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 crucial requirement for devising permissionless consensus protocols. This notion, originally formalized by Pass and Shi (ASIACRYPT 2017) through their "sleepy model", captures the essence of a system's ability to handle participants joining or leaving during a protocol execution. A dynamically available consensus protocol preserves safety and liveness while allowing dynamic participation. Blockchain protocols, such as Bitcoin's consensus protocol, have implicitly adopted this concept.

In the context of Ethereum's consensus protocol, Gasper, Neu, Tas, and Tse (S&P 2021) presented an attack against LMD-GHOST – the component of Gasper designed to ensure dynamic availability. Consequently, LMD-GHOST results unable to fulfill its intended function of providing dynamic availability for the protocol. Despite attempts to mitigate this issue, the modified protocol still does not achieve dynamic availability, highlighting the need for more secure dynamically available protocols.

In this work, we present RLMD-GHOST, a synchronous consensus protocol that not only ensures dynamic availability but also maintains safety during bounded periods of asynchrony. This protocol is particularly appealing for practical systems where strict synchrony assumptions may not always hold, contrary to general assumptions in standard synchronous protocols.

Additionally, we present the "generalized sleepy model", within which our results are proven. Building upon the original sleepy model proposed by Pass and Shi, our model extends it with more generalized and stronger constraints on the corruption and sleepiness power of the adversary. This approach allows us to explore a wide range of dynamic participation regimes, spanning from complete dynamic participation to no dynamic participation, i.e., with every participant online. Consequently, this model provides a foundation for analyzing dynamically available protocols.

1 Introduction

1.1 Balancing dynamic availability and asynchrony resilience

Tolerating *dynamic participation* has emerged as a desirable feature of consensus protocols operating in the permissionless setting of blockchains. A *dynamically available* protocol preserves safety and liveness during events involving portions of the participants going offline. This concept, which was implicitly taken into account by the Bitcoin consensus protocol [14], was later formalized by Pass and Shi [18] through the *sleepy model* of consensus. In particular, Pass and Shi model a system where participants can be either online or offline, with their online status adversarially controlled throughout the execution, subject to the honest and online participants always outnumbering the adversarial ones.

A notable limitation of dynamically available consensus protocols is their inability to tolerate network partitions [8, 18, 11]. No consensus protocol can simultaneously satisfy liveness (under dynamic participation) and safety (under temporary network partitions, or temporary asynchrony). In other words, it is impossible for a consensus protocol (for state-machine replication) to yield a single output chain, while simultaneously offering dynamic availability and ensuring transaction finality, even during asynchronous periods or network partitions. As a result, dynamically available protocols are generally assumed to be synchronous [18, 13, 12].

Neu, Tas, and Tse [16], while formalizing the security requirements of Ethereum's consensus protocol, Gasper [5], demonstrate that the original version of LMD-GHOST, Gasper's dynamically available component, is not secure even in a context of full participation, *i.e.*, with all the participants in the protocol, or *validators*, being online. This finding is supported by the presentation of a *balancing* attack [16, 20].

D'Amato, Neu, Tas, and Tse [7] have proposed Goldfish as a solution to address the challenges posed by LMD-GHOST. Goldfish is synchronous consensus protocol that offers safety and liveness even when there is variable participation, making it dynamically available. However, the protocol's lack of *resilience to temporary asynchrony* makes it impractical to replace LMD-GHOST in Ethereum: even a very short period of asynchrony can result in a catastrophic failure, jeopardizing the safety of any previously confirmed block.

Our work acknowledges the limitations of both LMD-GHOST and Goldfish, and proposes a new family of synchronous consensus protocols, namely Recent Latest Message Driven GHOST (RLMD-GHOST), generalizing (variants of) LMD-GHOST and Goldfish. To analyze its properties, we introduce the *generalized sleepy model*, which can capture a broad spectrum of dynamic participation regimes falling between complete dynamic participation and no dynamic participation, and within which we formally define the concept of *asynchrony resilience*. Through the RLMD-GHOST family, we explore the trade-off space between resilience to temporary asynchrony and dynamic availability.

1.2 Technical outline

1.2.1 LMD-GHOST and its security

The consensus protocol of Ethereum, Gasper [5], is defined by two *components* [16]: LMD-GHOST, a synchronous consensus protocol, and Casper [4], a partially synchronous protocol (or *gadget*) that finalizes blocks on the chain output by LMD-GHOST and that keeps such finalized blocks safe during period of network asynchrony.

LMD-GHOST is named after its key component, the Latest Message Driven Greediest Heaviest Observed Sub-Tree rule (LMD-GHOST) *fork-choice function* (Section 4.2), introduced by Zamfir [23]. This function takes as input the sequence of blocks and the votes from each validator v_i, *i.e.*, a *local view* of v_i, and outputs a *canonical chain*. To do so, it starts from the genesis block B_{genesis} and walks down the sequence of blocks: at each block B, it chooses as the next block the child of B with the *heaviest subtree*, *i.e.*, with most (stake weighted) *latest* votes on its descendants.

LMD-GHOST can be thought of as a *propose-vote* protocol, proceeding in *slots*, each with an associated validator, a *proposer*, tasked with creating and proposing a new block. Votes are then cast by a randomly selected subset of the validator set, a *committee*. In other words, the protocol implements *subsampling* of validators. Both the proposer and voters utilize the output of the LMD-GHOST fork-choice function to decide what block to extend, and what block to vote for, respectively.

Balancing attack Neu, Tas, and Tse [16] show that this protocol is vulnerable to a *balancing attack* [16, 20], exploiting the fact that validators can have different local views, leading to conflicting votes. An adversarial proposer of slot t can equivocate and produce two conflicting blocks, which it reveals to two equal-sized subsets of the honest validators in the committee of slot t, so that their votes are split between the two blocks. In slot $t+1$, it then releases withheld votes from the adversarial validators in the committees of slot t to split validators of slot $t+1$ into two equal-sized subsets, one which sees one chain as leading and votes for it, and one which sees the other chain as leading and votes for it. This can be repeated indefinitely, maintaining the split between the two chains and preventing either liveness or safety of LMD-GHOST, as any decision made while the attack is ongoing would not be safe. The proposer boost technique [3] was later introduced as a mitigation to help *coordinate the voters in slots with an honest proposer*. It requires honest voters to temporarily grant extra weight to the current proposal.

Ex-ante reorgs Still, the resulting protocol remains prone to *ex-ante reorgs* [20], and has not been proven secure, even in a regime of full participation. Consider an adversary controlling a fraction β of the validators, among which the proposer for slot $t+1$, which *privately* creates a block B' on top of block B, *i.e.*, the block for slot t. Moreover, the adversary controls roughly a fraction β of the validators sampled to vote at slot $t+1$, and they privately vote for B'. Honest validators in the committee 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, which is the current tip of the canonical chain in their local view, and all honest validators in the committee of slot $t+2$

vote for it, so B'' accrues roughly $1-\beta$ of one committee's weight, say $(1-\beta)W_c$. Afterwards, the adversary publishes block B' and its votes from slot $t+1$ *and* $t+2$ for it. At this point, B has weight roughly $2\beta W_c$. If $2\beta > 1-\beta$, *i.e.*, if $\beta > \frac{1}{3}$, B' becomes canonical, reorging B''. More generally, an adversary controlling k slots in a row (which happens with probability β^k) can withhold votes from those slots, and release them after one honest slot, resulting in $(k+1)\beta W_c$ weight for a withheld adversarial block. If $\beta > \frac{1}{k+2}$, this result in a reorg of the honestly proposed block, so even relatively weak adversaries can perform ex-ante reorgs, albeit with low probability.

Note that subsampling is crucial in the success of the attack, because the adversary is able to accumulate the votes of the validators it controls across the k committees. Without subsampling, the adversarial votes from the last slot would simply replace the ones from the previous slots, due to the latest message rule. In other words, *without subsampling there is no way for the adversary to overcome the weight of all honest validators voting together in one slot*, as they are a majority over the entire validator set. We will later make use of this fact in our protocol.

1.2.2 Goldfish

To cope with the problems of LMD-GHOST, D'Amato, Neu, Tas, and Tse [7] devise Goldfish (Section 4.3), a synchronous consensus protocol that enjoys safety and liveness *under fully variable participation*, and thus that is dynamically available. Moreover, Goldfish is *reorg resilient*: blocks proposed by honest validators are guaranteed inclusion in the chain. Goldfish is based on two techniques: *view-merge* [6, 10][1] and *vote expiry.* It inherits the propose-vote structure of LMD-GHOST, and it adds a third round dedicated to view-merge. This new structure can be generalized by a family of protocols which we call *propose-vote-merge* protocols (Section 3).

View-merge is an improvement over the proposer boost technique, and serves the same purpose, *i.e.*, coordinating honest voters in slots with an honest proposer. In particular, it lets an honest proposer synchronize the local views of honest voters with its own, and *ensures* that they cast votes for the proposed block. With vote expiry, only votes from the latest slot influence the fork-choice. The adversary has then no way to accumulate weight *across committees*, and thus *no way to overpower the coordinated voting of the honest voters in a committee.* Therefore, vote expiry solves two problems at once: the ex-ante reorgs caused by subsampling, and the lack of dynamic availability due the adversary exploiting votes of offline validators, as these are now simply expired.

As we now informally explain, view-merge and vote expiry together make for a reorg resilience protocol, under the assumption that in every committee a majority of all *online* validators are honest[2]. View-merge ensures that an honest proposal B from slot t receives *all* votes from *honest and online* validators in the committee of slot t, which by assumption outnumber the adversarial votes in it. By vote expiry, the votes from the committee of slot t are the only ones which count in slot $t+1$, *i.e.*, voters in slot $t+1$ use them as input to their fork-choice function when determining what to vote for. Since a majority of the votes which are considered are for B, all votes from honest and online validators in slot $t+1$ are for B as well. By induction, this is true for all future slots achieving reorg resilience of honest proposals. As we explore in Section 3, reorg resilience then immediately implies that the κ-deep confirmation rule is both safe and live, implying dynamic availability for Goldfish.

Nonetheless, 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 other words, Goldfish is not *asynchrony resilient.* This is simply due to the strict vote expiry, which can cause the weight supporting a block to drop to zero at any time, if no *new* votes supporting it are received within one slot.

[1] View-merge was first introduced in the Highway protocol [10], under the name of vote buffering, which is also the name used in Goldfish [7].

[2] The sleepy model assumes that at any given time a majority of online validators are honest, which is a slightly different assumption that assuming such condition for a committee. Still, the latter can be recovered by strengthening the former, by requiring a $\frac{1}{2}+\epsilon$ majority in the whole validator set, and by having large enough committees so that an honest majority in a committee holds except with negligible probability.

1.2.3 RLMD-GHOST

Strict vote expiry does not seem compatible with any reasonable notion of resilience to asynchrony. On the other end, not expiring votes at all is not compatible with dynamic availability, as we have observed in LMD-GHOST, where votes of *honest but offline* validators can be exploited by the adversary to conclude long reorgs. A natural approach, and the one we take in this work, is then to *relax* vote expiry. In particular, we introduce Recent Latest Message Driven GHOST (RLMD-GHOST) (Section 5), *family* of propose-vote-merge consensus protocols parameterized by the *vote expiry period* η, *i.e.*, only votes from the most recent η slots are utilized in the protocol with parameter η. RLMD-GHOST extends and generalizes both LMD-GHOST and Goldfish. As LMD-GHOST, RLMD-GHOST implements the latest message rule (LMD). As Goldfish, it implements view-merge and vote expiry. Relaxing vote expiry forces us to do away with subsampling, since we want to avoid reintroducing the possibility of ex-ante reorgs through accumulating votes from multiple committees, as this would break the reorg resilience property upon which the security of both Goldfish and RLMD-GHOST rests (and which is itself highly desirable). Therefore, every validator votes in every slot of RLMD-GHOST. For $\eta = 1$, RLMD-GHOST reduces to a variant of Goldfish without subsampling of validators. For $\eta = \infty$, RLMD-GHOST reduces to LMD-GHOST. To be precise, we do not refer here to LMD-GHOST as currently implemented in Ethereum but to a variant, described in this work, without subsampling *and implementing the view-merge technique instead of proposer boost* (*i.e.*, also a propose-vote-merge protocol).

Intuitively, considering votes from a longer period ($\eta > 1$) results in a protocol that is more tolerant of asynchrony, since a correspondingly longer period of asynchrony is needed for all votes to expire. On the other end, relaxing vote expiry should weaken dynamic availability because, as in LMD-GHOST, the fork-choice can now be affected by the (still unexpired) votes of offline validators. In the reorg resilience argument for Goldfish, after an honest slot t with proposal B, honest voters at slot $t+1$ still vote for B, because the honest votes from slot t outnumber adversarial votes, *and there are no further votes to consider.* With a longer expiry period $\eta > 1$, this is not the case anymore, because at slot $t+1$ all votes from slots $[t+1-\eta, t]$ have to be considered.

1.2.4 Generalized sleepy model

In order to describe the security of RLMD-GHOST, we introduce an extension of the sleepy model, which we refer to as the *generalized sleepy model* (Section 2), that allows us to capture a weaker notion of dynamic availability and to precisely define a notion of *asynchrony resilience.* In defining this model, we again take inspiration from the reorg resilience argument for Goldfish. In particular, computing the fork-choice at slot t now requires considering all votes from slots $[t-\eta, t-1]$. If there has been an honest proposal in slot $t-1$, the only votes which are guaranteed to vote in support of it are those from honest validators which were online in slot $t-1$, which we define as H_{t-1}. All other votes are either adversarial or from offline validators, and potentially dangerous in both cases. Letting $H_{s,t}$ be the honest validators online in slots $[s,t]$, and A_t be the validators corrupted up to slot t, we require that $|H_{t-1}| > |A_t \cup (H_{t-\eta,t-2} \setminus H_{t-1})|$. Observe that when $\eta = 1$, we recover a standard majority assumption, *i.e.*, $|H_{t-1}| > |A_t|$, much like the one found in the sleepy model. The more general condition is precisely the key assumption of our *generalized sleepy model*, which allows us to describe a whole spectrum of dynamic participation regimes, by varying the parameter. If the assumption holds for a certain η, then the corresponding RLMD-GHOST protocol is reorg resilient, and consequently secure. Finally, we extend the generalized sleepy model further, to allow for bounded periods of asynchrony, and define an associated notion of asynchrony resilience, which is satisfied by RLMD-GHOST. Limitations of RLMD-GHOST are analyzed in Appendix A.

Related works to this paper are discussed in Section 6, while conclusion and future work are presented in Section 7. 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.

2 Model and Preliminary Notions

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 validator 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 corrupt validators over the course of the 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 by v_i.

Time Time is divided into discrete *rounds*. We consider a synchronous model in which validators have synchronized clocks and 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* (see Section 3.4), after which they become *active* [7]. All adversarial validators are always awake, and are not prescribed to follow any protocol. Therefore, we always use active, awake, and asleep to refer to honest validators. As for corruptions, the adversary is adaptive also for sleepiness, *i.e.*, the sleepiness schedule is also chosen dynamically by the adversary. Note that awake and active validators coincide in the sleepy model [18].

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 [2], *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 [9].

View Due to adversarial validators and message delays, validators may have different sets of received messages. A *view* (at a given round r), denoted by $\mathcal{V}$, is a subset of all the messages that a validator 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. In particular, a block contains a reference to its parent block, and verifying its validity requires also being able to verify the parent's 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 consider 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 [22], which we denote with $\mathcal{G}_f$ and introduce in Section 4.1.2.

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.

2.1 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 [18, 7], and then a two-parameters family, which generalizes it further by introducing, and accounting for, *bounded* periods of asynchrony.

2.1.1 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 [18], 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, Neu, Tas, and Tse [7] introduce the notion of an active validator[3] and assume a modified condition, *i.e.*, $h_{r-3\Delta} > f_r$. In this condition, that is tailored to their protocol, $\mathsf{Goldfish}$, $h_{r-3\Delta}$ is considered instead of h_r because, if r is a voting round in $\mathsf{Goldfish}$, validators corrupted after round r can still retroactively broadcast votes for that round, and these votes are relevant until 3Δ rounds later (but no longer, due to vote expiry). 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 [4]. 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 $H_i := \emptyset$). 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:

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

Equation 1 requires that the number of active validators at slot $t-1$, *i.e.*, $|H_{t-1}|$, must exceed the number of adversarial validators, *i.e.*, $|A_t|$, together with all the other validators that have been active at some point in $[t-\tau, t-2]$, and that are not active at slot $t-1$, *i.e.*, $|H_{t-\tau,t-2} \setminus H_{t-1}|$. Intuitively, this condition is tailored to a protocol implementing vote expiry (with period $\eta = \tau$) because the votes which are considered at slot t are those from slots $[t-\tau, t-1]$. Out of these, the only honest votes which we can rely on are those

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

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

from H_{t-1}, whereas unexpired votes from honest validators which were not active in slot $t-1$ might help the adversary.

We refer to the sleepy model in which the adversary is constrained by τ-sleepiness as the *τ-sleepy model.* Note that, for $\tau = 1$, this reduces to the sleepy model from Goldfish, as this condition reduces to the majority condition $h_{r-3\Delta} > f_r$ of 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 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.

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 constrains 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.

2.1.2 (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. In particular, we define a temporary period of asynchrony as it follows.

Definition 2 (Temporary period of asynchrony)**.** We say that an interval (t_1, t_2) of consecutive slots 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.

We consider a system where synchrony holds *except for one such* π-tpa, for some $\pi \in \mathbb{N} \cup \{\infty\}$. We refer to this network model as *synchronous network with a temporary period of asynchrony* tpa. Since a 1-tpa is empty, this is a generalization of the usual synchronous network model. 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 3 ((τ, π)-compliant execution)**.** For $\tau > \pi$, or $\tau = \pi = \infty$, an execution in the 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}, because we rely on H_{t_1} to preserve the canonical chain throughout this period.

This is also why we have the third condition, as it guarantees that validators H_{t_1} are able to observe the votes cast at round $3\Delta t_1 + \Delta$, which inform any votes they might cast during the period of asynchrony. 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. 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 synchronous 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 $|H_{t_1} \setminus A_\infty| > \frac{n}{2}$, capturing the intuition that we can in principle get asynchronous safety as long as an honest majority of validators (which are never corrupted) agrees on something.

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

2.2 Security

Security Parameters We largely follow here the notation and definitions of [7]. 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*, or w.o.p, if it happens except with probability which is $\text{negl}(\kappa) + \text{negl}(\lambda)$. Properties of cryptographic primitives hold except with probability $\text{negl}(\lambda)$, *i.e.*, with overwhelming probability, but we leave this implicit in the remainder of this work.

Confirmed chain The protocols which we consider always specify a *confirmation rule*, with whom validators can identify a *confirmed prefix* of the canonical chain. Alongside the canonical chain, validators then also keep track of a *confirmed chain.* 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). This is the output of the protocol, for which safety properties should hold, and thus with respect to which the security of the protocol is defined.

Definition 4 (Secure protocol [7])**.** We say that a consensus protocol outputting a confirmed chain Ch is *secure*, and has confirmation time T_{conf}[5], if Ch satisfies:

- **Safety:** For any two rounds r, r', 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 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.

[5]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).

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}.

Definition 5 (Dynamic availability)**.** We say that a consensus protocol is *τ-dynamically-available* if and only if it satisfies τ-security 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 satisfies *reorg resilience* if any honest proposal B from a slot t 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 synchronous network model with a tpa (t_1, t_2) satisfies *asynchrony resilience* if any honest proposal from a slot $t \leq t_1$ 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.*

3 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 k rounds, each having a proposer v_p, chosen through a proposer selection mechanism, e.g., the one outlined in Section 2. In this work, unless otherwise specified, we analyze the case $k = 3\Delta$.

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 3.3. 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*, as described in Section 3.5, 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 3.6 we prove properties about propose-vote-merge protocols in a fork-choice-agnostic way. Then, we will instead focus on protocols using GHOST-based fork-choice functions.

3.1 Message types

In propose-vote-merge protocols there are three message types, namely 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

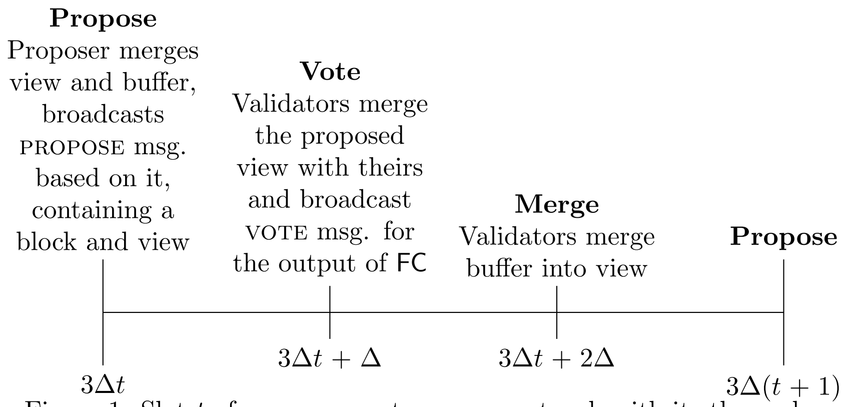
Propose
Proposer merges view and buffer, broadcasts PROPOSE msg. based on it, containing a block and view
Vote
Validators merge the proposed view with theirs and broadcast VOTE msg. for the output of FC
Merge
Validators merge buffer into view
Propose
$3\Delta t$
$3\Delta t + \Delta$
$3\Delta t + 2\Delta$
$3\Delta(t+1)$

Figure 1: Slot t of a propose-vote-merge protocol, with its three phrases.

block[6]. 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 $\mathcal{V}_i$ the view of validator v_i. 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.

3.2 Protocol

We now define a propose-vote-merge protocol with a generic fork-choice function FC. Recall that a fork-choice function uniquely identifies a protocol of this family. A slot of the protocol proceeds in three phase, depicted in Figure 1 and implemented in Algorithm 1. Observe that validators (which have synchronized clocks) update the variables t and r representing slot and round, respectively, throughout the protocol's execution.

PROPOSE: At round $3\Delta t$, proposer v_p merges its view $\mathcal{V}_p$ with its buffer $\mathcal{B}_p$, *i.e.*, $\mathcal{V}_p \leftarrow \mathcal{V}_p \cup \mathcal{B}_p$, and sets $\mathcal{B}_p \leftarrow \emptyset$. Then, v_p runs the fork-choice function 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 [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 $\mathcal{B}_i \leftarrow \emptyset$.

3.3 View-merge

The MERGE phase, along with all other operations involving views and buffers discussed in the previous section, are implementing the *view-merge* technique. View-merge has been introduced by Kane, Fackler, Gagol, and Straszak [10] to guarantee liveness of the Highway protocol, and then by D'Amato, Neu, Tas, and Tse [7] to ensure *reorg resilience*, *i.e.*, proposals made by honest validators stay in the canonical chain, under synchrony. The core concept of the view-merge technique involves synchronizing the views of all honest validators with the view $\mathcal{V}_p$ of the proposer for a specific slot *before* the validators broadcast their votes in that slot. To do so, view-merge works as follows:

1. A validator's buffer is merged into its view only at one specific time, *i.e.*, Δ rounds preceding the start of a new slot. Therefore, no new messages enter its view this way before voting in the next slot.

2. The one exception to the above is the proposer of the next slot, which merges the buffer right before proposing, Δ *rounds after all other validators*. It then proposes its resulting view $\mathcal{V}_p$ and a block extending the canonical chain as determined by FC, according to $\mathcal{V}_p$.

[6]For simplicity, we omit a reference to the parent block. As mentioned in Section 2, we leave questions of validity implicit.

Algorithm 1 Propose-vote-merge protocol for v_i

```
1: State
2:     $\mathcal{V}_i \leftarrow \{\mathcal{B}_{\text{genesis}}\}$: view of validator $v_i$
3:     $\mathcal{B}_i \leftarrow \emptyset$: buffer of validator $v_i$
4:     $\mathsf{ch}_i \leftarrow B_{\text{genesis}}$: canonical chain of validator $v_i$
5:     $t \leftarrow 0$: the current slot
6:     $r \leftarrow 0$: the current round
   PROPOSE
7: at $r = 3\Delta t$ do
8:     if $v_i = v_p^t$ then
9:         $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
10:        $\mathcal{B}_i \leftarrow \emptyset$
11:        $B' \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
12:        $B \leftarrow \mathsf{NewBlock}(B')$
13:        // append a new block on top of $B'$
14:        $\mathsf{ch}_i \leftarrow B$
15:        gossip message [PROPOSE, $B$, $\mathcal{V}_i \cup \{B\}$, $t$, $v_i$]
   VOTE
16: at $r = 3\Delta t + \Delta$ do
17:     $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
18:     gossip message [VOTE, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$]
   MERGE
19: at $r = 3\Delta t + 2\Delta$ do
20:     $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
21:     $\mathcal{B}_i \leftarrow \emptyset$
22: upon receiving a message [PROPOSE, $B$, $\mathcal{V}$, $t$, $v_p^t$] do
23:     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
24:     if $r \in [3\Delta t, 3\Delta t + \Delta]$ then
25:         $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$
26: upon receiving a message [VOTE, $B$, $t'$, $v_i$] from $v_i$ do
27:     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup$ {[VOTE, $B$, $t'$, $v_i$]}
28: upon receiving a message [BLOCK, $b$, $t'$, $v_i$] from $v_i$ do
29:     $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup$ {[BLOCK, $b$, $t'$, $v_i$]}
```

3. Validators merge the proposed view 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 views of other validators, because all messages merged by validators in the first step will also be merged by the proposer Δ rounds later. Then, the final merged view of all validators before voting 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, agreeing with the proposed block. 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 1. *Suppose that t is a pivot slot. 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 block B by v_p, the honest proposer of slot t, *i.e.*, $\mathcal{V}_p$ is the view of v_p 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 (see Section 2) 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 $\mathcal{B}_i$. So, messages in $\mathcal{V}_i$ are delivered to the proposer by round $3\Delta t$, so $\mathcal{V}_i \subseteq \mathcal{V}_p$.

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 $\mathsf{FC}(\mathcal{V}_p \cup \{B\}, t) = B$. □

3.4 Joining protocol

In Section 2 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 recall such a protocol, as presented in [7].

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. Validator 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 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.

3.5 Confirmation rule

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.

Lemma 2. *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. □

3.6 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 2. The two key ingredients for reorg resilience are the view-merge property (Lemma 1), 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 $\mathsf{FC} =$ GHOST-Eph, the fork-choice of Goldfish [7], if 1-sleepiness is satisfied. More generally, we later show that it holds for $\mathsf{FC} =$ RLMD-GHOST with vote expiry parameter η, *i.e.*, the fork-choice rule introduced in this work and presented in Section 5, if η-sleepiness is satisfied.

Proposition 1. *Suppose 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 (Lemma 1), 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 1 (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. We prove reorg resilience by induction on the slot. Note that validators only ever update their canonical chain at rounds $\{3\Delta s, 3\Delta s + \Delta\}$, for all slots $s \geq t$, 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. Lemma 1 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 [7], 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 1, 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 2 (Dynamic-availability)**.** *An execution of a propose-vote-merge protocol satisfying reorg resilience also satisfies security with overwhelming probability with $T_{\mathsf{conf}} = 2\kappa$ slots. In particular, τ-reorg-resilience implies τ-dynamic-availability.*

Proof. Theorem 1 and Lemma 2 imply security (Definition 4) with overwhelming probability, as we now explain. For a round r, denote by $\text{slot}(r)$ the slot to which that round belongs. We show liveness with confirmation time $T_{\mathsf{conf}} = 2\kappa$ slots. Consider a round r, with $t = \text{slot}(r)$, a round r' with $t' = \text{slot}(r') \geq t + 2\kappa$, and an honest validator v_i active at round r'. By Lemma 2, w.o.p, there exists a pivot slot $t'' \in [t+1, t+\kappa]$. By Theorem 1, 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 $r' \geq r$, and any two honest validators v_i and v_j, active at rounds r and r', respectively. Let also $t = \text{slot}(r)$. Lemma 2 implies that w.o.p. there is at least a pivot slot $t' \in [t - \kappa, t)$, and by Theorem 1 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, it follows that either $\mathsf{Ch}_i^r \preceq \mathsf{Ch}_j^{r'}$ or $\mathsf{Ch}_j^{r'} \preceq \mathsf{Ch}_i^r$. □

4 Propose-vote-merge protocols based on GHOST

4.1 Prerequisites

In this section we recall the LMD-GHOST fork-choice function as utilized in the Ethereum consensus protocol, Gasper [5]. We start by presenting the fork-choice function GHOST, the main building block of LMD-GHOST, and of all the fork-choice functions we consider in this work.

4.1.1 GHOST

GHOST is a fork-choice function based on the fork-choice procedure introduced in [22] 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 3. *Let $\mathcal{V}$ be a view in which over a majority of the votes are for a descendant of a block B. Then,* 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 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 2 GHOST Fork-Choice function

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

4.1.2 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, $\mathsf{FC}(\mathcal{V}, t) := \text{GHOST}(\mathsf{FIL}(\mathcal{V}, t))$, *i.e.*, $\mathsf{FC} := \text{GHOST} \circ \mathsf{FIL}$. GHOST itself is contained in $\mathcal{G}_f$, characterized by the identity filter.

Equivocation discounting All fork-choice functions we consider from now on implement a technique to deal with equivocations, used in the Ethereum protocol, as a spam-resistance measure, *i.e.*, *equivocation discounting* [1][7]. 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 $\mathsf{FC}_{eq} \coloneqq \mathrm{GHOST} \circ \mathsf{FIL} \circ \mathsf{FIL}_{eq}$. All fork-choice functions considered in the following sections implement equivocation discounting, even if not explicitly stated.

4.1.3 LMD-GHOST

LMD-GHOST is an adaptation of the original GHOST [22], in which the protocol considers only each validator's most recent vote (LMD), which is assumed to be the most meaningful. LMD-GHOST belongs to the class $\mathcal{G}_f$ of fork-choice functions based on GHOST [22], and is characterized by a view filter $\mathsf{FIL}_{\mathrm{lmd}}$. $\mathsf{FIL}_{\mathrm{lmd}}(\mathcal{V}, t)$ 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.

Algorithm 3 LMD-GHOST Fork-Choice function

```
1: function LMD-GHOST($\mathcal{V}, t$)
2:     return GHOST($\mathsf{FIL}_{\mathrm{lmd}}(\mathsf{FIL}_{\mathrm{eq}}(\mathcal{V}, t))$)
3: function $\mathsf{FIL}_{\mathrm{lmd}}(\mathcal{V}, t)$
4:     $\mathcal{V}' \leftarrow \mathcal{V}$ without all but the most recent (latest) votes
5:             of each validator
6:     return $(\mathcal{V}', t)$
7: function $\mathsf{FIL}_{\mathrm{eq}}(\mathcal{V}, t)$
8:     $\mathcal{V}' \leftarrow \mathcal{V}$ without all votes by equivocators in $\mathcal{V}$
9:     return $(\mathcal{V}', t)$
```

4.2 LMD-GHOST with view-merge and no subsampling

The first propose-vote-merge protocol that we consider is a variation of the original LMD-GHOST protocol, with the addition of view-merge and without considering subsampling of validators, the latter in order to prevent *ex-ante reorgs*[7].

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 4.1.3. From now on, we refer to this propose-vote-merge protocol simply as LMD-GHOST.

We first introduce the following notation. We denote with $\widetilde{H}_t$ the set of honest voters of slot t that *are never corrupted*, *i.e.*, $\widetilde{H}_t \coloneqq \lim_{s\to\infty} H_t \setminus A_s = H_t \setminus A_\infty$. 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 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 3 (Strong reorg resilience)**.** *Consider an honest proposal B from a slot t in which network synchrony hold and $|\widetilde{H}_t| > \frac{n}{2}$. Suppose that 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 $\widetilde{H}_t$.*

[7]The Ethereum protocol has (disjoint) committees of (the whole set 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 the LMD-GHOST protocol implemented in Ethereum is not a reorg resilient protocol, even in the full participation setting.

Proof. By Lemma 1, all honest voters of slot t broadcast 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 and the result follows. $\square$

On the other hand LMD-GHOST is significantly limited in its support of dynamic participation, as shown in the following theorem. In particular, we present 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. The reason why this attack is possible is due to the fact that τ-sleepiness only considers votes from the last τ slots, but LMD-GHOST does not have vote expiry, so all votes are relevant to the fork-choice. 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.

Theorem 4. *LMD-GHOST is not τ-dynamically-available for any finite τ and any confirmation rule with finite confirmation time T_{conf}.*

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 such execution 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[8]. 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. $\square$

4.3 Goldfish

Goldfish is a simplified variant of LMD-GHOST, introduced by D'Amato, Neu, Tas, and Tse [7], 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 1-reorg-resilient and 1-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 [7], and consider a version of Goldfish which fulfills the specification of a propose-view-merge protocol. This variant does not consider

[8]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 [15].

subsampling and replaces the VRF lottery considered in the original protocol with an SSLE proposer selection mechanism (see Section 2). Note that accommodating these features of the original Goldfish protocol would only require a small extension of the family of propose-vote-merge protocols, which we do not consider here for simplicity. In fact, the analysis of [7] is almost exactly identical to the analysis of this variant of Goldfish, in particular relying on reorg resilience in the same way. Crucially, in the case of Goldfish, Proposition 1 is true even if subsampling is considered, due to the strict vote expiry.

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

4.3.1 Vote Expiry

The notion of *vote expiry* has been introduced 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 $\eta \in \mathbb{N}$ with $\eta \geq 0$, 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. In particular, by assuming an expiration period, the capabilities of the adversary are limited, as votes broadcast before slot $t-\eta$ are not considered.

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

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

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

Vote expiry allows the protocol to support dynamic participation.

4.3.2 Properties and limitations

D'Amato, Neu, Tas, and Tse [7] show that Goldfish is 1-reorg-resilient and 1-dynamic-available, as also follows from Theorems 6 and 7 in our analysis of RLMD-GHOST, for the special case $\eta = 1$. On the other hand, 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 with a single adversarial validator, and with all validators 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} itself. 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 there is a single adversarial validator, 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 5. *Goldfish is not (τ, π)-asynchrony-resilient for any $\tau > \pi \geq 2$.*

5 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 4.3.1) with the latest message driven (LMD) fork-choice (see Section 4.1.3). 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 [21]. 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

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

5.1 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 5.2, 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 4, motivating why the η-sleepiness assumption is required.

5.2 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 3, 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 2), the following results are also satisfied for $\tau \geq \eta$. In Appendix A, we show that these results are tight.

Theorem 6 (Reorg resilience)**.** *RLMD-GHOST is η-reorg-resilient.*

Theorem 7 (Dynamic availability)**.** *RLMD-GHOST is η-dynamically-available.*

Theorem 8 (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 8 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.

6 Related works

The sleepy model of consensus, introduced by Pass and Shi [18], abstracts the functioning implicitly introduced by the Bitcoin protocol [14], modeling a distributed system in which participants can be either online or offline; in other words, where the participation is *dynamic*, assuming only that at any given time a majority of online participants are honest. Moreover, Pass and Shi show that the longest chain protocol[9] is safe and live within its model, *i.e.*, that it is dynamically available. Longest chain protocols are also meaningfully resilient to temporary asynchrony: the deeper a block is in the longest chain, the longer the period of asynchrony required for it to be reorged. In other words, blocks *accumulate safety over time*. On the other end, longest chain protocols are not reorg resilient, as newly proposed blocks can displace those at the tip of the chain.

Other than longest chain, to our knowledge only two other types of dynamically available protocols have been designed so far. One is Goldfish [7], a variant of LMD-GHOST [5, 23] which we have discussed extensively in this work. Unlike longest chain protocols, Goldfish is reorg resilient, but not at all tolerant of asynchrony. Finally, another class of approaches to integrate dynamic participation within a consensus protocol has been recently devised [13, 12]. Similarly to Goldfish, these do not tolerate even temporary asynchrony, raising the same questions about the practicality of their usage.

Momose and Ren [13] 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 upon a graded agreement protocol using a quorum-based design.

Malkhi, Momose, and Ren [12] improves on the latency of Momose and Ren [13], and present a synchronous protocol that is live under *fully fluctuating participation*. They present a Byzantine atomic broadcast protocol in the sleepy model [18] with 3 round latency, but tolerating only one-third Byzantine nodes, rather than one-half. This solution has been recently improved [19] achieving optimal $\frac{1}{2}$ corruption threshold.

7 Conclusion and future work

Dynamically available protocols have recently been explored in the context of blockchain protocols, based on (variants of) the sleepy model [18]. In this work we presented a generalization of this model, with more generalized and stronger constraints in the corruption and sleepiness power of the adversary, and we formally proved properties and limitations of (a variant of) LMD-GHOST [23], the dynamically available component of Gasper [5], Goldfish [7], a synchronous dynamically available and reorg resilient protocol, and RLMD-GHOST, our novel protocol. Table 1 summarizes the properties achieved by these protocols. RLMD-GHOST results in a provably secure synchronous dynamically available protocol that can be regarded as a potential future substitute for the existing LMD-GHOST component in Gasper.

A characterization through components of Gasper has been first formalized by Neu, Tas, and Tse [16]. Motivated by understanding Gasper [5] design goals, the authors introduce the *partially synchronous sleepy model*, and then define the desiderata of the Ethereum's consensus protocol through the notion of an *ebb-and-flow* protocol. In the partially synchronous 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 particular, a (secure) ebb-and-flow protocol is constituted by both a dynamically available protocol that, if $\mathsf{GST} = 0$, *i.e.*, under synchrony, is guaranteed to be safe and live at all times (according to Definition 4), *and* a finalizing protocol 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 represented by LMD-GHOST, while the latter by Casper [4]. Understanding the interaction between RLMD-GHOST and a finality component is an interesting open problem, especially with respect to asynchrony. Specifically, we reserve for future research the development of a secure ebb-and-flow protocol which incorporates RLMD-GHOST as a dynamically available component, along with a partially synchronous finality component. Moreover, it is an open question whether the techniques of this work can be applied to the family of quorum-based dynamically

[9]Inspired by the Bitcoin protocol, but replacing leader election through proof of work with a VRF-based election

	Dynamic Availability	**Asynchrony Resilience**
LMD-GHOST	✗ (only $\tau = \infty$, Theorem 4)	✓ ($\tau, \pi = \infty$, Theorem 8)
Goldfish	✓ ($\tau = 1$, Theorem 7)	✗ (Theorem 5)
RLMD-GHOST, expiry period η	✓ ($\tau = \eta$, Theorem 7)	✓ ($\tau, \pi = \eta, \eta - 1$, Theorem 8)

Table 1: The properties achieved by LMD-GHOST with view-merge and no subsampling (Section 4.2), Goldfish (Section 4.3), and RLMD-GHOST with expiry period $\eta \in (1, \infty)$ (Section 5).

available protocols [13, 12], to strengthen their resilience of asynchrony at the cost of weakened tolerance to dynamic participation, as we have done with Goldfish.

References

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

[2] 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`.

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

[4] 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`.

[5] 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`.

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

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

[8] Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[9] 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`.

[10] 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`.

[11] Andrew Lewis-Pye and Tim Roughgarden. Resource pools and the CAP theorem. *CoRR*, abs/2006.10698, 2020. URL: `https://arxiv.org/abs/2006.10698`.

[12] 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`.

[13] 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`.

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

[15] 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`.

[16] 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.

[17] 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`.

[18] 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.

[19] Chainlink Labs Research. Minority corruption resilience in byzantine generals with unknown and fluctuating participation, 2022. URL: `https://blog.chain.link/minority-corruption-resilience-in-byzantine-generals-with-unknown-and-fluctuating-participation/`.

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

[21] 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`.

[22] 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.

[23] 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

We first show limitations on the reorg resilience of RLMD-GHOST when $\eta > \tau$, except for $\eta = 1$, *i.e.*, Goldfish. We construct a τ-compliant execution in which old, unexpired votes of honest validators which are currently asleep are used by the adversary to fuel a reorg.

Theorem 9. *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, broadcasting 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. □

The second limitative result we present concerns dynamic availability. Unsurprisingly, an expiry period $\eta > \tau$ means that $\mathsf{RLMD\text{-}GHOST}$ is also not τ-dynamically-available.

Theorem 10. *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. 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 broadcast 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 corrupt 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 11. *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. 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, $\mathsf{FC} =$ RLMD-GHOST.

B.1 Protocol with fast confirmation

In the following, and in Algorithm 6, we update the confirmed chain explicitly, contrary to Algorithm 1. 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 3.2

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 [16, 17], 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 [10]. 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.

[10]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

```
1: State
2:     $\mathcal{V}_i \leftarrow \{B_{\text{genesis}}\}$: view of validator $v_i$
3:     $\mathcal{B}_i \leftarrow \emptyset$: buffer of validator $v_i$
4:     $\mathsf{ch}_i \leftarrow B_{\text{genesis}}$: canonical chain of validator $v_i$
5:     $\mathsf{Ch}_i \leftarrow B_{\text{genesis}}$: confirmed chain of validator $v_i$
6:     $t \leftarrow 0$: the current slot
7:     $r \leftarrow 0$: the current round
8:     $\mathit{sentvote} \leftarrow$ FALSE: indicates whether $v_i$ has voted in slot $t$
   PROPOSE
9: at $r = 3\Delta t$ do
10:    if $v_i = v_p^t$ then
11:        $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
12:        $\mathcal{B}_i \leftarrow \emptyset$
13:        $B' \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
14:        $B \leftarrow \mathsf{NewBlock}(B')$                // append a new block on top of $B'$
15:        $\mathsf{ch}_i \leftarrow B$
16:        send message [PROPOSE, $B$, $\mathcal{V}_i \cup \{B\}$, $t$, $v_i$] through gossip
   VOTE AND CONFIRM
17: at $r = 3\Delta t + \Delta$ do
18:    if $\neg\mathit{sentvote}$ then
19:        $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
20:        send message [VOTE, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$] through gossip
21:        $\mathit{sentvote} \leftarrow$ TRUE
22:    $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
23:    $\mathcal{B}_i \leftarrow \emptyset$
24:    $B_{\text{fast}} \leftarrow B_{\text{genesis}}$
25:    $S_{\text{fast}} \leftarrow \{B \prec \mathsf{ch}_i \colon |\{v_i \colon \exists B' \succ B\ (\text{VOTE}, B', t, v_i) \in \mathcal{V}_i\}| \geq \frac{2}{3}n\}$
26:    if $S_{\text{fast}} \neq \emptyset$ then:
27:        $B_{\text{fast}} \leftarrow \arg\max_{S_{\text{fast}}} |B|$
28:    if $\neg(B_{\text{fast}} \prec \mathsf{Ch}_i \wedge \mathsf{ch}_i^{\lceil \kappa} \prec \mathsf{ch}_i)$ then:
29:        $\mathsf{Ch}_i \leftarrow \arg\max_{\mathsf{ch} \in \{\mathsf{ch}_i^{\lceil \kappa}, B_{\text{fast}}\}} |\mathsf{ch}|$
   MERGE
30: at $r = 3\Delta t + 2\Delta$ do
31:    $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{B}_i$
32:    $\mathcal{B}_i \leftarrow \emptyset$
33:    $\mathit{sentvote} \leftarrow$ FALSE
34: upon receiving a gossiped message [PROPOSE, $B$, $\mathcal{V}$, $t$, $v_p^t$] do
35:    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
36:    if $\neg\mathit{sentvote}$ and $r \in [3\Delta t, 3\Delta t + \Delta]$ then
37:        $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{V}$
38:        $\mathsf{ch}_i \leftarrow \mathsf{FC}(\mathcal{V}_i, t)$
39:        send message [VOTE, $\mathsf{FC}(\mathcal{V}_i, t)$, $t$, $v_i$] through gossip
40:        $\mathit{sentvote} \leftarrow$ TRUE
41: upon receiving a gossiped message $V =$ [VOTE, $B$, $t'$, $v_i$] from $v_i$ do
42:    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{V\}$
43: upon receiving a gossiped message $B =$ [BLOCK, $b$, $t'$, $v_i$] from $v_i$ do
44:    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{B\}$
```

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 1) in our security analysis of fast confirmations, in the sense that it provides the base case for the induction of Theorem 1, 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. $\square$

Theorem 12 (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 1, using Lemma 5 instead of Lemma 1 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. $\square$

Theorem 13 (Dynamic availability)**.** *RLMD-GHOST with fast confirmations is η-dynamically-available.*

Proof. η-liveness follows directly from Theorem 7, 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 12. 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 7 then implies the η-safety of the protocol. $\square$

Theorem 14 (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 1, 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. $\square$