Recent Latest Message Driven GHOST: Balancing Dynamic Availability With Asynchrony Resilience

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

Luca Zanolini
Ethereum Foundation
luca.zanolini@ethereum.org

Abstract

Dynamic participation has recently become a key requirement to devise permissionless consensus protocols, as it adds a degree of robustness to events that include portions of participants going offline, preserving safety and liveness of such *dynamically available* protocols. This notion, formalized by Pass and Shi (ASIACRYPT 2017) with the *sleepy model*, has been implicitly adopted to model several blockchain protocols such as, for example, the Ethereum's consensus protocol, Gasper.

Neu, Tas, and Tse (S&P 2021) show that LMD-GHOST, the dynamic availability component of Gasper, is actually not secure even in a context of full-participation, *i.e.*, with all the validators online. Mitigations have shortly after been developed to cope with its problems, but the resulting protocol still falls short of achieving dynamic availability, motivating the research of more secure dynamically available protocols.

In this work we present RLMD-GHOST, a synchronous dynamically available protocol that does not lose safety during bounded periods of asynchrony. This protocol results appealing especially for practical systems, where strict synchrony assumptions might not always hold, contrary to what is generally assumed with standard synchronous protocols. Moreover, we introduce the *generalized sleepy model*, in which our results will be proved. This model takes up from the original sleepy model presented by Pass and Shi and extends it with more generalized and stronger constraints in the corruption and sleepiness power of the adversary. This allows us to explore a broad space of dynamic participation regimes which falls between complete dynamic participation and no dynamic participation, i.e., with every participant online, offering a foundation for the analysis of dynamic available protocols.

1 Introduction

With the advent of the blockchain era, dynamic participation became a key requirement to devise consensus protocols, as it adds a degree of robustness to events that include portions of participants going offline, preserving safety and liveness of such *dynamically available* protocols. This idea, introduced by the Bitcoin consensus protocol [15], has been first formalized by Pass and Shi [21] with the *sleepy model* of consensus. In this model, participants can be either online or offline, and their online status may change at any point during the protocol execution.

One limitation of dynamically available 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), and dynamically available protocols are generally assumed to be synchronous [21, 14, 13].

Neu *et al.* [18], motivated by formally understanding the Ethereum consensus protocol's [7] design goals, introduce the *partially synchronous sleepy model*, a model under which such protocol can be expressed. In a partially synchronous model in the sleepy model, (i) before a global stabilization time (GST) message delays are chosen by an adversary, and after that 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. In this model, Neu *et al.* [18] precisely formalize the notion of a dynamically available protocol: a protocol that outputs a ledger that, if $\mathsf{GST} = 0$, *i.e.*, under synchrony, is guaranteed to be safe and live at all times, provided that at all times fewer than $\frac{1}{2}$ of the awake participants (or, validators) are adversarial. More generally, a (secure) 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 $\max\{\mathsf{GST}, \mathsf{GAT}\}$. In the context of Ethereum, the available ledger is output by LMD-GHOST [25], and the finalized ledger by the *finality gadget* Casper FFG [6]. In the process of formalizing the security requirements of [7], Neu *et al.* [18] show that the (original version of) LMD-GHOST is actually not secure even in a context of full-participation, *i.e.*, with all the validators online, by presenting a *balancing* attack [18, 23]. The proposer boost technique [5] was later introduced as a mitigation, though the resulting protocol falls short of being dynamically available (see Appendix C.1 in [10]).

To cope with the problems of LMD-GHOST, D'Amato *et al.* [10] devise Goldfish, a protocol that enjoys safety and liveness *under fully variable participation*, and thus that is dynamically available. Goldfish is based on two techniques, *view-merge* [9, 12] [1], which allows validators to join the sets of the messages they received (at some point during the execution) before making any protocol's decision, and *vote expiry*, where only messages received within a specific time window influence the protocol's behavior. The protocol can be securely combined with an accountability gadget [19], fulfilling the ebb-and-flow properties [18].

However, Goldfish is not considered practically viable to replace LMD-GHOST in Ethereum, due to its brittleness to temporary asynchrony: even a single slot of asynchrony can lead to a catastrophic failure, jeopardizing the safety of *any* previously confirmed block.

In this work, we overcome the limitation of Goldfish by presenting RLMD-GHOST, a protocol which generalizes both LMD-GHOST and Goldfish. As the former, RLMD-GHOST implements the latest message rule (LMD). As the latter, it implements view-merge and vote expiry. Differently from Goldfish, where only votes from the most recent slot are considered, RLMD-GHOST is parameterized by a *vote expiry period* η, *i.e.*, only messages from the most recent η slots are utilized. For $\eta = 1$, RLMD-GHOST reduces to Goldfish, and for $\eta = \infty$ to (a more secure variant of the original) LMD-GHOST.

The remainder of this work is structured as follows. In Section 2 we discuss related works. We present our system model in Section 3. In particular, we introduce the *generalized sleepy model*, in which our results will be proved. This model takes up from the original sleepy model presented by Pass and Shi [21] and extends it with more generalized and stronger constraints in the corruption and sleepiness power of the adversary. This allows us to explore a broad space of dynamic participation regimes which fall between complete dynamic participation and no dynamic participation, *i.e.*, with every participants online.

In Section 4 we recall LMD-GHOST, as originally presented, and formally prove its properties and limitations. In Section 5 we define a class of protocols, called *propose-vote-merge* protocols, aiming at grouping together LMD-GHOST, Goldfish, and RLMD-GHOST under an unique framework, so that properties proved for a generic propose-vote-merge protocol are applicable to all of them. Finally, we present a modified version of LMD-GHOST, implementing view-merge, and recall Goldfish. For both we analyze, in our generalized sleepy model, properties and limitation, paving the way to our protocol, RLMD-GHOST. It is worth noting that, as we show, LMD-GHOST is fundamentally incompatible with the notion of dynamic availability, as not expiring messages opens up to different vectors of attack.

In Section 6 we introduce RLMD-GHOST. Through this generalization of Goldfish and LMD-GHOST, we explore a trade-off between resilience to temporary asynchrony and dynamic availability. RLMD-GHOST with an expiry period $1 < \eta < \infty$ 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 the vote expiry period*, and satisfies an appropriate notion of dynamic availability, albeit weaker than the standard one. Limitations of RLMD-GHOST are presented in Appendix A.

Finally, in Appendix B we extend RLMD-GHOST achieving a faster confirmation time for proposals, *optimistically.* This is aligned with the way in which Goldfish also achieves fast confirmation, with the difference that we do not require an increase in the length of the slots, due to using a slightly different proposer selection mechanism.

Conclusions are drawn in Section 7.

2 Related works

The sleepy model of consensus, introduced by Pass and Shi [21], abstracts the functioning implicitly introduced by the Bitcoin protocol [15], modeling a distributed system in which participants can be either online

[1] View-merge was first introduced in the Highway protocol [12], under the name of vote buffering, which is also the name used in Goldfish [10]

or offline; in other terms where the participation is *dynamic*. This creates a separation from the standard models considered in the literature, in which honest participants are assumed to be always online, and execute the protocol assigned to them. Moreover, Pass and Shi devise a consensus protocol that achieves safety and liveness, by assuming only that at any given time a majority of online participants are honest.

Buterin *et al* [7] devise Gasper, a proof-of-stake consensus protocol consisting of two protocols: Casper [6], a gadget on top of a block proposal mechanism whose role is to finalize blocks, and LMD-GHOST, a fork-choice function where participants (or validators) vote for blocks to signal support for those blocks. The Latest Message Driven Greediest Heaviest Observed Sub-Tree rule (LMD-GHOST) is a fork-choice function introduced by Zamfir *et al.* [25] while looking for a "correct-by-construction" consensus protocol. It is then used by the identically named protocol LMD-GHOST, as presented in [7]. LMD-GHOST is an adaptation of the original GHOST [24], in which the protocol considers only each validator's most recent vote (LMD), which is assumed to be the most meaningful. The sleepy model [21] can be applied [18] to encapsulate the behavior of Gasper.

Neu *et al* [18], motivated by formally understanding the Gasper [7] design goals, introduce the *partially synchronous sleepy model*, and define the desiderata of the Ethereum's consensus protocol through the notion of an *ebb-and-flow protocol*. In particular, a (secure) 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 $\max\{\mathsf{GST}, \mathsf{GAT}\}$. In the context of Gasper, the former is defined by LMD-GHOST [25], while the latter from Casper [6].

However, under a deeper analysis, Neu *et al* [18] show that LMD-GHOST is not dynamically available, by presenting an attack to its liveness.

Neu *et al.* [10] later introduce Goldfish, a simplified variant of LMD-GHOST, and prove that it is secure and reorg resilient in synchronous networks with dynamic participation, assuming a majority of the validators follows the protocol honestly. Goldfish is composable with finality gadgets and accountability gadgets, making it an ebb-and-flow protocol.

Other approaches to integrate dynamic participation within a consensus protocol have been recently devised [14, 13].

Momose-Ren [14] present a quorum-based atomic broadcast protocol in the sleepy model that simultaneously supports dynamic participation, albeit requiring periods of stable participation *for liveness*, and achieves constant latency. They extend the classic BFT approach from static quorum size to *dynamic quorum* size, *i.e.*, according to the current participation level, while preserving properties of static quorum. In particular, their protocol is built from a graded agreement protocol using a quorum-based design.

Malkhi-Momose-Ren [13] improves on the latency of Momose-Ren [14], and is live under fully fluctuating participation. They present a Byzantine atomic broadcast protocol in the sleepy model [21] with 3 round latency, but tolerating only one-third Byzantine nodes, rather than one-half. Moreover, they observe that all malicious nodes are always online in the sleepy model, and extend it to the *sleepy model with growing adversary*, which allows new malicious nodes to join the system.

3 Model and Preliminary Notions

3.1 System model

Validators. We consider a system of n *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.

Failures. 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.

Links. We assume that a best-effort gossip primitive that will reach all validators is available. 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. 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 k rounds, for a constant k. We are interested in the case $k = 3\Delta$, so our presentation will assume this length for slots, unless otherwise specified.

Sleepiness. The adversary $\mathcal{A}$ can decide for each round which honest validator is *awake* or *asleep* at that round. 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 5.4), after which they become *active* [10]. 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 [21].

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 a new block. 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 mechanism is not within the goals of this work, we assume the existence of a proposer selection mechanism satisfying the requirements of a Single Secret Leader Election (SSLE) scheme [3], *i.e.*, *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. Such a mechanism has been researched for usage in the Ethereum consensus protocol [11]. Note that satisfying fairness and uniqueness alone is trivial, *e.g.*, with a round-robin election, and that all analyzed protocols still maintain their properties with such a proposer mechanism, *if the adversary is static both in its choice of corruption and sleepiness schedules.* Moreover, note that a popular choice for a proposal mechanism is a VRF-based lottery, which satisfies both unpredictability and fairness. This is for example the case of Goldfish [10], a protocol which we also discuss in this work. Such a mechanism still satisfies uniqueness *in slots with an honest proposer and under synchrony*, in the sense that, when an honest and active validator is in possession of a winning ticket, and network synchrony holds, all active validators agree on the who the leader is. This approach is sufficient to retain all properties we are going to prove, *even with an adaptive adversary.*

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). There are validity conditions on messages, and we say that a view is valid if all messages within it are *verifiably valid within the view itself*, *i.e.*, all messages they reference are also contained in the view and themselves valid, *i.e.*, blocks contain a reference to a parent block, and verifying their validity requires also being able to verify its validity (and recursively that of the entire chain). We do not discuss questions of availability and validity further, and just leave it implicit that we only ever talk valid messages and views.

Blocks and chains. For two chains ch_1 and ch_2, we say $\mathsf{ch}_1 \preceq \mathsf{ch}_2$ 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 $\mathsf{ch}' \preceq \mathsf{ch}$ and A is the head of ch', we also say $\mathsf{ch}' \preceq B$ and $A \preceq B$.

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. In this work we are mainly interested in a particular class of fork-choice functions based on GHOST, which we denote with $\mathcal{G}_f$ and introduce in Section 4.2.

Terminology. We often use the terms *honest proposal*, *honest slot*, and *honest view* to refer to a block proposal made by an honest validator, a slot with an honest proposer, and a view of an honest validator, respectively. We also use the term *pivot slot* to refer to a slot in which the proposer is active at proposal time, *i.e.*, to a slot in which an honest proposal is made, and we say that such a slot has an *active proposer*. Finally, we say *honest voters of slot* t to refer to the active validators at the voting round $3\Delta t + \Delta$ of slot t.

3.2 Generalized sleepy model

We now specify how the adversary is constrained in using its corruption and sleepiness power. We do so by formulating first a one-parameter family of adversarial restrictions, which generalizes the usual sleepy model [21, 10], and then a two-parameters family, which generalizes it further by introducing, and accounting for, *bounded* periods of asynchrony.

3.2.1 τ-sleepy model

We denote with h_r the number of honest validators that are active at round r, with $h_0 > 0$ a lower bound on h_r, and with f_r the number of adversarial validators at round r. In the sleepy model [21], the adversary is constrained in its choice of sleepiness and corruption schedules by the requirement that awake validators outnumber adversarial validators in every round by a constant factor $c > 1$. As awake and active validators coincide in this model, the requirement is $h_r > cf_r$.

D'Amato *et al.* [10] introduce the notion of active validator [2] and assume a modified condition, *i.e.*, $h_{r-3\Delta} > f_r$. In this condition, that is tailored for their protocol, $\mathsf{Goldfish}$, $h_{r-3\Delta}$ is considered instead of f_r because, if r is a voting round in $\mathsf{Goldfish}$, validators corrupted after round r can still retroactively cast votes for that round, which (votes) are relevant until 3Δ rounds later. In practice, all that is required is that $h_{3\Delta(t-1)+\Delta} > f_{3\Delta t+\Delta}$ for any *slot* t, *i.e.*, the condition only needs to hold for *voting rounds*.

In this work, we follow this distinction between awake and active validators, and we use H_t and A_t, for a slot t, to refer to the set of active and adversarial validators at round $3\Delta t + \Delta$, respectively [3]. Moreover, we define $H_{s,t}$ as 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 $H_i \coloneqq \emptyset$). We then require that, for some fixed parameter $1 \leq \tau \leq \infty$, the following condition, which we refer to as *τ-sleepiness at slot* t, holds for any slot t after GST:

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

We refer to the sleepy model in which the adversary is constrained by τ-sleepiness after GST as the *τ-sleepy model.* Note that, for $\tau = 1$, this reduces to the sleepy model from $\mathsf{Goldfish}$, as this condition reduces to the majority condition $h_{r-3\Delta} > f_r$ of $\mathsf{Goldfish}$ for voting rounds $r = 3\Delta t + \Delta$, because $H_{t-1,t-2} = \emptyset$. We therefore also refer to the 1-sleepy model simply as sleepy model.

Definition 1 (τ-compliant execution)**.** An execution in the partially synchronous network model is *τ-compliant* if it satisfies τ-sleepiness. We refer to the set of such protocol executions as E_τ. In other words, the τ-sleepy model restricts the allowable set of executions to τ-compliant executions, *i.e.*, to E_τ, constraining the adversarial sleepiness and corruption power accordingly. We refer to 1-compliant executions simply as compliant executions.

[2] There, validators which have completed the joining protocol are simply called awake, and validators which are executing the joining protocol are called dreamy

[3] Recall that we focus on protocols of slot length 3Δ rounds. For the protocols introduced in Section 5, $3\Delta t + \Delta$ is a voting round, as in $\mathsf{Goldfish}$.

Hierarchy of τ-sleepy models. As τ increases, so do the restrictions that τ-sleepy models put on the adversary, *i.e.*, the τ_1-sleepy model makes stronger assumptions than the τ_2-sleepy model for $\tau_1 > \tau_2$. Another way to say this is that $\tau_1 > \tau_2$ implies $E_{\tau_1} \subset E_{\tau_2}$. This is immediate from τ-sleepiness, *i.e.*, Equation 1. The only term that depends on τ is $|H_{t-\tau,t-2} \setminus H_{t-1}|$, which is monotonically increasing in τ. Therefore, τ_1-sleepiness implies τ_2-sleepiness, so a τ_1-compliant execution is also a τ_2-compliant execution. In other words, increasing τ makes it harder for τ-sleepiness to be satisfied, *i.e.*, it constrains the adversarial corruption and sleepiness power more.

As we mentioned, $\tau = 1$ corresponds to sleepy model from Goldfish, 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 $3\Delta(t-1)+\Delta$, 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.

Relationship with eventually stable participation. Momose and Ren [14] introduce the concept of T-eventually stable participation. The authors define a validator v_i to be *insomniac* during a time interval $[t, t']$ if v_i is always awake during $[t, t']$. For any time t after some stabilization time, Momose and Ren require that honest and insomniac validators during $[t, t+T]$ are a majority of all of the validators which are awake during $[t, t+T]$. If we instead require a majority of always *active* validators, then $3\Delta\tau$-eventually stable participation implies τ-sleepiness. Let the set of honest validators that are always active in the interval $[3\Delta(t-\tau)+\Delta, 3\Delta t+\Delta]$ be S_1, and the set of validators which are active (here including the adversarial ones) at any point during those rounds be S_2. We have that $S_1 \subseteq H_{t-1}$, $A_t \cup (H_{t-\tau,t-2} \setminus H_{t-1}) \subseteq S_2$ and $S_1 \subseteq S_2$. Moreover, $A_t \cup (H_{t-\tau,t-2} \setminus H_{t-1})$ and S_1 are disjoint, so $|A_t \cup (H_{t-\tau,t-2} \setminus H_{t-1})| + |S_1| \leq |S_2|$. Therefore, $|S_1| > \frac{1}{2}|S_2|$ implies $|H_{t-1}| \geq |S_1| > |A_t \cup (H_{t-\tau,t-2} \setminus H_{t_1})|$, *i.e.*, τ-sleepiness is satisfied at slot t. The converse does not hold, so the two models are *not* equivalent. We choose to use τ-sleepiness because it is a minimal condition for the family of protocols which we analyze in Section 6.

3.2.2 (τ, π)-sleepy model

We generalize the τ-sleepy model further, by introducing the notion of a *temporary period of asynchrony of less than π slots*, abbreviated by π-tpa. We consider a partially synchronous network where synchrony holds after GST, *except for one such π-tpa*, for some $\pi \in \mathbb{N} \cup \{\infty\}$. We refer to this network model as *partially synchronous network with a* tpa. Since a 1-tpa is empty, this is a generalization of the usual partially synchronous network model, as any usual partially synchronous execution is an execution with a 1-tpa. We also specify a suitable notion of compliance for executions in this network model, which defines the (τ, π)-*sleepy model*, generalizing the τ-sleepy model.

Definition 2 (Temporary period of asynchrony)**.** We say that an interval (t_1, t_2) of consecutive slots after GST is a *temporary period of asynchrony*, abbreviated by tpa, if synchrony does not hold in (t_1, t_2). If $t_2 - t_1 \leq \pi$, we also refer to it as π-tpa.

Definition 3 ((τ, π)-compliant execution)**.** For $\tau > \pi$, or $\tau = \pi = \infty$, an execution in the partially synchronous network model with a tpa is (τ, π)-compliant if the tpa is in particular a π-tpa (t_1, t_2) and the following conditions hold:

- τ-sleepiness at slot t holds for $t \notin (t_1, t_2]$
- $|H_{t_1} \setminus A_t| > |A_t \cup (H_{t-\tau,t-1} \setminus H_{t_1})|$ for $t \in (t_1, t_2+1]$
- H_{t_1} are awake at round $3\Delta t_1 + 2\Delta$

We say that a (τ, π)-compliant execution satisfies (τ, π)-sleepiness, and call the set of (τ, π)-compliant executions $E_{\tau,\pi}$. The (τ, π)-sleepy model restricts the allowable set of executions to $E_{\tau,\pi}$.

During the π-tpa, the network is asynchronous and *all* honest validators can be asleep. On the other hand, there are more restrictions on the adversary corruption schedule, *i.e.*, the adversary cannot corrupt too many validators in H_{t_1}. This is because we rely on H_{t_1} to preserve the canonical chain throughout this period. Moreover, not too many honest validators can be woken up during this period, because waking up during asynchrony allows the adversary to manipulate their votes. Note that, $\forall t > t_2$, τ-sleepiness holds at slot t, and the network is synchronous. Therefore, if we take GST to be at slot t_2, this is also a τ compliant execution, so all results which hold for the τ-sleepy model after GST also hold for the (τ, π)-sleepy model after the tpa. As a consequence, unless otherwise specified, we will mainly consider the τ-sleepy model. The (τ, π)-sleepy model will be used when interested in analyzing the behaviour of a protocol under (bounded) asynchrony. In particular, we use it to define what it means for a protocol to be *resilient to (temporary) asynchrony* (Definition 7).

Hierarchy of (τ, π)-sleepy models. Like E_τ, $E_{\tau,\pi}$ is monotonically decreasing in τ, *i.e.*, $\tau_1 > \tau_2$ implies $E_{\tau_1,\pi} \subset E_{\tau_2,\pi}$. Moreover, it is monotonically *increasing* in π, *i.e.*, $\pi_1 < \pi_2$ implies $E_{\tau,\pi_1} \subset E_{\tau,\pi_2}$, because a π_1-tpa is also a π_2-tpa. For $\pi \leq 1$, a π-tpa is empty, and (τ, π)-compliance only requires τ-sleepiness at all slots, *i.e.*, $E_{\tau,\pi} = E_\tau$. The (τ, π)-sleepy model is then indeed a generalization of the τ-sleepy model. For $\pi = \infty$, a π-tpa can be an unbounded period of asynchrony starting after slot t_1, and (τ, π)-compliance is only defined for $\tau = \infty$ as well. It requires $|H_{t_1} \setminus A_\infty| > |A_\infty \cup (H_{0,\infty} \setminus H_{t_1})|$, where A_∞ and $H_{0,\infty}$ are defined in the obvious way as limits. $H_{t_1} \setminus A_\infty$ are the honest voters of slot t_1 which are never corrupted, and $H_{0,\infty}$ are all honest validators which ever vote. If all validators vote at least once in the entire execution, then the requirement simply becomes $|H_{t_1} \setminus A_\infty| > \frac{n}{2}$. We will see in Theorem 1 that this is precisely the condition which we require for $\mathsf{LMD\text{-}GHOST}$ to achieve asynchronous safety.

Aware validators. Given a π-tpa (t_1, t_2), we say a validator $v_i \in H_{t_1}$ is *aware at round* r for r in slots $(t_1, t_2]$ if v_i is active at round r. For r not in slots $(t_1, t_2]$, we say that v_i is aware at round r if it is active at round r. We motivate this notion after using it to define *asynchrony resilience* (Definition 7).

3.3 Security

Security Parameters. We largely follow here the notation and definitions of [10]. 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 $\mathrm{negl}(\kappa) + \mathrm{negl}(\lambda)$. Properties of cryptographic primitives hold except with probability $\mathrm{negl}(\lambda)$, *i.e.*, with overwhelming probability, but we leave this implicit in the remainder of this work.

Confirmed chain. Alongside the canonical chain, validators also keep track of a *confirmed chain*, which is the output of the protocol. We refer to the confirmed chain of validator v_i at round r as Ch_i^r (cf. ch_i^r for the canonical chain).

Definition 4 (Secure protocol [10])**.** We say that a protocol outputting a confirmed chain Ch is *secure* after time T_{sec}, and has confirmation time T_{conf} [4], 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 $\mathsf{Ch}_i^r \preceq \mathsf{Ch}_j^{r'}$ or $\mathsf{Ch}_j^{r'} \preceq \mathsf{Ch}_i^r$.
- **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 E_τ. A protocol satisfies τ-security if it satisfies τ-safety and τ-liveness.

Observe that, for $\tau_1 > \tau_2$, since the τ_1-sleepy model makes stronger assumptions than the τ_2-sleepy model, security in the τ_1-sleepy model is weaker than security in the τ_2-sleepy model, *i.e.*, *τ_2-security implies τ_1-security.* This is immediate from $E_{\tau_1} \subset E_{\tau_2}$, because τ_2-security is precisely security in all executions E_{τ_2}, which implies security in E_{τ_1}.

[4] 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 5 (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.

Definition 6 (Reorg resilience)**.** An execution in the partially synchronous network model satisfies *reorg resilience* if any honest proposal B from a slot t after $\mathsf{GST} + \Delta$ [5] is always in the canonical chain of all active validators at rounds $\geq 3\Delta t + \Delta$. A protocol is τ*-reorg-resilient* if all τ-compliant executions satisfy reorg resilience.

Definition 7 (Asynchrony resilience)**.** An execution in the partially synchronous network model with a tpa (t_1, t_2) satisfies *asynchrony resilience* if any honest proposal from a slot $t \leq t_1$ after $\mathsf{GST} + \Delta$ is always in the canonical chain of all *aware* validators at rounds $\geq 3\Delta t + \Delta$. A protocol is (τ, π)-asynchrony-resilient if all (τ, π)-compliant executions satisfy asynchrony resilience.

In Definition 3, we assume that validators H_{t_1} are also awake at round $3\Delta t_1 + 2\Delta$, so that they observe their own votes, *i.e.*, each validator in H_{t_1} has all honest votes from slot t_1 in their view going forward. They are then the only validators which we can require to see all honest proposals from before the tpa as canonical *during* the tpa. For example, we cannot require a validator which is asleep at slot t_1, but active at slot $t_1 + 1$, to see an honest proposal from slot t_1 as canonical, because asynchrony has already started and they might not have received the proposal at all. After the tpa, the requirement can again apply to all active validators. In other words, *we define asynchrony resilience as reorg resilience of proposals made before the* tpa*, in the views of aware validators.*

4 Prerequisites

In this section we recall the $\mathsf{LMD\text{-}GHOST}$ [7] protocol. We start by presenting the fork-choice function GHOST, the main building block of the fork-choice function of $\mathsf{LMD\text{-}GHOST}$, and of all the protocols we consider in this work. Moreover, we present properties and limitation of $\mathsf{LMD\text{-}GHOST}$, defining the starting point of this work.

4.1 GHOST

GHOST (Algorithm 1) is a fork-choice function [6] based on the fork-choice procedure introduced in [24] by Sompolinsky and Zohar, a greedy algorithm that grows a blockchain on sub-branches with the *most activity*. Except, this one is *vote-based* rather than *block-based*, *i.e.*, here we weigh sub-trees based on number of votes rather than blocks. Given a set of votes M, we define the *weight* function $w(B, M)$ to output the number of votes in M for B or descendants of B, *i.e.*, on the sub-tree rooted at B. Starting at the first block of the canonical chain, *i.e.*, B_{genesis}, and considering the set M of votes in $\mathcal{V}$, GHOST iterates over a sequence of blocks from $\mathcal{V}$, selecting as the next block the descendant of the current block with the highest weight. This continues until it reaches a block that does not have any descendant in $\mathcal{V}$, which is output. We now state and prove a simple property of the GHOST fork-choice, which we are going to repeatedly use throughout the work, without explicitly mentioning it, whenever wanting to prove that a block is in the canonical chain in some view.

Lemma 1. *Let $\mathcal{V}$ be a view in which over a majority of the votes are for a descendant of a block B. Then,* $\text{GHOST}(\mathcal{V}, t)$ *is a descendant of B,* i.e.*, B is in the canonical chain output by the* GHOST *fork-choice.*

Proof. Let M be all votes in $\mathcal{V}$. Consider any height less than or equal to the height of B. In any fork at such a height, there is one branch that contains B, and thus also the whole sub-tree rooted at B. Say the block on that branch at that height is B', and consider any competing sibling B''. Since over a majority of

[5] The reason why we require that slot t comes after $\mathsf{GST} + \Delta$, *i.e.*, that $\mathsf{GST} \leq 3\Delta t - \Delta$, can be found in the assumptions of the *view-merge property*, Lemma 2. It is a crucial component of the reorg resilience guarantees of the protocols we analyze, and it requires synchrony holding from round $3\Delta t - \Delta$ in order to hold at slot t.

[6] Observe that, in order for GHOST to be a fork-choice function, a slot t is included among the inputs, despite not being used in Algorithm 1.

the votes in M are for the sub-tree rooted at B, and all votes on the sub-tree rooted at B' are not votes on the sub-tree rooted at B'', $w(B', M) > \frac{|M|}{2} > w(B'', M)$. Thus, B' is selected by the GHOST fork-choice algorithm at that height. Therefore, $B \preceq \text{GHOST}(\mathcal{V}, t)$. □

Algorithm 1 GHOST Fork-Choice function

```
function GHOST($\mathcal{V}, t$)
    $B \leftarrow B_{\text{genesis}}$
    $M \leftarrow$ all votes in $\mathcal{V}$
    while $B$ has descendant blocks in $\mathcal{V}$ do
        $B \leftarrow \underset{B' \in \mathcal{V},\ \text{child of}\ B}{\arg\max}\ w(B', M)$
        // ties are broken according to a deterministic rule
    return $B$
```

4.2 Filtered GHOST

We define the family of GHOST-based fork-choice functions $\mathcal{G}_f$. A fork-choice function $\mathsf{FC} \in \mathcal{G}_f$ is characterized by a view filter FIL, which takes as input a view $\mathcal{V}$ and a slot t, and outputs $(\mathcal{V}', t)$, where $\mathcal{V}$ is another view such that $\mathcal{V}' \subseteq \mathcal{V}$. Then, $\mathsf{FC}(\mathcal{V}, t) \coloneqq \text{GHOST}(\mathsf{FIL}(\mathcal{V}, t))$, *i.e.*, $\mathsf{FC} \coloneqq \text{GHOST} \circ \mathsf{FIL}$. GHOST itself is contained in $\mathcal{G}_f$, characterized by the identity filter. Moreover, both LMD-GHOST and GHOST-Eph (the fork-choice function of the Goldfish protocol [10]) belong to $\mathcal{G}_f$: the former is characterized by a filter that removes all but the *latest* votes (see Section 5.7), and the latter by a filter $\mathsf{FIL}(\mathcal{V}, t)$ that removes all the *expired* votes from $\mathcal{V}$, *i.e.*, from slots $< t - 1$ (see Section 5.8.1). As we see in Section 6, both LMD-GHOST and GHOST-Eph are special cases of the RLMD-GHOST fork-choice, which is also a member of $\mathcal{G}_f$.

Equivocation discounting. All fork-choice functions we consider from now on implement a technique to deal with equivocations, used in the Ethereum protocol [2] and considered in Goldfish [10] as well, as a spam-resistance measure, *i.e.*, *equivocation discounting.* It consists of excluding votes from equivocating validators from one's view, before running the fork-choice function on it. We describe equivocation discounting using view filters. Consider the view filter FIL_{eq} such that $\mathsf{FIL}_{eq}(\mathcal{V}, t)$ removes all votes by *equivocating validators in* $\mathcal{V}$, *i.e.*, validators for which $\mathcal{V}$ contains multiple, equivocating, votes for some slot t. Given a fork-choice function $\mathsf{FC} \in \mathcal{G}_f$, characterized by the view filter FIL, we apply equivocation discounting to FC by composing FIL with FIL_{eq}, *i.e.*, by considering the fork-choice function $\mathsf{FC}_{eq} \coloneqq \text{GHOST} \circ \mathsf{FIL} \circ \mathsf{FIL}_{eq}$. All fork-choice functions considered in the following sections implement equivocation discounting through this composition, even if not explicitly stated.

4.3 LMD-GHOST

The Latest Message Driven Greediest Heaviest Observed Sub-Tree rule (LMD-GHOST) is a fork-choice function introduced by Zamfir *et al.* [25] while looking for a "correct-by-construction" consensus protocol. It is then used by the identically named protocol LMD-GHOST, as presented in [7]. LMD-GHOST is an adaptation of the original GHOST [24], in which the protocol considers only each validator's most recent vote (LMD), which is assumed to be the most meaningful. As already observed, LMD-GHOST belongs to the class $\mathcal{G}_f$ of fork-choice functions based on GHOST [24], and is characterized by a view filter $\mathsf{FIL}_{\text{lmd}}$. $\mathsf{FIL}_{\text{lmd}}(\mathcal{V}, t)$ 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. In the following section, we consider a propose-vote protocol in which proposals and votes are made according to the LMD-GHOST fork-choice (Algorithm 2), exactly as in Gasper [7]. On the other hand, we do *not* consider a partition of the validator set

[7] Algorithm 2 differs from the one specified as part of the Gasper protocol [7]. In [7], the set of votes which are considered are "the most recent attestations of the validators (one per validator)", which is not well-defined because it lacks a rule for picking one of several equivocating votes. The rule implemented by the Ethereum fork-choice specification was to only consider the first message seen for a certain validator and epoch, because the update rule for the stored latest messages required (and still requires) `target.epoch > store.latest_messages[i].epoch` [1]. This rule allows the adversary to split honest views

into committees, only one of which votes in every slot, *i.e.*, we do not consider subsampling. This is because we are interested in the asynchronous safety properties of LMD-GHOST, which are most easily described when all validators vote in every slot. Moreover, subsampling only adds further attack vectors, in particular more ways to perform the *ex-ante reorgs* we describe in Section 5.7.1.

Algorithm 2 LMD-GHOST Fork-Choice function

```
function LMD-GHOST($\mathcal{V}, t$)
    return GHOST($\mathsf{FIL}_{\text{lmd}}(\mathsf{FIL}_{\text{eq}}(\mathcal{V}, t))$)
function $\mathsf{FIL}_{\text{lmd}}(\mathcal{V}, t)$
    $\mathcal{V}' \leftarrow \mathcal{V}$ without all but the most recent (latest) votes of each validator
    return $(\mathcal{V}', t)$
function $\mathsf{FIL}_{\text{eq}}(\mathcal{V}, t)$
    $\mathcal{V}' \leftarrow \mathcal{V}$ without all votes by equivocators in $\mathcal{V}$
    return $(\mathcal{V}', t)$
```

4.3.1 Finality

We define the notion of a certificate with quorum Q, which is the basis for a notion of finality in LMD-GHOST. This notion is not explored in Gasper [7], which chooses to instead pair LMD-GHOST with Casper-FFG [6], a separate finality gadget. On the other hand, it is explored in the Highway Protocol [12], which first introduces vote buffering, the technique which we introduce in Section 5.3 as *view-merge*, in order to solve liveness issues of LMD-GHOST. In fact, the notion of m-finality which we describe below corresponds to that given by a $(m, 1)$-summit in the Highway protocol [8].

Definition 8 (Certificate)**.** Given a slot t, we define an *m-quorum certificate* for slot t for a block B as a set Q of at least m votes broadcast in t, all (votes) for some descendant of B, and all from distinct validators.

We now define a notion of finality with quorum Q of at least m votes, which gives safety against $f < m - \frac{n}{2}$ adversarial validators. For this, we require blocks to be allowed to contain votes, so that a vote for a block is also an acknowledgment of the votes it contains. The honest proposal behavior is then to include all possible votes from the previous slot.

Definition 9 (Finality)**.** A block B is *m-finalized*, with $m > \frac{n}{2}$, if for some slot t there exist:

1. an m-quorum certificate Q for slot t;
2. a block A from slot $t + 1$ containing Q; and
3. an m-quorum certificate Q' for slot $t + 1$ for *A from the same set of validators that Q is from.*

If these conditions hold we say that B is finalized at slot $t + 1$. Finally, the set of validators whose votes are in the two m-quorum certificates Q and Q' is called an *m-clique.*

On subjective finality. This notion of m-finality provides safety with adversarial resilience $f < m - \frac{n}{2}$, rather than the usual $f < 2m - n$, obtained with quorum size m. This is because validators at all times simply follow the fork-choice, and thus majority rule, so if $m - \frac{n}{2}$ validators in the m-clique *unjustifiably* change their vote later, even all honest validators might follow the new majority. Crucially, they do not require a new conflicting m-quorum certificate to do so, because m-finality is a subjective condition which can be detected by observers, rather than being part of the protocol. In particular, *there is no locking*, because there is no

permanently through the well timed release of equivocations, causing a permanent chain split [20]. Equivocation discounting was therefore implemented [2], and we consider here the resulting protocol.

[8] In the Highway protocol, votes include the whole current view of the validator that broadcast them. This is necessary to achieve finality with the usual fault tolerance $f < 2m - n$ with a quorum size m, whereas an $(m, 1)$-summit, like our notion of finality, gives safety up to $f < m - \frac{n}{2}$. Higher order summits (m, k) are considered, *i.e.*, longer sequences of quorum certificates, which achieve the usual fault-tolerance $f < 2m - n$ in the limit. Other than for this purpose, including an entire view in every vote is not necessary. If we stick to the notion of finality with $k = 1$, it is not necessary for votes to include a view, *i.e.*, the resulting protocol is still safe and live, albeit with a reduced fault tolerance. We show this in Section 5.7

quorum size prescribed by the protocol, which is what allows notions of finality with arbitrary quorum size $> \frac{n}{2}$ to coexist within it. For example, in Tendermint [4], a PBFT-style [8] protocol, an honest validator that has *pre-committed* will not *pre-vote* on anything else, *i.e.*, it is *locked* on the pre-commit, unless they see a newer $\frac{2n}{3}$-quorum, allowing it to *safely* release its lock. Violating safety then requires two conflicting $\frac{2n}{3}$-quorums, leading to the usual adversarial tolerance, whereas in our case the adversary always just has to produce a $\frac{n}{2}$-quorum (a majority), regardless of what the (subjectively) chosen quorum size is. Moreover, since the notion of finality is subjective in our context, liveness-favoring observers which do not foresee the right adversarial threshold might experience safety violations.

Accountable safety. This notion of finality also provides *accountability*. For an m-finalized block B to not be canonical in a view that contains the finalization evidence for it, it must be the case that $m - \frac{n}{2}$ validators have either equivocated in slot t or $t-1$, or broadcast a vote not for a descendant of B at a slot $> t+1$ *unjustifiably*, *i.e.*, *without being allowed to do so by their fork-choice function.* Validators involved in a conflicting finalization can be required to provide evidence for conflicting votes, which honest validators will always be able to do, and the chain of evidence can eventually be traced back to some unjustified voting. We do not discuss this aspect further, as we are mainly interested in analyzing the resistance to asynchronous periods of the latest message driven fork-choice.

4.3.2 Properties

We denote with $\widetilde{H}_t$ the set of honest voters of slot t that *are never corrupted*, *i.e.*, $\widetilde{H}_t := \lim_{s\to\infty} H_t \setminus A_s$. We refer to validators that are never corrupted as *permanently honest*, and to $\widetilde{H}_t$ as the *permanently honest voters of slot t*. In the following theorem we show that LMD-GHOST can achieve asynchronous safety, if a permanently honest majority of validators votes for the same block at a slot, and quickly observes those votes.

Theorem 1 (Asynchronous safety). *Consider a set S of more than $\frac{n}{2}$ permanently honest voters of slot t. Suppose that all validators in S broadcast a vote for a descendant of block B at slot t, and call this set of votes Q. Suppose also that Q is contained in every view of validators in S before voting in slot $t+1$. From then on, B is asynchronously safe, as it is always canonical in any view which contains Q.*

Proof. Consider any view $\mathcal{V}$ which contains Q. If for any validator in S, its latest message in $\mathcal{V}$ is for a descendant of B, then clearly B is canonical in $\mathcal{V}$, since $|S| > \frac{n}{2}$. Therefore, B is canonical in any view which contains Q, as long as it does not contain any vote broadcast by a validator in S at a slot $\geq t$, and which is not for a descendant of B. However, validators in S do not ever broadcast such votes, as we prove by induction on the slot where such a vote could be broadcast.

The base case for this proof by induction is slot t, and it holds because, by assumption, validators in S broadcast votes that are then contained in Q at slot t. Moreover, they broadcast no further votes at slot t, since permanently honest validators never equivocate. For the inductive step, consider some slot $s > t$. All votes from validators in S from slots $\in [t, s)$ are for descendants of B. By assumption, $s > t$ implies that the view of any validator in S at slot s contains Q, and so all latest messages from validators in S in such a view are from slots $\geq t$, and thus for descendants of B. As previously noted, this implies that B is canonical in this view, since $|S| > \frac{n}{2}$. Therefore, a vote broadcast at slot s (if any) is for a descendant of B, which concludes the proof. □

We now show that no two conflicting blocks can be m-finalized in LMD-GHOST, *i.e.*, *m-finality is safe*, if $f < m - \frac{n}{2}$ (recall that f is the maximum number of validators ever corrupted by the adversary).

Theorem 2 (Safety of finality). *Suppose $f < m - \frac{n}{2}$, and let B be a block m-finalized at slot $t+1$. B is always canonical in any view that contains the first m-quorum certificate of the two finalizing it. Moreover, no conflicting block A can be m'-finalized at a slot $\geq t$ for any $m' > \frac{n}{2}$. In particular, no two conflicting blocks can be m-finalized.*

Proof. Since $f < m - \frac{n}{2}$, the m-clique contains more than $\frac{n}{2}$ permanently honest validators. By definition of m-finality, at slot $t+1$ they all vote for a block which contains the first m-quorum certificate, Q, so in particular Q is contained in their view at the voting round of slot $t+1$. We can then apply Theorem 1 and conclude that B is canonical in any view which contains Q. If $\mathcal{V}$ is a view of one of the permanently honest validators in the m-clique, from the voting round of a slot $> t+1$, view $\mathcal{V}$ contains Q, and thus B is canonical in it. Therefore, no such validator votes for a block other than a descendant of B at any slot $> t+1$. They also do not do so at slots t and $t+1$, since honest validators do not equivocate. Since there are more than $\frac{n}{2}$ such validators, there cannot be any m'-quorum certificate for a conflicting block A from slots $\geq t$. This implies that A cannot be m'-finalized at any slot $\geq t$. If, in particular, $m' = m$ then, by the same argument, A being m-finalized at a slot $s < t$ implies that no conflicting block can be finalized at a slot $\geq s-1$, contradicting the fact that B is m-finalized at slot $t+1$. Therefore, no two conflicting blocks can be m-finalized. □

4.3.3 Limitations

We now recall some critical security issues of the original LMD-GHOST protocol. As mitigation, the proposer boost technique [5] has been proposed and implemented in the Ethereum protocol, with the goal of allowing honest proposers to steer the fork-choice towards the chain they see as canonical. In this work, we consider instead an alternative technique, *view-merge* (Sections 5.3), which similarly empowers honest proposers. In Section 5.7, we consider LMD-GHOST with the addition of view-merge, and show that it resolves the security issues described below.

Reorgs. A reorg is an event where a block that was part of the canonical chain becomes no longer part of the canonical chain because a competing block beat it out. Reorgs can occur naturally due to network latency, or maliciously, caused by an adversary who seeks to exploit reorgs for its own gain. Reorgs can be classified in two categories: *ex-post reorgs*, *i.e.*, an adversary observes a block which subsequently attempts to fork out, and *ex-ante reorgs*, *i.e.*, an adversary attempts to fork out a future block that is unknown to the adversary at the start of the attack.

LMD-GHOST, as shown by Schwarz-Schilling *et al.* [22], is prone to ex-ante reorgs, *even by a single validator*. In particular, an adversary, being the proposer for slot $t+1$, can perform a reorg of one block as we briefly describe. As the adversary is the proposer for slot $t+1$, in such slot it privately creates block B' on top of block B, *i.e.*, the block for slot t, and votes for it. Honest validators of slot $t+1$ do not see any block and thus vote for B. In the next slot, *i.e.*, slot $t+2$, an honest proposer publishes block B'' building on B block, which is the current head in their view. At the same time, the adversary publishes block B' and its vote for it. All honest validators of slot $t+2$ vote for B' as head of the chain, because it has more weight than B''. Finally, the block for slot $t+3$ is proposed building on block B', resulting in block B'' being reorged out.

Security problems. Neu *et al.* [18] have shown that LMD-GHOST suffers from a security issue. They presented an attack, called *balancing attack* [16, 23], in the synchronous network model with adversarial network delay. Subsequently, a similar attack, without requiring adversarial network delay, has been presented [17]. Briefly, with this attack, the adversary aims to keep honest validators split between two chains indefinitely. Key technique to maintain this split is that some adversarial validators withhold their votes and release them only at specific times and to specific subsets of honest nodes in order to influence the fork-choice procedure of honest nodes and thus steer which honest nodes vote either one or the other chain. Since the chains are kept *balanced* in terms of weight, this is possible even with little adversarial stake. This attack implies that LMD-GHOST is not secure with *any* confirmation rule: while the attack is ongoing, a confirmation rule satisfying liveness will eventually confirm something, which the adversary can reorg, violating safety. In Section 5.7, we show how to overcome these limitations by enhancing the LMD-GHOST protocol with the view-merge technique (see Section 5.3), and prove some security results for the resulting protocol.

5 Propose-vote-merge protocols

In this section we give a characterization of a class of protocols that we call *propose-vote-merge* protocols. These are protocols that proceed in *slots* consisting of 3Δ rounds, each having a proposer v_p, chosen through a proposer selection mechanism, e.g., the one outlined in Section 3.

At the beginning of each slot t, a block is proposed by v_p. All active validators (or *voters*) vote after Δ rounds (what they vote for will become clear shortly.) The last Δ rounds of the slot are needed for the *view-merge* synchronization technique, as explained in Section 5.3. 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 equipped with 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*, with respect to which the security of the protocol is defined. We differentiate propose-vote-merge protocols based on which fork-choice they implement; in other words, these protocols are *uniquely characterized by* FC. In section 5.6 we prove properties about propose-vote-merge protocols in a fork-choice-agnostic way. In the remaining part of this work, we consider instead protocols using GHOST-based fork-choice functions (see Section 4.1).

5.1 Message types

There are three message types, PROPOSE, BLOCK, and VOTE messages. We make no distinctions between network-level representation of blocks and votes, and their representation in a validator's view, *i.e.*, there is no difference between BLOCK and VOTE messages and blocks and votes, and we usually just refer to the latter. In the following description, t is a slot and v_i a validator. A block, or BLOCK message, is a tuple [BLOCK, b, t, v_i], where b is a *block body*, *i.e.*, the protocol-specific content of the block [9]. A vote, or VOTE message, is a tuple [VOTE, B, t, v_i], where B is a block. A proposal, or PROPOSE message, is a tuple [PROPOSE, B, $\mathcal{V}_i$, t, v_i] where B is a block and $\mathcal{V}_i$ a view. Votes are gossiped at any time, and the same goes for blocks, regardless of whether they are received directly or as part of a vote or a proposal, *i.e.*, a validator receiving a vote or proposal also gossips the block that it contains. Finally, a proposal from slot t is gossiped only during the first Δ rounds of slot t.

5.2 Protocol

We now define a propose-vote-merge protocol with generic fork-choice function FC. Recall that a fork-choice function uniquely identifies a protocol of this family. The protocol proceeds in three phases as it follows, and it is implemented in Algorithm 3. Observe that validators (which have synchronized clocks) update the variables t and r representing slot and round, respectively, through the protocol's execution.

PROPOSE: At round $3\Delta t$, 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: In rounds $[3\Delta, 3\Delta t + \Delta]$, 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 $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$. At round $3\Delta t + \Delta$, regardless of whether or not v_i received a proposal message, v_i broadcasts the vote message [VOTE, $\mathsf{FC}(\mathcal{V}_i, t)$, t, v_i], and updates its canonical chain by setting $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$.

MERGE: At round $3\Delta t + 2\Delta$, 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 $\mathcal{B}_i \leftarrow \emptyset$.

5.3 View-merge

View-merge is a technique first used by Kane, Fackler, Gagol, and Straszak [12] to guarantee liveness of the Highway protocol, and then by D'Amato, Neu, Tas, and Tse [10] to ensure *reorg resilience*, *i.e.*, proposals made by honest validators stay in the canonical chain, under synchrony. The idea behind the view-merge technique is to synchronize the views of all the honest validators, *before* they cast a vote in a slot, with the view $\mathcal{V}_p$ of the proposer of that slot. To do so, view-merge works as follows [9]:

1. Validators freeze their view Δ rounds before the beginning of a new slot, caching new messages in their buffer for later processing.

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

Algorithm 3 Propose-vote-merge protocol for validator v_i

```
State
    $\mathcal{V}_i \leftarrow \{\mathcal{B}_{\text{genesis}}\}$: view of validator $v_i$
    $\mathcal{B}_i \leftarrow \emptyset$: buffer of validator $v_i$
    $\mathsf{ch}_i \leftarrow B_{\text{genesis}}$: canonical chain of validator $v_i$
    $t \leftarrow 0$: the current slot
    $r \leftarrow 0$: the current round
PROPOSE
at $r = 3\Delta t$ do
    if $v_i = v_p^t$ then
        $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
        $\mathcal{B}_i \leftarrow \emptyset$
        $B' \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
        $B \leftarrow \mathsf{NewBlock}(B')$                // append a new block on top of $B'$
        $\mathsf{ch}_i \leftarrow B$
        send message [PROPOSE, $B$, $\mathcal{V}_i \cup \{B\}$, $t$, $v_i$] through gossip
VOTE
at $r = 3\Delta t + \Delta$ do
    $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
    send message [VOTE, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$] through gossip
MERGE
at $r = 3\Delta t + 2\Delta$ do
    $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
    $\mathcal{B}_i \leftarrow \emptyset$
upon receiving a gossiped message [PROPOSE, $B$, $\mathcal{V}$, $t$, $v_p^t$] do
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
    if $r \in [3\Delta t, 3\Delta t + \Delta]$ then
        $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$
upon receiving a gossiped message $V =$ [VOTE, $B$, $t'$, $v_i$] from $v_i$ do
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{V\}$
upon receiving a gossiped message $B =$ [BLOCK, $b$, $t'$, $v_i$] from $v_i$ do
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
```

2. Proposer v_p does not freeze its view. At the beginning of its slot, it proposes, according to its view $\mathcal{V}_p$, a block extending the canonical chain as determined by FC. Then, v_p broadcasts its view $\mathcal{V}_p$ together with the block to every other validator.

3. Validators merge the view received from v_p into their local view, execute FC based on this merged view, and vote.

Observe that, if the network delay is less that Δ rounds, the view of the proposer is a superset of the frozen views of other validators, so the final merged view is equal to the view of the proposer. If this is the case, since the output of the fork choice is a function of the view of a validator, every honest validator will have the same fork choice output. As a consequence, active validators vote for honest proposals under synchrony. We refer to this as the *view-merge property*, and prove it here for *all propose-vote-merge protocols*.

Lemma 2. *Suppose that t is a pivot slot, and network synchrony holds in rounds $[3\Delta t - \Delta, 3\Delta t + \Delta]$. Then, all honest voters of slot t,* i.e.*, H_t, vote for the honest proposal B of slot t.*

Proof. Let $\mathcal{V}_p \cup \{B\}$ be the view proposed with B by v_p, the honest proposer of slot t, *i.e.*, $\mathcal{V}_p$ is their view at round $3\Delta t$. Since v_p is honest, B extends $\mathsf{FC}(\mathcal{V}_p, t)$, and thus $\mathsf{FC}(\mathcal{V}_p \cup \{B\}, t) = B$ by the consistency property of FC. Consider an honest voter of slot t, *i.e.*, a validator $v_i \in H_t$, and let $\mathcal{V}_i$ be its view at round $3\Delta t + \Delta$, before merging $\mathcal{V}_i$ with the proposed view $\mathcal{V}_p \cup \{B\}$. Observe that, since v_i is active in round $3\Delta t + \Delta$, it must has already been awake at round $3\Delta(t-1) - 2\Delta$, because otherwise it would need to follow the joining protocol until round $3\Delta t + 2\Delta$, and would thus not currently be active. Therefore, v_i was already active at round $3\Delta(t-1) - 2\Delta$, and in particular it merged its buffer $\mathcal{B}_i$ in its local view then. So, $\mathcal{V}_i$ is the view that v_i had after merging the buffer $\mathcal{B}_i$. Since synchrony holds in rounds $[3\Delta t - \Delta, 3\Delta t]$, messages in

$\mathcal{V}_i$ are delivered to the proposer by round $3\Delta t$, so $\mathcal{V}_i \subseteq \mathcal{V}_p$. Since synchrony holds in rounds $[3\Delta t, 3\Delta t + \Delta]$, the proposal message is received by v_i before voting. Then, v_i merges the proposed view $\mathcal{V}_p \cup \{B\}$ with its view $\mathcal{V}_i$, resulting in the view $\mathcal{V}_i \cup (\mathcal{V}_p \cup \{B\}) = \mathcal{V}_p \cup \{B\}$. Validator v_i votes for the output of its fork-choice at round $3\Delta t + \Delta$, which is $\mathsf{FC}(\mathcal{V}_p \cup \{B\}, t) = B$. □

5.4 Joining protocol

In Section 3 we described a model in which 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*. We now present such a protocol, as presented in [10].

When an honest validator v_i wakes up at some round $r \in (3\Delta(t-1) + 2\Delta, 3\Delta t + 2\Delta]$, 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. v_i then waits for the *next view-merge opportunity*, at round $3\Delta t + 2\Delta$, 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.

Recall that we denote with H_t the set of active validators at round $3\Delta t + \Delta$, *i.e.*, the voting round of slot t. Since validators that are awake but not yet active do not vote, validators in H_t are exactly the honest validators that cast votes at slot t, motivating why we also refer to them as *honest voters of slot t*.

To understand the reasoning behind the joining protocol, consider for example a validator v_i which wakes up at round $3\Delta t$, where t is an adversarial slot and proposer v_p does not propose a block. If v_i immediately executed the protocol, it would cast a vote at round $3\Delta t + \Delta$, without having added any new message to their view, which would thus be missing all votes sent while it was asleep. Therefore, its view might be very outdated, and it might cast a vote for some block which is not in the canonical chain. If instead validators which become awake were to include all new messages in their view, and immediately start executing the protocol, the view-merge technique would fail to guarantee that such validators vote for honest proposals. It is therefore crucial that newly awake validators wait for the next view-merge opportunity, and cast votes only afterwards.

5.5 Confirmation rule

A confirmation rule allows validators to identify a *confirmed prefix* of the canonical chain, for which safety properties hold. The finality rule for LMD-GHOST presented in Section 4.3 is a special case of a confirmation rule that is *asynchronously safe*. In propose-vote-merge protocols, the confirmed chain Ch is the κ-deep prefix *in terms of slots* of the canonical chain ch, *i.e.*, its prefix corresponding to blocks proposed at slots $\leq t - \kappa$, which we denote with $\mathsf{ch}^{\lceil \kappa}$. A validator v_i updates its confirmed chain Ch_i whenever it updates its canonical chain ch_i by computing the fork-choice, *i.e.*, at round $3\Delta t + \Delta$, and possibly also $3\Delta t$, if they are the proposer of slot t, so that at any time we have $\mathsf{Ch}_i = \mathsf{ch}_i^{\lceil \kappa}$. We show that, with overwhelming probability, all intervals $[t - \kappa, t)$ of κ consecutive slots contain a *pivot slot*, *i.e.*, a slot with an honest proposer which is active at proposal time, and thus which makes a proposal. To achieve this, we rely on the lower bound h_0 on the active validators h_r at any round, which guarantees that in any slot there is at least a probability $\frac{h_0}{n}$ of an honest proposal being made. Note that this makes the confirmation time always based on the worst case participation. We can compare this to Goldfish [10], where the same κ-deep in slots confirmation rule achieves a better confirmation time, *i.e.*, the same failure probability can be achieved with a lower κ. This is because the probability of a honest proposal being made is greater than $\frac{1}{2}$, due to the proposal mechanism not satisfying uniqueness, and instead allowing for the proposer to be chosen among the active validators.

Lemma 3. *With overwhelming probability, all slot intervals of length κ contain at least a pivot slot.*

Proof. By assumption of *fairness* of the proposal mechanism, the proposer v_p of slot t is active at round $3\Delta t$ with probability $\frac{h_{3\Delta t}}{n} \geq \frac{h_0}{n}$, for $h_0 > 0$. Given any κ slots, the probability of none of the κ slots having an active proposer is $\leq (\frac{n-h_0}{n})^\kappa$, *i.e.*, negligible in κ. The number of slot intervals of length κ which we need to consider is equal to the time horizon T_{hor} over which the protocol is executed, which is polynomial in κ, so the probability of even one occurrence of κ consecutive slots without a pivot slot is also negligible. □

5.6 Properties

We now prove properties that propose-vote-merge protocols with a generic fork-choice function FC satisfy. An important property is reorg resilience (Definition 6), which we show implies security (Definition 4) in Theorem 4. The two key ingredients for reorg resilience are the view-merge property (Lemma 2), and the following proposition. While the former holds for any propose-vote-merge protocol, the latter does only for some of them, and only in some τ-sleepy model. In particular, we later prove that it holds for FC = GHOST-Eph, the fork-choice of Goldfish [10], if sleepiness is satisfied. More generally, we later show that it holds for FC = RLMD-GHOST with vote expiry parameter η, *i.e.*, the fork-choice rule introduced in this work and presented in Section 6, if η-sleepiness is satisfied.

Proposition 1. *Suppose network synchrony holds for rounds* $[3\Delta(t-1)+\Delta, 3\Delta t+\Delta]$*, and that all honest voters of slot* $t-1$ *vote for a descendant of block* B*. Then,* B *is in the canonical chain of all active validators in rounds* $\{3\Delta t, 3\Delta t+\Delta\}$*. In particular, all honest voters of slot* t *vote for descendants of* B*.*

We now show that, if Proposition 1 holds for an execution, then the execution satisfies reorg-resilience. The idea is the following: by the view-merge property, all active validators vote for honest proposals, and Proposition 1 ensures that this keeps holding also in future slots. We prove this result in the following theorem, which immediately implies that a protocol is τ-reorg-resilient if Proposition 1 holds for it in the τ-sleepy model.

Theorem 3 (Reorg resilience)**.** *Let us consider an execution of a propose-vote-merge protocol in which Proposition 1 holds. Then, this execution satisfies reorg resilience.*

Proof. Consider a honest proposal B from slot t after $\mathsf{GST}+\Delta$, *i.e.*, with $3\Delta t \geq \mathsf{GST}+\Delta$. We prove reorg resilience by induction on the slot. Note that validators only ever update their canonical chain at rounds $\{3\Delta s, 3\Delta s+\Delta\}$, upon computing the fork-choice. Therefore, the following statement holding for all $s \geq t$ is sufficient for reorg-resilience, as it implies that B is canonical in all rounds $\geq 3\Delta t+\Delta$.

Induction hypothesis: B is canonical in the views of active validators at rounds $r \in \{3\Delta s, 3\Delta s+\Delta\}$, for a slot $s \geq t$ and $r \geq 3\Delta t+\Delta$.

Base case: The proposal slot t. Synchrony holds from round $3\Delta t-\Delta \geq \mathsf{GST}$, so Lemma 2 applies and implies that all honest voters at slot t vote for B, which is in particular canonical in their views.

Inductive step: Suppose now that the statement holds for $s \geq t$. In particular, all honest voters of slot s vote for a descendant of B, because it is canonical in their view in the voting round $3\Delta s+\Delta$. Proposition 1 then implies the desired statement for $s+1$. □

If an execution satisfies reorg resilience we obtain that, by applying the same arguments as in [10], the κ-deep confirmation rule is secure in it, in the sense that the confirmed chain satisfies Definition 4. In particular, τ-reorg-resilience implies τ-dynamic-availability. Because of Thereom 3, we then only need to show that Proposition 1 holds for τ-compliant executions in order to show that a protocol is τ-dynamically-available.

Theorem 4 (Dynamic-availability)**.** *An execution of a propose-vote-merge protocol satisfying reorg-resilience also satisfies security with overwhelming probability after round* $T_{\mathsf{sec}} = \mathsf{GST}+\Delta+3\Delta(\kappa+1)$*, with* $T_{\mathsf{conf}} = 2\kappa$ *slots. In particular,* τ*-reorg-resilience implies* τ*-dynamic-availability.*

Proof. Theorem 3 and Lemma 3 imply security with overwhelming probability, as we now explain. For a round r, denote by $\mathrm{slot}(r)$ the slot to which that round belongs. We show liveness with confirmation time $T_{\mathsf{conf}} = 2\kappa$ slots after time $\mathsf{GST}+\Delta$. Consider a round $r \geq \mathsf{GST}+\Delta$, with $t = \mathrm{slot}(r)$, a round r' with $t' = \mathrm{slot}(r') \geq t+2\kappa$, and an honest validator v_i active at round r'. By Lemma 3, with overwhelming probability, there exists a pivot slot $t'' \in [t+1, t+\kappa]$. By Theorem 3, the proposal B from slot t'' is in the canonical chain of all active validators in later slots, so in particular it is in $\mathsf{ch}_i^{r'}$. Since $t'' \leq t+\kappa \leq t'-\kappa$, B is κ-deep in $\mathsf{ch}_i^{r'}$, and so it is in the confirmed chain $\mathsf{Ch}_i^{r'}$ as well.

To show safety, let us consider any two rounds $r' \geq r \geq T_{\mathsf{sec}} = \mathsf{GST}+\Delta+3\Delta(\kappa+1)$, and any two honest validators v_i and v_j at rounds r and r', respectively. Let also $t = \mathrm{slot}(r)$. Since $r \geq \mathsf{GST}+\Delta+3\Delta(\kappa+1)$, *i.e.*, r is at least κ slots after round $\mathsf{GST}+\Delta$, all slots $[t-\kappa, t)$ are after round $\mathsf{GST}+\Delta$, so Theorem 3 applies to them. Lemma 3 implies that there is at least a pivot slot $t' \in [t-\kappa, t)$, and its proposal B is canonical in all active views from round $3\Delta t' + \Delta$. Therefore, B is in the canonical chain of v_i at round r and, since it is from a slot $\geq t - \kappa$, $\mathsf{Ch}_i^r \preceq B$. Block B is also in the canonical chain of v_j at round r', *i.e.*, either $B \preceq \mathsf{Ch}_j^{r'}$ or $\mathsf{Ch}_j^{r'} \preceq B$. In the first case, $\mathsf{Ch}_i^r \preceq B \preceq \mathsf{Ch}_j^{r'}$. In the second case, we have both $\mathsf{Ch}_i^r \preceq B$ and $\mathsf{Ch}_j^{r'} \preceq B$. Therefore, Ch_i^r and $\mathsf{Ch}_j^{r'}$ cannot be conflicting, and it follows that either $\mathsf{Ch}_i^r \preceq \mathsf{Ch}_j^{r'}$ or $\mathsf{Ch}_j^{r'} \preceq \mathsf{Ch}_i^r$. □

5.7 LMD-GHOST with View-Merge

The first propose-vote-merge protocol that we consider is a variation of LMD-GHOST, with the addition of view-merge, in order to prevent ex-ante reorgs and balancing attacks (Section 4.3.3) [10]. To specify a propose-vote-merge protocol, we only need to define the fork-choice function which uniquely characterizes it, which in this case is of course LMD-GHOST, as introduced in Section 4.3. From now on, we refer to this propose-vote-merge protocol simply as LMD-GHOST. Since it supports asynchronous safety, we are mainly interested in analyzing its properties with finality as the confirmation rule, rather than the κ-deep confirmation rule. Moreover, we show in Theorem 7 that for LMD-GHOST the latter rule (and indeed, *any* confirmation rule) is not secure, according to Definition 4, under dynamic participation, so the usage of the κ-deep confirmation rule is not motivated in this context. We show that in this protocol m-finality satisfies liveness whenever $f < n - m$. Since by Theorem 2 m-finality is safe up to $f < m - \frac{n}{2}$, this means that LMD-GHOST can achieve $\frac{n}{4}$ fault tolerance both for liveness and safety, by $\frac{3}{4}$-finality. That said, recall that m is *not* a protocol parameter, but rather a subjective choice made by validators [11], contrary to usual quorum-based protocols in which the quorum size is globally agreed upon. Moreover, safety is *accountable*; a concrete economic penalty can be associated to a safety violation, possibly making a lower safety threshold entirely acceptable.

In the following theorem we analyze the reorg resilience of LMD-GHOST. In particular, given $|\widetilde{H}_t| > \frac{n}{2}$ and an honest proposal B from a slot t, then B is always canonical in all honest views that contain all slot t votes from $\widetilde{H}_t$, *without requiring synchrony at any future slot.* In other words, honest proposals made during synchrony immediately become asynchronously safe! This is much stronger than the usual notion of reorg resilience, which requires *continued synchrony.*

Theorem 5 (Strong reorg resilience)**.** *Consider an honest proposal B from a slot t in which $|\widetilde{H}_t| > \frac{n}{2}$. Suppose that the network is synchronous in rounds $[3\Delta t - \Delta, 3\Delta t + 2\Delta]$, and that the validators in $\widetilde{H}_t$ do not fall asleep in rounds $[3\Delta t + \Delta, 3\Delta t + 2\Delta]$. Then, B is always canonical in all honest views which contain all slot t votes from $\widetilde{H}_t$.*

Proof. By Theorem 2, all honest voters of slot t cast a vote for B at round $3\Delta t + \Delta$. Synchrony in the subsequent Δ rounds means that all such votes are received by those same validators before they merge their buffers, since by assumption they do not fall asleep. Those votes are then in all of their views by the end of slot t. Since, by assumption, $|\widetilde{H}_t| > \frac{n}{2}$, Theorem 1 implies the desired result, by letting $S = \widetilde{H}_t$. □

Finally, we show that a block proposed after $\mathsf{GST} + \Delta$ at slot t becomes m-finalized at slot $t+1$, assuming that m honest validators are active at such slots and both slots have honest proposers.

Theorem 6 (Liveness of finality)**.** *Let t and $t+1$ be two subsequent pivot slots after $\mathsf{GST} + \Delta$ in which a set S of $m > \frac{n}{2}$ honest validators is always active. Then, the proposal B made by the honest proposer of slot t is m-finalized at slot $t+1$.*

Proof. Since the validators in S are active throughout slot t, $|\widetilde{H}_t| \geq m > \frac{n}{2}$. By Theorem 5, B is canonical in all views that contain the slot t votes from $\widetilde{H}_t$. By synchrony, B is then canonical in all views of active validators at slot $t+1$, including those in S. The honest proposer for slot $t+1$ then extends B with some block A that contains the m-quorum certificate from slot t. By Theorem 2, all validators in S vote for A at slot $t+1$ as well. Therefore, B is m-finalized. □

[10] In the Ethereum protocol, the proposer boost technique [5] is used instead, with the drawback that adversarial resilience is at most $\frac{n}{3}$ (see Appendix C.1 in [10]). Moreover, the Ethereum protocol only has a committee of validators voting in each slot, *i.e.*, it implements subsampling. Neither proposer boost nor view-merge can fully prevent ex-ante reorgs in that setting, leading to a protocol with different security guarantees than what is described here, in particular not a reorg resilient protocol, even in the full participation setting.

[11] Or by any observer of the chain

5.7.1 Limitations

The following result shows that, even during synchrony, LMD-GHOST is not τ-dynamically-available for any finite τ, *regardless of the choice of confirmation rule.* Since the ∞-sleepy model allows only an extremely restrictive form of dynamic participation, almost equivalent to requiring $|\widetilde{H}_t| > \frac{n}{2}$ at all times, this is a fairly strong limitative result. We do so by presenting a scenario in which the adversary is able to cause a reorg of a confirmed block, compromising τ-safety and, consequently, τ-dynamic-availability, while never violating τ-sleepiness.

Theorem 7. *LMD-GHOST is not τ-dynamically-available for any finite τ and any confirmation rule with finite confirmation time T_{conf}, even with $\mathsf{GST} = 0$.*

Proof. For some $\tau < \infty$ and a confirmation rule with confirmation time T_{conf}, we show that τ-safety and τ-liveness are in conflict for LMD-GHOST. We look at a specific execution, which we assume satisfies liveness, and show that it does not satisfy safety. Moreover, we show that it is τ-compliant. Therefore, there are τ-compliant executions in which either liveness or safety is not satisfied, and consequently LMD-GHOST is not τ-dynamically-available.

Without loss of generality, we fix a finite $\tau \geq T_{\mathsf{conf}}$ (we do not need to consider $\tau < T_{\mathsf{conf}}$ since τ_1-dynamic-availability implies τ_2-dynamic-availability for $\tau_1 \leq \tau_2$). We consider a validator set of size $n = 2m + 1$, partitioned in three sets, V_1, V_2, and V_3, with $V_1 = \{v_1\}$, $|V_2| = m + 1$, $|V_3| = m - 1$. Validators in V_2 and V_3 are all initially honest, while v_1 is adversarial. Let $t - 1$ and t be two adversarial slots, *i.e.*, controlled by v_1. In slot t, validator v_1 publishes conflicting blocks A and B, one as a proposal for slot $t - 1$ and the other for slot t. By round $3\Delta t + \Delta$, the adversary delivers only A to validators in V_2, and only B to validators in V_3, so that the former vote for A and the latter for B in slot t [12]. At this point, the adversary puts all validators in V_3 to sleep, and then does nothing for $N \gg \tau$ slots, *i.e.*, until slot $t + N$. Meanwhile, validators in V_2 keep voting for A, since V_2 contains $m + 1 > \frac{n}{2}$ validators, so A stays canonical in all of the views of every member of V_2. Since $\tau \geq T_{\mathsf{conf}}$, this execution satisfying liveness implies that some honest proposal made after slot t is confirmed in this period, and thus that block A is confirmed, since all honest proposals made in this period are descendants of A. For any slot $s \in [0, t+1]$, we have that $|H_{s-1}| = |V_2 \cup V_3| = 2m$, so τ-sleepiness is satisfied. For $s \in [t + 2, t + \tau]$, we have that $|H_{s-1}| = |V_2| = m + 1 > m = |V_1 \cup V_3| = |A_s \cup (H_{s-\tau,s-2} \setminus H_{s-1})|$, so τ-sleepiness is also satisfied. For $s \in [t + \tau + 1, t + N - 1]$, the first two terms are unchanged, while $H_{s-\tau,s-2} \setminus H_s = \emptyset$, because the last vote broadcast by the validators in $|V_3|$ is from slot $t < s - \tau$. τ-sleepiness is then still satisfied. At slot $t + N$, the adversary corrupts a single validator $v_2 \in V_2$, and starts voting for B with both v_1 and v_2. Now, B has $m + 1$ votes, and descendants of A only m, so B becomes canonical and stay so. After T_{conf} slots, it is confirmed by all validators in V_2, meaning we have a safety violation. The adversary does not perform any more corruptions nor puts to sleep any more validators, and does not wake up validators in V_3. Therefore, for all slots $s \geq t + N$, we have $A_s = \{v_1, v_2\}$, $V_2 \setminus \{v_2\} \subseteq H_{s-1}$ and $H_{s-\tau,s-2} \setminus H_{s-1} = \emptyset$. τ-sleepiness is then satisfied, because $|H_{s-1}| \geq m > 2 = |A_s \cup H_{s-\tau,s-2} \setminus H_s|$. Therefore, the executions is τ-compliant, and thus the protocol does not satisfy τ-security. □

5.8 Goldfish protocol

Goldfish is a simplified variant of LMD-GHOST, introduced by D'Amato *et al.* [10], which very nearly belongs to the family of propose-vote-merge protocols. Goldfish can tolerate dynamic participation, it supports subsampling of validators, and it is reorg resilient and dynamically available in synchronous networks with dynamic participation. During each slot in Goldfish, only votes from the immediately preceding slot influence the protocol's behavior.

In the following analysis, we depart slightly from the original formulation [10], and consider a version of Goldfish which fulfils the specification of a propose-view-merge protocol. This variant does not consider subsampling and replaces the VRF lottery considered in the original protocol with an SSLE proposer selection mechanism (Section 3). As discussed in Section 5.5, the confirmation rule is unchanged, but a consequence of having a proposer selection mechanism satisfying uniqueness is that a longer confirmation time, *i.e.*, a higher κ, is required to ensure the same failure probability.

[12] In practice, such splitting of honest views can be done without targeting specific validators, and rather only aiming to get an approximate partition of the honest validators in two. This kind of attack method has been discussed at length in the context of balancing attacks [17]

Goldfish implements GHOST-Eph as fork-choice function. GHOST-Eph is a fork-choice function in $\mathcal{G}_f$ that takes a view $\mathcal{V}$ and a slot t as input, and finds the canonical chain determined by the votes within $\mathcal{V}$ that were cast for slot $t-1$.

5.8.1 Vote Expiry

The notion of *vote expiry* has been introduced by D'Amato *et al.* in the context of Goldfish, where all but the most recent votes are discarded. We generalize here this notion by introducing an expiration period η. In particular, given a slot t and a constant $\eta \geq 0$, the *expiration period* for slot t is the interval $[t-\eta, t)$, and only votes broadcast within this period influence the protocol's behavior at slot t.

Formally, we can define the GHOST *fork-choice function with expiry period* η as a fork-choice function in $\mathcal{G}_f$. It is characterized by the filter function $\mathsf{FIL}_{\eta\text{-exp}}(\mathcal{V}, t)$ which removes all votes from slots $< t-\eta$ from $\mathcal{V}$, and outputs the resulting view. For $\eta = \infty$, we get back the regular GHOST fork-choice function, whereas for $\eta = 1$, we get GHOST-Eph.

Algorithm 4 GHOST-Eph Fork-Choice function

```
function GHOST-Eph($\mathcal{V}, t$)
    return GHOST($\mathsf{FIL}_{1\text{-exp}}(\mathsf{FIL}_{\mathrm{eq}}(\mathcal{V}, t))$)
function $\mathsf{FIL}_{\eta\text{-exp}}(\mathcal{V}, t)$
    $\mathcal{V}' \leftarrow \mathcal{V}$ without all votes from slots $< t - \eta$
    return $(\mathcal{V}', t)$
```

Vote expiry allows the protocol to support dynamic participation. Recall from Section 5.7.1 that, in LMD-GHOST, the set of latest (votes) messages cast in the past by honest validators that have since fallen asleep can be used by an adversary to execute reorgs of arbitrary length even under synchrony, preventing τ-security for any finite τ (and with any confirmation rule). LMD-GHOST is then only secure in the ∞-sleepy model, which is extremely restrictive of dynamic participation. By assuming an expiration period, the capabilities of an adversary are limited, as votes broadcast before slot $t-\eta$ are not considered.

5.8.2 Properties

In Theorems 9 and Theorem 10, we prove a general result about a family of protocols generalizing Goldfish and LMD-GHOST. In the special case of Goldfish, it corresponds to the known properties from [10], *i.e.*, reorg-resilience and dynamic availability, in the usual sleepy model (for $\tau = 1$).

5.8.3 Limitations

Goldfish is brittle to temporary asynchrony. Due to the expiry period being $\eta = 1$, even a single violation of the bound of Δ rounds on the network delay can lead to a catastrophic failure, jeopardizing the safety of *any* previously confirmed block. This holds even if $f = 1$ and all validators are awake. Suppose for example that network delay is $> \Delta$ rounds between rounds $[3\Delta t + \Delta, 3\Delta t + 2\Delta]$. This causes all honest votes broadcast at round $3\Delta t + \Delta$ to not be delivered to any honest validator by round $3\Delta t + 2\Delta$. Then, such votes are not in any honest view after merging the buffer. Suppose also that the proposer of slot $t+1$ is malicious, and proposes a block B extending a block A which is not in the canonical chain. Moreover, the proposed view contains no votes other than a single slot t vote for A, from v_p^{t+1} themselves. Then, the view of an honest validator at the voting round $3\Delta(t+1) + \Delta$ contains only two slot t votes: its own, and the one for A. If A wins the tiebreaker, all honest validators thus vote for B, making it canonical and reorging all previously confirmed blocks. Since the period of asynchrony lasts for (less than) a single slot, all validators are always awake and $f = 1$, it is clear that the described execution is $(\infty, 2)$-compliant. Since $E_{\tau,\pi}$ is monotonically decreasing in τ and increasing in π, and thus $E_{\infty,2} \subseteq E_{\tau,\pi}$ for all $\tau > \pi \geq 2$, the above amounts to a proof of the following theorem.

Theorem 8. *Goldfish is not (τ, π)-asynchrony-resilient for any $\tau > \pi \geq 2$.*

6 Recent Latest Message Driven (RLMD) GHOST

In this section we present our propose-vote-merge protocol, RLMD-GHOST, which generalizes both LMD-GHOST and Goldfish. It is characterized by the GHOST-based fork-choice FC = RLMD-GHOST, which combines vote expiry (see Section 5.8.1) with the latest message driven (LMD) fork-choice (see Section 4.3). 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, *i.e.*, $\mathsf{FIL}_{\text{rlmd}} = \mathsf{FIL}_{\text{lmd}} \circ \mathsf{FIL}_{\eta\text{-exp}} \circ \mathsf{FIL}_{eq}$.

For $\eta = 1$, RLMD-GHOST coincides with GHOST-Eph, and thus RLMD-GHOST with Goldfish, because $\mathsf{FIL}_{1\text{-exp}}$ only considers votes from slot $t-1$, which are a subset of the latest votes, so that the LMD rule does not add any further filtering. For $\eta = \infty$, it coincides with LMD-GHOST, because no messages expire, so $\mathsf{FIL}_{\infty\text{-exp}}$ does not perform any filtering. Unless specified, we henceforth refer to RLMD-GHOST with a generic parameter η.

As Goldfish, RLMD-GHOST can support *fast confirmations* of honest proposals optimistically, *i.e.*, when honest participation is high. Moreover, due to the proposer selection mechanism satisfying uniqueness, it can do so without increasing the slot length to 4Δ rounds, *if the optimistic assumption is expanded to also assume that network latency is* $\frac{\Delta}{2}$. We discuss this at length in Appendix B.

Finally, observe that, if $\eta > 1$, enabling subsampling allows for ex-ante reorgs [23]. Since reorg resilience is central in the security analysis of propose-vote-merge protocols, this leads to entirely different security arguments being necessary, and consequently to reduced adversarial tolerance.

Algorithm 5 RLMD-GHOST Fork-Choice function

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

6.1 Tradeoff between dynamic availability and resilience to asynchrony

The expiry parameter η allows us to explore a tradeoff space between dynamic availability and resilience to asynchrony. At the extremes, $\eta = \infty$ gives us LMD-GHOST, a protocol which achieves asynchronous safety, but requires $> \frac{n}{2}$ honest participants to always be awake, and $\eta = 1$ gives us Goldfish, a dynamically available protocol which does not tolerate even a single slot of asynchrony. As we show in Section 6.2, RLMD-GHOST with $1 < \eta < \infty$ sits somewhere in between, achieving weaker forms of both properties, namely η-dynamic-availability and $(\eta, \eta-1)$-asynchrony-resilience, *i.e.*, it can tolerate $< \eta - 1$ slots of asynchrony, but it is only secure with the stronger assumptions of the η-sleepy model. The greater resilience to asynchrony is unsurprisingly due to the longer expiration period, which allows the latest messages of honest validators to persist even if periods of asynchrony prevent those validators from renewing their votes in the views of other validators. The same considerations apply to sudden drops in active participation, violating η-sleepiness assumptions. Even temporary violations of the basic 1-sleepiness assumption, *i.e.*, a *temporary adversarial majority* (for example if all honest validators are asleep), can be tolerated. In all such cases, the longer expiry period prevents the set of active validators whose votes are unexpired, and thus relevant to the fork-choice function, from suddenly shrinking or disappearing altogether. On the other hand, taking into account votes of validators which might be asleep weakens dynamic availability, as we have seen for LMD-GHOST in Theorem 7, motivating why the η-sleepiness assumption is required.

6.2 Properties

We start by showing that Proposition 1 holds in η-compliant executions of RLMD-GHOST.

Lemma 4. *Proposition 1 holds for RLMD-GHOST in η-compliant executions.*

Proof. Let $\mathcal{V}$ be the view of an active validator at a round $\in \{3\Delta t, 3\Delta t + \Delta\}$. By the synchrony assumption, and since the buffer is merged at round $3\Delta(t-1) + 2\Delta$, all honest votes from slot $t-1$ are in $\mathcal{V}$ and, by assumption, they are for descendants of B. The only votes to consider in order to decide whether B is canonical in $\mathcal{V}$ are those from slots $\in [t-\eta, t-1]$, because votes from slots prior to $t-\eta$ are expired at slot t. Votes that are not for descendants of B might be those from adversarial validators in A_t, or from validators in $H_{t-\eta,t-2} \setminus H_{t-1}$, *i.e.*, those that have voted in at least some slot $\in [t-\eta, t-2]$, but did not vote in slot $s-1$. Observe that $H_{t-1} \cap A_t$ might not be empty; there might be validators that were active in slot $t-1$ but were (shortly after) corrupted. Therefore, $\mathcal{V}$ might contain more than one vote from slot $t-1$ from some of these validators.

Let $E \subset H_{t-1} \cap A_t$ be the set of validators in $H_{t-1} \cap A_t$ for which $\mathcal{V}$ contains more than one vote from slot $t-1$. Due to equivocation discounting, votes from validators in E will not count. Observe that the number of votes that are not for descendants of B and that are counted in $\mathcal{V}$ is upper bounded by $|(A_t \setminus E) \cup (H_{t-\eta,t-2} \setminus H_{t-1})| = |(A_t \cup (H_{t-\eta,t-2} \setminus H_{t-1})) \setminus E| = |A_t \cup (H_{t-\eta,t-2} \setminus H_{t-1})| - |E|$, where the first equality follows from $E \subset H_{t-1}$. Since $\mathcal{V}$ contains votes for descendants of B for all validators in H_{t-1}, the number of votes for descendants of B and that are counted in $\mathcal{V}$ is lower bounded by $|H_{t-1} \setminus E| = |H_{t-1}| - |E|$. Since this is an η-compliant execution, η-sleepiness holds, *i.e.*, $|H_{t-1}| > |A_t \cup (H_{t-\eta,t-2} \setminus H_{t-1}|)$, so B is canonical in $\mathcal{V}$. □

As shown in Section 5, since Proposition 1 holds for RLMD-GHOST in η-compliant executions, the next two theorems follow. Observe that, by the hierarchy of sleepy models (see Section 3), the following results are also satisfied for $\tau \geq \eta$. In Appendix A, we show that these results are tight.

Theorem 9 (Reorg resilience)**.** *RLMD-GHOST is η-reorg-resilient.*

Theorem 10 (Dynamic availability)**.** *RLMD-GHOST is η-dynamically-available.*

Theorem 11 (Asynchrony resilience)**.** *RLMD-GHOST is $(\eta, \eta-1)$-asynchrony-resilient.*

Proof. Consider an $(\eta, \eta-1)$-compliant execution, with a $(\eta-1)$-tpa (t_1, t_2), and an honest proposal B from a slot $t \leq t_1$ after GST $+ \Delta$. First, since synchrony holds for slots $[t, t_1]$, and thus network synchrony holds until round $3\Delta t_1 + 2\Delta$, all the properties of RLMD-GHOST hold until then, including reorg resilience. In particular, starting from round $\geq 3\Delta t + \Delta$, B is in the canonical chain of all active validators in those slots, as they coincide with the aware validators. We then only need to consider aware validators at slots $s > t_1$. Suppose B is in the canonical chain of all aware validators at all slots $< s$. In particular, $B \preceq \mathsf{ch}_i^r$ for a validator $v_i \in H_{t_1}$ which is active at a round $r \in [3\Delta t_1 + \Delta, 3\Delta(s-1) + \Delta]$, because validators in H_{t_1} are always aware when active. Therefore, validators in $H_{t_1} \setminus A_s$ only ever broadcast votes for descendants of B in slots $[t_1, s-1]$.

Consider first the case $s \in (t_1, t_2]$. Then, the aware validators at a round $r \in \{3\Delta s, 3\Delta s + \Delta\}$ are exactly the validators H_{t_1} which are active in r. Consider then such a validator $v_i \in H_{t_1}$, and its view $\mathcal{V}_i$ at round r. View $\mathcal{V}_i$ contains all honest votes from slot t_1 because, by definition of $(\eta-1)$-tpa, v_i was awake at round $3\Delta t_1 + 2\Delta$, at which point it received all honest votes from slot t_1 and merged them into its view. Such votes are not expired at slot s, since $t_2 - t_1 \leq \eta - 1$ implies $t_1 > t_2 - \eta \geq s - \eta$, *i.e.*, t_1 is within the expiry period $[s-\eta, s-1]$ for slot s. All validators in $H_{t_1} \setminus A_s$ are not equivocators in $\mathcal{V}_i$, since they are not corrupted by round $3\Delta s + \Delta$. Therefore, their latest votes in $\mathcal{V}_i$ all count for a descendant of B in $\mathcal{V}_i$. The other votes which are counted in $\mathcal{V}_i$ are those from A_s and $H_{s-\eta,s-1} \setminus H_{t_1}$. Since the execution is $(\eta, \eta-1)$-compliant, we have $|H_{t_1} \setminus A_s| > |A_s \cup (H_{s-\eta,s-1} \setminus H_{t_1})|$, and thus B is canonical in $\mathcal{V}_i$.

Consider now the case $s = t_2 + 1$. Now, aware views coincide with active views, so we let $\mathcal{V}_j$ be the view of an active validator v_j at a round $r \in \{3\Delta(t_2+1), 3\Delta(t_2+1) + \Delta\}$. Since synchrony holds from slot t_2, view $\mathcal{V}_j$ contains all latest votes from $H_{t_1} \setminus A_{t_2+1}$, which are all from slots $[t_1, t_2]$, and thus for descendants of B by assumption. Moreover, $t_1 \geq t_2 - (\eta - 1) = (t_2 + 1) - \eta$, so all such votes are not expired at slot $t_2 + 1$. We can again conclude that B is canonical in $\mathcal{V}_j$, because $|H_{t_1}| > |A_s \cup (H_{s-\eta,s-1} \setminus H_{t_1})|$ holds for $s = t_2 + 1$ as well.

Finally, suppose $s > t_2 + 1$. Since aware and active views coincide at slot $s-1$, B is canonical in all active views at slot $s-1$ by assumption, so all honest votes from slot $s-1$ are for a descendant of B. Since synchrony holds as well, we can apply 1 and conclude that B is canonical at all active views at slot s. □

For RLMD-GHOST with $\eta \leq 2$, Theorem 11 does not say anything, because a π-tpa is empty for $\pi \leq 1$. For $\eta = 1$, this is entirely to be expected, given the limitations of Goldfish in this sense (see Section 5.8.3). For $\eta = \infty$, Theorem 11 gives us *asynchronous safety of honest proposals* from slots $\leq t_1$ in the views of validators $\widetilde{H}_{t_1}$, which are the only validators which are aware whenever they are active. As we have already mentioned, τ-compliance for $\tau = \infty$ essentially requires $|\widetilde{H}_{t_1}| > \frac{n}{2}$, which is precisely our assumption in Theorem 1.

7 Conclusion

Dynamically available protocols have recently been explored in the context of blockchain protocols, based on (variants of) the sleepy model [21]. In this work we presented a generalization of such sleepy model, with more generalized and stronger constraints in the corruption and sleepiness power of the adversary, and we formally proved properties and limitations of (two variants of) LMD-GHOST [25], the dynamically available component of Gasper [7], Goldfish [10], a synchronous dynamically available and reorg resilient protocol, and RLMD-GHOST, our novel protocol. In particular, Table 1 summarizes the results presented in this work, in our generalized sleepy model.

	Dynamic Availability	Asynchrony Resilience
LMD-GHOST without view-merge	✗ (Section 4.3.3)	✗ (Section 4.3.3)
LMD-GHOST with view-merge	✗ (only $\tau = \infty$, Theorem 7)	✓ ($\tau, \pi = \infty$, Theorem 11)
Goldfish	✓ ($\tau = 1$, Theorem 10)	✗ (Section 5.8.3)
RLMD-GHOST, expiry period η	✓ ($\tau = \eta$, Theorem 10)	✓ ($\tau, \pi = \eta, \eta - 1$, Theorem 11)

Table 1: A summary of the properties achieved by LMD-GHOST without view-merge (Section 4.3), LMD-GHOST with view-merge (Section 5.7), Goldfish (Section 5.8), and RLMD-GHOST (Section 6), all assumed to implement equivocation discounting (Section 4.2). Observe that asynchronous resilience is defined as reorg resilience of honest proposals made before a period of asynchrony. The original LMD-GHOST protocol (Section 4.3) is not asynchrony resilient: it supports asynchronous safety (Theorem 1), but this is not live, as showed by the balancing attacks [18] presented in Appendix 4.3.3.

RLMD-GHOST results in a synchronous protocol that has interesting practical properties: it is η-dynamically available (Theorem 10) and η-reorg resilient (Theorem 9)), with η being the expiry period in which votes are considered to make protocol's decisions, and it is resilient to asynchronous periods lasting less than $\eta - 1$ slots. Because of these properties, we believe that RLMD-GHOST can be considered a viable future replacement the current LMD-GHOST component of Gasper, achieving a sufficient degree of dynamic availability.

References

[1] Ethereum fork-choice specification. URL: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/fork-choice.md.

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

[3] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 12–24. ACM, 2020. doi:10.1145/3419614.3423258.

[4] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. URL: http://arxiv.org/abs/1807.04938, arXiv:1807.04938.

[5] Vitalik Buterin. Proposal for mitigation against balancing attacks to lmd ghost. URL: https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] George Kadianakis. Whisk: A practical shuffle-based ssle protocol for ethereum. URL: https://ethresear.ch/t/whisk-a-practical-shuffle-based-ssle-protocol-for-ethereum/11763.

[12] Daniel Kane, Andreas Fackler, Adam Gagol, and Damian Straszak. Highway: Efficient consensus with flexible finality. *CoRR*, abs/2101.02159, 2021. URL: https://arxiv.org/abs/2101.02159, arXiv:2101.02159.

[13] 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.

[14] 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.

[15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01. URL: https://bitcoin.org/bitcoin.pdf.

[16] 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.

[17] Joachim Neu, Ertem Nusret Tas, and David Tse. Attacking Gasper without adversarial network delay, 2021. URL: https://ethresear.ch/t/attacking-gasper-without-adversarial-network-delay/10187.

[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, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 446–465. IEEE, 2021.

[19] Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. In *International Conference on Financial Cryptography and Data Security*, FC '22, 5 2022. URL: `https://arxiv.org/abs/2105.06075`.

[20] Joachim Neu, Ertem Nusret Tas, and David Tse. Balancing attack: LMD edition, 2022. URL: `https://ethresear.ch/t/balancing-attack-lmd-edition/11853`.

[21] 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.

[22] 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.

[23] 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`.

[24] 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.

[25] 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`.

A Limitations of RLMD-GHOST

The first theorem shows limitations on the reorg resilience of RLMD-GHOST when $\eta > \tau$, except for $\eta = 1$, *i.e.*, Goldfish. The proof is very similar to that of Theorem 7. We again construct a τ-compliant execution in which old votes of honest validators which are currently asleep are used by the adversary to fuel a reorg. The difference is that now there is an expiration period, so the reorg we construct can only use unexpired votes. The reason we are still successful, despite the τ-sleepiness assumption, is that τ-sleepiness at slot t only considers votes from slots $\geq t - \tau$, which, when $\eta > \tau$, does not account for the still unexpired votes from slot $t - \eta$. In Theorem 7, we have exploited precisely the same mismatch between η and τ, in that case existing for any finite τ, since $\eta = \infty$ for LMD-GHOST. The second limitative theorem concerns dynamic availability. Unsurprisingly, an expiry period $\eta > \tau$ means that RLMD-GHOST is also not τ-dynamically-available. Finally, the third limitative theorem concerns asynchrony resilience, when $\pi \geq \eta$. Both of these results construct appropriately compliant executions in which the relevant property is not satisfied. In the former, we again exploit the mismatch between η and τ. In the latter, we do the same with π and η, by having a tpa which is too long for the expiration period to handle, *i.e.*, so that all votes prior to the tpa are expired after it.

Theorem 12. *RLMD-GHOST is not τ-reorg-resilient for any $1 \leq \tau < \eta$.*

Proof. We prove this theorem with an example. Let us consider a validator set of size $n = 2m+1$, partitioned in three sets, V_1, V_2, and V_3, with $V_1 = \{v_1\}$, $|V_2| = m + 1$, $|V_3| = m - 1$. Validators in V_2 and V_3 are all initially honest, while v_1 is adversarial. Let $t - 1$ and t be two adversarial slots, *i.e.*, controlled by v_1. In slot t, validator v_1 publishes conflicting blocks A and B, one as a proposal for slot $t - 1$ and the other for slot t. By round $3\Delta t + \Delta$, the adversary delivers only A to validators in V_2, and only B to validators in V_3, so that the former vote for A and the latter for B in slot t. At this point, the adversary puts all validators in V_3 to sleep, and then does nothing until slot $t + \eta - 1$. Meanwhile, validators in V_2 keep voting for A, since V_2 contains $m + 1 > \frac{n}{2}$ validators, so A stays canonical in all of the views of every member of V_2. Suppose in particular that the proposer of slot $t+1$ is in V_2, so that it makes a proposal C extending A. We now show that the adversary can induce a reorg of C, exploiting the votes of the asleep validators V_3, so that reorg resilience is not satisfied in this execution. We then only have to show that the execution is τ-compliant, in order to show that the protocol is not τ-reorg-resilient.

At the voting round $3\Delta(t+\eta-1)+\Delta$, the adversary votes for B with v_1. After the voting round, it corrupts two validators $v_2, v_3 \in V_2$, and starts voting for B with them, casting late votes for slot $t+\eta-1$. These votes are delivered to all awake validators by round $3\Delta(t+\eta-1)+2\Delta$, and are therefore in all of their views at the voting round of slot $t+\eta$. The votes of v_2 and v_3 are equivocations, so they are discounted, both for B and for A. Slot t is in $[t, t+\eta)$, the expiration period for slot $t+\eta$, so the votes of V_3 count at this slot. Therefore, in all views of the remaining honest validators in V_2, B has m votes, *i.e.*, those of V_3 and v_1, and descendants of A only $m-1$, because two have been discounted. B is then canonical in such views, and reorg resilience (of C) is violated. The adversary does not perform any more corruptions nor puts to sleep any more validators, and does not wake up validators in V_3.

We now show that this execution is τ-compliant. For any slot s, we show that τ-sleepiness holds at slot s, *i.e.*, that Equation 1 holds. For any slot $s \leq t+1$, this is clear, because we have $|H_{s-1}| = |V_2 \cup V_3| = 2m > 1 = |V_1| = |A_s \cup (H_{s-\tau,s-2} \setminus H_{s-1})|$. For any slot $s \in [t+2, t+\eta-1]$, we have $H_{s-1} = V_2$ and $A_s = V_1$, because the two corruptions only happen after round $3\Delta(t+\eta-1)+\Delta$. Therefore, $|H_{s-1}| = |V_2| = m+1 > m = |V_1 \cup V_3| \geq |A_s \cup (H_{s-\tau,s-2} \setminus H_{s-1})|$, so τ-sleepiness is satisfied. For any slot $s \geq t+\eta$, we have $A_s = \{v_1, v_2, v_3\}$, $V_2 \setminus \{v_2, v_3\} \subseteq H_{s-1}$ and $H_{s-\tau,s-2} \setminus H_{s-1} = \emptyset$, because $\eta > \tau$ implies $s-\tau \geq t+\eta-\tau > t$, so $V_3 \cap H_{s-\tau,s-2} = \emptyset$. τ-sleepiness is then satisfied, because $|H_{s-1}| \geq m-1 > 3 = |A_s| = |A_s \cup H_{s-\tau,s-2} \setminus H_{s-1}|$. Since the execution is τ-compliant, but does not satisfy reorg resilience, the protocol is not τ-reorg-resilient. □

Theorem 13. *RLMD-GHOST is not τ-dynamically-available for any $1 \leq \tau < \eta$ and for any confirmation rule with $T_{\mathsf{conf}} < \lfloor \frac{n-5}{4} \rfloor \eta = O(\eta \cdot n)$ slots, even with $\mathsf{GST} = 0$. In particular, it is not τ-dynamically available with the κ-deep confirmation rule, for $\kappa < \lfloor \frac{n-5}{4} \rfloor \eta$.*

Proof. Consider a validator set of size $n = 2m+1$, partitioned in three sets, $\mathcal{C}_0$, $\mathcal{A}_0$, and $\mathcal{S}_0$, standing for *corrupted, active, and sleepy*, respectively, with $\mathcal{C}_0 = \{v_1\}$, $|\mathcal{A}_0| = m+2$, $|\mathcal{S}_0| = m-2$. Validators in $\mathcal{A}_0$ and $\mathcal{S}_0$ are all initially honest, while v_1 is adversarial. Let $t-1$ and t be two adversarial slots, *i.e.*, controlled by v_1. In slot t, validator v_1 publishes conflicting blocks A and B, one as a proposal for slot $t-1$ and the other for slot t. By round $3\Delta t + \Delta$, the adversary delivers only A to validators in $\mathcal{A}_0$, and only B to validators in $\mathcal{S}_0$, so that the former vote for A and the latter for B in slot t.

At this point, the adversary puts all validators in $\mathcal{S}_0$ to sleep, and then does nothing until immediately after round $3\Delta(t+\eta-2)+\Delta$, *i.e.*, the voting round of slot $t+\eta-2$, at which point it corrupts two validators $\{v_2, v_3\} \in \mathcal{A}_0$. Up until this point, all validators in $\mathcal{A}_0$ have kept voting for A, since $|\mathcal{A}_0| = m+2 > \frac{n}{2}$ validators. At slot $t+\eta-1$, the adversarial validators initially do not cast votes. By round $3\Delta(t+\eta-1)+2\Delta$, the adversary wakes up validators in $\mathcal{S}_0$, so that they are active in slot $t+\eta$. At slot $t+\eta$, the adversary publishes three votes for B from slot $t+\eta-1$, from validators $\{v_1, v_2, v_3\}$. By round $3\Delta(t+\eta)+\Delta$, it delivers these only to validators in $\mathcal{A}_0$. Since the expiration period $[t, t+\eta-1]$ for slot $t+\eta$ contains t, the votes of $\mathcal{S}_0$ count at slot $t+\eta$. Therefore, in the views of the validators in $\mathcal{S}_0$ at slot $t+\eta$, descendants of A have a total of $m+2$ votes, from all validators in $\mathcal{A}_0$, including the newly corrupted ones, while B only has $m-2$ votes from $\mathcal{S}_0$. Thus, A is canonical in their views, and they vote for it. The views of the m remaining honest validators in $\mathcal{A}_0$ also include the three adversarial votes for B from slot $t+\eta-1$, so descendants of A only have m votes, while B has $m+1$. B then is canonical in their views, and they vote for it. The three adversarial validators also do so, so B receives $m+3$ votes and is canonical in the following slots. After the voting round of slot $t+\eta$, the adversary then puts all but two of the $m-2$ validators in $\mathcal{S}_0$ to sleep.

In slot $t+\eta+1$, there are then $m+2$ active validators: the two which are still active from $\mathcal{S}_0$, and m which are still honest from $\mathcal{A}_0$. There are also three adversarial validators and $m-4$ validators from $\mathcal{S}_0$ asleep from the previous slot. We are therefore in the same situation as in slot $t+1$, except we have two more adversarial validators (from $\mathcal{A}_0$) and two less asleep validators (from $\mathcal{S}_0$). We let $\mathcal{C}_1$ be the three adversarial validators, $\mathcal{A}_1$ be the $m+2$ active validators and $\mathcal{S}_1$ be the $m-4$ asleep validators. The adversary repeats the same pattern. It corrupts two more validators from $\mathcal{A}_1$ after the voting round of slot $(t+\eta)+(\eta-2) = t+2\eta-2$, and at round $3\Delta(t+2\eta-1)+2\Delta$ wakes up all validators in $\mathcal{S}_0$ so that they are active by slot $t+2\eta$. It then votes with all of the five adversarial validators for the branch of A at slot $t+2\eta-1$, but delivers such votes only to validators in $\mathcal{A}_1$ by the voting round of slot $t+2\eta$. Then, at slot $t+2\eta$, the branch of A has $m+1$

votes in the views of validators in $\mathcal{A}_1$, *i.e.*, the adversarial votes plus the votes from the $m-4$ validators in $\mathcal{S}_1$, which were put to sleep after voting for A at slot $t+\eta$. Therefore, it is canonical in their views and they vote for it, and so does the adversary. On the other hand, the views of validators in $\mathcal{S}_1$ at that round do not include the adversarial votes for A, and so all validators in $\mathcal{S}_1$ vote for B.

All but *four* of them are now put to sleep, so that at slot $t+2\eta+1$ there are $m+2$ active validators, $m-6$ asleep validators and five adversarial validators. We let these new sets of validators be $\mathcal{A}_2, \mathcal{S}_2, \mathcal{C}_2$, respectively. Again, the adversary has reorged from one branch to the other, while only needing to "convert" two asleep validators into two adversarial validators, and while otherwise preserving the same setup. They can repeat this until the number of adversarial validators reaches $m-1$, which does not allow for two additional corruptions. After the k^{th} reorg, at slot $t+k\eta+1$, there are $m+2$ active validators $\mathcal{A}_k$, $2k+1$ adversarial validators $\mathcal{C}_k$, and $m-2(k+1)$ asleep validators $\mathcal{S}_k$. Therefore, the adversary can repeat this up to $k \leq \lfloor \frac{m-2}{2} \rfloor = \lfloor \frac{n-5}{4} \rfloor$ times. Each time they do so, they can reorg from one branch to the other after η slots, for a total of $\lfloor \frac{n-5}{4} \rfloor \eta$ slots. By assumption, $T_{\mathsf{conf}} < \lfloor \frac{n-5}{4} \rfloor \eta$ slots. If no confirmation has been made after T_{conf} slots, then liveness is violated. If one has been made, then the confirmed block can still be reorged by slot $\lfloor \frac{n-5}{4} \rfloor \eta$, and the conflicting branch eventually confirmed afterwards, violating safety.

To complete the proof, we only need to verify that τ-sleepiness is satisfied. For slots $s \leq t$, we have $|H_{s-1}| = 2m > 1 = |A_s \cup (H_{s-\tau,s-2} \setminus H_{s-1})|$, so it is indeed satisfied. Consider now some $1 \leq k \leq \frac{m-2}{2}$, and slots $[t+(k-1)\eta+1, t+k\eta]$. For $s \in [t+(k-1)\eta+1, t+k\eta-1]$, we have $|H_{s-1}| \geq |\mathcal{A}_k| = m+2$, $|A_s| \leq |\mathcal{C}_k| = 2k+1$ and $|H_{s-\tau,s-2} \setminus H_{s-1}| = |\mathcal{S}_{k-1}| = m-2k$, so $|H_{s-1}| \geq m+2 > m+1 = (2k+1)+(m-2k) \geq |A_s \cup (H_{s-\tau,s-2} \setminus H_{s-1})|$, and τ-sleepiness at slot s is satisfied. For $s = t+k\eta$, we have $|H_{s-1}| = m$, because two more validators have been corrupted, $|A_s| = |\mathcal{C}_k| = 2k+1$, and $H_{s-\tau,s-2} \setminus H_{s-1} = \emptyset$, because $\tau < \eta$ implies $s-\tau = t+k\eta-\tau > t+(k-1)\eta$, which is the last slot in which $\mathcal{S}_{k-1}$ were active . Since $2k+2 \leq m$, we have that $2k+1 < m$, so τ-sleepiness at slot $t+k\eta$ is indeed satisfied. In slots after the last reorg, all honest validators are active, and there are $\geq m+2 > \frac{n}{2}$ of them, so τ-sleepiness is also satisfied. □

Theorem 14. *RLMD-GHOST with finite η is not (τ,π)-asynchrony-resilient for any $\tau > \pi \geq \max(\eta, 2)$, nor for $\tau = \pi = \infty$.*

Proof. For $\eta = 1$, *i.e.*, Goldfish, we have already shown this in Theorem 8. For $\eta > 1$, we have to show that RLMD-GHOST is not asynchrony resilient for any $\tau > \pi \geq \eta$, which we do by showing that RLMD-GHOST is not (∞,π)-asynchrony-resilient, by constructing an (∞,η)-compliant execution in which asynchrony-resilience does not hold. Since $E_{\tau,\pi}$ is monotonically decreasing in τ and monotonically increasing in π, $E_{\infty,\eta} \subset E_{\tau,\pi}$ for any $\tau > \pi \geq \eta$, and similarly $E_{\infty,\eta} \subset E_{\infty,\infty}$, so the desired result follows. We consider a validator set $\{v_1, v_2, v_3\}$, where all validators are honest at all times, and consider an execution with a η-tpa $(t, t+\eta)$, which is $\neq \emptyset$ since $\eta \geq 2$. In particular, network synchrony does not hold at slot $t+\eta-1$. Before round $3\Delta(t+\eta-1)+2\Delta$, validator v_3 is asleep. It wakes up at that round, and stays awake thereafter, so $v_3 \in H_s$ for $s \geq t+\eta$. Both validators v_1 and v_2 are active at all rounds $\leq 3\Delta t + 2\Delta$, so $H_s = \{v_1, v_2\}$ for $s \leq t$. Validator v_1 subsequently falls asleep, and only wakes up again in round $3\Delta(t+\eta)+2\Delta$, while validator v_2 is always awake. Upon waking up at round $3\Delta(t+\eta-1)+2\Delta$, validator v_3 does not see any message before merging the buffer into its view, due to asynchrony. Validator v_3 is the proposer of slot $t+\eta$, and, due to the lack of messages in its view, it proposes a block B extending B_{genesis}, which conflicts with all previous honest proposals. Validator v_3 then also votes for B at slot $t+\eta$, while v_2 does not. All three honest validators are active at round $3\Delta(t+\eta)+2\Delta$, so they receive these votes and merge them into their view. The latest vote from v_1 is from slot t, and is expired at slot $t+\eta+1$. Therefore, the only unexpired latest votes at the voting round of slot $t+\eta+1$ are those from v_2 and v_3 from slot $t+\eta$. If B wins the tiebreaker, it is then canonical in the views of the three validators. All honest proposals from slots $\leq t$ are then not canonical in these active views, which are also aware views since we are at a slot $> t+\eta$, so asynchrony-resilience is not satisfied in this execution. In order to show the desired result, we then only need to show that the execution is (∞,η)-compliant. For slots $s \notin (t, t+\eta]$, we have to show that ∞-sleepiness holds. It suffices to show that $|H_{s-1}| \geq 2 > \frac{n}{2}$. For $s \leq t$, we have $H_{s-1} = \{v_1, v_2\}$, while for $s > t+\eta$ we have $\{v_2, v_3\} \subseteq H_{s-1}$, so this is indeed the case. For slots $s \in (t, t+\eta+1]$, we have $H_t \setminus A_s = H_t = \{v_1, v_2\}$, so the condition which needs to hold during the η-tpa is satisfied. Moreover, H_t are awake at round $3\Delta t + 2\Delta$, satisfying even the last condition of (∞,η)-compliance. □

B Fast confirmations

We specify the protocol with fast confirmations and then analyze its properties. It requires a small modification to the generic propose-vote-merge protocol, changing the vote behavior so that validators vote *as soon as they see a proposal, or at round* $3\Delta t + \Delta$*, whichever comes first* (which is also exactly the attestation behavior specified by the Ethereum consensus protocol). In the following, $\mathsf{FC} = \text{RLMD-GHOST}$.

B.1 Protocol with fast confirmation

In the following, and in Algorithm 6, we update the confirmed chain explicitly, contrary to Algorithm 3. This is because the confirmed chain Ch_i in the latter is at any point simply a function of the canonical chain ch_i, *i.e.*, $\mathsf{Ch}_i = \mathsf{ch}_i^{\lceil\kappa}$. With fast confirmations, this is no longer the case. Moreover, in Algorithm 6 we use $\mathsf{FIL}(\mathcal{V}, t).\mathcal{V}$ to refer to the view output by a filter. Observe that validators (which have synchronized clocks) update the variables t and r representing slot and round, respectively, through the protocol's execution.

Propose: Unchanged from Section 5.2

Vote: In rounds $[3\Delta t, 3\Delta t + \Delta]$, 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 $\mathcal{V}$. After doing so, or at round $3\Delta t + \Delta$ if no proposal is received, it updates its canonical chain by setting $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$, and broadcasts the vote message [vote, $\mathsf{FC}(\mathcal{V}_i, t)$, t, v_i].

Confirm: At round $3\Delta t + \Delta$, a validator v_i merges its view with its buffer, *i.e.*, $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$, and sets $\mathcal{B}_i \leftarrow \emptyset$. It then 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 Ch_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* Ch_i *to some prefix of it* (we do not needlessly revert confirmations).

Merge: At round $3\Delta t + 2\Delta$, 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 $\mathcal{B}_i \leftarrow \emptyset$.

B.2 Safety and liveness tradeoff

The protocol with fast confirmations is safe when *both* fast confirmations and standard (κ-deep) confirmations are safe, and live whenever *at least one* is live. In particular, liveness is guaranteed by the liveness of the standard confirmations. On the other hand, the safety resilience of this protocol can be worse than $\frac{n}{2}$, which is what the original protocol tolerates *when all honest validators are awake*, since η-sleepiness reduces to $|H_{t-1}| > |A_t|$. In the previous section we have in particular specified fast confirmations to require a quorum of size $\frac{2}{3}n$, which results in both liveness and safety resilience *of fast confirmations* of $\frac{n}{3}$. We have chosen this quorum size because we are interested in using $\mathsf{RLMD\text{-}GHOST}$ in combination with a finality gadget, following the pattern of [18, 19], in which confirmations of the available protocol are input to the gadget, to preserve dynamic availability of the combined protocol. In this setting, we then want fast confirmations to require no further assumptions compared to the finality gadget, so that they can be live whenever the conditions are right for the gadget to be live, speeding up its action [13]. Due to this choice, safety resilience of the resulting protocol is then reduced to $\frac{n}{3}$ as well (cf. $\mathsf{Goldfish}$, where the chosen quorum is $\frac{3}{4}n$, so that the final protocol is still safe for $f < \frac{n}{2}$). On the other hand, we show in the next section that violating safety of a fast confirmation with quorum $\frac{2}{3}n$ requires $\frac{n}{3}$ equivocations, thus also making $\frac{n}{3}$ validators slashable. Therefore, the security guarantee of the resulting protocol *under network synchrony* is that a safety violation requires *either* safety of standard confirmations to be violated, implying a violation of η-sleepiness, and in particular $f \geq \frac{n}{2}$ if all honest validators are awake, *or* $\frac{n}{3}$ adversarial validators have to be slashable for equivocation. All safety results in the next section follow this formulation.

[13] In this setting, we would then also consider doing away with the extra optimistic assumption about network latency, and going back to 4Δ rounds instead, so that fast confirmations are always live after $\max(\mathsf{GST}, \mathsf{GAT})$, without needing network latency $\leq \frac{\Delta}{2}$.

Algorithm 6 Propose-vote-merge protocol for validator v_i

```
State
    $\mathcal{V}_i \leftarrow \{B_{\text{genesis}}\}$: view of validator $v_i$
    $\mathcal{B}_i \leftarrow \emptyset$: buffer of validator $v_i$
    $\mathsf{ch}_i \leftarrow B_{\text{genesis}}$: canonical chain of validator $v_i$
    $\mathsf{Ch}_i \leftarrow B_{\text{genesis}}$: confirmed chain of validator $v_i$
    $t \leftarrow 0$: the current slot
    $r \leftarrow 0$: the current round
    $\textit{sentvote} \leftarrow \textsc{false}$: indicates whether $v_i$ has voted in slot $t$
PROPOSE
at $r = 3\Delta t$ do
    if $v_i = v_p^t$ then
        $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
        $\mathcal{B}_i \leftarrow \emptyset$
        $B' \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
        $B \leftarrow \mathsf{NewBlock}(B')$                    // append a new block on top of $B'$
        $\mathsf{ch}_i \leftarrow B$
        send message [$\textsc{propose}$, $B$, $\mathcal{V}_i \cup \{B\}$, $t$, $v_i$] through gossip
VOTE AND CONFIRM
at $r = 3\Delta t + \Delta$ do
    if $\neg\textit{sentvote}$ then
        $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
        send message [$\textsc{vote}$, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$] through gossip
        $\textit{sentvote} \leftarrow \textsc{true}$
    $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
    $\mathcal{B}_i \leftarrow \emptyset$
    $B_{\text{fast}} \leftarrow B_{\text{genesis}}$
    $S_{\text{fast}} \leftarrow \{B \prec \mathsf{ch}_i \colon |\{v_i \colon \exists B' \succ B\ (\textsc{vote}, B', t, v_i) \in \mathcal{V}_i\}| \geq \frac{2}{3}n\}$
    if $S_{\text{fast}} \neq \emptyset$ then:
        $B_{\text{fast}} \leftarrow \underset{S_{\text{fast}}}{\arg\max} |B|$
    if $\neg(B_{\text{fast}} \prec \mathsf{Ch}_i \wedge \mathsf{ch}_i^{\lceil \kappa} \prec \mathsf{ch}_i)$ then:
        $\mathsf{Ch}_i \leftarrow \underset{\mathsf{ch} \in \{\mathsf{ch}_i^{\lceil \kappa}, B_{\text{fast}}\}}{\arg\max} |\mathsf{ch}|$
MERGE
at $r = 3\Delta t + 2\Delta$ do
    $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
    $\mathcal{B}_i \leftarrow \emptyset$
    $\textit{sentvote} \leftarrow \textsc{false}$
upon receiving a gossiped message [$\textsc{propose}$, $B$, $\mathcal{V}$, $t$, $v_p^t$] do
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
    if $\neg\textit{sentvote}$ and $r \in [3\Delta t, 3\Delta t + \Delta]$ then
        $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$
        $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
        send message [$\textsc{vote}$, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$] through gossip
        $\textit{sentvote} \leftarrow \textsc{true}$
upon receiving a gossiped message $V =$ [$\textsc{vote}$, $B$, $t'$, $v_i$] from $v_i$ do
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{V\}$
upon receiving a gossiped message $B =$ [$\textsc{block}$, $b$, $t'$, $v_i$] from $v_i$ do
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
```

B.3 Properties

Other than the small modifications we have made, the protocol with fast confirmation behaves exactly as the original protocol, because the confirmation rule does not in any way influence the protocol execution. Therefore, properties like reorg resilience and asynchrony resilience are preserved (up to accounting for those changes in the proofs). In the following, we then only discuss results for which the confirmation rule is relevant, *i.e.* security results. Firstly, we show a result that fulfills the same role of the view-merge property

(Lemma 2) in our security analysis of fast confirmations, in the sense that it provides the base case for the induction of Theorem 3, allowing us to prove an analogous reorg resilience result for fast confirmations, which implies safety. Since liveness is obtained for free from the liveness of the protocol without fast confirmations, this shows η-dynamic-availability. Finally, we show that fast confirmations are themselves live when there are at least $\frac{2}{3}n$ honest validators awake and the real network latency is $\leq \frac{\Delta}{2}$. To make it easier to state the results, we work here with a slightly modified definition of τ-compliant execution. In addition to satisfying τ-sleepiness, we require that in no honest view $\geq \frac{n}{3}$ validators are seen as equivocators. This is a very weak requirement, since equivocation is a slashable offense.

Lemma 5. *Suppose network synchrony holds for rounds $[3\Delta t + \Delta, 3\Delta t + 2\Delta]$, and that an honest validator fast confirms block B at slot t. Suppose also that, in the view of any active validator at slot $t+1$, $< \frac{n}{3}$ validators are seen as equivocators. Then, all honest voters of slot $t+1$ vote for descendants of B.*

Proof. Upon fast confirming B at round $3\Delta t + \Delta$, the honest validator broadcasts B and all votes $\geq \frac{2}{3}n$ votes which are responsible for the fast confirmation, so that they are in the view of all awake validators at round $3\Delta t + 2\Delta$, by synchrony. Therefore, they are also in the view of all active validators at round $3\Delta(t+1) + \Delta$. Consider one such view $\mathcal{V}$. By assumption, $< \frac{n}{3}$ validators are seen as equivocators in $\mathcal{V}$, so over $\frac{n}{3}$ out of the $\frac{2}{3}n$ votes are not discounted. Since they are from slot t, they are latest votes, and are the ones which count for the respective validators. Therefore, $w(B, \mathsf{FIL}_{ulmd}(\mathcal{V}, t+1).\mathcal{V}) > \frac{n}{3}$. On the other hand, $\mathcal{V}$ contains at most $\frac{n}{3}$ votes from slot t, conflicting with B and by a validator which is not seen as an equivocator in $\mathcal{V}$. Therefore, $w(B', \mathsf{FIL}_{ulmd}(\mathcal{V}, t+1).\mathcal{V}) \leq \frac{n}{3}$ for any B' conflicting with B, so B is canonical in $\mathcal{V}$, and an active validator with view $\mathcal{V}$ votes for a descendant of B. □

Theorem 15 (Reorg resilience of fast confirmations)**.** *Consider an η-compliant execution of RLMD-GHOST. 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 $\geq 3\Delta(t+1) + \Delta$.*

Proof. The proof follows that of Theorem 3, using Lemma 5 instead of Lemma 2 as the base case. The assumption of Lemma 5 about equivocators is satisfied by (the new) definition of η-compliance. Proposition 1, which we have proven for η-compliant executions of RLMD-GHOST in Lemma 4, is still used for the inductive step. □

Theorem 16 (Dynamic availability)**.** *RLMD-GHOST with fast confirmations is η-dynamically-available.*

Proof. η-liveness follows directly from Theorem 10, in particular from η-liveness of RLMD-GHOST without fast confirmations. This is because fast confirmations are not needed for the confirmed chain to make progress, and so liveness of the standard confirmations suffices. We then only need to show that it satisfies η-safety. If an honest validator fast confirms a block B at slot t in an η-compliant execution, then B is in the canonical chain of all active validators at rounds $\geq 3\Delta(t+1) + \Delta$, by Theorem 15. At slot $t + \kappa$, B is then in the κ-slots-deep prefix of the canonical chain of all active validators and thus confirmed by them with the standard confirmation rule. Therefore, a safety violation involving conflicting confirmed chains Ch_i^r and $\mathsf{Ch}_j^{r'}$ can be reduced to a safety violation for the standard confirmation rule, for rounds $r + 3\Delta(\kappa+1)$ and $r' + 3\Delta(\kappa+1)$. Theorem 10 then implies the η-safety of the protocol. □

Theorem 17 (Liveness of fast confirmations)**.** *An honest proposal B from a slot t after* $\mathsf{GST} + \Delta$ *in which $|H_t| \geq \frac{2}{3}n$ and network latency is $\leq \frac{\Delta}{2}$ is fast confirmed by all active validators at round $3\Delta t + \Delta$.*

Proof. Firstly, note that validators in H_t are active in all rounds $[3\Delta(t_1) + 2\Delta, 3\Delta t + \Delta]$, because falling asleep at any point in those rounds would force them to go through the joining protocol again, and thus they would not be active prior to at least round $3\Delta t + 2\Delta$. Since network latency is $\leq \frac{\Delta}{2}$, all validators in H_t receive the honest proposal by round $3\Delta t + \frac{\Delta}{2}$. By the view-merge property, Lemma 2, they all vote for B. Again by the assumption on network latency, they all receive such votes by round $3\Delta t + \Delta$, at which point they are merged into their views. Therefore, all of their views contain $|H_t| \geq \frac{2}{3}n$ votes for B from slot t, and B is fast confirmed. □