_2.t = \mathcal{C}_4.t$

2. **$\mathbf{E_2}$ (Surround voting)** $\mathcal{C}_3.t < \mathcal{C}_1.t < \mathcal{C}_2.t < \mathcal{C}_4.t$

Latest justified checkpoint and block A checkpoint is justified in a view $\mathcal{V}$ if $\mathcal{V}$ contains the chain of supermajority links justifying it. We refer to the justified checkpoint $\mathcal{C}$ of highest slot $\mathcal{C}.t$ in $\mathcal{V}$ as the *latest justified checkpoint* in $\mathcal{V}$, or $\mathcal{LJ}(\mathcal{V})$, and to $\mathcal{LJ}(\mathcal{V}).B$ as the *latest justified block* in $\mathcal{V}$, or $LJ(\mathcal{V})$. Ties are broken arbitrarily, and the occurrence of a tie implies that $\frac{n}{3}$ validators are slashable for equivocation. For brevity, we also use $\mathcal{LJ}_i$ to refer to $\mathcal{LJ}(\mathcal{V}_i)$, the latest justified checkpoint in the view $\mathcal{V}_i$ of validator v_i.

Finality A checkpoint $\mathcal{C}$ is *finalized* if it is justified and there exists a supermajority link $\mathcal{C} \to \mathcal{C}'$ with $\mathcal{C}'.t = \mathcal{C}.t + 1$. A block B is finalized if there exists a finalized checkpoint $\mathcal{C}$ with $B = \mathcal{C}.B$.

5.4 Voting

Fork-choice Similarly to Gasper [4], we adopt a hybrid *justification-respecting* fork-choice, HFC, in this case with RLMD-GHOST [8] as the underlying fork-choice, rather than LMD-GHOST. HFC$(\mathcal{V}, t)$ simply starts from $LJ(\mathcal{V})$, the *latest justified block* in $\mathcal{V}$, instead of B_{genesis}, and then proceeds as RLMD-GHOST, *i.e.*, runs GHOST using the view filtered by $\mathsf{FIL}_{\text{rlmd}}$. Formally, we can define it simply by using another view filter, $\mathsf{FIL}_{\text{FFG}}$, *i.e.*, HFC = RLMD-GHOST $\circ\, \mathsf{FIL}_{\text{FFG}}$. $\mathsf{FIL}_{\text{FFG}}(\mathcal{V}, t)$ outputs $(\mathcal{V}', t)$, where $\mathcal{V}'$ filters out blocks in $\mathcal{V}$ that conflict with $LJ(\mathcal{V})$. In other words, it filters out *branches which do not contain* $LJ(\mathcal{V})$, so $LJ(\mathcal{V})$ is guaranteed to be canonical.

Algorithm 2 HFC, the justification-respecting RLMD-GHOST fork-choice

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

3: function $\mathsf{FIL}_{\text{FFG}}(\mathcal{V}, t)$
4:     $\mathcal{V}' \leftarrow \mathcal{V} \setminus \{B \in \mathcal{V}, B.\text{tag} = \text{BLOCK} : LJ(\mathcal{V}) \not\preceq B \land B \not\preceq LJ(\mathcal{V})\}$
5:     return $(\mathcal{V}', t)$
```

Voting rules Consider a validator v_i voting at slot t. Head votes work exactly as in RLMD-GHOST, or any propose-vote-merge protocol, *i.e.*, validators vote for the output of their fork-choice: when it is time to vote, validator v_i casts vote $[\text{HEAD-VOTE}, \text{HFC}(\mathcal{V}_i, t), t, v_i]$. FFG votes always use the *latest justified checkpoint as source*. The target block is the *highest confirmed descendant of the latest justified block, or the latest justified block itself if there is none* [5]. The target checkpoint is then $\mathcal{C}_{\text{target}} = (\arg\max_{B \in \{LJ_i, \mathsf{chAva}\}} |B|, t)$, and the FFG vote of v_i is $[\text{FFG-VOTE}, \mathcal{LJ}_i, \mathcal{C}_{\text{target}}, v_i]$, voting for the link $\mathcal{LJ}_i \to \mathcal{C}_{\text{target}}$.

5.5 Protocol execution

We now present our single slot finality protocol. Note that the PROPOSE and HEAD-VOTE phases are *exactly* as in a generic propose-vote-merge protocol [8]. Moreover, a slot t in our protocol begins at round $4\Delta t$. At any time, the finalized chain chFin_i of validator v_i just consists of the finalized blocks according to its view $\mathcal{V}_i$, so we omit explicit updates to chFin in the following.

PROPOSE: At round $4\Delta t$, proposer v_p merges its view $\mathcal{V}_p$ with its buffer $\mathcal{B}_p$, *i.e.*, $\mathcal{V}_p \leftarrow \mathcal{V}_p \cup \mathcal{B}_p$, and sets $\mathcal{B}_p \leftarrow \emptyset$. Then, v_p runs the fork-choice function HFC with inputs its view $\mathcal{V}_p$ and slot t, obtaining the head of the chain $B' = \text{HFC}(\mathcal{V}_p, t)$. Proposer v_p extends B' with a new block B, and updates its canonical chain accordingly, by setting $\mathsf{ch}_p \leftarrow B$. Finally, it broadcasts the proposal $[\text{PROPOSE}, B, \mathcal{V}_p \cup \{B\}, t, v_p]$.

HEAD-VOTE: In rounds $[4\Delta t, 4\Delta t + \Delta]$, a validator v_i, upon receiving a proposal message (PROPOSE, B, $\mathcal{V}$, t, v_p) from v_p, merges its view with the proposed view $\mathcal{V}$ by setting $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$. After that, or at round $4\Delta t + \Delta$ if no proposal is received, it updates its canonical chain by setting $\mathsf{ch}_i \leftarrow \text{HFC}(\mathcal{V}_i, t)$, and casts the head vote (HEAD-VOTE, $\text{HFC}(\mathcal{V}_i, t)$, t, v_i).

CONFIRM: At round $4\Delta t + 2\Delta$, a validator v_i selects for fast confirmation the highest *canonical* block $B_{\text{fast}} \prec \mathsf{ch}_i$ such that $\mathcal{B}_i$ contains $\geq \frac{2}{3}n$ votes from slot t for descendants of B_{fast}, from distinct validators. It then updates its confirmed chain chAva_i to the highest of B_{fast} and $\mathsf{ch}_i^{\lceil \kappa}$, the κ-deep prefix of its canonical chain, *as long as this does not result in updating* chAva_i *to some prefix of it* (we do not needlessly revert confirmations).

FFG-VOTE: At round $4\Delta t + 2\Delta$, after updating chAva_i, a validator v_i casts the FFG vote (FFG-VOTE, $\mathcal{LJ}_i$, $\mathcal{C}_{\text{target}}$, v_i), where $\mathcal{C}_{\text{target}} = (\underset{B \in \{LJ_i, \mathsf{chAva}_i\}}{\arg\max} |B|, t)$

MERGE: At round $4\Delta t + 3\Delta$, every validator v_i merges its view with its buffer, *i.e.*, $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$, and sets $\mathcal{B}_i \leftarrow \emptyset$.

6 Analysis

6.1 Synchrony

Throughout this part of the analysis, we assume that $< \frac{n}{3}$ validators are ever slashable for equivocation, by which here we mean signing multiple head votes for a single slot, rather than violating $\mathbf{E_1}$. Observe that, in RLMD-GHOST with fast confirmations, this assumption 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 canonical chain. On the other hand, the protocol we present here utilizes confirmations as a prerequisite for justification, and justification does affect the canonical chain, since HFC filters out branches conflicting with the latest justified block. Therefore, we require this assumption for all of the properties which we are going to prove. To avoid stating it repeatedly, we further restrict η-compliant executions to those executions in which the assumption holds.

Now, we state without proof a property that follows from the usage of the view-merge technique, since it enables proposers to synchronize the view of honest voters with theirs. It corresponds to Lemma 2 as

[5]Were we to change the confirmation rule so that the latest justified block is always confirmed, the target block for FFG votes would then always simply be the *tip of* chAva. Given that we take confirmation as a prerequisite for justification, to ensure that the finality gadget does not interfere with the security of the available chain, this is a very natural choice, because it is the highest block which we can attempt to justify. Changing the confirmation rule in this manner is entirely possible, precisely because a justified block must have been confirmed by at least one honest validator, so it is safe to confirm it. We only decide to not do so in order to minimize the difference between this protocol and RLMD-GHOST, which simplifies proving that they are equivalent under synchrony, as we do in Theorem 3

Algorithm 3 Propose-vote-merge protocol for validator v_i

```
1: State
2:     $\mathcal{V}_i \leftarrow \{\mathcal{B}_{\text{genesis}}\}$: view of validator $v_i$
3:     $\mathcal{B}_i \leftarrow \emptyset$: buffer of validator $v_i$
4:     $\mathsf{ch}_i \leftarrow B_{\text{genesis}}$: canonical chain of validator $v_i$
5:     $t \leftarrow 0$: the current slot
6:     $r \leftarrow 0$: the current round
   PROPOSE
7: at $r = 4\Delta t$ do
8:     if $v_i = v_p^t$ then
9:         $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
10:        $\mathcal{B}_i \leftarrow \emptyset$
11:        $B' \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
12:        $B \leftarrow \mathsf{NewBlock}(B')$
13:        $\mathsf{ch}_i \leftarrow B$
14:        send message [PROPOSE, $B$, $\mathcal{V}_i \cup \{B\}$, $t$, $v_i$] through gossip
   HEAD-VOTE
15: at $r = 4\Delta t + \Delta$ do
16:     $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
17:     send message [HEAD-VOTE, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$] through gossip
   CONFIRM AND FFG-VOTE
18: at $r = 4\Delta t + 2\Delta$ do
19:     $B_{\text{fast}} \leftarrow B_{\text{genesis}}$
20:     $S_{\text{fast}} \leftarrow \{B \prec \mathsf{ch}_i \colon |\{v_i \colon \exists B' \succ B\ [\text{HEAD-VOTE}, B', t, v_i] \in \mathcal{B}_i\}| \geq \frac{2}{3}n\}$
21:     if $S_{\text{fast}} \neq \emptyset$ then:
22:         $B_{\text{fast}} \leftarrow \arg\max_{S_{\text{fast}}} |B|$
23:     if $\neg(B_{\text{fast}} \prec \mathsf{chAva}_i \wedge \mathsf{ch}_i^{\lceil \kappa} \prec \mathsf{chAva}_i)$ then:
24:         $\mathsf{chAva}_i \leftarrow \arg\max_{\mathsf{ch} \in \{\mathsf{ch}_i^{\lceil \kappa}, B_{\text{fast}}\}} |\mathsf{ch}|$
25:     $\mathcal{C}_{\text{target}} \leftarrow (\arg\max_{B \in \{LJ_i, \mathsf{chAva}_i\}} |B|, t)$
26:     send message [FFG-VOTE, $\mathcal{LJ}_i$, $\mathcal{C}_{\text{target}}$, $v_i$] through gossip
   MERGE
27: at $r = 4\Delta t + 3\Delta$ do
28:     $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
29:     $\mathcal{B}_i \leftarrow \emptyset$
30: upon receiving a gossiped message [PROPOSE, $B$, $\mathcal{V}$, $t$, $v_p^t$] do
31:     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
32:     if $r \in [4\Delta t, 4\Delta t + \Delta]$ then
33:         $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$
34: upon receiving a gossiped message $V =$ [HEAD-VOTE, $B$, $t'$, $v_i$] from $v_i$ do
35:     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{V\}$
36: upon receiving a gossiped block $B =$ (BLOCK, $b$, $t'$, $v_i$) from $v_i$ do
37:     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
```

presented by D'Amato and Zanolini [8], with an addition regarding synchronization of the latest justified checkpoint.

Lemma 1. *Suppose that t is an honest slot with an active proposer and that network synchrony holds in rounds $[4\Delta t - \Delta, 4\Delta t + \Delta]$. Say the proposed block is B, and the latest justified checkpoint in the view of the proposer is $\mathcal{LJ}_p$. Then, at round $4\Delta t + \Delta$, all active validators broadcast a head vote for the honest proposal B of slot t. Moreover, $\mathcal{LJ}_i = \mathcal{LJ}_p$ for any such active validator v_i.*

Our single slot finality protocol 3 implements the HFC fork-choice function (Algorithm 2), dealing with checkpoints and justifications. However, one could implement it also with different fork-choice functions. In particular, if we substitute HFC with RLMD-GHOST (with expiry period of length η), *i.e.*, if we ignore justifications and consider the *vanilla* fork-choice function introduced by D'Amato and Zanolini [8], then the resulting protocol is equivalent to the $\mathsf{RLMD\text{-}GHOST}$ protocol with fast confirmation (Appendix B [8]), because FFG votes have no effect at all, and as such it is η-reorg-resilient, and η-dynamically-available. Moreover, the following two results about fast confirmations also apply.

Theorem 1 (Reorg resilience of fast confirmations)**.** *Consider an η-compliant execution. A block fast confirmed by an honest validator at a slot t after GST is always in the canonical chain of all active validators at rounds $\geq 4\Delta(t+1)+\Delta$.*

Theorem 2 (Liveness of fast confirmations)**.** *An honest proposal B from a slot t after $\mathsf{GST}+\Delta$ in which $|H_t| \geq \frac{2}{3}n$ is fast confirmed by all active validators at round $4\Delta t+\Delta$.*

We show that, under synchrony, *i.e.*, with $\mathsf{GST}=0$, these properties are preserved by our justification-respecting protocol, which uses HFC instead. To do so, we show that η-compliant executions with $\mathsf{FC}=$ RLMD-GHOST (with expiry period of length η) and with $\mathsf{FC}=$ HFC are *equivalent*, meaning the transcript is the same regardless of which fork-choice function is used. All properties of the protocol with $\mathsf{FC}=$ RLMD-GHOST in such η-compliant executions then also apply to the protocol with $\mathsf{FC}=$ HFC. In particular, it is also η-reorg-resilient and η-dynamically-available, and it also satisfies reorg resilience and liveness of fast confirmations, *i.e.*, Theorems 1 and Theorem 2. From now on, when presenting $\mathsf{FC}=$ RLMD-GHOST, we assume an expiry period of length η. This implies that also $\mathsf{FC}=$ HFC has the same expiry period.

Theorem 3 (Execution equivalence)**.** *Consider an η-compliant execution with $\mathsf{GST}=0$ and $\mathsf{FC}=$ HFC, and the same execution with $\mathsf{FC}=$ RLMD-GHOST. The two transcripts correspond exactly.*

Proof. Since the only difference between the two protocols is the fork-choice function FC, the two transcripts correspond as long as the outputs of HFC and RLMD-GHOST obtained by active validators are always the same in the two executions. FC is used only twice in Algorithm 3, in Line 11 for proposing, and in Lines 16-17, with the same input, for broadcasting a head vote. We are going to prove by induction that the canonical chain of an active validator *at any voting round* is the same in both executions. Since Line 16 sets $\mathsf{ch}_i \leftarrow \mathsf{FC}(V_i, t)$, and this value is the same as in Line 17, we only need to show that the fork-choice output in Line 11 coincides in the two executions as well. In Line 17, an honest validator uses the fork-choice output to construct their head vote, so head votes correspond in both executions. Moreover, the view-merge property applies in both executions, so honestly proposed blocks correspond to the honest head votes from their slot. Therefore, head votes coinciding in the two executions implies that honestly proposed blocks coincide as well. Since honestly proposed blocks extend the output of the fork-choice at Line 11, this output is then also the same in both executions, completing the proof. We now carry out the induction.

Induction hypothesis: At any slot $t' \leq t$ and for $r = 4\Delta t' + \Delta$, ch_i^r coincides in both executions, for any active validator i.

Base case: In rounds $[0, \Delta]$, the two executions are exactly the same, because the only justified checkpoint is B_{genesis}, so HFC $=$ RLMD-GHOST. Therefore, the statement holds for $t=0$.

Inductive step: Suppose now that the statement holds for t, and consider round $r = 4\Delta(t+1)+\Delta$. Consider an active validator v_i with view $\mathcal{V}_i$ at round r, and latest justified block $B = LJ(\mathcal{V}_i)$. Let t' be minimal such that there exists a justified checkpoint $\mathcal{C} = (B, t')$, *i.e.*, slot t' is the first slot in which block B was justified. The supermajority link with target $\mathcal{C}$ contains at least one FFG vote from an honest validator v_k. By minimality of t', B could not have been already justified in the view of v_k when broadcasting an FFG vote at slot t'. Therefore, by Lines 25-26 of Algorithm 3, it must be the case that $B \prec \mathsf{chAva}_k$ at round $4\Delta t' + 2\Delta$, *i.e.*, that it had been confirmed by v_k. If it was fast confirmed at a slot $\leq t'$, then, in the execution with $\mathsf{FC}=$ RLMD-GHOST, Theorem 1 implies that $B \prec \mathsf{ch}_j^{r'}$ for all active validators v_j at any round $r' \geq 4\Delta(t'+1)+\Delta$, and so in particular that $B \prec \mathsf{ch}_i^r$, since $t > t'$. If instead $B \prec \mathsf{ch}_k^{\lceil\kappa}$ at round $4\Delta t' + 2\Delta$, *i.e.*, B is confirmed by v_k due to being κ-deep in its canonical chain, then with overwhelming probability there exists a pivot slot $t'' \in [t'-\kappa, t')$ (Lemma 3 [8]), with proposed block B'. In the execution with $\mathsf{FC}=$ RLMD-GHOST, η-reorg-resilience then implies that $B' \prec \mathsf{ch}_j^{r'}$ for all active validators v_j at any round $r' \geq 4\Delta t'' + \Delta$. In particular, $B' \prec \mathsf{ch}_k^{r'}$ at round $r' = 4\Delta t' + 2\Delta$, and $B' \prec \mathsf{ch}_i^r$. The former implies $B \prec B'$, since $B.t \leq t' - \kappa \leq B'.t$, and we then have $B \prec B' \prec \mathsf{ch}_i^r$. Regardless of how B has been confirmed by v_k, we have $B \prec \mathsf{ch}_i^r$. Therefore, $LJ(\mathcal{V}_i) = B \prec$ RLMD-GHOST$(\mathcal{V}_i, t+1)$, which in turn implies $\text{RLMD-GHOST}(\mathcal{V}_i, t+1) = \text{RLMD-GHOST} \circ \mathsf{FIL}_{\text{FFG}}(\mathcal{V}_i, t+1) = \text{HFC}(\mathcal{V}_i, t+1)$. Therefore, after v_i updates its canonical chain ch_i at round r by setting $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}, t+1)$, with FC dependent on the execution, ch_i is the same in both executions. □

6.2 Partial synchrony

Throughout this section, we assume that $f < \frac{n}{3}$. First, we prove that the finalized chain is accountably safe, exactly as done in Casper [3]. Then, we show that honest proposals made after $\max(\mathsf{GST}, \mathsf{GAT}) + \Delta$ are justified within their proposal slot, which implies liveness of the finalized chain. This, together with Theorem 3, show that our protocol is an η-secure ebb-and-flow protocol.

Theorem 4 (Accountable safety)**.** *The finalized chain is accountably safe,* i.e.*, two conflicting finalized blocks imply that at least $\frac{n}{3}$ adversarial validators can be detected to have violated either $\mathbf{E_1}$ or $\mathbf{E_2}$.*

Proof. We assume throughout that there are no double justifications, *i.e.*, there are no checkpoints $\mathcal{C} \neq \mathcal{C}'$ with $\mathcal{C}.t = \mathcal{C}'.t$, and we refer to this as the non-equivocation assumption. If that's not the case, clearly $\geq \frac{n}{3}$ validators are slashable for violating $\mathbf{E_1}$. Consider two conflicting finalized blocks B and B'. By definition, there are also finalized checkpoints $\mathcal{C}$ and $\mathcal{C}'$ with $B = \mathcal{C}.B$, $B' = \mathcal{C}'.B$. Say $\mathcal{C}$ is finalized by the chain of supermajority links $(B_{\text{genesis}},\ 0) \to \mathcal{C}_1 \cdots \to \mathcal{C}_k = \mathcal{C} \to \mathcal{C}_{k+1}$, with $\mathcal{C}_{k+1}.t = \mathcal{C}.t + 1$, and $\mathcal{C}'$ by the chain $(B_{\text{genesis}},\ 0) \to \mathcal{C}'_1 \cdots \to \mathcal{C}'_{k'} = \mathcal{C}' \to \mathcal{C}'_{k'+1}$, with $\mathcal{C}'_{k'+1}.t = \mathcal{C}'.t + 1$. Let $t_i = \mathcal{C}_i.t$, and $t'_i = \mathcal{C}'_i.t$. By the non-equivocation assumption, $t_k \neq t'_k$, and without loss of generality we take $t_k < t'_k$. Let $j = \min\{i \leq k' \colon t_k < t'_i\}$, so $t_k < t'_j \leq t'_{k'}$, and $t'_{j-1} \leq t_k$ by minimality of t'_j. By the non-equivocation assumption, $t'_j = t_{k+1}$ implies that $\mathcal{C}_{k+1} = \mathcal{C}'_j$. We then have $B = \mathcal{C}.B \prec \mathcal{C}_{k+1}.B = \mathcal{C}'_j.B \prec \mathcal{C}'.B = B'$, contradicting that B and B' are conflicting. Therefore, $t'_j > t_k + 1 = t_{k+1}$ as well. Similarly, $t'_{j-1} < t_k$. Therefore, we have $t'_{j-1} < t_k < t_{k+1} < t'_k$, *i.e.*, $\mathcal{C}'_{j-1}.t < \mathcal{C}_k.t < \mathcal{C}_{k+1}.t < \mathcal{C}'_j.t$. The intersection of the two sets of voters of the supermajority links $\mathcal{C}_k \to \mathcal{C}_{k+1}$ and $\mathcal{C}'_{j-1} \to \mathcal{C}'_j$ contains at least $\frac{n}{3}$ validators, which are then slashable for violating $\mathbf{E_2}$. □

Lemma 2. *If an honest proposer v_p proposes a block B at a slot t after* $\max(\mathsf{GST}, \mathsf{GAT}) + \Delta$*, and the latest justified checkpoint in the view of the proposer is $\mathcal{LJ}_p$, then the checkpoint (B, t) is justified in all honest views at round $4\Delta t + 3\Delta$, by supermajority link $\mathcal{LJ}_p \to (B, t)$.*

Proof. Since t is after $\mathsf{GAT} + \Delta$, all $> \frac{2}{3}n$ honest validators are awake since at least round $4\Delta t - \Delta$, so at slot t they have completed the joining protocol and are active. Moreover, the view-merge property applies to all of them. Consider now an honest validator v_i. By the view-merge property, v_i broadcasts a head vote for B at round $4\Delta t + \Delta$. Also by the view-merge property, $\mathcal{LJ}_i = \mathcal{LJ}_p$ at round $4\Delta t + \Delta$, but $\mathcal{LJ}_i$ does not change until round $4\Delta t + 3\Delta$, since $\mathcal{V}_i$ does not. Therefore, $\mathcal{LJ}_i = \mathcal{LJ}_p$ at round $4\Delta + 2\Delta$. By that round, all $\geq \frac{2}{3}$ honest head votes for B are received by all honest validators, including v_i. Since also $B \prec \mathsf{ch}_i$, v_i fast confirms B, and thus broadcasts an FFG vote $\mathcal{LJ}_i \to (B, t) = \mathcal{LJ}_p \to (B, t)$. All honest validators receive such votes by round $4\Delta t + 3\Delta$, and merge them into their view then. Therefore, checkpoint (B, t) is justified in all honest views at that round. □

Theorem 5 (Liveness)**.** *Consider two consecutive honest slots t and $t+1$ after* $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$*. The block B proposed at slot t is finalized at the end of slot $t+1$.*

Proof. By Lemma 2, (B, t) is justified in all honest views at round $4\Delta t + 3\Delta$. Since at the beginning of slot $t+1$ there cannot be any justified checkpoint with slot $> t$, and there cannot be any other justified checkpoint with slot t, (B, t) is therefore the latest justified block in the view of the proposer of slot $t+1$. B is then canonical in its view, and it proposes a block B' which extends B. Again by Lemma 2, $(B', t+1)$ is justified in all honest views at round $4\Delta(t+1) + \Delta$, by the supermajority link $(B, t) \to (B', t+1)$. Therefore, B is finalized in all honest views. □

7 Single slot finality

The protocol from Algorithm 3 is a an η-secure ebb-and-flow protocol (satisfying η-dynamic-availability instead of dynamic availability) which (at best) finalizes a block in every slot, but it does not achieve *single* *slot finality*, *i.e.*, it cannot finalize a proposal *within its proposal slot*. At best, it lags behind by one slot, finalizing a proposal from slot t at the end of slot $t+1$. A straightforward extension of our protocol which achieves single slot finality is one with 5Δ rounds per slot, allowing for an additional FFG voting phase. This would be very costly in Ethereum, for two reasons. First, it would in practice significantly increase the slot time, because each voting round requires aggregating hundreds of thousands (if not millions) of BLS signatures, likely requiring a lengthier multi-step aggregation process. Moreover, it would be expensive in terms of bandwidth consumption and computation, because such votes would have to all be gossiped and verified by each validator, costly even if already aggregated.

We decide to describe here an alternative way to enhance to protocol for the purpose of achieving single slot finality, without suffering from these drawbacks. We introduce a new type of message, *acknowledgments*, and a new slashing condition alongside them. Acknowledgments do not influence the protocol in any way, except in case of slashing, and are mainly intended to be consumed by external observers which want to have the earliest possible finality guarantees. Therefore, they do not need to be gossiped to and verified by all validators. They can then simply be gossiped in smaller sub-networks (similar to the *attestation subnets* which Ethereum employs today), requiring limited bandwidth and verification resources. If an observer wants to have faster finality guarantees than they could have by simply following the chain or listening to the global gossip, they can opt to participate in all such sub-networks, and collect all acknowledgments. As doing so is permissionless, it can also be expected that aggregate acknowledgments, or equivalent proofs, might become available through some other channels. In essence, an acknowledgment is just a self-imposed constraint, preventing a validator which knows of a justification from later ignoring it.

Acknowledgment An acknowledgment is a tuple $[\textsc{Ack}, \mathcal{C}, t, v]$, where $\mathcal{C}$ is a checkpoint with $\mathcal{C}.t = t$. We also refer to this as an acknowledgment *of* $\mathcal{C}$. A *supermajority acknowledgment of* $\mathcal{C}$ is a set of $\geq \frac{2}{3}n$ distinct acknowledgments of $\mathcal{C}$.

Acknowledgment broadcasting protocol At round $4\Delta t + 3\Delta$, after merging the buffer, validator v_i broadcasts the acknowledgment $[\textsc{Ack}, \mathcal{LJ}_i, t, v_i]$ *if* $\mathcal{LJ}_i.t = t$, *i.e.*, if $\mathcal{LJ}_i$ has been justified in the current slot.

Finality with acknowledgments An observer which receives a supermajority acknowledgment of a *justified* checkpoint $\mathcal{C}$ *may* consider $\mathcal{C}$ to be finalized.

Slashing rule for finality voting When validator v_i broadcasts an acknowledgment $(\mathcal{C}, t)$, it *acknowledges* that, at the end of slot t, it knows about $\mathcal{C}$ being justified. Since the FFG voting rules prescribe that the source of an FFG vote should be the latest known justified checkpoint, subsequent FFG votes with a source whose slot is $< t$ constitute a provable violation, which is analogous to surround voting. Accordingly, we formulate a third slashing rule, which ensures that finality via a supermajority acknowledgment is accountably safe. Validator v_i is slashable for an FFG vote $(\mathcal{C}_1, \mathcal{C}_2)$ and an acknowledgment $(\mathcal{C}, t)$, if they satisfy $\mathbf{E_3}$, *i.e.*, $\mathcal{C}_1.t < \mathcal{C}.t < \mathcal{C}_2.t$. In other words, the link $\mathcal{C}_1 \to \mathcal{C}_2$ *surrounds* the acknowledged checkpoint.

Theorem 6 (Accountable safety with acknowledgments)**.** *The finalized chain is accountably safe even when it is updated via acknowledgments as well,* i.e., *two conflicting finalized checkpoints imply that more than $\frac{n}{3}$ adversarial validators can be detected to have violated* $\mathbf{E_1}$*,* $\mathbf{E_2}$ or $\mathbf{E_3}$*.*

Proof. The proof largely follows that of Theorem 4. We again consider two conflicting finalized blocks B and B', and corresponding finalized checkpoints $\mathcal{C}$ and $\mathcal{C}'$. Regardless of whether finalization is through a supermajority link or a supermajority acknowledgment, $\mathcal{C}$ and $\mathcal{C}'$ have to be justified, by chains of supermajority links $(B_{\text{genesis}},\ 0) \to \mathcal{C}_1 \cdots \to \mathcal{C}_k = \mathcal{C}$ and $(B_{\text{genesis}},\ 0) \to \mathcal{C}'_1 \cdots \to \mathcal{C}'_{k'} = \mathcal{C}'$. Let $t_i = \mathcal{C}_i.t$, and $t'_i = \mathcal{C}'_i.t$. By the non-equivocation assumption, we again have $t_k \neq t'_k$, and without loss of generality we take $t_k < t'_k$. As before, we let $j = \min\{i \leq k' \colon t_k < t'_i\}$, so $t_k < t'_j \leq t'_{k'}$, and $t'_{j-1} \leq t_k$ by minimality of t'_j. Moreover, also by the non-equivocation assumption, $t'_{j-1} < t_k$. If $\mathcal{C}$ is finalized through a supermajority link, the proof of Theorem 4 already shows that at least $\frac{n}{3}$ validators must have violated $\mathbf{E_2}$, and it is still applicable here because it does not use the last supermajority link in the chain finalizing $\mathcal{C}'$ (which may or may not exist here). If instead $\mathcal{C}$ is finalized through a supermajority acknowledgment, *i.e.*, there are $\frac{2}{3}n$ acknowledgments of $\mathcal{C}$, then at least $\frac{n}{3}$ validators have violated $\mathbf{E_3}$, because $\mathcal{C}'_{j-1}.t < \mathcal{C}.t < \mathcal{C}'_j.t$. □

Theorem 7 (Single Slot Finality)**.** *An honest proposal from a slot t after* $\max(\mathsf{GST}, \mathsf{GAT}) + 4\Delta$ *is finalized in round* $4\Delta(t+1)$ *by a supermajority acknowledgment.*

Proof. Say the honestly proposed block is B. By Lemma 2, $\mathcal{C} = (B, t)$ is justified in all honest views at round $4\Delta t + 3\Delta$. Therefore, all honest validators broadcast an acknowledgment of $\mathcal{C}$. Any observer which listens for acknowledgments would receive all such messages by rounds $4\Delta(t+1)$, and thus possess a supermajority acknowledgment of $\mathcal{C}$. They may then consider $\mathcal{C}$, and thus also B, to be finalized. ☐

References

[1] Aditya Asgaonkar. Remove equivocating validators from fork choice consideration. URL: `https://github.com/ethereum/consensus-specs/pull/2845`.

[2] Vitalik Buterin. Paths toward single-slot finality, 2023. URL: `https://notes.ethereum.org/@vbuterin/single_slot_finality`.

[3] 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`.

[4] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and Casper. *arXiv:2003.03052 [cs.CR]*, 2020. URL: `https://arxiv.org/abs/2003.03052`.

[5] 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. URL: `https://dl.acm.org/citation.cfm?id=296824`.

[6] Francesco D'Amato. View-merge as a replacement for proposer boost. URL: `https://ethresear.ch/t/view-merge-as-a-replacement-for-proposer-boost/13739`.

[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. `arXiv:2209.03255`, `doi:10.48550/arXiv.2209.03255`.

[8] Francesco D'Amato and Luca Zanolini. Recent latest message driven ghost: Balancing dynamic availability with asynchrony resilience, 2023. URL: `https://arxiv.org/abs/2302.11326`, `doi:10.48550/ARXIV.2302.11326`.

[9] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[10] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Byzantine consensus under fully fluctuating participation. *IACR Cryptol. ePrint Arch.*, page 1448, 2022. URL: `https://eprint.iacr.org/2022/1448`.

[11] 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`.

[12] Joachim Neu, Ertem Nusret Tas, and David Tse. A balancing attack on Gasper, the current candidate for Eth2's beacon chain. URL: `https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079`.

[13] 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.

[14] 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.

[15] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *International Conference on Financial Cryptography and Data Security*, FC '22, 2022. Forthcoming. URL: `https://arxiv.org/abs/2110.10086`.

[16] 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.

[17] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.

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