Goldfish: No More Attacks on Proof-of-Stake Ethereum

Francesco D'Amato
francesco.damato@ethereum.org

Joachim Neu
jneu@stanford.edu

Ertem Nusret Tas
nusret@stanford.edu

David Tse
dntse@stanford.edu

Abstract—**The latest message driven (LMD) greedy heaviest observed sub-tree (GHOST) consensus protocol is a critical component of proof-of-stake (PoS) Ethereum. In its current form, the protocol is brittle, as evidenced by recent attacks and patching attempts. We report on Goldfish, a considerably simplified candidate under consideration for a future Ethereum protocol upgrade. We prove that Goldfish satisfies the properties required of a drop-in replacement for LMD GHOST: Goldfish is *secure* in synchronous networks under dynamic participation, assuming a majority of the nodes (called *validators*) follows the protocol. Goldfish is *reorg resilient* (*i.e.*, honestly produced blocks are guaranteed inclusion in the ledger) and supports *fast confirmation* (*i.e.*, the expected confirmation latency is independent of the desired security level). We show that subsampling validators can improve the *communication efficiency* of Goldfish, and that Goldfish is *composable with finality gadgets and accountability gadgets*, which improves state-of-the-art ebb-and-flow protocols. Attacks on LMD GHOST exploit lack of coordination among honest validators, typically provided by a locking mechanism in classical BFT protocols. However, locking requires votes from a quorum of all participants and is not compatible with dynamic availability. Goldfish is powered by a novel coordination mechanism to synchronize the honest validators' actions under dynamic participation. Experiments with our implementation of Goldfish demonstrate the practicality of this mechanism for Ethereum.**

I. Introduction

A. A History of Attacks and Patches for LMD GHOST

The latest message driven (LMD) greedy heaviest observed sub-tree (GHOST) [1], [2] consensus protocol is a key component of the Gasper protocol [3] that powers proof-of-stake (PoS) Ethereum's beacon chain since 'the Merge' (Fig. 1). LMD GHOST proceeds in synchronized slots, at the beginning of which a pseudo-randomly elected leader proposes a block with new transactions at the tip of the *canonical chain*. Subsequently, members of a pseudo-randomly elected committee cast votes for the tip towards block confirmation (Fig. 2). To determine the tip of the canonical chain, the proposer and voters of a slot t walk down the tree of blocks starting at the genesis block and following the LMD GHOST fork-choice rule [3, Alg. 3.1] (cf. Alg. 2): At each block B, the validator chooses the child of B whose subtree is *heaviest*, *i.e.*, received the largest number of *unique latest* votes for its blocks. Here, by the LMD rule, only the *latest* vote cast by each committee member of previous slots is considered.

The initial version of LMD GHOST specified by Gasper [3] is susceptible to the *balancing attack*, which was first shown for synchronous and partially synchronous networks with adversarial message delay [4], [5], and later for networks with non-adversarial, merely random network delay [6]–[8]. In the attack, the adversary exploits the lack of a *coordination mechanism* for synchronizing the views of honest validators; so that different committee members vote for conflicting blocks at each slot. For instance, suppose the adversary controls the proposer and one voter at some slot t. The adversarial validator proposes two conflicting blocks (B_{L} and B_{R}) and shows one block to half of the honest voters and the other to the remaining half. As a result, the votes cast by the committee of slot t are split evenly between the two blocks. At slot $t+1$, the honest proposer builds its block B_{H} on B_{L}. However, before the committee of slot $t+1$ can vote, the adversarial voter of slot t broadcasts a vote for B_{R} to slot $t+1$'s committee. This makes B_{R} heavier in terms of votes than B_{L} and B_{H}, *swaying* slot $t+1$'s committee to vote for B_{R} instead of B_{H}. By swaying only part of the committee, the adversary can maintain the split among voters and repeat the attack.

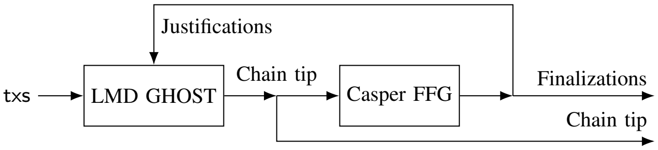

Fig. 1. Gasper [3] consists of two sub-protocols: LMD GHOST ('fork choice rule') and Casper FFG [13] ('finality gadget'). The desired properties for Gasper were formalized by the *ebb-and-flow* design objective [4], [6], [14], which consists of security of the full chain under dynamic participation of validators, and accountable security of a finalized prefix.

Designing a coordination mechanism for voters is key to preventing the swaying of honest validators' votes. For this purpose, in response to the balancing attack, a patch called *proposer boosting* was added to the protocol [9]. Proposer boosting gives proposals a temporary weight to ensure that the committees vote for their slot's now heavier proposals. However, it was subsequently demonstrated that the LMD functionality alone can be exploited to conduct a balancing-type attack *despite* proposer boosting [10], [11], and that LMD GHOST without the LMD aspect would suffer from a so called *avalanche attack* [10], [12]. Again in response, a patch called *equivocation discounting* was added to the protocol. Not least because of its complexity, the protocol with these patches has so far defied security analysis—both in terms of giving a formal security proof as well as further attacks. This leaves room for an uncomfortable amount of doubt about the security of Ethereum's ecosystem worth hundreds of billions of USD.

The authors are listed alphabetically.

B. Requirements for LMD GHOST in the Context of Ethereum

As LMD GHOST's vulnerability to attacks stems from the lack of a coordination mechanism, for inspiration towards a secure Ethereum, we analyze how existing protocols synchronize the views of honest validators. For example, classical Byzantine fault tolerant (BFT) consensus protocols such as PBFT [15], Tendermint [16], HotStuff [17] and Streamlet [18] (hereafter called PBFT-like protocols) require a *quorum* of votes from a large fraction of the whole validator set to *certify* blocks. These quorums are subsequently used to convince honest validators to vote for descendants of *certified* blocks, thus serving as a coordination mechanism. This is typically enforced by a *locking* rule that dictates honest validators to *lock* on such certified blocks and vote for them or their descendants thereafter. However, as blocks cannot be certified without an *absolute* quorum of votes by a fraction of all validators, none of the aforementioned protocols satisfies liveness under *dynamic participation*.

Dynamic participation was first formalized by the sleepy model of consensus [19], where the number and identity of validators actively participating in the protocol can change over time. Guaranteeing **safety and liveness under dynamic participation** is crucial in the semi-permissionless model of public PoS blockchains, which have little control over the participants and limited ability to react quickly to unforeseen dropouts due to regulatory requirements or software/hardware updates. At the time of writing, approximately 70% of Ethereum validators[1] follow U.S. Office of Foreign Assets Control (OFAC) regulations, and ignore certain transactions. It is conceivable that in the future, these 70% of validators could selectively abstain from voting for certain blocks, thus behaving like temporary crash faults. Satisfying security under dynamic participation is indeed one reason behind LMD GHOST's fork-choice rule that avoids absolute quorums and selects blocks with *relatively* heavier quorums.

Given tolerance to dynamic participation as a requirement, we next turn to PoS variants of Nakamoto's *longest chain* (LC) protocol (*e.g.*, Ouroboros [20]–[22], Sleepy/SnowWhite [19], [23]), the first BFT PoS protocols for the dynamic participation model. In LC protocols, there is no committee of voters. Instead, each new block fundamentally acts as a vote in favor of its ancestor blocks. As validators do not rely on votes from a large committee to confirm blocks, they opt to wait a long time horizon before confirming blocks; so that the honest validators can observe all previously proposed blocks in the prefix of the longest chain (*i.e.*, votes for the LC) and converge on this prefix. In contrast to LMD GHOST, where the timed release of adversarial votes sway different validators to vote for different blocks, the relatively small number of blocks (as opposed to votes) and the long confirmation latency[2] enable the longest chain to serve as a synchronization mechanism.

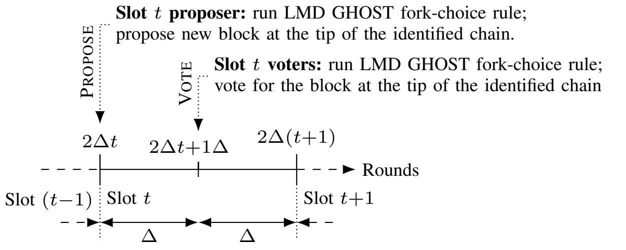
Slot t proposer: run LMD GHOST fork-choice rule;
propose new block at the tip of the identified chain.
PROPOSE
VOTE
Slot t voters: run LMD GHOST fork-choice rule;
vote for the block at the tip of the identified chain
$2\Delta t$
$2\Delta t{+}1\Delta$
$2\Delta(t{+}1)$
Rounds
Slot $(t{-}1)$
Slot t
Slot $t{+}1$
Δ
Δ

Fig. 2. LMD GHOST has time slots of two phases of Δ duration each. Each time slot has a pseudorandomly elected proposer and a committee of voters. PROPOSE: At the start of a slot, the *proposer* runs the LMD GHOST fork-choice rule [3, Alg. 3.1] (cf. Alg. 2) and proposes a block extending the tip of the identified chain. VOTE: Midway into a slot, *voters* run the LMD GHOST fork-choice rule and vote for the block at the tip of the identified chain.

Security of LC protocols come at the expense of a long confirmation latency that is linear in the security parameter κ. However, a desirable property for LMD GHOST was to support **fast confirmations**, *i.e.*, having an expected confirmation latency that does not depend on the security parameter κ. This enables achieving low failure probability without increasing latency. Moreover, in LC protocols, an honestly produced block can be displaced by each adversarial block, *e.g.*, using the selfish mining attack [24]. In the case of Ethereum, where blocks contain excessively valuable transactions [25], validators are incentivized to carry out selfish mining and similar block reorganization attacks such as *undercutting* [26], [27] and *time-bandit* [28] attacks. For resilience against such attacks, a protocol for the role of LMD GHOST must satisfy **reorg resilience**: whenever an honest validator is selected as the proposer, its proposal block eventually enters all ledgers output by honest validators, with the proposal's prefix determined at the time of its production.

C. Key Techniques of Goldfish

Failure of LC protocols to satisfy reorg resilience and fast confirmation due to 'too few votes spread across too much time' suggests to employ a committee of voters that can create a quorum of votes supporting honest proposals soon after they are broadcast. However, as absolute quorums are incompatible with liveness under dynamic participation, rather than using the absolute number of votes, a dynamically available protocol must use their relative weights to favor blocks with stronger support during fork choice. Together, these observations corroborate some structural elements of LMD GHOST for the design of our new protocol Goldfish: a slot structure with a proposer and committee that votes, using relative weights of quorums. To provide the coordination mechanism that is lacking in LMD GHOST and thereby satisfy its desiderata, Goldfish employs additional techniques:

Message buffering[3] requires each validator to buffer votes received from the network and carefully time the inclusion of

[1] https://www.mevwatch.info/ (Jan. 2023)

[2] Confirmation latency denotes the time for any transaction sent by an honest validator to enter the ledgers of all honest validators. It is a *random variable* that depends on the sequence of block proposers.

[3] The message buffering technique was publicized under the name *view-merge* by an author of this work in a blog post [29] dated Oct. 2021. We have later observed that a similar technique was used in the Highway protocol in unpublished work [30] in Jan. 2021.

these votes into its local view, with priority given to the votes relayed by the proposer. As a result, in slots with an honest proposer, all honest validators adopt the view of the proposer and vote in favor of the proposed block. This ensures that honest proposals are guaranteed to remain in the canonical chain, implying reorg resilience. Going back to the balancing attack example in Sec. I-A, with message buffering, honest validators will not immediately consider the adversary's late released vote for block B_{R}. They will include the late released vote into their views only after voting for B_{H}, thus preventing the adversary from swaying them. Since honest proposals contain fresh transactions and stabilize their prefix, and long streaks of adversarial proposals are exponentially unlikely, safety and liveness of the protocol *when every validator votes in every slot* follow readily from message buffering.

Vote expiry means that during each time slot, only votes from the immediately preceding slot influence the protocol's behavior.[4] Hence, stale votes of offline validators cannot support blocks conflicting with proposals of future slots, and the protocol preserves reorg resilience and security *under dynamic participation*. Vote expiry also allows the protocol to *subsample small committees* of voters per slot from the full set of validators (for improved communication efficiency), keeping the number of votes small that affect actions of honest validators in the short term. Thus, only few protocol messages need to be buffered and considered by honest validators at any point in time. Vote expiry is therefore also a prerequisite for the feasibility and efficiency of message buffering.

D. Properties Achieved by Goldfish

Goldfish achieves all the requirements for LMD GHOST identified in Sec. I-B: (a) Goldfish is *provably secure*, *i.e.* safe and live, under *dynamic participation* assuming honest majority of *validators*, and network synchrony with adversarial network delay, up to a known upper bound Δ. (b) Goldfish is *reorg resilient*, *i.e.*, honest proposals enter all ledgers output by honest validators. (c) Goldfish satisfies *optimistic fast confirmation*: under optimistic conditions, *i.e.*, when participation is high and $\frac{3}{4}$ fraction of validators are honest, it confirms transactions with constant expected latency independent of κ.

Besides the properties above, Goldfish supports *subsampling* of validators, to improve communication efficiency and achieve resilience to *adaptive corruption*. Goldfish periodically and pseudo-randomly selects a small group of validators to run the protocol on behalf of the whole validator set. This results in a considerably lower communication overhead and allows for a large validator set. Furthermore, the selected validators send only a single protocol message. Thus, the protocol satisfies *player-replaceability* [31], [32] and is secure against adaptive adversaries, which can corrupt validators during protocol execution.

As desired of LMD GHOST (Fig. 1), Goldfish is also composable with *finality and accountability gadgets* such as [4], [6], [13], [14]. The goal of these gadgets is to checkpoint blocks within the confirmed blockchain such that these blocks and their prefixes remain safe under network partitions. The composite protocol (cf. Fig. 4) can achieve the *ebb-and-flow consensus formulation* [4], [6] desired of Ethereum's beacon chain, which is to produce an available full ledger that is secure under dynamic participation, and a prefix ledger that is accountably secure under network partition.

Goldfish is intentionally simple, and similar to LMD GHOST as currently deployed. Moreover, message buffering and vote expiry can be realized with modest changes to the vote accounting logic of LMD GHOST. Requiring only modest implementation changes, Goldfish can thus serve as a drop-in replacement for LMD GHOST and is a credible candidate under consideration for a future upgrade of Ethereum consensus. Given the earlier attacks on variants of LMD GHOST, Goldfish is the first positive result with security proof for a close variant, strengthening the confidence in this family of protocols. Simplicity of Goldfish also makes it a good pedagogical example as a feature-rich consensus protocol for synchronous networks under dynamic participation.

E. Related Works

For Goldfish, we consider the sleepy model [19] under synchrony. The first secure consensus protocol for the sleepy model [19] (dynamic availability) was Nakamoto's LC protocol, first based on proof-of-work (PoW) in Bitcoin [41], [42], and subsequently on PoS in Ouroboros [20]–[22] and Sleepy Consensus/Snow White [19], [23] (see Tab. I for a comparison of Goldfish with related works). Parallel composition of LC protocol instances was suggested in [33], [34] to overcome the scaling of LC protocols' confirmation latency with the security parameter κ. For the same goal, Thunderella [35] proposed combining an asynchronous protocol achieving optimistic fast confirmation with a slow LC protocol for when the adversarial fraction is high. However, as observed in Sec. I-B, LC protocols and Thunderella that builds on a LC protocol are not reorg-resilient. Moreover, under optimistic conditions, Thunderella recovers fast confirmation only after a period of LC confirmation delay, whereas Goldfish can instantaneously start fast-confirming blocks under optimistic conditions.

Many classical PBFT-like consensus protocols [15], [17], [18] have constant (expected) confirmation latency and can be reorg resilient, but do not tolerate dynamic participation. Highway [30] enables confirming blocks using different absolute quorum sizes; however it does not support dynamic participation. An early 'classical' BFT protocol for a model with unknown (but static) participation is due to Khanchandani and Wattenhofer [36], [43]. A subsequent protocol by Goyal et al. [37] supports dynamic participation with confirmation latency independent of the participation level, but still linear in the security parameter κ [40]. Confirmation latency independent of the security parameter is achieved in the permissionless PoW setting with omission faults by [44].

A recent work by Momose and Ren [40] presents the first permissioned/PoS protocol that supports dynamic participation with confirmation latency independent of the security

[4] Alleged forgetfulness of its animal namesake inspired Goldfish's name.

TABLE I

COMPARISON OF GOLDFISH WITH RELATED WORKS REGARDING KEY DESIDERATA. OPTIMISTIC ('opt.') FAST CONFIRMATION REQUIRES HIGH PARTICIPATION AND LESS THAN $\frac{1}{4}$ ADVERSARIAL FRACTION. DYNAMIC PARTICIPATION '✓ (slow)' INDICATES THE PROTOCOL REMAINS LIVE ONLY UNDER SLOW FLUCTUATIONS IN PARTICIPATION. A NUMBER NEXT TO '✓' FOR FAST CONFIRMATION DENOTES THE MINIMUM CONFIRMATION LATENCY. RESPONSIVE ('resp.') CONFIRMATION MEANS WITH DELAY OF THE ACTUAL NETWORK DELAY RATHER THAN DELAY BOUND Δ.

	LC [22] [23], [33], [34]	Thunderella [35]	PBFT-like prot. [15]–[18]	Highway [30]	Khanchandani et al. [36]	Goyal et al. [37]	Malkhi et al. [38], [39]	Momose et al. [40]	Goldfish (this work)
Dynamic participation	✓	✓ (slow)	✗	✗	✓ (slow)	✓	✓	✓	✓
Reorg resilience	✗	✗	✓	✓	✓	✓	✓	✓	✓
Adversarial resilience	1/2	1/2	1/3	flexible	1/3	1/2	1/3	1/2	1/2
Fast confirmation	✗	opt. (resp.)	✓ (resp.)	✓	✓	✗	✓ (3Δ)	✓ (37Δ)	opt. (4Δ)

parameter and participation level. Whereas [40] ensures fast confirmation under all validator fluctuations with adversarial resilience $\frac{1}{2}$, its confirmation latency is 37Δ, much larger than the latency of Goldfish (4Δ) under optimistic conditions. In the contemporary but independent work [38], [39], the prerequisites for liveness were relaxed and latency was improved to 3Δ, at the expense of reduced adversarial resilience (from $\frac{1}{2}$ down to $\frac{1}{3}$). By using authenticated channels, these works maintain security with $\frac{1}{3}$ resilience under all types of validator fluctuations, even when the number of adversarial validators goes down (all prior dynamically available PoS protocols assume constant or increasing number of adversarial validators). However, the assumption of authenticated channels to support 'un-corruption' or 'sleeping adversary nodes' is not practical in the semi-permissionless model of Ethereum.

Latest works [45], [46] retain $\frac{1}{2}$ resilience while supporting faster confirmation and participation fluctuation than [40].

F. Follow-Up Works & Adoption

A challenge that Goldfish shares with other reorg-resilient fast-confirming dynamically-available protocols (*e.g.*, [39], [40]), is that even short periods of asynchrony can have an outsize impact on security. In Goldfish, this is due to vote expiry: if new votes are not received timely to replace expiring ones, blocks can be reorged (up to the latest checkpoint, if used with a finality/accountability gadget). A Goldfish variant in a follow-up work [47] addresses this issue, by trading off a longer vote expiry period for a less dynamic participation model. In turn, the current candidate protocol [48] to provide single-slot finality for Ethereum is based on this protocol.

G. Outline

We recapitulate the model of synchronous networks with dynamic participation and asynchronous periods in Sec. II, before describing our basic Goldfish protocol in Sec. III, and its optimistic fast confirmation rule in Sec. IV. We prove the security properties in Sec. V. We conclude with a discussion of implementation aspects and experimental results in Sec. VI.

II. MODEL AND PRELIMINARIES

We review cryptographic primitives, how to model environment and adversary, and the consensus security desiderata.

A. Preliminaries

1) Security parameters: We denote by λ and κ the security parameters associated with the cryptographic primitives employed by Goldfish, and with Goldfish itself, respectively.[5] We say that an event happens with probability *negligible* in a security parameter μ, denoted by $\mathrm{negl}(\mu)$, if its probability is $o(1/\mu^d)$ for all $d > 0$. Overall, we say that an event happens *with overwhelming probability* (*w.o.p.*) if it happens except with probability (w.p.) $\mathrm{negl}(\kappa) + \mathrm{negl}(\lambda)$.

2) Digital signatures:

Definition 1 (Informal, cf. [49], [50])**.** A *signature* scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ consists of probabilistic poly-time (PPT) algorithms so that:

- $(\mathsf{ssk}, \mathsf{spk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$ creates a secret/public key pair.
- $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{ssk}, m)$ creates a signature on a message.
- $\{0, 1\} \leftarrow \mathsf{Sig.Verify}(\mathsf{spk}, m, \sigma)$ verifies a signature.
- *Correctness:* With overwhelming probability, for all messages, $\mathsf{Sig.Verify}(\mathsf{spk}, m, \mathsf{Sig.Sign}(\mathsf{ssk}, m)) = 1$.
- *Security (existential unforgeability):* An adversary with access to spk and to a signing oracle $\mathsf{Sig.Sign}(\mathsf{ssk}, .)$ cannot produce a valid (m, σ) other than via the oracle.

3) Verifiable random functions: A verifiable random function (VRF) [51] is used for leader election and subsampling of the validators within the Goldfish protocol.

Definition 2 (Informal, cf. [21, Sec. 3.2, Fig. 2], [32], [52])**.** A *verifiable random function* (VRF) scheme $\mathsf{Vrf} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ consists of PPT algorithms so that:

- $(\mathsf{vsk}, \mathsf{vpk}) \leftarrow \mathsf{Vrf.Gen}(1^\lambda)$ samples a VRF with associated secret/public key pair for evaluation/verification.
- $(y, \pi) \leftarrow \mathsf{Vrf.Eval}(\mathsf{vsk}, x)$ obtains the output y of the VRF at input x, and the evaluation proof π.
- $\{0, 1\} \leftarrow \mathsf{Vrf.Verify}(\mathsf{vpk}, x, (y, \pi))$ verifies an evaluation.
- *Correctness:* With overwhelming probability, for all inputs, $\mathsf{Vrf.Verify}(\mathsf{vpk}, x, \mathsf{Vrf.Eval}(\mathsf{vsk}, x)) = 1$.
- *Uniqueness:* Per input x, there is only one output y: if $\mathsf{Vrf.Verify}(\mathsf{vpk}, x, (y, \pi)) = 1$ for $(y, \pi) = (y_1, \pi_1)$ and $(y, \pi) = (y_2, \pi_2)$, then $y_1 = y_2$.
- *'Pseudorandomness':* Conceptually, the VRF behaves like a random oracle that is *unpredictable* (*i.e.*, without knowledge of vsk, the VRF output cannot be distinguished from

[5] The parameter κ with which to control Goldfish's consensus security failure probability is its (slow-path) confirmation latency, cf. Alg. 1, l. 29.

a random string) and *verifiable* (*i.e.*, given vpk, an alleged output of the VRF can be verified). For a formal definition, see [21, Sec. 3.2, Fig. 2].

B. Model

1) Validators: Goldfish is run among n validators, with identities $\mathsf{id} \in [n] \triangleq \{1, ..., n\}$. Each validator id generates a secret/public key pair $(\mathsf{ssk}_{\mathsf{id}}, \mathsf{spk}_{\mathsf{id}})$ and $(\mathsf{vsk}_{\mathsf{id}}, \mathsf{vpk}_{\mathsf{id}})$ for the signature and the VRF scheme, respectively. The public keys are known to all validators (*public-key infrastructure*, PKI).

Goldfish is designed to be used in the proof-of-stake setting. As is customary in the security analysis of PoS protocols [19], [20], we analyze Goldfish in the static stake setting, where each validator represents a protocol participant controlling unit stake[6]. Once proven secure under static stake, Goldfish can support dynamic stake via reconfiguration schemes [21], [23].

2) Environment and adversary: Time is divided into discrete *rounds* and the validators have synchronized clocks.[7] Validators receive transactions (txs) from the environment, and continuously output transaction ledgers to it ($\mathsf{ch}_r^{\mathsf{id}}$ for validator id and round r). The environment allows validators to *broadcast* messages to each other. The adversary is a probabilistic poly-time (PPT) algorithm that can leverage three aspects of the model (*corruption*, *sleepiness*, and *network delay*) in its attempt to undermine consensus. We first discuss these three aspects, and then the limits of the adversary.

3) Corruption: The adversary chooses f validators to corrupt (adaptively, subject to constraints detailed in Sec. II-B6), hereafter called *adversarial* validators (non-corrupt validators are *honest*). The internal state of corrupted validators is handed over to the adversary, which can then make them deviate from the protocol in arbitrary and coordinated fashion (*Byzantine faults*) for the remainder of the execution (permanent corruption). We define the adversarial fraction $\beta \triangleq f/n$.

4) Sleepiness: The adversary decides for each round and each honest validator whether it is *asleep* or not. Asleep validators do not execute the protocol (*temporary crash faults*). Messages delivered to an asleep validator get picked up by it only once the validator is no longer asleep. When a validator stops being asleep, it becomes *dreamy*. During this phase, it *joins* the protocol, usually over multiple rounds, using a special *joining procedure* specified by the protocol. Upon completion of this procedure, the honest validator becomes *awake* and then follows the 'standard path' of the protocol. Adversarial validators are always awake. The number of awake validators at any round is bounded below by a constant $n_0 > 0$.

5) Network delay: Messages sent between validators are delivered with an adversarially determined delay that can differ for each recipient. Upon picking up messages (once no longer asleep after delivery), an honest validator re-broadcasts them.

6) Adversary limits: A *partially synchronous network in the sleepy model* [4] has a global stabilization time (GST), a global awake time (GAT), and a delay upper-bound Δ. GST and GAT are constants unknown to the honest validators chosen *adaptively* by the adversary, *i.e.*, as causal functions of the execution, whereas Δ is a constant known to the validators. Before GST, message delays are arbitrarily adversarial (*asynchronous*). After GST, message delays are subject to the delay upper bound Δ (*synchronous*). Similarly, before GAT, the adversary can set the sleep schedule for honest validators. After GAT, all honest validators are awake.

Message delays and sleeping schedule are chosen adaptively. For corruption, Goldfish supports two assumptions. Either, we require *mildly* adaptive corruption, where it takes 3Δ rounds for corruption to take effect, together with the constraint that for every round r, the number of adversarial validators at round r must be less than the number of honest awake validators at round $r - 3\Delta$. Or, analogously to earlier works [21], [22], [32], through the use of key evolving signature and VRF schemes, we allow for fully adaptive corruption, together with the constraint that for every round r, the number of adversarial validators at round r must be less than the number of honest awake validators at round r. The precise technical assumptions are stated by Def. 5.

C. Consensus Security Desiderata

1) Security: We next formalize the notion of security *after a certain time*. Security is parameterized by κ, which, in the context of longest-chain protocols and Goldfish, determines the confirmation delay for transactions (*i.e.*, these protocols come with a security-latency trade-off). In our analysis, we consider a finite time horizon T_{hor} that is polynomial in κ. We denote a consensus protocol's output ledger, *e.g.*, the Goldfish ledger, in the view of a validator i at round r by ch_r^i. We write $\mathsf{ch}_1 \preceq \mathsf{ch}_2$ to express that the ledger ch_1 is a prefix of (or the same as) ledger ch_2.

Definition 3 (Security)**.** Let T_{conf} be a polynomial function of the security parameter κ. We say that a state machine replication protocol that outputs a ledger ch is *secure after time* T_{sec}, and has transaction confirmation time T_{conf}, iff:

- **Safety:** For any two rounds $r, r' \geq T_{\mathsf{sec}}$, and any two honest validators i, j awake at rounds r and r', respectively, either $\mathsf{ch}_r^i \preceq \mathsf{ch}_{r'}^j$ or $\mathsf{ch}_{r'}^j \preceq \mathsf{ch}_r^i$.
- **Liveness:** If a transaction has been received by some awake honest validator by some round $r \geq T_{\mathsf{sec}}$, then for any round $r' \geq r + T_{\mathrm{conf}}$ and any honest validator i awake at round r', the transaction will be included in $\mathsf{ch}_{r'}^i$.

The protocol satisfies $\bar{f}$-*safety* ($\bar{f}$-*liveness*) if it satisfies safety (liveness) as long as the number of adversarial validators f stays below $\bar{f}$ for all rounds. Similarly, the protocol satisfies $1/2$-*safety* ($1/2$-*liveness*) if it satisfies safety (liveness) if the fraction of adversarial validators β is bounded above away from $1/2$ for all rounds.

[6] Participants with large amount of stake manifest as multiple unit-stake validators controlled by the same entity. Indeed, in Ethereum, each unit stake of 32 ETH corresponds to one validator [53].

[7] Bounded clock offsets can be lumped into the subsequently discussed network delay upper bound Δ.

2) Accountable safety: Accountable safety provides a *trust-minimizing* strengthening of safety, with the aim to hold validators accountable for their actions. In case of a safety violation in a protocol with accountable safety resilience $\bar{f} > 0$, one can, after collecting evidence from sufficiently many honest validators, generate cryptographic proof that identifies $\bar{f}$ adversarial validators as protocol violators [6], [54]. By definition, the proof does not falsely accuse any honest validator, except with negligible probability.

3) The ebb-and-flow formulation: As $\mathsf{Goldfish}$ outputs a *dynamically available* ledger (*i.e.*, live under dynamic participation), by the availability-accountability dilemma [6], its output ledger cannot satisfy *accountable safety*. Similarly, it cannot satisfy safety under a partially synchronous network (*i.e.*, *finality*), by an analogue of the CAP theorem [55], [56]. However, $\mathsf{Goldfish}$ can be composed with an accountability gadget in order to obtain a separate prefix ledger that attains accountable safety under partial synchrony while staying consistent with the output of $\mathsf{Goldfish}$ [6]. Denoting the output of $\mathsf{Goldfish}$ as the available ledger $\mathsf{ch}_{\mathrm{ava}}$ and that of the accountability gadget as the accountable final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$, the desiderata are captured in the *ebb-and-flow formulation* [4]:

Definition 4 (Ebb-and-flow formulation [4], [6])**.**

1) (**P1: Accountability and finality**) Under a partially synchronous network in the sleepy model, the accountable final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$ has accountable safety resilience $n/3$ at all times, (except w.p. $\mathrm{negl}(\lambda)$), and there exists a constant $\mathbf{C}$ such that $\mathsf{ch}_{\mathrm{acc}}$ provides $n/3$-liveness with confirmation time T_{conf} after round $\max(\mathsf{GST}, \mathsf{GAT}) + \mathbf{C} \cdot \kappa$ (w.o.p.).
2) (**P2: Dynamic availability**) Under a synchronous network in the sleepy model (*i.e.*, for $\mathsf{GST} = 0$), the available ledger $\mathsf{ch}_{\mathrm{ava}}$ provides $1/2$-safety and $1/2$-liveness at all times (w.o.p.).
3) (**Prefix**) For each honest id and round r, $\mathsf{ch}^{\mathsf{id}}_{\mathrm{acc},r} \preceq \mathsf{ch}^{\mathsf{id}}_{\mathrm{ava},r}$.

The accountable final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$ can experience liveness violations before GST or GAT, due to lack of timely communication among sufficiently many honest validators, but $\mathsf{ch}_{\mathrm{acc}}$ remains accountably safe throughout. The available ledger $\mathsf{ch}_{\mathrm{ava}}$ can experience safety violations before GST, but remains live throughout. When conditions improve, $\mathsf{ch}_{\mathrm{acc}}$ catches up with $\mathsf{ch}_{\mathrm{ava}}$. This ebb-and-flow behavior lends the formulation its name. Providing the irreconcilable properties in two separate but *consistent* ledgers provides a user-dependent resolution to the CAP theorem [55], [56].

III. Protocol

We first describe the $\mathsf{Goldfish}$ protocol that is being proposed as a drop-in replacement for LMD GHOST in Ethereum's beacon chain. We then describe how $\mathsf{Goldfish}$ can be securely integrated with accountability and finality gadgets.

A. The $\mathsf{Goldfish}$ Protocol

The protocol (cf. Alg. 1) proceeds in *slots* of 3Δ rounds.

1) VRF-based lotteries: The VRF PKI enables cryptographic lotteries. A *lottery* $(\mathtt{tag}, \mathsf{thr})$ is defined by a fixed $\mathtt{tag}$ and threshold $\mathsf{thr} \in [0,1]$. Each validator id receives for each time slot t a lottery *ticket* (id, t). To *open* the ticket, id computes

$$\varrho \triangleq (y, \pi) \leftarrow \mathrm{Open}_{\mathsf{id}}^{(\mathtt{tag},\mathsf{thr})}(t) \triangleq \mathsf{Vrf.Eval}(\mathsf{vsk}_{\mathsf{id}}, \mathtt{tag} \,\|\, t). \quad (1)$$

An *opened ticket* with *opening* $\varrho$is *winning* for $(\mathtt{tag}, \mathsf{thr})$ iff:

$$\begin{aligned} &\mathrm{IsWinning}^{(\mathtt{tag},\mathsf{thr})}((\mathsf{id}, t), \varrho) \\ &\triangleq (\varrho.y \leq \mathsf{thr}\, 2^{\lambda}) \wedge \mathsf{Vrf.Verify}(\mathsf{vpk}_{\mathsf{id}}, \mathtt{tag} \,\|\, t, (\varrho.y, \varrho.\pi)). \end{aligned} \quad (2)$$

Finally, winning opened tickets are totally ordered by increasing *precedence*, $\mathrm{Prio}(\varrho) \triangleq \frac{\varrho.y}{2^{\lambda}} \in [0,1]$.

2) Data structures: Blocks and *votes* are central to $\mathsf{Goldfish}$. A block $B \triangleq (\mathtt{block}, (\mathsf{id}, t), \varrho, h, \mathsf{txs}, \sigma)$ consists of tag '$\mathtt{block}$', ticket (id, t) and opening ϱ to the $(\mathtt{block}, \mathsf{thr}_{\mathrm{b}})$ block production lottery, hash h committing to the new block's parent block and transactions txs (as block 'content'), and signature σ binding together block production opportunity and the block's content. A special *genesis block* $B_0 \triangleq (\mathtt{block}, (\perp, 0), \perp, \perp, \emptyset, \perp)$ is known to all validators.

A block B is *valid* iff:

$$\mathrm{IsValid}(B_0) \triangleq \mathtt{1} \quad (3)$$

$$\begin{aligned} \mathrm{IsValid}(B) \triangleq\ &\mathrm{IsWinning}^{(\mathtt{block},\mathsf{thr}_{\mathrm{b}})}((B.\mathsf{id}, B.t), B.\varrho) \\ &\wedge \mathsf{Sig.Verify}(\mathsf{spk}_{B.\mathsf{id}}, \mathtt{block} \,\|\, B.h \,\|\, B.\mathsf{txs}, B.\sigma) \\ &\wedge \mathrm{IsValid}(^*[B.h]) \wedge (B.t > {}^*[B.h].t). \end{aligned} \quad (4)$$

Here, $^*[B.h]$ means the parent block that $B.h$ commits to (namely, $^*[x]$ represents the block committed by hash x). The context within which these references get resolved is detailed with the different network message types below.

A vote $v \triangleq (\mathtt{vote}, (\mathsf{id}, t), \varrho, h, \sigma)$ consists of tag '$\mathtt{vote}$', ticket (id, t) and opening ϱ to the $(\mathtt{vote}, \mathsf{thr}_{\mathrm{v}})$ voting lottery, hash h committing to the block voted for (as vote 'content'), and signature σ binding together voting opportunity and the vote's content. Every vote v is tied to its time slot $v.t$ via the lottery ticket (id, t). A vote v is *valid* iff:

$$\begin{aligned} \mathrm{IsValid}(v) \triangleq\ &\mathrm{IsWinning}^{(\mathtt{vote},\mathsf{thr}_{\mathrm{v}})}((v.\mathsf{id}, v.t), v.\varrho) \\ &\wedge \mathsf{Sig.Verify}(\mathsf{spk}_{v.\mathsf{id}}, \mathtt{vote} \,\|\, v.h, v.\sigma) \\ &\wedge \mathrm{IsValid}(^*[v.h]) \wedge (v.t \geq {}^*[v.h].t). \end{aligned} \quad (5)$$

We call *block-vote-set* (short *bvset*) a set of blocks and votes. Commitments to blocks for the purpose of the references $v.h$ or $B.h$ are computed using $H(.)$. For a bvset $\mathcal{T}$ we denote by $\mathcal{T}[h]$ the block $B \in \mathcal{T}$ with $H(B) = h$, and $\perp$ if non-existent.

In $\mathsf{Goldfish}$, votes and blocks are encapsulated and exchanged in two network message types, *pieces* and *proposals*. A piece $M \triangleq (\mathtt{piece}, x)$ consists of tag '$\mathtt{piece}$' and for payload x either a vote or a block, and is *valid* iff:

$$\mathrm{IsValid}(M) \triangleq \mathrm{IsValid}(M.x). \quad (6)$$

Pieces are used to propagate blocks and votes and abstract Ethereum's peer-to-peer broadcast object propagation. In determining a piece's validity, block references $^*[.]$ are resolved

Algorithm 1 Goldfish executed by validator id with signature secret/public key $(\mathsf{ssk}_{\mathsf{id}}, \mathsf{spk}_{\mathsf{id}})$, VRF secret/public key $(\mathsf{vsk}_{\mathsf{id}}, \mathsf{vpk}_{\mathsf{id}})$, bvtree $\mathcal{T}$ and buffer $\mathcal{B}$. Here, notation '**at**' means executing the code block at the specified round, $\mathsf{ch}^{\mathsf{id}}$ denotes the Goldfish chain momentarily confirmed at id. For GHOST-Eph$(\mathcal{T}, t)$, see Alg. 2.

```
 1: $(\mathcal{B}, \mathcal{T}, t) \leftarrow (\emptyset, \emptyset, 0)$        ▷ Initialize buffer $\mathcal{B}$ and bvtree $\mathcal{T}$
 2: ▷ At all rounds, only valid messages not from later than $t$ are picked up from the
    network, re-broadcast, and put into $\mathcal{B}$ as specified in Sec. III-A4
 3: for $t = 1, 2, \ldots$ do                                      ▷ Slots
 4:     at $3\Delta t$ do                                           ▷ PROPOSE phase
 5:         $\varrho \leftarrow \mathrm{Open}_{\mathsf{id}}^{(\mathtt{block}, \mathsf{thr}_{\mathrm{b}})}(t)$          ▷ Check if eligible to propose
 6:         if $\mathrm{IsWinning}^{(\mathtt{block}, \mathsf{thr}_{\mathrm{b}})}((\mathsf{id}, t), \varrho)$ then
 7:             $\mathcal{T}' \leftarrow \mathrm{MERGE}(\mathcal{T}, \mathcal{B})$                  ▷ Bvtree to propose
 8:             $B \leftarrow \text{GHOST-EPH}(\mathcal{T}', t-1)$             ▷ Parent block
 9:             $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{ssk}_{\mathsf{id}}, \mathtt{block} \parallel H(B) \parallel \mathsf{txs})$
10:             $B \leftarrow (\mathtt{block}, (\mathsf{id}, t), \varrho, H(B), \mathsf{txs}, \sigma)$         ▷ New block
11:             $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{ssk}_{\mathsf{id}}, \mathtt{propose} \parallel \mathcal{T}' \parallel B)$
12:             Broadcast $(\mathtt{propose}, \mathcal{T}', B, \sigma)$               ▷ Propose
13:     at $3\Delta t + \Delta$ do                                      ▷ VOTE phase
14:         ▷ Filter for proposals from slot $t$
15:         $\mathcal{B}' \leftarrow \{(\mathcal{T}', B) \mid (\mathtt{propose}, \mathcal{T}', B, .) \in \mathcal{B} \land B.t = t\}$
16:         ▷ Identify the leader of slot $t$ and its proposal
17:         $(\mathcal{T}'^*, B^*) \leftarrow \arg\min_{(\mathcal{T}', B) \in \mathcal{B}'} \mathrm{Prio}(B.\varrho)$
18:         ▷ Merge own buffer and that of the leader into own bvtree
19:         $\mathcal{T} \leftarrow \mathrm{MERGE}(\mathcal{T}, \mathcal{T}'^* \cup \{B^*\})$
20:         $\varrho \leftarrow \mathrm{Open}_{\mathsf{id}}^{(\mathtt{vote}, \mathsf{thr}_{\mathrm{v}})}(t)$            ▷ Check if eligible to vote
21:         if $\mathrm{IsWinning}^{(\mathtt{vote}, \mathsf{thr}_{\mathrm{v}})}((\mathsf{id}, t), \varrho)$ then
22:             $B \leftarrow \text{GHOST-EPH}(\mathcal{T}, t-1)$              ▷ Target block
23:             $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{ssk}_{\mathsf{id}}, \mathtt{vote} \parallel H(B))$
24:             $v \leftarrow (\mathtt{vote}, (\mathsf{id}, t), \varrho, H(B), \sigma)$               ▷ New vote
25:             Broadcast $(\mathtt{piece}, v)$                          ▷ Vote
26:     at $3\Delta t + 2\Delta$ do                                     ▷ CONFIRM phase
27:         $\mathcal{T} \leftarrow \mathrm{MERGE}(\mathcal{T}, \mathcal{B})$                  ▷ Merge buffer and bvtree
28:         $B \leftarrow \text{GHOST-EPH}(\mathcal{T}, t)$           ▷ Canonical GHOST-Eph chain
29:         $\mathsf{ch}^{\mathsf{id}} \leftarrow B^{\lceil \kappa}$    ▷ Output ledger: $B$'s $\kappa$-deep prefix in terms of slots
```

with respect to the bvset $\mathcal{T}$ each validator maintains as part of its state, see Sec. III-A3. If a validator does not have any matching block in $\mathcal{T}$, it cannot currently determine the piece's validity. It then keeps the piece 'in limbo' for re-examination until its (in-)validity is established.[8]

A proposal $P \triangleq (\mathtt{propose}, \mathcal{T}, B, \sigma)$ consists of tag '$\mathtt{propose}$', bvset $\mathcal{T}$ and block B (as proposal content), and signature σ tying the proposal to the block production opportunity of B. Thus, a proposal P is *valid* iff:

$$\begin{aligned}\mathrm{IsValid}(P) \triangleq\; & \mathrm{IsValid}(P.B) \land \mathrm{IsConsistent}(P.\mathcal{T} \cup \{P.B\}) \\ & \land \mathsf{Sig.Verify}(\mathsf{spk}_{P.B.\mathsf{id}}, \mathtt{propose} \parallel P.\mathcal{T} \parallel P.B, P.\sigma) \\ & \land (\forall x \in P.\mathcal{T}\colon\; \mathrm{IsValid}(x) \land (x.t < P.B.t)) \qquad (7)\end{aligned}$$

where $\mathrm{IsConsistent}(\mathcal{T})$ is a predicate that is satisfied on a bvset $\mathcal{T}$ iff $B_0 \in \mathcal{T}$ and for every vote and block in $\mathcal{T}$ the referenced target/parent block is also in $\mathcal{T}$. We call a bvset $\mathcal{T}$ with $\mathrm{IsConsistent}(\mathcal{T})$ a *block-vote-tree* (short *bvtree*). In determining the validity of proposal P, block references $*[.]$ are resolved with respect to $P.\mathcal{T}$.

3) Validator state: Each validator keeps track of the current time slot t. It also maintains a bvtree $\mathcal{T}$ based on which it takes consensus decisions and actions. Finally, each validator maintains a *buffer* $\mathcal{B}$ of network messages (*i.e.*, pieces and proposals) that 'sits between' network and consensus protocol.

Algorithm 2 GHOST-Eph fork-choice rule.

```
$\mathrm{CHILDREN}(\mathcal{T}, B) \triangleq \{B' \in \mathcal{T} \mid B'.h = H(B)\}$
$\mathrm{VOTES}(\mathcal{T}, B, t) \triangleq |\{\mathsf{id}' \mid (\mathtt{vote}, (\mathsf{id}', t), ., h, .) \in \mathcal{T} \land B \preceq \mathcal{T}[h]\}|$
function GHOST-EPH($\mathcal{T}, t$)
    $B \leftarrow B_0$                        ▷ Start fork-choice at genesis block
    forever do
        ▷ Choose the heaviest subtree (breaking ties deterministically) rooted at
        one of the children blocks $B'$ of $B$, by number of validators that have cast a vote
        in slot $t$ for $B'$ or one of its descendants; $B' = \perp$ if $\mathrm{CHILDREN}(\mathcal{T}, B) = \emptyset$
        $B' \leftarrow \arg\max_{B' \in \mathrm{CHILDREN}(\mathcal{T}, B)} \mathrm{VOTES}(\mathcal{T}, B', t)$
        if $B' = \perp$ then return $B$
        $B \leftarrow B'$
```

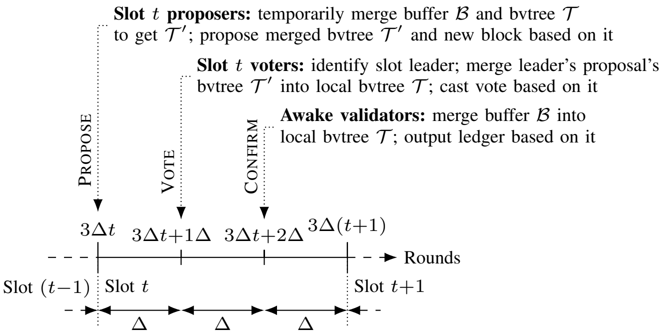
Slot t proposers: temporarily merge buffer $\mathcal{B}$ and bvtree $\mathcal{T}$ to get $\mathcal{T}'$; propose merged bvtree $\mathcal{T}'$ and new block based on it
Slot t voters: identify slot leader; merge leader's proposal's bvtree $\mathcal{T}'$ into local bvtree $\mathcal{T}$; cast vote based on it
Awake validators: merge buffer $\mathcal{B}$ into local bvtree $\mathcal{T}$; output ledger based on it
PROPOSE
VOTE
CONFIRM
$3\Delta t$
$3\Delta t{+}1\Delta$
$3\Delta t{+}2\Delta$
$3\Delta(t{+}1)$
Rounds
Slot $(t{-}1)$
Slot t
Slot $t{+}1$
Δ
Δ
Δ

Fig. 3. Throughout the execution, validators *buffer* received proposals and pieces, and *merge* the blocks and votes contained therein into their bvtrees only as *explicitly* instructed. Goldfish has time slots of three phases of Δ rounds each. Each time slot has proposers (one of which will later be recognized as the slot's leader) and a committee of voters. PROPOSE: At the start of a slot, *proposers* temporarily merge their buffers into their local bvtrees, and propose their temporary bvtrees and a new block based on it. VOTE: One-thirds into a slot, *voters* identify the slot's leader's proposal, merge the proposed bvtree into their local bvtrees, and cast a vote based on their local bvtrees. CONFIRM: Two-thirds into a slot, *all awake validators* merge their buffers into their local bvtrees, and confirm a ledger based on their local bvtrees.

4) Message handling: Recall that messages are delivered to validators irrespective of their sleep status. However, validators pick up delivered messages only once awake. Invalid messages are discarded. If a piece's validity cannot be determined due to missing references, it is held in limbo until its (in-)validity is determined. Pieces and proposals 'from the future' (*i.e.*, in time slot t, pieces M with $M.x.t > t$ and proposals P with $P.B.t > t$) are also held in limbo. Upon picking up a *valid non-in-limbo* message from the network, the validator re-broadcasts it, and adds it to $\mathcal{B}$. If the message is a proposal P, the validator also re-broadcasts the blocks and votes in $P.\mathcal{T} \cup \{P.B\}$ as pieces, and adds those pieces to its own $\mathcal{B}$.

5) Message buffering: The validator unpacks messages from $\mathcal{B}$ and *merges* them into $\mathcal{T}$ in a way that preserves $\mathrm{IsConsistent}(\mathcal{T})$. For this purpose, $\mathrm{MERGE}(\mathcal{T}, \mathcal{B})$ outputs the largest bvtree $\mathcal{T}'$ that is a subset of the union of $\mathcal{T}$ and the pieces in $\mathcal{B}$. Merging of $\mathcal{B}$ into $\mathcal{T}$ takes place only at carefully chosen points in time as instructed (Alg. 1, ll. 19, 27). This *message buffering* is a key ingredient of Goldfish. First, Δ rounds into a slot, each awake validator identifies a slot leader and merges the bvtree proposed by the leader into its bvtree (Alg. 1, l. 19). Second, 2Δ rounds into a slot, each awake validator merges its buffer into its bvtree (Alg. 1, l. 27).

[8] Vote expiry (Sec. III-A6) and reorg resilience (Thm. 3) enable timely garbage collection of pieces with missing referenced blocks.

6) Vote expiry: To determine the canonical chain, validators use the GHOST-Eph fork-choice function with ephemeral votes (Alg. 2). The function takes a bvtree $\mathcal{T}$ and slot t as input, and finds the canonical GHOST-Eph chain determined by the votes within $\mathcal{T}$ *that were cast for slot* t. More specifically, starting at the genesis block, the function iterates over a sequence of blocks from the bvtree, selecting as the next block the child of the current block with the maximum number of validators that have cast a slot t vote for a block within the child's subtree. This continues until it reaches a leaf of the bvtree, and outputs a complete chain from leaf to root. The fork-choice rule ignores votes from other than slot t in its decision (votes are *ephemeral*), lending GHOST-Eph its name.

7) The complete Goldfish *protocol:* The three phases (PROPOSE, VOTE, CONFIRM) of each slot t are shown in Fig. 3. We describe them from the perspective of an awake honest validator id.

PROPOSE: At round $3\Delta t$, id checks if its lottery ticket (id, t) is winning for $(\texttt{block}, \mathsf{thr}_{\mathrm{b}})$ (Alg. 1, l. 6). If so, id *temporarily merges* its bvtree with its buffer (Alg. 1, l. 7), identifies the GHOST-Eph chain tip using only slot $t-1$ votes (Alg. 1, l. 8), and proposes its temporary bvtree and a new block based on it (Alg. 1, l. 12). Note that in a practical implementation, the proposals need not contain the whole bvtree, but merely the votes therein (see Sec. VI).

VOTE: At round $3\Delta t + \Delta$, id identifies as *leader* for slot t any one of the proposals with smallest precedence (Alg. 1, l. 17). It *merges* the leading proposal's bvtree into its bvtree $\mathcal{T}$ (Alg. 1, l. 19). Validator id then checks if its lottery ticket (id, t) is winning for $(\texttt{vote}, \mathsf{thr}_{\mathrm{v}})$ (Alg. 1, l. 21). If so, id identifies the GHOST-Eph chain tip using only slot $t-1$ votes (Alg. 1, l. 22), and votes for it (Alg. 1, l. 25).

CONFIRM: At round $3\Delta t + 2\Delta$, id *merges* its buffer $\mathcal{B}$ into its bvtree $\mathcal{T}$ (Alg. 1, l. 27). It then identifies the GHOST-Eph chain tip using only slot t votes (Alg. 1, l. 28), and outputs as confirmed ledger $\mathsf{ch}^{\mathsf{id}}$ the transactions of those blocks in the GHOST-Eph chain that are from slots $\leq t - \kappa$ ('κ-deep in time', Alg. 1, l. 29). Since the Goldfish ledger in view of an awake honest validator id is only updated at this point, we may view the ledger as indexed by time slot t: $\mathsf{ch}_t^{\mathsf{id}}$.

8) Joining procedure: At each round, each honest validator is either asleep, dreamy or awake (Sec. II-B4). Once an honest validator is no longer asleep, it remains *dreamy* until the round of the next CONFIRM phase.[9] While being dreamy, the validator does not follow Alg. 1, except for relaying messages. With the CONFIRM phase, the validator returns to being *awake* and fully resumes Alg. 1.

9) Key mechanism of Goldfish*: Message buffering* ensures that if in slot t the leading proposal is honest, then all honest voters in t will vote for the proposed block. This is because in PROPOSE, the leader's temporary bvtree is a superset of all honest validators' bvtrees, and thus in VOTE all honest validators adopt that leader's bvtree. *Vote expiry* (together with majority honest validators) ensures that if in slot t all honest voters have voted into the subtree rooted at some block B, then all honest voters in slot $t+1$ will also vote into the subtree rooted at B. An inductive argument immediately yields reorg resilience of Goldfish. Furthermore, w.o.p., every interval of κ slots has at least one honest leading proposer. The prefix of that proposal stabilizes (by reorg resilience), and the proposal includes unconfirmed transactions, leading to safety and liveness of the κ-deep confirmation rule.

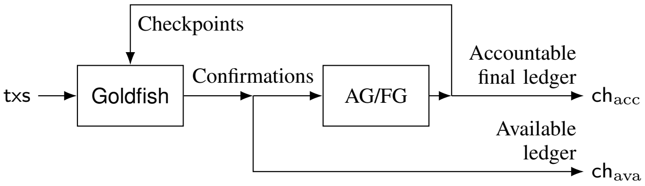

Fig. 4. An accountability/finality gadget (AG/FG; a.k.a. *overlay*) checkpoints decisions of the dynamically available protocol Goldfish (a.k.a. *underlay*). A feedback loop ensures that Goldfish respects earlier checkpoints. This construction satisfies the *ebb-and-flow* design objective of Ethereum, to produce an available full ledger that is secure under dynamic participation of validators, and a prefix ledger that is accountably secure under network partition [4], [6].

Algorithm 3 GHOST-Eph (cf. Alg. 2) modified (green) to respect the latest checkpoint B. See Alg. 2 for CHILDREN and VOTES.

```
function GHOST-EPH($\mathcal{T}, t, B$)
    ▷ Start fork-choice from latest checkpoint $B$
    forever do
        ▷ Choose the heaviest subtree rooted (breaking ties deterministically) at one of the children blocks $B'$ of $B$, by number of validators that have cast a vote in slot $t$ into the subtree rooted at $B'$; $B' = \bot$ if $\textsc{Children}(\mathcal{T}, B) = \emptyset$
        $B' \leftarrow \arg\max_{B' \in \textsc{Children}(\mathcal{T}, B)} \textsc{Votes}(\mathcal{T}, B', t)$
        if $B' = \bot$ then return $B$
        $B \leftarrow B'$
```

10) Validator replaceability: Due to subsampling, once a validator takes an action in Goldfish, it does not play any further role, at least for a long time. As a result, Goldfish supports player replaceability [31], [32], [57] and can withstand a mildly adaptive adversary (Sec. II-B6). Analogously to earlier works [21], [22], [32], [58], fully adaptive corruption can be allowed through the use of key evolving signature and VRF schemes. In both cases, the adversary cannot corrupt an honest validator and make it send conflicting protocol messages 'fast enough' to harm the protocol execution.

B. *Goldfish with Accountability Gadgets*

For the composition of Goldfish with accountability gadgets and finality gadgets, we follow the construction of [6], [14] (Fig. 4, Alg. 4). In this construction, a partially synchronous accountably-safe consensus protocol such as Streamlet, Tendermint, or HotStuff [16]–[18], [59], with accountable safety resilience of $n/3$ out of n validators, is used to determine checkpoints of Goldfish's output ledger. To ensure that Goldfish respects earlier checkpoints, its fork-choice rule is modified to respect earlier checkpoint decisions (cf. Alg. 3). The most recent checkpoint forms the accountably-safe finalized prefix ledger $\mathsf{ch}_{\mathrm{acc}}$, while Goldfish's output forms the dynam-

[9] We assume that messages arrive at validators while asleep (Sec. III-A4). To allow for extra time to download messages missed during sleep, dreaminess can be extended accordingly, but should always end at a CONFIRM phase.

Algorithm 4 Composition of Goldfish and accountability gadget (cf. Fig. 4, [6, Alg. 1]), executed by validator id. Here, Goldfish (cf. Alg. 1) uses a modified GHOST-Eph rule (Alg. 3), starting the recursion from the latest checkpoint, *i.e.*, the last block of $\mathsf{ch}^{\mathsf{id}}_{\mathrm{acc}}$. Throughout, Goldfish maintains the available chain $\mathsf{ch}^{\mathsf{id}}_{\mathrm{ava}}$. RunAccountabilityGadget attempts the next iteration of the gadget, where valid checkpoint candidates are determined using $\mathsf{ch}^{\mathsf{id}}_{\mathrm{ava}}$. Iterations may fail ($\perp$), *e.g.*, if the gadget invokes a malicious leader.

```
1: $\mathsf{ch}^{\mathsf{id}}_{\mathrm{acc}} \leftarrow B_0$                        ▷ 'Zero-th' checkpoint: Goldfish's genesis block
2: for $c = 1, 2, \dots$ do                                   ▷ Checkpoint iterations
3:     checkpoint $\leftarrow$ RunAccountabilityGadget($\mathsf{ch}^{\mathsf{id}}_{\mathrm{ava}}$)
4:     if checkpoint $\neq \perp$ then
5:         $\mathsf{ch}^{\mathsf{id}}_{\mathrm{acc}} \leftarrow$ checkpoint                    ▷ Update latest checkpoint
6:         Sleep for $T_{\mathrm{chkpt}}$ rounds
```

ically available full ledger $\mathsf{ch}_{\mathrm{ava}}$ (cf. *ebb-and-flow*, Def. 4). As Goldfish now respects checkpoints, $\mathsf{ch}_{\mathrm{acc}} \preceq \mathsf{ch}_{\mathrm{ava}}$ holds.

The full protocol proceeds in checkpointing iterations (cf. Alg. 4). Iterations may fail, *e.g.*, when the consensus protocol of the gadget invokes a malicious leader, or during asynchrony before GST, or while many validators are asleep before GAT. Successful checkpoint iterations are separated by at least T_{chkpt} rounds of inactivity of the gadget. In App. F, we apply the techniques of earlier analyses [6], [14] to the combination of Goldfish and the accountability gadget, to show how to tune T_{chkpt} as a function of the network delay Δ and the confirmation parameter κ, and to formally prove that the combination satisfies the ebb-and-flow desiderata.

IV. Optimistic Fast Confirmations

The Goldfish protocol described in Sec. III-A has reorg resilience as an advantage over protocols which use blocks as votes (*e.g.*, longest chain [19], [20], [41], GHOST [1]). On the other hand, Goldfish's κ-slots deep confirmation rule, which leads to $\Theta(\kappa)$ latency in both the worst and the expected case, falls behind many propose-and-vote style protocols that achieve *constant* expected latency (*e.g.*, PBFT [15], Tendermint [16], HotStuff [17], Streamlet [18]). By introducing a fast confirmation rule and adding a Fast-Confirm phase to the Goldfish slot structure, we can achieve constant expected confirmation latency *under optimistic conditions*, *i.e.*, under high participation and honest supermajority (Fig. 5, Alg. 5). In particular, validators can now confirm blocks proposed by honest leaders immediately, in the Fast-Confirm phase of the slot, under optimistic conditions. The κ-deep confirmation rule (Alg. 4, l. 29), to which we from now on refer as *standard* confirmation rule, still applies and guarantees liveness when optimistic conditions do not hold.

a) Fast confirmation phase: Slots now consist of 4Δ rounds and four phases (Propose, Vote, Fast-Confirm, Confirm), with the addition of phase Fast-Confirm at round $4\Delta t + 2\Delta$ (Fig. 5, Alg. 5).

In Fast-Confirm, a validator id first merges its buffer into its bvtree $\mathcal{T}$ (Alg. 5, l. 9). It then marks a block B as fast confirmed if $|\textsc{Votes}(\mathcal{T}, B, t)| \geq n(\frac{3}{4} + \frac{\epsilon}{2})\mathsf{thr}_{\mathrm{v}}$ for some $\epsilon > 0$,[10] and updates $\mathsf{ch}^{\mathsf{id}}_{\mathrm{fast}}$ to the highest fast confirmed block (Alg. 5, l. 10).

Algorithm 5 Goldfish executed by validator id, using both (optimistic) fast confirmation and standard confirmation (cf. Alg. 1). See Alg. 2 for Votes.

```
1: $(\mathcal{B}, \mathcal{T}, t) \leftarrow (\emptyset, \emptyset, 0)$              ▷ Initialize buffer $\mathcal{B}$ and bvtree $\mathcal{T}$
2: ▷ At all rounds, only *valid* messages *not from later than* $t$ are picked up from the
   network, re-broadcast, and put into $\mathcal{B}$ as specified in Sec. III-A4
3: for $t = 1, 2, \dots$ do                                                 ▷ Slots
4:     at $4\Delta t$ do                                               ▷ Propose phase
5:         *Same as* Propose *phase in Alg. 1*
6:     at $4\Delta t + \Delta$ do                                            ▷ Vote phase
7:         *Same as* Vote *phase in Alg. 1*
8:     at $4\Delta t + 2\Delta$ do                                    ▷ Fast-Confirm phase
9:         $\mathcal{T} \leftarrow$ Merge($\mathcal{T}, \mathcal{B}$)                          ▷ Merge buffer and bvtree
10:        $\mathsf{ch}^{\mathsf{id}}_{\mathrm{fast}} \leftarrow \arg\max_{B \in \mathcal{T}:\ |\textsc{Votes}(\mathcal{T},B,t)| \geq n(\frac{3}{4}+\frac{\epsilon}{2})\mathsf{thr}_{\mathrm{v}}} |B|$
11:    at $4\Delta t + 3\Delta$ do                                         ▷ Confirm phase
12:        $\mathcal{T} \leftarrow$ Merge($\mathcal{T}, \mathcal{B}$)                          ▷ Merge buffer and bvtree
13:        $B \leftarrow$ GHOST-Eph($\mathcal{T}, t$)                      ▷ Canonical GHOST-Eph chain
14:        $\mathsf{ch}^{\mathsf{id}} \leftarrow \arg\max_{\mathsf{ch} \in \{\mathsf{ch}^{\mathsf{id}}_{\mathrm{fast}}, B^{\lceil\kappa}\}} |\mathsf{ch}|$              ▷ Output Goldfish ledger
```

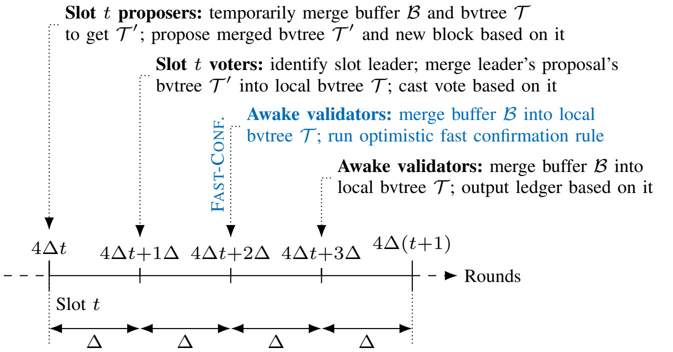
Slot t proposers: temporarily merge buffer $\mathcal{B}$ and bvtree $\mathcal{T}$ to get $\mathcal{T}'$; propose merged bvtree $\mathcal{T}'$ and new block based on it
Slot t voters: identify slot leader; merge leader's proposal's bvtree $\mathcal{T}'$ into local bvtree $\mathcal{T}$; cast vote based on it
Awake validators: merge buffer $\mathcal{B}$ into local bvtree $\mathcal{T}$; run optimistic fast confirmation rule
Fast-Conf.
Awake validators: merge buffer $\mathcal{B}$ into local bvtree $\mathcal{T}$; output ledger based on it
$4\Delta t$
$4\Delta t{+}1\Delta$
$4\Delta t{+}2\Delta$
$4\Delta t{+}3\Delta$
$4\Delta(t{+}1)$
Rounds
Slot t
Δ
Δ
Δ
Δ

Fig. 5. To enable optimistic fast confirmations, a Fast-Confirm phase (blue) of Δ rounds is inserted between Vote and Confirm phase (cf. Fig. 3). Fast-Confirm: Two-fourth into a slot, *all awake validators* merge their buffers into their local bvtrees, and run the optimistic fast confirmation rule based on their local bvtrees.

The other three phases are unchanged, other than for how the Goldfish ledger is output in Confirm (Alg. 5, l. 14). Validator id outputs the highest of $\mathsf{ch}^{\mathsf{id}}_{\mathrm{fast}}$ and the κ-deep prefix $B^{\lceil\kappa}$, where B is the tip of its canonical chain GHOST-Eph (cf. Alg. 4, l. 29, where $B^{\lceil\kappa}$ is output instead). For simplicity, we have omitted in Alg. 5 the mechanism to avoid temporary ledger 'roll back' (to ensure $\forall \mathsf{id}, t' \geq t\colon \mathsf{ch}^{\mathsf{id}}_{t} \preceq \mathsf{ch}^{\mathsf{id}}_{t'}$).

The reason for the extra Δ rounds, as opposed to just running the fast confirmation rule in the Confirm phase, is to guarantee that, whenever an honest validator fast confirms a block, all honest awake validators see the votes responsible for fast confirmation by the time their bvtrees are updated for the last time in the given slot, at round $4\Delta t{+}3\Delta$. This ensures that the fast confirmed block eventually enters the Goldfish ledger in the view of all honest awake validators (Thm. 5), which in turn implies that fast confirmations are safe (Thm. 6). The security of the protocol with the fast confirmation rule is proven in Sec. V-C.

b) Joining procedure: The joining protocol is conceptually unchanged. Once a validator stops being asleep, it remains

[10] The parameter $\epsilon > 0$ can be made arbitrarily small in the limit $n \to \infty$.

dreamy until the next CONFIRM phase, at which point it turns fully awake and resumes protocol execution by merging its buffer with its bvtrees 3Δ rounds into a slot (phase CONFIRM, cf. Alg. 5, l. 12).

c) Composition with accountability and finality gadgets: When composing accountability gadgets and Goldfish with the fast confirmation rule, we stipulate that the validators input to the gadget only those blocks confirmed via the standard confirmation rule ($\text{GHOST-EPH}(\mathcal{T}, t)^{\lceil\kappa}$) in their view. This is necessary to ensure that all honest validators promptly agree on the confirmation status of the blocks input to the gadget for checkpointing, which in turn is a prerequisite for the liveness of the accountable final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$. Otherwise, it is possible that a block fast confirmed by one honest validator might not become confirmed in the view of another honest validator until after κ slots, stalling the checkpointing process of the accountability gadget for that block. Thus, the fast confirmation rule is primarily for reducing the latency of the available ledger $\mathsf{ch}_{\mathrm{ava}}$, and does not affect the time for a block to enter the accountable final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$.

d) Trading-off safety and liveness resiliences: With fast confirmation, Goldfish has two 'parallel' confirmation rules, 'fast' and 'standard'. The overall protocol is safe only when both rules are safe, and live when one of the rules is live. To match the $\frac{1}{2}$-safety of standard confirmation, the 'quorum' for fast confirmation was chosen as $n(\frac{3}{4}+\frac{\epsilon}{2})\mathsf{thr}_{\mathrm{v}}$ votes. With this parameterization, however, Goldfish cannot guarantee any fast confirmation in the presence of $\frac{n}{4}+\frac{\epsilon}{2}$ adversarial validators. It is possible to vary the quorum to trade-off safety and liveness of the fast path (and thereby of the overall protocol). With a quorum of $n(\frac{2}{3}+\frac{\epsilon}{2})\mathsf{thr}_{\mathrm{v}}$ for fast confirmation, Goldfish satisfies safety and liveness with $T_{\mathrm{conf}} = \Theta(\kappa)$ if $\beta < \frac{1}{3} - \frac{3}{2}\epsilon$, and liveness with constant expected confirmation time if all validators are awake.

V. ANALYSIS

A. *Goldfish*

In the subsequent analysis, a valid proposal P (cf. Sec. III-A2) is *for slot* t iff $t = P.B.t$, and it *has precedence* p iff $p = \mathrm{Prio}(P.B.\rho)$. A validator id is *eligible to propose at slot* t if its ticket (id, t) is winning for the lottery $(\mathtt{block}, \mathsf{thr}_{\mathrm{b}})$. Similarly, a validator id is *eligible to vote at slot* t if its ticket (id, t) is winning for the lottery $(\mathtt{vote}, \mathsf{thr}_{\mathrm{v}})$. Recall that awake honest validators consider the proposal with lowest precendence received by $3\Delta t + \Delta$ the *leader* of slot t (Alg. 1, l. 16). We hereafter use blocks and the sequences of blocks they induce via the parent-block chain relation interchangeably. A block B_1 is a *descendant* (resp., *ancestor*) of block B_2 iff the underlying chains satisfy $B_2 \preceq B_1$ (resp., $B_1 \preceq B_2$). Two blocks B_1, B_2 are *conflicting* if B_1 is neither an ancestor nor a descendant of B_2.

Let A_r and H_r denote the number of adversarial and honest validators awake at round r, respectively. Our security theorems hold for *compliant executions* that satisfy the relations on A_r and H_r laid out in Sec. II-B6:

Definition 5. In the absence of key-evolving cryptographic primitives (signatures and VRFs), an execution is (γ, τ)-*compliant* iff:

- $\forall r\colon \frac{A_r}{A_r + H_{r-\tau}} \leq \beta < \gamma - \epsilon$.
- The corruption is mildly adaptive: If the adversary decides to corrupt an honest validator at round r, then the validator becomes adversarial no earlier than at round $r + \tau$.

With key-evolving primitives, an execution is *compliant* iff:

- $\forall r\colon \frac{A_r}{A_r + H_r} \leq \beta < \gamma - \epsilon$.

Moreover, in both cases, $H_r > \gamma n_0 = \Theta(\kappa)$ for all rounds r, and the time horizon T_{hor} of the protocol execution satisfies $T_{\mathrm{hor}} = \mathrm{poly}(\kappa)$.

Intuitively, in compliant executions, honest voters outnumber adversarial voters (as long as votes have not yet expired); and every long interval of slots contains at least one slot in which all honest validators recognize the same honest validator as the slot leader.

Lemma 1. *Suppose the Goldfish execution is* $(\frac{1}{2}, 3\Delta)$-*compliant. Then, w.o.p., for every slot t, adversarial validators at round* $3\Delta(t+1)+\Delta$ *eligible to vote at slot t are less than honest validators awake at round* $3\Delta t + \Delta$ *and eligible to vote at slot t.*[11] *Also w.o.p., all slot intervals of length* κ *have at least one slot t where an honest validator is recognized as the slot t leader by all awake honest validators at round* $3\Delta t + \Delta$.[12]

Lem. 1's proof uses correctness, uniqueness and pseudorandomness of VRF-based lotteries along with Chernoff bounds. It is omitted for brevity and can be found in App. E.

The main security results are as follows:

Theorem 1. *Suppose a* $(\frac{1}{2}, 3\Delta)$-*compliant execution of Goldfish in the synchronous sleepy network model of Sec. II-B, and validator* id *with proposal* P^* *is recognized as the leader of a slot t by all awake honest validators at round* $3\Delta t + \Delta$ *(Alg. 1, l. 16). Then, w.o.p.,* $P^*.B \preceq B$ *for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any round* $r \geq 3\Delta t + 2\Delta$.

Theorem 2 (Security)**.** *Suppose a* $(\frac{1}{2}, 3\Delta)$-*compliant execution of Goldfish in the synchronous sleepy network model. Then, w.o.p., Goldfish is secure with transaction confirmation time* $T_{\mathrm{conf}} = 2\kappa + 2$ *slots.*

Theorem 3 (Reorg resilience)**.** *Suppose a* $(\frac{1}{2}, 3\Delta)$-*compliant execution of Goldfish in the synchronous sleepy network model, and validator* id *with proposal* P^* *is recognized as*

[11] For concreteness, the Ethereum validator set has over 400,000 validators as of 5-Sept-2022. Suppose we subsample with $\mathsf{thr}_{\mathrm{b}} = \frac{1}{32}$, *i.e.*, with committee size unchanged in expectation, and that $\epsilon = 0.05$, *i.e.*, that 55% of validators are assumed to be honest. Then, the probability of an adversarial majority at a single slot (assuming perfect randomness) is roughly $4 \cdot 10^{-15}$. There are 2628000 slots in a year, so the expected number of years before seeing an adversarial majority at a slot is $\frac{4 \cdot 10^{15}}{2628000} \approx 10^7$ years.

[12] The proposer-lottery threshold $\mathsf{thr}_{\mathrm{b}}$ can be tuned following Algorand [31, Appendix-B.1] so that each slot has at least one eligible proposer.

the leader of a slot t by all awake honest validators at round $3\Delta t + \Delta$ (Alg. 1, l. 16). Then, w.o.p.,

$$\exists r' \colon \forall r \geq r' \colon \forall \mathsf{id} \colon \quad P^*.B \preceq \mathsf{ch}_r^{\mathsf{id}}, \tag{8}$$

where $\mathsf{ch}_r^{\mathsf{id}}$ denotes Goldfish's ledger at validator id and round r. In particular, $r' = 3\Delta(t+\kappa) + 2\Delta$ satisfies the above.

We first prove Thms. 2 and 3 from Thm. 1 and Lem. 1. Then, we prove Thm. 1 from Lems. 1, 2 and 3.

Proof of Thm. 2. By Lem. 1, w.o.p., all slot intervals of length κ have at least one slot t, where an honest validator with proposal P^* is recognized as the slot leader by all awake honest validators at round $3\Delta t + \Delta$, and, by Thm. 1, $P^*.B \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any $r \geq 3\Delta t + 2\Delta$.

Liveness: A transaction tx is input to an honest validator at some round r. At most 6Δ rounds (*i.e.*, 2 slots) later the transaction is propagated to all honest validators and we have reached the beginning of a slot t_0. For the next κ slots all honest proposers will include tx if they extend a tip whose chain does not include tx yet. By the earlier argument, one of these proposals will be an ancestor of any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any $r' \geq 3\Delta(t_0+\kappa)+2\Delta$. From κ slots later onwards, all awake honest validators include the transaction in their ledger (Alg. 1, l. 29). Thus, Goldfish is live with $T_{\mathrm{conf}} = 2\kappa + 2$ slots.

Safety: Pick any two honest validators id_1 and id_2, and two slots t_1 and $t_2 \geq t_1$. By the earlier argument, there exists a block B' proposed (by an honest validator) at some slot $t' \in [t_1 - \kappa, t_1]$ such that $B' \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any $r' \geq 3\Delta t' + 2\Delta$. As $t' \geq t_1 - \kappa$ but by Goldfish's confirmation rule blocks in $\mathsf{ch}_{t_1}^{\mathsf{id}_1}$ are from no later than $t_1 - \kappa$, $\mathsf{ch}_{t_1}^{\mathsf{id}_1} \preceq B$. Similarly, if $t' \geq t_2 - \kappa$, then $\mathsf{ch}_{t_2}^{\mathsf{id}_2} \preceq B$; otherwise, $B \preceq \mathsf{ch}_{t_2}^{\mathsf{id}_2}$. In both cases, either $\mathsf{ch}_{t_1}^{\mathsf{id}_1} \preceq \mathsf{ch}_{t_2}^{\mathsf{id}_2}$ or $\mathsf{ch}_{t_2}^{\mathsf{id}_2} \preceq \mathsf{ch}_{t_1}^{\mathsf{id}_1}$. □

Proof of Thm. 3. By Thm. 1, $P^*.B \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any $r \geq 3\Delta t + 2\Delta$. From κ slots later onwards, all awake honest validators include the transaction in their ledger (Alg. 1, l. 29). □

Proof of Thm. 1 follows from Lems. 1, 2 and 3, and is provided at the end of this section. The structure of the argument is inductive: Lem. 2 shows that in a slot t with honest leader, all honest voters vote for the leader's proposal. Lem. 3 shows that if in slot t all honest voters have voted for a descendant of a certain block, then in slot $t+1$ all honest voters will vote for a descendant of that block.

Lemma 2. *Suppose an execution of Goldfish in the synchronous sleepy network model. Suppose validator id^* with proposal P^* is recognized as the leader of a slot t by all awake honest validators at round $3\Delta t + \Delta$ (Alg. 1, l. 16). Then, all honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t, vote for $P^*.B$ at slot t.*

Proof. Let $\mathcal{T}' = P^*.\mathcal{T}$, and $\mathcal{B}^*$ and $\mathcal{T}^*$ denote the buffer and bvtree of id^* at round $3\Delta t$. Since id^* is honest, it must have broadcast P^* at round $3\Delta t$ with bvtree $\mathcal{T}' = \textsc{Merge}(\mathcal{T}^*, \mathcal{B}^*)$ and a new block $P^*.B$ with parent GHOST-$\textsc{Eph}(\mathcal{T}', t-1)$ (Alg. 1, ll. 7, 8, 12).

By synchrony, any message that a non-asleep honest validator id could have added to its bvtree $\mathcal{T}_{\mathsf{id}}$ by $3\Delta(t-1) + 2\Delta$, is received by id^* by $3\Delta t$, and thus in $\mathcal{T}'$. As awake honest validators do not update their bvtrees and no honest validators turn awake in the interval $(3\Delta(t-1)+2\Delta, 3\Delta t+\Delta)$, for any honest validator id awake at round $3\Delta t + \Delta$, $\mathcal{T}_{\mathsf{id}} \subseteq \mathcal{T}'$ prior to Alg. 1, l. 19.

Since id^* is recognized as the leader of slot t by all awake honest validators at round $3\Delta t + \Delta$, at that round, each awake honest validator id merges its bvtree with $\mathcal{T}' \cup \{P^*.B\}$ (Alg. 1, l. 19) and reaches $\mathcal{T}_{\mathsf{id}} = \mathcal{T}' \cup \{P^*.B\}$. Consequently, each honest validator id awake at round $3\Delta t + \Delta$ and eligible to vote at slot t votes for $P^*.B$ due to the recursive structure of the GHOST-Eph rule (Alg. 2). □

Lemma 3. *Suppose a $(\frac{1}{2}, 3\Delta)$-compliant execution of Goldfish in the synchronous sleepy network model. Consider a slot t where all honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t, vote for a descendant of B. Then, w.o.p., all honest validators awake at round $3\Delta(t+1) + \Delta$ and eligible to vote at slot $t+1$, vote for a descendant of B.*

Proof. By Lem. 1, w.o.p., for every slot t, the number of adversarial validators at round $3\Delta(t+1) + \Delta$ and eligible to vote at slot t is less than the number of honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t.

Let t be a slot such that all honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at t voted for a descendant of B. Pick any honest validator id awake at round $3\Delta(t+1)+\Delta$ and eligible to vote at slot $t+1$. Since id must have been awake at least since round $3\Delta t + 2\Delta$, its bvtree at round $3\Delta t + 2\Delta$ contains all votes broadcast by honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t (Alg. 1, l. 19). The same is true for its bvtree at round $3\Delta(t+1) + \Delta$, even after id merges its bvtree with that of any proposal (Alg. 1, l. 7). Moreover, the number of honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t is greater than the number of adversarial validators at round $3\Delta(t+1) + \Delta$ that are eligible to vote at slot t.

Consequently, upon invoking the GHOST-Eph fork-choice rule at round $3\Delta(t+1) + \Delta$ (Alg. 1, l. 22), id observes that at every iteration of the fork choice (Alg. 2, l. 7), blocks consistent with B have more votes than blocks conflicting with B. Thus, at round $3\Delta(t+1) + \Delta$, fork choice returns a descendant of B, and id votes for it. □

Proof of Thm. 1. From Lems. 1, 2 and 3, it follows by induction that w.o.p., for all $t' \geq t$, all honest validators awake at round $3\Delta t' + \Delta$ and eligible to vote at slot t', vote for a descendant of $P^*.B$.

By synchrony, the honest votes of slot t' reach all honest validators awake at $3\Delta t' + 2\Delta$ by then, when they also merge the votes into their bvtrees. The number of honest validators awake at round $3\Delta t' + \Delta$ and eligible to vote at slot t' is greater than the number of adversarial validators by round $3\Delta(t'+1)+\Delta$ that are eligible to vote at slot t' (by Lem. 1). Upon invoking the GHOST-Eph rule of Alg. 1, ll. 8, 22, 28 at $3\Delta t' + 2\Delta$, $3\Delta(t'+1)$ and $3\Delta(t'+1)+\Delta$, respectively, an awake honest validator id (who must have been awake since at least $3\Delta t' + 2\Delta$, due to the joining procedure) observes that at every iteration of the fork choice (Alg. 2, l. 7), blocks consistent with $P^*.B$ have more votes than blocks conflicting with $P^*.B$. Thus, id's fork choice reaches a descendant of $P^*.B$. □

B. *Goldfish with Accountability Gadget*

We provide a formal statement for Def. 4:

Theorem 4 (Ebb-and-flow property)**.** *Goldfish combined with accountability gadgets (cf. Sec. III-B) satisfies the ebb-and-flow property:*

1) *(**P1: Accountability and finality**) Under a partially synchronous network in the sleepy model, the accountable final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$ has accountable safety resilience $n/3$ at all times, (except w.p. $\mathrm{negl}(\lambda)$), and there exists a constant $\mathbf{C}$ such that if the execution is $(\frac{1}{3}, 3\Delta)$-compliant, $\mathsf{ch}_{\mathrm{acc}}$ provides liveness with transaction confirmation time $T_{\mathrm{conf}} = \Theta(\kappa^2)$ after round $\max(\mathsf{GST}, \mathsf{GAT}) + \mathbf{C}\kappa$ (w.o.p.).*
2) *(**P2: Dynamic availability**) Under a synchronous network in the sleepy model (i.e., for $\mathsf{GST} = 0$), if the execution is $(\frac{1}{2}, 3\Delta)$-compliant, the available ledger $\mathsf{ch}_{\mathrm{ava}}$ is secure at all times (w.o.p.).*
3) *(**Prefix**) For each honest id and round r, $\mathsf{ch}^{\mathsf{id}}_{\mathrm{acc},r} \preceq \mathsf{ch}^{\mathsf{id}}_{\mathrm{ava},r}$.*

A proof sketch for Thm. 4 is provided in App. C, and the full proof can be found in App. F. Proof of Thm. 4 follows the same blueprint as the original construction of accountability gadgets in [6, Appendices B, C].

C. *Goldfish with Fast Confirmation*

In the following analysis, we consider a synchronous network in the sleepy model as described in Sec. II. Recall that the total number of validators is n (cf. Sec. II). Since Goldfish slots consist of 4Δ rounds in the case of fast confirmation, we hereafter assume that the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant. We show that Thm. 2 holds for Goldfish with fast confirmations (w.o.p.) in compliant executions. To do so, we first prove Thm. 5, an analogue of Thm. 1 for fast confirmations, showing that fast confirmed blocks are always in the canonical chain of awake validators at later rounds.

Proposition 1. Suppose $T_{\mathrm{hor}} = \mathrm{poly}(\kappa)$. Then, w.o.p., there can be at most $(1+\epsilon)n\,\mathsf{thr}_{\mathrm{v}}$ validators that are eligible to vote at any given slot. If the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant, then, w.o.p., for all slots t, the number of adversarial validators at round $4\Delta(t+1)+\Delta$, eligible to vote at slot t, is less than $\frac{1}{2}n\,\mathsf{thr}_{\mathrm{v}}$.

Proof follows from a Chernoff bound.

Lemma 4. *Suppose the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant in the synchronous sleepy network model, and an honest validator id^* fast confirms a block B at slot t. Then, w.o.p, all honest validators awake at round $4\Delta(t+1)+\Delta$ and eligible to vote at slot $t+1$, vote for a descendant of B at slot $t+1$.*

Proof is provided in App. D and follows from Props. 1 and 1 and a quorum intersection argument.

Theorem 5. *Suppose the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant in the synchronous sleepy network model, and an honest validator id^* fast confirms a block B at slot t. Then, w.o.p., $B \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any round $r \geq 4\Delta(t+1)+\Delta$.*

Proof is provided in App. D and follows from Lems. 1, 4 and 3 and the inductive argument used in the proof of Thm. 1.

Theorem 6. *Suppose the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant. Then, Goldfish with fast confirmations satisfies safety (w.o.p.).*

Proof is provided in App. D and follows from Thm. 2. In $(\frac{1}{2}, 4\Delta)$-compliant executions, we automatically get liveness of Goldfish with fast confirmations from the liveness of the standard confirmation rule, since fast confirmation is not needed for a block to be confirmed. Under optimistic conditions, liveness of fast confirmations holds as well. We prove that a block within an honest, valid proposal is immediately fast confirmed within the same slot by the awake honest validators, if there are over $(\frac{3}{4}+\frac{3}{2}\epsilon)n$ awake, honest validators at the voting time of the given slot, implying the liveness of fast confirmations under optimistic conditions.

Theorem 7. *Suppose the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant. Then, Goldfish with fast confirmations satisfies liveness with $T_{\mathrm{conf}} = \Theta(\kappa)$ (w.o.p.).*

Consider a slot t, such that there are $(\frac{3}{4}+\frac{3}{2}\epsilon)n\,\mathsf{thr}_{\mathrm{v}}$ honest validators eligible to vote at slot t and awake at round $4\Delta t + \Delta$. Suppose an honest validator id with proposal P^* is recognized as the leader of a slot t by all awake honest validators at round $4\Delta t + \Delta$ (Alg. 1, l. 16).

Then all honest validators awake at round $4\Delta t + 2\Delta$ fast confirm $P^*.B$ in Alg. 5, l. 10.

Proof is provided in App. D. Liveness follows from Thm. 2 and fast confirmation from Lem. 2.

VI. Implementation and Experiments

In this section we discuss implementation aspects of Goldfish and experimentally study the behavior of Goldfish under different dynamic participation scenarios. To this end, we have implemented Goldfish in approx. 2,500 lines of Rust code[13], with BLAKE3 hashes [60] and BLS signatures [61] over the BLS12-381 curve [62] for signatures and VRFs. A particular focus of our inquiry is on the communication-efficient implementation of proposals and the message buffering mechanism,

[13] Source code: https://github.com/tse-group/goldfish-experiments

and on the interplay between the parameterization of the block production lottery threshold $\mathtt{thr}_\mathrm{b}$, the protocol's communication load, and its behavior under low participation.

A. Proposal Size and Wire Format

In the Goldfish variant of Sec. III-A, which is streamlined for ease of exposition rather than performance, each proposal includes the proposer's entire bvtree $\mathcal{T}'$ (Alg. 1, l. 7). This raises concerns about the resulting communication load. Proposal messages would grow over time with the number of blocks, and could be further inflated by *equivocation spamming* (where an adversary uses *one* winning lottery ticket to create *many equivocating* blocks or votes, cf. [63], [64]).

The following implementation details resolve these concerns. It suffices for a proposal to only include votes from the most recent VOTE phase, as votes from earlier phases will not carry weight during fork-choice due to vote expiry. Another tweak is *equivocation discounting*, *i.e.*, not counting votes during fork-choice from validators who have sent votes for two or more different blocks during the latest VOTE phase. We discuss equivocation discounting at length, and show it to not compromise security, in App. A. As *any two* equivocating votes suffice as evidence for an honest validator to discount *all* votes of an equivocating adversary, the above two measures mean that every proposal needs to include at most two votes per validator eligible to vote in the previous slot.

Notice also that it suffices in practice for proposals to include *references* (hashes) to blocks and votes. In fact, an honest proposer's role in the message buffering technique is only to point validators to messages (*which they already have in their buffer* because at least the honest proposer would have relayed them) that they can safely merge into their bvtree. Finally, only blocks with nonzero fork-choice weight need to be referenced, because blocks with zero weight cannot possibly be decisive in fork-choice regarding the proposer's block. Observe that nonzero weight blocks are either referenced by votes, or by a nonzero weight child block. Thus, it ultimately suffices for proposals to only reference at most two votes per validator eligible to vote in the previous slot.

As a concrete example, if Goldfish is used among $n = 1{,}024$ validators without voter subsampling, so $\mathtt{thr}_\mathrm{v} = 1$, with $32\,\mathrm{Byte}$ hashes, then even in the worst case a proposal is only of size $64\,\mathrm{kByte}$ plus one block. This example is representative for a deployment in Ethereum, where votes get aggregated by $1{,}024$ aggregators per slot, Goldfish's fork-choice would operate on aggregates, and at most two aggregates per aggregator need to be referenced in a proposal. Comparing $64\,\mathrm{kByte}$ to the current block size of $80\,\mathrm{kByte}$[14], message buffering seems feasible in terms of network load.

Garbage collection: Proposal messages are discarded after their slot's VOTE phase. Vote expiry allows to discard votes within two slots. Blocks (including 'in limbo') are discarded once inconsistent with confirmed blocks (*i.e.*, after at most κ slots) and with the gadget output (if the gadget is used).

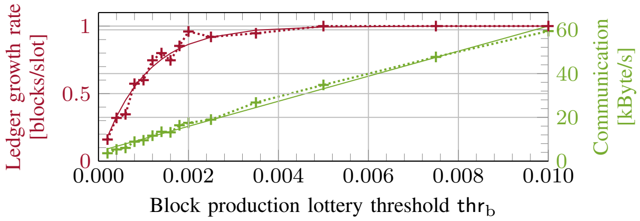

Fig. 6. Ledger growth rate and average broadcast load of Goldfish as a function of block production lottery threshold, for experiments (⋯+⋯, ⋯+⋯) with $n = 1000$, $\mathtt{thr}_\mathrm{v} = 0.1$, $\Delta = 4\,\mathrm{s}$, under full honest participation. For the block production lottery, we expect the number of proposals per slot to be binomially distributed with mean $n\,\mathtt{thr}_\mathrm{b}$. The measurements fit the predictions for the probability of zero proposals in a given slot ($1 - e^{-n\,\mathtt{thr}_\mathrm{b}}$, ——), and that the communication load is affine ($5723 \cdot \mathtt{thr}_\mathrm{b} + 4.472$ with coefficients to four digit accuracy, ——) with the constant term accounting for votes.

B. Block Production Lottery Threshold

From Sec. V it is clear how to tune the vote lottery threshold $\mathtt{thr}_\mathrm{v}$ such that, w.o.p., all voter committees over a given execution horizon have an honest majority. Given a number of validators n and a threshold $\mathtt{thr}_\mathrm{v}$, the size of a proposal and the communication load resulting from votes are constant in expectation. The block production lottery threshold $\mathtt{thr}_\mathrm{b}$ is the remaining parameter affecting the overall broadcast load through the expected number of proposals per slot $n\,\mathtt{thr}_\mathrm{b}$ (Fig. 6). For low $\mathtt{thr}_\mathrm{b} < 1/n$, communication load is low but ledger growth is impaired because many slots have no proposal. For high $\mathtt{thr}_\mathrm{b} > 1/n$, most slots have more than one proposal, leading to communication overhead but also close-to-optimal ledger growth. For a reasonable tradeoff of ledger growth and communication load in the non-degraded common case of near-full participation, we tune $\mathtt{thr}_\mathrm{b} = 3/n$.

C. Behavior under Dynamic Participation

Based on Fig. 6, we expect a confirmation performance degradation under low participation if $\mathtt{thr}_\mathrm{b} = 3/n$. (If good performance is to be ensured even under very low participation $n_0 \ll n$, tune $\mathtt{thr}_\mathrm{b}$ to n_0 rather than to n.) To study the impact of dynamic participation on Goldfish with $\mathtt{thr}_\mathrm{b} = 3/n$, we run it (Fig. 7) in four different dynamic participation environments inspired by [40]: ⓐ *Stable participation:* Starting from 50% participation, randomly increase or decrease participation by 3% per Goldfish *phase* of length Δ (unless this would exceed $[10\%, 90\%]$). ⓑ *Unstable participation:* Select a participation level uniformly at random in $[10\%, 90\%]$ per Δ. ⓒ *High participation:* Reset participation to 80%, randomly increase or decrease participation by 3% per Δ (staying in $[70\%, 90\%]$). ⓓ *Low participation:* Reset participation to 20%, randomly increase or decrease participation by 3% per Δ (staying in $[10\%, 30\%]$). Once the participation level was drawn according to this schedule, from instant to instant the environment selects a random set of asleep (awake) validators to wake up (put to sleep), respectively, to meet the participation levels.

We see in Fig. 7 that when participation is above the fast-confirmation threshold (⋯⋯), transactions are fast-confirmed swiftly (——) (ⓐ, ⓒ). When participation is volatile (ⓑ),

[14] https://etherscan.io/chart/blocksize

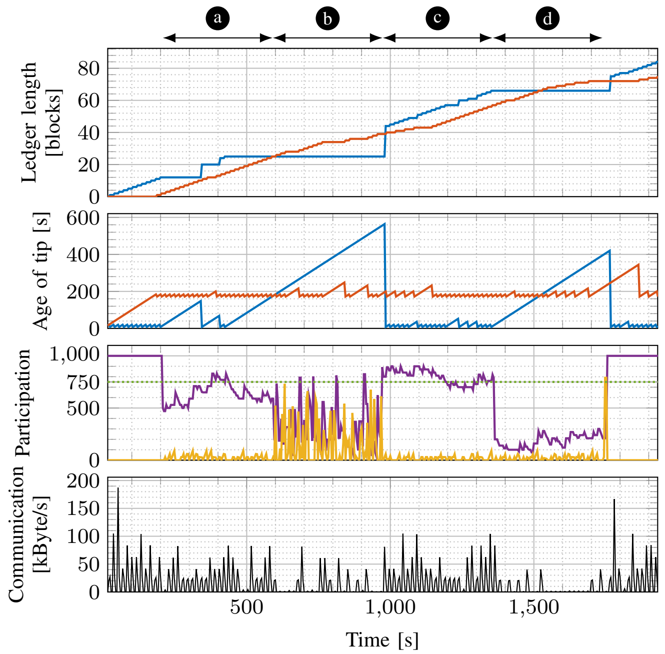

Fig. 7. Ledger length and age of the most recently confirmed block under fast (——) and slow (——) confirmation, and resulting broadcast load (——), for Goldfish with $n = 1000$, $\mathsf{thr}_{\mathrm{v}} = 0.1$, $\Delta = 4\,\mathrm{s}$, $\mathsf{thr}_{\mathrm{b}} = 3/n$, $\kappa = 10$, block size 80 kByte, under different environments of dynamic participation (——, see Sec. VI-C). When participation is above the fast-confirmation threshold of approx. $3n/4$ (┈┈┈), transactions are confirmed swiftly (4Δ, ——), otherwise fast confirmation stalls. When participation is volatile (cf. 600 s to 950 s), many honest validators are *dreamy* (——, cf. Sec. III-A8). Then, or when participation is steady but at a low level (cf. 1400 s to 1700 s), *effective* participation (by honest validators who are neither asleep nor dreamy) is low. Slow confirmation ($4\Delta\kappa$ base latency, ——) takes place throughout, but since $\mathsf{thr}_{\mathrm{b}} = 3/n$, slow confirmation degrades (cf. Fig. 6) when effective participation is low (cf. slots with no proposal around 650 s or 1600 s, ——, lead to latency spikes, ——).

many honest validators are *dreamy* (——, cf. Sec. III-A8). Then, or when participation is steady but at a low level (ⓓ), *effective* participation (by validators neither asleep nor dreamy) is low. Slow confirmation (——) takes place throughout, but degrades (cf. Fig. 6) when effective participation is low (cf. slots with no proposal around 650 s or 1600 s, ——, lead to latency spikes, ——). Communication load is modest (——).

Acknowledgment

We thank Aditya Asgaonkar, Carl Beekhuizen, Vitalik Buterin, Justin Drake, Dankrad Feist, Sreeram Kannan, Georgios Konstantopoulos, Barnabé Monnot, Dan Robinson, Danny Ryan, Caspar Schwarz-Schilling, and Fan Zhang for fruitful discussions. The work of JN was conducted in part while at Paradigm. JN, ENT and DT are supported by a gift from the Ethereum Foundation. JN is supported by the Protocol Labs PhD Fellowship and the Reed-Hodgson Stanford Graduate Fellowship. ENT is supported by the Stanford Center for Blockchain Research.

References

[1] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography*, ser. LNCS, vol. 8975. Springer, 2015, pp. 507–527.

[2] A. Kiayias and G. Panagiotakos, "On trees, chains and fast transactions in the blockchain," in *LATINCRYPT*, ser. LNCS, vol. 11368. Springer, 2017, pp. 327–351.

[3] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," arXiv:2003.03052v3 [cs.CR], 2020. [Online]. Available: http://arxiv.org/abs/2003.03052v3

[4] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 446–465.

[5] ——. (2020) A balancing attack on Gasper, the current candidate for Eth2's beacon chain. [Online]. Available: https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079

[6] ——, "The availability-accountability dilemma and its resolution via accountability gadgets," in *Financial Cryptography*, ser. LNCS, vol. 13411. Springer, 2022, pp. 541–559.

[7] ——. (2021) Attacking Gasper without adversarial network delay. [Online]. Available: https://ethresear.ch/t/attacking-gasper-without-adversarial-network-delay/10187

[8] C. Schwarz-Schilling, J. Neu, B. Monnot, A. Asgaonkar, E. N. Tas, and D. Tse, "Three attacks on proof-of-stake ethereum," in *Financial Cryptography*, ser. LNCS, vol. 13411. Springer, 2022, pp. 560–576.

[9] V. Buterin. (2020) Proposal for mitigation against balancing attacks to LMD GHOST. [Online]. Available: https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation

[10] J. Neu, E. N. Tas, and D. Tse, "Two more attacks on proof-of-stake GHOST/Ethereum," in *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, ser. ConsensusDay '22. ACM, 11 2022.

[11] ——. (2022) Balancing attack: LMD edition. [Online]. Available: https://ethresear.ch/t/balancing-attack-lmd-edition/11853

[12] ——. (2022) Avalanche attack on proof-of-stake GHOST. [Online]. Available: https://ethresear.ch/t/avalanche-attack-on-proof-of-stake-ghost/11854

[13] V. Buterin and V. Griffith, "Casper the friendly finality gadget," arXiv:1710.09437v4 [cs.CR], 2017. [Online]. Available: http://arxiv.org/abs/1710.09437v4

[14] S. Sankagiri, X. Wang, S. Kannan, and P. Viswanath, "Blockchain CAP theorem allows user-dependent adaptivity and finality," in *Financial Cryptography (2)*, ser. LNCS, vol. 12675. Springer, 2021, pp. 84–103.

[15] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI*. USENIX Association, 1999, pp. 173–186.

[16] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on bft consensus," arXiv:1807.04938v3 [cs.DC], 2018. [Online]. Available: http://arxiv.org/abs/1807.04938v3

[17] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *PODC*. ACM, 2019, pp. 347–356.

[18] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *AFT*. ACM, 2020, pp. 1–11.

[19] R. Pass and E. Shi, "The sleepy model of consensus," in *ASIACRYPT (2)*, ser. LNCS, vol. 10625. Springer, 2017, pp. 380–409.

[20] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO (1)*, ser. LNCS, vol. 10401. Springer, 2017, pp. 357–388.

[21] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT (2)*, ser. LNCS, vol. 10821. Springer, 2018, pp. 66–98.

[22] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *CCS*. ACM, 2018, pp. 913–930.

[23] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *Financial Cryptography*, ser. LNCS, vol. 11598. Springer, 2019, pp. 23–41.

[24] I. Eyal and E. G. Sirer, "Majority is not enough: bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, pp. 95–102, 2018.

[25] (2023) Maximal extractable value (MEV). [Online]. Available: https://ethereum.org/en/developers/docs/mev/

[26] K. Liao and J. Katz, "Incentivizing blockchain forks via whale transactions," in *Financial Cryptography Workshops*, ser. LNCS, vol. 10323. Springer, 2017, pp. 264–279.

[27] M. Carlsten, H. A. Kalodner, S. M. Weinberg, and A. Narayanan, “On the instability of bitcoin without the block reward,” in *CCS*. ACM, 2016, pp. 154–167.

[28] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, “Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability,” in *IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 910–927.

[29] C. Beekhuizen, C. Schwarz-Schilling, and F. D’Amato. (2021) Change fork choice rule to mitigate balancing and reorging attacks. [Online]. Available: https://ethresear.ch/t/change-fork-choice-rule-to-mitigate-balancing-and-reorging-attacks/11127

[30] D. Kane, A. Fackler, A. Gagol, and D. Straszak, “Highway: Efficient consensus with flexible finality,” arXiv:2101.02159v2 [cs.DC], 2021. [Online]. Available: http://arxiv.org/abs/2101.02159v2

[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling byzantine agreements for cryptocurrencies,” in *SOSP*. ACM, 2017, pp. 51–68.

[32] J. Chen and S. Micali, “Algorand: A secure and efficient distributed ledger,” *Theor. Comput. Sci.*, vol. 777, pp. 155–183, 2019.

[33] V. K. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, “Prism: Deconstructing the blockchain to approach physical limits,” in *CCS*. ACM, 2019, pp. 585–602.

[34] M. Fitzi, P. Gaži, A. Kiayias, and A. Russell, “Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition,” Cryptology ePrint Archive, Paper 2018/1119, 2018. [Online]. Available: https://eprint.iacr.org/2018/1119

[35] R. Pass and E. Shi, “Thunderella: Blockchains with optimistic instant confirmation,” in *EUROCRYPT (2)*, ser. LNCS, vol. 10821. Springer, 2018, pp. 3–33.

[36] P. Khanchandani and R. Wattenhofer, “Byzantine agreement with unknown participants and failures,” in *IPDPS*. IEEE, 2021, pp. 952–961.

[37] V. Goyal, H. Li, and J. Raizes, “Instant block confirmation in the sleepy model,” in *Financial Cryptography (2)*, ser. LNCS, vol. 12675. Springer, 2021, pp. 65–83.

[38] D. Malkhi, A. Momose, and L. Ren. Instant finality in byzantine generals with unknown and dynamic participation. [Online]. Available: https://blog.chain.link/instant-finality-in-byzantine-generals-with-unknown-and-dynamic-participation/

[39] ——, “Byzantine consensus under fully fluctuating participation,” Cryptology ePrint Archive, Paper 2022/1448, 2022. [Online]. Available: https://eprint.iacr.org/2022/1448

[40] A. Momose and L. Ren, “Constant latency in sleepy consensus,” in *CCS*. ACM, 2022, pp. 2295–2308.

[41] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” https://bitcoin.org/bitcoin.pdf, 2008.

[42] J. A. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol: Analysis and applications,” in *EUROCRYPT (2)*, ser. LNCS, vol. 9057. Springer, 2015, pp. 281–310.

[43] P. Khanchandani and R. Wattenhofer, “Brief announcement: Byzantine agreement with unknown participants and failures,” in *PODC*. ACM, 2020, pp. 178–180.

[44] Y. Pu, L. Alvisi, and I. Eyal, “Safe permissionless consensus,” in *DISC*, ser. LIPIcs, vol. 246. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 33:1–33:15.

[45] G. Losa and E. Gafni, “Consensus in the unknown-participation message-adversary model,” arXiv:2301.04817v1 [cs.DC], 2023. [Online]. Available: http://arxiv.org/abs/2301.04817v1

[46] D. Malkhi, A. Momose, and L. Ren. Minority corruption resilience in byzantine generals with unknown and fluctuating participation. [Online]. Available: https://blog.chain.link/minority-corruption-resilience-in-byzantine-generals-with-unknown-and-fluctuating-participation/

[47] F. D’Amato and L. Zanolini, “Recent latest message driven ghost: Balancing dynamic availability with asynchrony resilience,” arXiv:2302.11326v2 [cs.DC], 2023. [Online]. Available: http://arxiv.org/abs/2302.11326v2

[48] ——, “A simple single slot finality protocol for ethereum,” arXiv:2302.12745v1 [cs.DC], 2023. [Online]. Available: http://arxiv.org/abs/2302.12745v1

[49] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[50] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, 2015. [Online]. Available: http://toc.cryptobook.us/book.pdf

[51] S. Micali, M. O. Rabin, and S. P. Vadhan, “Verifiable random functions,” in *FOCS*. IEEE Computer Society, 1999, pp. 120–130.

[52] Y. Dodis and A. Yampolskiy, “A verifiable random function with short proofs and keys,” in *Public Key Cryptography*, ser. LNCS, vol. 3386. Springer, 2005, pp. 416–431.

[53] (2023) Ethereum staking. [Online]. Available: https://ethereum.org/en/staking/

[54] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, “BFT protocol forensics,” in *CCS*. ACM, 2021, pp. 1722–1743.

[55] S. Gilbert and N. A. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[56] A. Lewis-Pye and T. Roughgarden, “Resource pools and the cap theorem,” arXiv:2006.10698v1 [cs.DC], 2020. [Online]. Available: http://arxiv.org/abs/2006.10698v1

[57] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, “Player-replaceability and forensic support are two sides of the same (crypto) coin,” Cryptology ePrint Archive, Paper 2022/1513, 2022. [Online]. Available: https://eprint.iacr.org/2022/1513

[58] G. Itkis and L. Reyzin, “Forward-secure signatures with optimal signing and verifying,” in *CRYPTO*, ser. LNCS, vol. 2139. Springer, 2001, pp. 332–354.

[59] J. Neu, E. N. Tas, and D. Tse, “Snap-and-chat protocols: System aspects,” arXiv:2010.10447v1 [cs.CR], 2020. [Online]. Available: http://arxiv.org/abs/2010.10447v1

[60] J. O’Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O’Hearn. (2020) BLAKE3. [Online]. Available: https://github.com/BLAKE3-team/BLAKE3-specs/blob/ea51a3ac997288bf690ee82ac9cfc8b3e0e60f2a/blake3.pdf

[61] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004.

[62] S. Bowe. (2017) BLS12-381: New zk-SNARK elliptic curve construction. [Online]. Available: https://electriccoin.co/blog/new-snark-curve/

[63] J. Neu, S. Sridhar, L. Yang, D. Tse, and M. Alizadeh, “Longest chain consensus under bandwidth constraint,” in *4th ACM Conference on Advances in Financial Technologies*, ser. AFT ’22. ACM, 2022. [Online]. Available: https://arxiv.org/abs/2111.12332

[64] L. Kiffer, J. Neu, S. Sridhar, A. Zohar, and D. Tse, “Security of nakamoto consensus under congestion,” Cryptology ePrint Archive, Paper 2023/381, 2023. [Online]. Available: https://eprint.iacr.org/2023/381

[65] R. Nakamura. (2019) Analysis of bouncing attack on FFG. [Online]. Available: https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113

[66] ——. (2019) Prevention of bouncing attack on FFG. [Online]. Available: https://ethresear.ch/t/prevention-of-bouncing-attack-on-ffg/6114

Appendix A
Equivocation Discounting to Mitigate Spamming

For ease of exposition, we have presented a version of Goldfish which deals with equivocating votes simply by accepting all of them, but counting at most one per subtree (Alg. 2, l. 7). This approach is vulnerable to spamming attacks, because it requires validators to accept all the votes they receive. Even a single adversarially controlled validator can be used to create an arbitrarily large number of equivocating votes at a slot, with the goal of creating network congestion and making it impossible for honest validators to download all of the other votes in time, which can result in a loss of safety.

Equivocations are attributable faults, punishable by slashing *a posteriori*, but this does not prevent the attack vector *a priori*, given that only one validator is required for it. To mitigate it, we introduce *equivocation discounting*. This general technique is already present in the current implementation of Ethereum, but the ephemerality of votes in Goldfish allows for a simpler rule, with clear bounds on the number of messages required for honest views to converge. This is particularly important in order to have guarantees about the functioning of the vote buffering technique, and in turn about the security of the whole protocol, which relies on reorg resilience. We formalize the simple equivocation discounting rule here, as a combination of a modification to the GHOST-Eph fork-choice, a download rule, and a validity condition for proposals.

a) Equivocation discounting:

(a) *Fork-choice discounting:* When running the GHOST-Eph fork-choice rule at slot t,, only count the valid slot $t-1$ votes from those validators for which your bvtree contains a single valid slot $t-1$ vote, *i.e.*, those which are not viewed to have equivocated *at slot* $t-1$.

(b) *Download rule:* Only download (or forward as part of the peer-to-peer gossip layer) votes from the current and prior slots, and at most two votes per *eligible* validator (*i.e.*, the opened ticket (id, t) for the validator id is winning for the tag $(\mathtt{vote}, \mathsf{thr}_{\mathrm{v}})$, cf. Sec. V).

(c) *Validity condition for proposals:* A proposal whose bvtree contains more than two valid votes for the same slot from some validator is invalid, and so is one which contains any invalid vote.

The download rule and validity condition ensure that a validator only ever needs to download at most two votes per subsampled validator of the current and previous slot. Setting the subsampling parameters so that this is manageable, we can ensure that equivocations cannot succeed at creating network congestion sufficient to prevent the functioning of vote buffering. Previously, this meant guaranteeing that an honest proposer's bvtree be a superset of honest validators' bvtrees. Instead, the success of vote buffering now only requires that a leader's view of votes from voters which have not equivocated in the last slot is a superset of the validators' views of such votes, and so is its view of *the list of equivocators from the previous slot*. Agreement on these two is sufficient for agreement on the fork-choice output, *i.e.*, Lem. 2 still holds. Note that the leader still only needs to include its bvtree in the proposal message, because following the download rule guarantees that it will contain exactly all valid votes from validators which have not equivocated in the previous slot, together with a pair of votes, *i.e.*, equivocation evidence, for validators which have.

The security analysis for Goldfish with equivocation discounting is then the same as that for vanilla Goldfish. Vote buffering implies that all honest validators vote together when the proposal with the minimum precedence is honest, as in Lem. 2, and all honest validators voting together implies that the proposal is never reorged, as in Lem. 3. The latter is not affected by equivocation discounting, because it relies on the valid votes of honest validators, which do not equivocate. From these two properties, we obtain reorg resilience as in Thm. 3, and from reorg resilience, we eventually obtain safety and liveness.

Optimistic fast confirmations are also compatible with equivocation discounting, without any loss of resilience. Liveness and fast confirmation of honest proposals follow from Thm. 7, since equivocation discounting plays no role in it. For safety, the key ingredient is Lem. 4, from which Thm. 6 follows unchanged. We thus prove Lem. 4 here for Goldfish with equivocation discounting, by making a very small modification to the argument:

Proof of Lem. 4 with equivocation discounting. By Prop. 1, w.o.p., the number of adversarial validators at round $4\Delta(t+1)+\Delta$, eligible to vote at slot t, is less than $\frac{1}{2}n\,\mathsf{thr}_{\mathrm{v}}$. An eligible awake honest validator sends a single slot t vote at round $4\Delta t+\Delta$, implying that over $(\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}} - \frac{1}{2}n\,\mathsf{thr}_{\mathrm{v}} = (\frac{1}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ validators broadcast a single slot t vote by round $4\Delta(t+1)+\Delta$, and that is for a descendant of B. By Prop. 1, w.o.p., for all slots t, there can be at most $(1+\epsilon)n\,\mathsf{thr}_{\mathrm{v}}$ validators that are eligible to vote at t. Hence, the number of valid slot t votes for the descendants of any block B' conflicting with B, and which are from validators which have not also cast one of the $(\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ votes for B, must be less than $(1+\epsilon)n\,\mathsf{thr}_{\mathrm{v}} - (\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}} = (\frac{1}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ at any given round. The validator id^* broadcasts B and over $(\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ valid votes for it (in pieces) at round $4\Delta t+2\Delta$. Each honest validator, awake at round $4\Delta(t+1)+\Delta$ and eligible to vote at slot $t+1$, observes these votes in its bvtree at the round of voting (Alg. 5, l. 12). Upon invoking the GHOST-Eph fork-choice rule at any of the rounds $4\Delta t+3\Delta$, $4\Delta(t+1)$ or $4\Delta(t+1)+\Delta$, using only the votes from validators which are not seen to be equivocating at slot $t-1$, the votes for the descendants of any block B' conflicting with B are then less than $(\frac{1}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$, and the votes for descendants of B are over $(\frac{1}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$. This implies that all honest validators, awake at round $4\Delta(t+1)+\Delta$ and eligible to vote at slot $t+1$, all vote for B or one of its descendants at slot $t+1$. □

Appendix B
From LMD GHOST to Goldfish

In this section, we outline the shortcomings of LMD GHOST in comparison to Goldfish, then discuss how Goldfish could replace it in the Ethereum protocol.

A. Limitations of Gasper

In the first iteration of Gasper's LMD GHOST, ex-ante reorgs and balancing attacks [5], [7], [8] prevent security even in the full participation setting and without subsampling. The proposer boost technique [9] mitigates these issues, but is itself not compatible with dynamic participation, and it entails a lower adversarial tolerance ($\frac{1}{4}$) than what is obtained with message buffering ($\frac{1}{2}$). Moreover, ex-ante reorgs [8] are still possible with subsampling, compromising reorg resilience, and the latest message rule (LMD) itself is not compatible with dynamic participation. Both of these issues are due to considering votes from older slots, and Goldfish solves them through vote expiry. In the following, we give a more detailed account of all of these limitations.

a) Interaction of LMD GHOST and Casper FFG: The combination of Goldfish with the accountability gadget in Sec. III follows the generic construction of [6], which is proven to be secure for any appropriately secure dynamically available protocol and accountable BFT protocol. On the other hand, the combination of LMD GHOST and Casper FFG in

HLMD GHOST, the hybrid fork-choice rule of [3], is ad-hoc and complicated to reason about. Firstly, it is known to be susceptible to a *bouncing attack* [65]. Instead of LMD GHOST starting its fork-choice iteration from the last block *finalized* by Casper FFG, it starts from the last *justified* block, in the terminology of Casper FFG, *i.e.*, the last block that has been the target of FFG votes by a supermajority of *all* n validators. This is sufficient to ensure accountable safety of the finalized checkpoints; however, it hinders safety of the available ledger $\mathsf{ch}_{\mathrm{ava}}$ (after $\max(\mathsf{GST}, \mathsf{GAT})$) under partial synchrony in the sleepy model, in particular negating the healing property (Lem. 10) of $\mathsf{ch}_{\mathrm{ava}}$, preventing us from proving the ebb-and-flow property. The current mitigation for the bouncing attack causes other problems such as the splitting attack [66], akin to the balancing attacks [4]. Another problematic interaction stems from the fact that the FFG votes at any *Ethereum epoch* point at the epoch boundary block of that epoch, regardless of its confirmation status by the underlying LMD GHOST rule. (In fact, there is no confirmation rule specified for LMD GHOST.) The accountability gadget can then in principle interfere with the available chain, jeopardizing its standalone security properties. Finally, the FFG voting schedule is staggered throughout an epoch, as FFG votes are cast together with LMD GHOST votes, so it is not clear how to ensure that the views of honest validators when casting FFG votes are consistent, which would at least ensure liveness of the accountable chain.

b) Stale votes in LMD GHOST: Without vote expiry, the votes of honest asleep validators can be weaponized by an adversary controlling a small fraction of the validator set to execute an arbitrarily long reorg. This implies that the protocol is not dynamically available with *any* confirmation rule with finite confirmation time T_{conf}. Consider for example a validator set of size $n = 2m + 1$, and a partition of the validator set into three sets, V_1, V_2, V_3, with $|V_1| = |V_2| = m$ and $|V_3| = 1$. The validators in V_1, V_2 are all honest, while the one in V_3 is adversarial. Suppose that the adversarial validator in V_3 is the leader of slots t, and that it broadcasts two proposals, with conflicting blocks B_1 and B_2. It does so in such a way that validators in V_1 see only B_1 before voting, and validators in V_2 only B_2. Validators in V_1 then vote for B_1, and so does the adversarial validator, while validators in V_2 vote for B_2. B_1 becomes canonical, since it has received $m + 1$ votes. The adversary then puts all validators in V_2 to sleep, and they do not become awake for the remainder of the protocol. The adversarial validator does not cast any more votes for a while. Meanwhile, validators in V_1, keep voting for descendants of B_1. After waiting for $> T_{\mathrm{conf}}$ slots, the adversarial validator votes for B_2. Since the m latest votes of the validators in V_2 are still for B_2, it now has $m + 1$ votes and becomes canonical, resulting in all awake honest validators experiencing a reorg of all blocks confirmed after slot t. If there are no such blocks, liveness is violated, and otherwise safety is violated.

c) Proposer boost: Proposer boost is not compatible with dynamic availability, because the artificial fork-choice weight it temporarily provides to proposals is independent of participation: the lower the participation, the more powerful the boost is relative to the weight of real attestations from awake validators, and thus the more it can be exploited by the adversary. When the weight of awake honest validators is less than the boost, the adversary has complete control of the fork-choice during the slots in which it is elected as the leader.

d) Reorg resilience: Even in the setting of full participation, where the adversary cannot take advantage of votes of asleep validators, LMD GHOST lacks reorg resilience. This is firstly due to subsampling without vote expiry, because it allows the adversary to accumulate fork-choice weight by withholding blocks and attestations, *i.e.*, to execute ex ante reorgs [8]. Without subsampling, LMD GHOST is indeed reorg resilient in the full participation setting, *if proposer boost is replaced by vote buffering*. In fact, Thm. 3 obtains reorg resilience as a consequence of two properties, Lems. 2 and 3, respectively the property that all honest awake validators vote for an honest proposal, and the property that all honest validators voting together guarantee the inclusion of honest blocks in the canonical GHOST-Eph chain, both of which also hold for LMD GHOST with vote buffering.

With proposer boost, LMD GHOST is not reorg resilient for $\beta \geq \frac{1}{4}$, even in the full participation setting and without subsampling, because those two properties are in conflict for such β, for any boost value W_p. The first property only holds if $W_p > 2\beta$, because the adversary can otherwise still conclude an ex ante reorg by revealing later votes, which move all adversarial weight β from the proposer's branch to a conflicting one, and outweigh the proposer boost W_p. On the other hand, the second property only holds if $W_p + \beta < 1 - \beta$, because otherwise an adversarial proposer can make use of boost to conclude an ex post reorg. Therefore, we can only have reorg resilience when $3\beta < W_p + \beta < 1 - \beta$, *i.e.*, for $\beta < \frac{1}{4}$, by setting $W_p = \frac{1}{2}$.

B. Replacing LMD GHOST with Goldfish in Gasper

For Goldfish to be used as a drop-in replacement for LMD GHOST in Ethereum, only a few adjustments are required. Most importantly, vote expiry and message buffering would have to be introduced, with the latter replacing proposer boost. In principle, the proposer selection mechanism does not need to be overhauled, as Goldfish can operate with RANDAO, the proposer selection mechanism of LMD GHOST. RANDAO always selects a unique proposer, which reduces the communication load, when compared to a VRF lottery. On the other hand, it is not compatible with adaptive security, because the selected proposer is publicly known in advance, and moreover the selection is biasable. A VRF lottery also enables the confirmation time to be independent of participation.

Finally, in order to benefit from the security guarantees of Goldfish in its combination with an accountability gadget, the interaction with Casper FFG would have to be modified to fit the construction from [6], which we have also employed in this work.

Appendix C
Proof Sketch for Goldfish with Accountability Gadget

In Goldfish with accountability gadgets, a partially synchronous accountably-safe consensus protocol is used to determine checkpoints. Security of this protocol ensures the *safety and accountability of the prefix ledger* $\mathsf{ch}_{\mathrm{acc}}$ in the partially synchronous sleepy network model. To ensure the *prefix property*, the fork-choice rule of Goldfish is modified to respect earlier checkpoint decisions (Alg. 3). This modification requires adjustments of the analysis of Goldfish, because it opens up the possibility that for a proposal P^* by an honest leader, $P^*.B \preceq B$ no longer holds for all blocks B identified in Alg. 1, ll. 8, 22, 28 by awake honest validators at future rounds, due to a new checkpoint conflicting with $P^*.B$.

To prevent checkpoints from undermining the security in this manner, and rigorously argue security of the combination despite the modified fork-choice rule, the framework of accountability gadgets [6], [14] relies on two principles:

- **Gap Property:** After a (successful) checkpointing iteration with a new checkpoint, honest validators wait for $T_{\mathrm{chkpt}} = \Theta(\kappa)$ rounds before participating in the next iteration.
- **Recency Property:** For checkpointing, honest validators suggest and approve only the blocks that were *recently* confirmed as part of $\mathsf{ch}_{\mathrm{ava}}$.

We prove that once the network heals and honest validators become awake at round $\max(\mathsf{GST}, \mathsf{GAT})$, $\mathsf{ch}_{\mathrm{ava}}$ regains its security with the help of these properties. Key is the following *healing property*.

Lemma 5 (Healing property (sketch))**.** *Suppose the number of adversarial validators is less than $n/3$ at all rounds.*

Then, under partial synchrony in the sleepy model, the available ledger $\mathsf{ch}_{\mathrm{ava}}$ is secure with transaction confirmation time $\Theta(T_{\mathrm{chkpt}})$ after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Theta(\kappa)$.

Formal statements for Lem. 5, its proof, and the full proof for **P1** are given in App. F.

Proof sketch for Lem. 5. Since the number of adversarial validators is less than $n/3$ at all rounds, by the security of the consensus protocol used by the accountability gadgets, all checkpoints are consistent with each other. After round $\max(\mathsf{GAT}, \mathsf{GST})$, all awake honest validators agree on the rounds they enter and complete subsequent checkpoint iterations (up to a difference of Δ rounds). By the gap property, honest validators wait for $T_{\mathrm{chkpt}} = \Theta(\kappa)$ rounds before participating in the next checkpointing iteration after a successful one. This ensures that no new checkpoints appear for T_{chkpt} rounds, during which the honest validators can treat the last checkpoint as the 'new' genesis block. Then, via the security analysis in Sec. V-A, there exists a slot with an honest leader such that for P^* proposed by the leader, $P^*.B \preceq B$ for all blocks B identified in Alg. 1, ll. 8, 22, 28 by awake honest validators during future rounds, until a new checkpoint is determined. By the recency property, for checkpointing, honest validators suggest and approve only the blocks that were *recently* confirmed as part of $\mathsf{ch}_{\mathrm{ava}}$. Since $P^*.B \preceq B$ for all recently confirmed blocks B at the start of the next checkpointing iteration, if a new checkpoint appears in the next iteration, the above prefix relation continues to hold for all future slots after the iteration. Thus, it is possible to state an analogue of Thm. 1 for honest leaders during T_{chkpt} rounds following successful checkpoint iterations and prove security with transaction confirmation time $T_{\mathrm{conf}} = \Theta(T_{\mathrm{chkpt}})$ via a similar reasoning to Sec. V-A. □

Liveness of $\mathsf{ch}_{\mathrm{ava}}$ together with the liveness of the accountability gadget's consensus protocol imply the *liveness of* $\mathsf{ch}_{\mathrm{acc}}$ in the partially synchronous sleepy network model.

In the synchronous sleepy network model, Sec. V-A implies that $\mathsf{ch}_{\mathrm{ava}}$ remains secure until the first checkpoint is determined. However, checkpoints cannot undermine its security since only confirmed blocks in $\mathsf{ch}_{\mathrm{ava}}$ are approved for checkpointing by honest validators. Proof of **P2** is given in App. F.

Appendix D
Proofs for Goldfish with Fast Confirmation

We state an analogue of Lem. 1, namely Lem. 6, to match the new slot structure in App. E.

Proof of Lem. 4. By Prop. 1, w.o.p., the number of adversarial validators at round $4\Delta(t+1)+\Delta$, eligible to vote at slot t, is less than $\frac{1}{2}n\,\mathsf{thr}_{\mathrm{v}}$. An eligible awake honest validator sends a single slot t vote at round $4\Delta t + \Delta$, implying that over $(\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}} - \frac{1}{2}n\,\mathsf{thr}_{\mathrm{v}} = (\frac{1}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ validators broadcast a single slot t vote by round $4\Delta(t+1)+\Delta$, and that is for a descendant of B. By Prop. 1, w.o.p., for all slots t, there can be at most $(1+\epsilon)n\,\mathsf{thr}_{\mathrm{v}}$ validators that are eligible to vote at t. Hence, the number of valid slot t votes for the descendants of any block B' conflicting with B must be less than $(1+\epsilon)n\,\mathsf{thr}_{\mathrm{v}} - (\frac{1}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}} = (\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ at any given round. The validator id^* broadcasts B and over $(\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_{\mathrm{v}}$ valid votes for it (in pieces) at round $4\Delta t + 2\Delta$. Each honest validator, awake at round $4\Delta(t+1)+\Delta$ and eligible to vote at slot $t+1$, observes these votes in its bvtree at the round of voting (Alg. 5, l. 12). Upon invoking the GHOST-Eph fork-choice rule at any of the rounds $4\Delta t + 3\Delta$, $4\Delta(t+1)$ or $4\Delta(t+1)+\Delta$ (Alg. 1, ll. 8, 22, 28), for any awake honest validator id with bvtree $\mathcal{T}'$, $\textsc{Votes}(\mathcal{T}', B, t) > \textsc{Votes}(\mathcal{T}', B', t)$ for any block B' conflicting with B. This implies that all honest validators, awake at round $4\Delta(t+1)+\Delta$ and eligible to vote at slot $t+1$ all vote for B or one of its descendants at slot $t+1$. □

Proof of Thm. 5. Follows by Lems. 6, 4 and 3, by the same inductive argument used in the proof of Thm. 1, in that case following from Lems. 1, 2 and 3. Here, Lem. 6 is the analogue of Lem. 1 with the new slot structure, and Lem. 4 provides the base case, substituting Lem. 2. □

Proof of Thm. 6. If an honest validator fast confirms a block B at slot t, then B is in the canonical GHOST-Eph chain of every awake honest validator at all slots larger than t by

Thm. 5. Therefore, B is in the κ-slots-deep prefix of the canonical GHOST-Eph chains of all awake honest validators at slot $t+\kappa$, and thus confirmed by them with the standard confirmation rule. Therefore, Thm. 2 implies the safety of the protocol. □

Proof of Thm. 7. Proof of liveness follows from Thm. 2.

For the second part of the proof, by Lem. 2, all of eligible and awake honest validators vote for $P^*.B$ at slot t. Then, the buffer of any honest validator awake at round $4\Delta t + 2\Delta$ contains at least $(\frac{3}{4}+\frac{\epsilon}{2})n\,\mathsf{thr}_\mathrm{v}$ votes (by Chernoff bound) for the block $P^*.B$, implying that all honest validators awake at rounds $4\Delta t+2\Delta$ fast confirm $P^*.B$ at the respective slots. □

Appendix E
Proof of Lem. 1

Proof of Lem. 1. By the *pseudorandomness* property of the VRF-based lottery (Secs. II-A3 and III-A1), for any given slot t and validators id_1 and id_2, $\mathsf{id}_1 \neq \mathsf{id}_2$,

$$\Pr\left[\mathrm{IsWinning}^{l_v}((\mathsf{id},t),\mathrm{Open}^{l_v}_{\mathsf{id}_1}(t))\right] = \mathsf{thr}_\mathrm{v} \tag{9}$$

$$\Pr\left[\mathrm{IsWinning}^{l_b}((\mathsf{id},t),\mathrm{Open}^{l_b}_{\mathsf{id}_1}(t))\right] = \mathsf{thr}_\mathrm{b} \tag{10}$$

$$\Pr\left[\mathrm{Prio}(\mathrm{Open}^{l_b}_{\mathsf{id}_1}(t)) < \mathrm{Prio}(\mathrm{Open}^{l_b}_{\mathsf{id}_2}(t))\right] = \frac{1}{2}, \tag{11}$$

where $l_v = (\texttt{vote}, \mathsf{thr}_\mathrm{v})$ and $l_b = (\texttt{block}, \mathsf{thr}_\mathrm{b})$ are the lotteries, and $\mathrm{Open}^{l_v}_{\mathsf{id}_1}(t)$, $\mathrm{Open}^{l_b}_{\mathsf{id}_1}(t)$, $\mathrm{Open}^{l_b}_{\mathsf{id}_2}(t)$, and $\mathrm{Open}^{l_v)}_{\mathsf{id}_2}(t)$ are independent random variables.

We first consider the protocol *without key-evolving primitives*. By the *uniqueness* property of the lottery (Sec. II), w.o.p., for all validators id and slots t, the ticket (id,t) can be opened at most one unique opening (Alg. 1, l. 20). Let $\tilde{H}_t$ denote the number of honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t. Let $\tilde{A}_t$ denote the number of adversarial validators at round $3\Delta(t+1)+\Delta$ that are eligible to vote at slot t. Recall that A_r and H_r denote the number of adversarial and honest validators awake at round r respectively (note that the honest validators have been awake since the closest round $3\Delta t + 2\Delta$ same as or preceding r). Let $n_t = H_{3\Delta t+\Delta} + A_{3\Delta(t+1)+\Delta} \geq n_0 = \Theta(\kappa)$.

By the pseudorandomness property, the adversary cannot predict in advance which honest validators will become eligible to vote or propose at a given slot. Moreover, if the adversary decides to corrupt the honest validators eligible to vote at a slot t after learning their identities at round $3\Delta t+\Delta$, it takes over 3Δ rounds for the corruption to take effect, implying that these validators cannot be counted as part of $\tilde{A}_t$. Hence, as $\frac{A_r}{A_r+H_{r-3\Delta}} \leq \beta < \frac{1}{2}-\epsilon$ for all rounds r, w.o.p.,

$$\mathbb{E}[\tilde{H}_t] = H_{3\Delta t+\Delta}\mathsf{thr}_\mathrm{v} \geq (\frac{1}{2}+\epsilon)n_t\mathsf{thr}_\mathrm{v}$$

$$\mathbb{E}[\tilde{A}_t] = A_{3\Delta(t+1)+\Delta}\mathsf{thr}_\mathrm{v} \leq (\frac{1}{2}-\epsilon)n_t\mathsf{thr}_\mathrm{v}$$

By a Chernoff bound,

$$\Pr\left[\tilde{H}_t < \frac{1}{2}n_t\mathsf{thr}_\mathrm{v}\right] \leq e^{-\frac{\epsilon^2}{1+2\epsilon}n_t\mathsf{thr}_\mathrm{v}}$$

$$\Pr\left[\tilde{A}_t > \frac{1}{2}n_t\mathsf{thr}_\mathrm{v}\right] \leq e^{-\frac{\epsilon^2}{1+3\epsilon}n_t\mathsf{thr}_\mathrm{v}}.$$

Thus, at any given slot t, $\tilde{H}_t > \tilde{A}_t$, except with probability

$$2\exp\,(-\frac{\epsilon^2}{1+3\epsilon}n_0\mathsf{thr}_\mathrm{v}).$$

By a union bound, every slot t has more honest validators awake at round $3\Delta t + \Delta$ and eligible to vote at slot t than adversarial validators at round $3\Delta(t+1)+\Delta$, eligible to vote at slot t (and more than $\frac{1}{2}n_0\mathsf{thr}_\mathrm{v}$ such honest validators), except with probability

$$2T_\mathrm{hor}\exp\left(-\frac{\epsilon^2}{1+3\epsilon}n_0\mathsf{thr}_\mathrm{v}\right) + \mathrm{negl}(\lambda) = \mathrm{negl}(\kappa) + \mathrm{negl}(\lambda),$$

since $n_0 = \Theta(\kappa)$ and $T_\mathrm{hor} = \Theta(\kappa)$. By the same reasoning, w.o.p., every slot t has more honest validators awake and eligible to propose for slot t at round $3\Delta t$ than adversarial validators at round $3\Delta t + \Delta$, eligible to propose for slot t.

Finally, for any given slot t, each valid slot t proposal broadcast within rounds $[3\Delta t, 3\Delta t+\Delta]$ has the same probability of achieving the minimum precedence up to terms negligible in λ[15]. Now, at a slot t, if an honest validator's proposal achieves the minimum precedence among the valid slot t proposals broadcast by Δ rounds into the slot, then that validator is identified as the slot leader by all honest validators awake at round $3\Delta t + \Delta$. Taking a fixed $t \geq \kappa$, the probability that no awake honest validator's proposal has the minimum precedence among the valid slot s proposals broadcast by Δ rounds into the slot, during the slots $s \in [t-\kappa, t]$, is upper bounded by $2^{-\kappa}+\mathrm{negl}(\kappa)+\mathrm{negl}(\lambda)$. Union bounding over all T_hor many such intervals, we find that w.o.p., all slot intervals of length κ have at least one slot t, where an honest validator is identified as the slot leader by all awake honest validators at round $3\Delta t + \Delta$.

Now with key-evolving primitives, we define $\tilde{H}_t = H_{3\Delta t+\Delta}$ and $\tilde{A}_t = A_{3\Delta t+\Delta}$. Similarly, we define $n_t = H_{3\Delta t+\Delta} + A_{3\Delta t+\Delta} \geq n_0 = \Theta(\kappa)$. In this case, $\frac{A_r}{A_r+H_r} \leq \beta < \frac{1}{2}-\epsilon$ for all rounds r. Note that the adversary cannot predict in advance which honest validators will become eligible to vote or propose at a given slot due to the pseudorandomness property of the lottery. Moreover, if the adversary corrupts the honest validators eligible to vote at a slot t after learning their identities at round $3\Delta t + \Delta$, it cannot make these validators broadcast new valid votes for slot t since the keys for slot t would have been evolved prior to adversarial corrruption (*i.e.*, these corrupted validators cannot be counted as part of $\tilde{A}_t$). Hence, the number of valid slot t votes adversarial validators can broadcast by round $3\Delta(t+1)+\Delta$ is upper bounded by the number of adversarial validators at round $3\Delta t+\Delta$ that are eligible to vote at slot t. Finally, by the same calculations as above, every slot t has more honest validators eligible to vote and awake at round $3\Delta t + \Delta$ than the adversarial validators

[15] We assume that $\mathrm{poly}(\kappa)\,\mathrm{negl}(\lambda) = \mathrm{negl}(\lambda)$.

at round $3\Delta(t+1)+\Delta$ eligible to vote at slot t (and more than $\frac{1}{2}n_0\mathsf{thr}_\mathrm{v}$ such honest validators), except with probability

$$2T_\mathrm{hor}\exp\left(-\frac{\epsilon^2}{1+3\epsilon}n_0\mathsf{thr}_\mathrm{v}\right)+\mathrm{negl}(\lambda)=\mathrm{negl}(\kappa)+\mathrm{negl}(\lambda).$$

Similarly, w.o.p., every slot t has more honest validators awake and eligible to propose for slot t at round $3\Delta t$ than adversarial validators at round $3\Delta t+\Delta$ eligible to propose for slot t. Thus, via the same argument, w.o.p., all slot intervals of length κ have at least one slot t, where an honest validator is identified as the slot leader by all awake honest validators at round $3\Delta t+\Delta$. □

Since Goldfish slots consist of 4Δ rounds in the case of fast confirmation, we state an analogue of Lem. 1 to match the new slot structure:

Lemma 6. *Suppose the Goldfish execution is $(\frac{1}{2}, 4\Delta)$-compliant.*

Then, w.o.p., for every slot t, the number of adversarial validators at round $4\Delta(t+1)+\Delta$, eligible to vote at slot t, is less than the number of honest validators, awake at round $4\Delta t+\Delta$ and eligible to vote at slot t.

Also w.o.p., all slot intervals of length κ have at least one slot t, where an honest validator is identified as the slot leader by all awake honest validators at round $4\Delta t+\Delta$.

Proof of Lem. 6 is very similar to the proof of Lem. 1, and follows from the same arguments using $(\frac{1}{2}, 4\Delta)$-compliant executions.

Appendix F
Analysis of Goldfish with Accountability Gadgets

We now prove the ebb-and-flow property for Goldfish combined with accountability gadgets (Fig. 4). The following analysis extensively refers to the details of the accountability gadgets described in [6, Section 4]. These gadgets can be *instantiated* with any BFT protocol that satisfies accountable safety (*e.g.*, PBFT [15], HotStuff [17]).

To distinguish the votes cast by validators as part of the accountability gadget iterations from those broadcast within Goldfish, we will refer to the former as *gadget votes*. Similarly, to distinguish the leaders of accountability gadget iterations from the leaders of Goldfish slots, we will refer to the former as the *iteration leaders*. We refer the reader to [6] for the accountability gadget specific definitions of the timeout parameter T_tmout and the confirmation delay T_bft of the BFT protocol. We highlight that honest iteration leaders propose only the blocks B^* that are *confirmed* in their view of ch_ava, *i.e.*, $B^* \preceq B^{\lceil\kappa}$ for B identified in Alg. 1, ll. 8, 22, 28 run using ch_ava. Similarly, honest validators send accepting gadget votes only for the checkpointing proposals that are *confirmed* in their view of ch_ava. We set T_chkpt, the time gap between the accountability gadget iterations, to be at least $6\Delta(\kappa+1)+T_\mathrm{tmout}+T_\mathsf{bft}$ (this is necessary for proving the ebb-and-flow property as will be evident in the following proofs). This makes the upper bound T_upper on the total duration of an iteration $T_\mathrm{chkpt}+T_\mathrm{tmout}+T_\mathsf{bft} = 6\Delta(\kappa+1)+2(T_\mathrm{tmout}+T_\mathsf{bft})=\Theta(\kappa)$.

We first show that ch_ava remains secure under synchrony in the sleepy network model, despite the added gadget.

Proposition 2. Suppose a $(\frac{1}{2}, 3\Delta)$-compliant execution of Goldfish in the synchronous sleepy network model of Sec. II-B. If a block B is observed to be checkpointed by an honest validator for the first time at some round r, then B is in the common prefix of the chains identified in Alg. 1, ll. 8, 22, 28 right before round r by all awake honest validators.

Proof. Since the execution is $(\frac{1}{2}, 3\Delta)$-compliant, for a block to become checkpointed, at least one honest validator must have sent an accepting gadget vote for that block. Let B_i denote the sequence of checkpointed blocks listed in the order of the rounds r_i at which, an awake honest validator observed B_i to be checkpointed for the first time. Proof is by induction on the indices of these blocks.

Induction Hypothesis: B_i is in the common prefix of the chains identified in Alg. 1, ll. 8, 22, 28 right before round r_i by all awake honest validators, and stays so until at least round r_{i+1}.

Base Case: Since an honest validator sends an accepting gadget vote only for a confirmed block (*i.e.*, κ slots deep), B_1 must have been confirmed by an honest validator at some slot t_1 before round r_1. As all honest validators start the fork-choice at the genesis block prior to r_1 and B_1 is confirmed in an honest view, it is in the prefix of a block proposed by an honest leader by Lem. 1 and Thm. 1. Hence, B_1 is in the common prefix of the chains identified in Alg. 1, ll. 8, 22, 28 right before round r_1 by all awake honest validators. It also stays in the common prefix until at least round r_2.

Inductive Step: By the induction hypothesis, checkpointing of the blocks $B_1, \ldots, B_{i-1}$ does not alter the fork-choice rule at Alg. 3, l. 2 for any awake honest validator. Hence, by the same reasoning above, B_i is in the common prefix of the chains identified in Alg. 1, ll. 8, 22, 28 right before round r_i by all awake honest validators, and stays so until at least round r_{i+2}. □

Lemma 7 (Safety and liveness of ch_ava under synchrony)**.** *Suppose a $(\frac{1}{2}, 3\Delta)$-compliant execution of Goldfish in the synchronous sleepy network model of Sec. II-B. Then, w.o.p., the available ledger ch_ava satisfies $1/2$-safety and $1/2$-liveness (at all times).*

Proof. By Prop. 2, checkpointing of blocks does not alter the fork-choice rule at Alg. 3, l. 2 for any awake honest validator. Concretely, if the honest validators started the fork-choice rule from the genesis block at all rounds instead of the latest checkpoint in view, then they would end up with the same execution. Thus, the security of ch_ava follows from Thm. 2. □

We next demonstrate the liveness of ch_acc after $\max(\mathsf{GST}, \mathsf{GAT})$. In the subsequent analysis, the total number of validators is denoted by n (cf. Sec. II). The accountability gadget is instantiated with a BFT protocol that has an accountable safety resilience of $n/3$.

Proposition 3 (Prop. 2 of [6])**.** The BFT protocol satisfies $n/3$-liveness after $\max(\mathsf{GST}, \mathsf{GAT})$ with transaction confirmation time $T_{\mathsf{bft}} < \infty$.

Proposition 4 (Prop. 3 of [6])**.** Consider a $(\frac{1}{3}, 3\Delta)$-compliant execution of Goldfish in the partially synchronous sleepy network model of Sec. II-B. Suppose a block from iteration c was checkpointed in the view of an honest validator at round r. Then, every honest validator enters iteration $c+1$ by round $\max(\mathsf{GST}, \mathsf{GAT}, r) + \Delta$.

Let c' be the largest iteration such that a block B was checkpointed in the view of some honest validator before $\max(\mathsf{GAT}, \mathsf{GST})$. (Let $c' = 0$ and B be the genesis block if there does not exist such an iteration.) If an honest validator enters an iteration $c'' > c'$ at some round $r \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{chkpt}}$, every honest validator enters iteration c'' by round $r + \Delta$.

Proof of Prop. 4 follows from the proof of [6, Prop. 3].

Proof. Suppose a block B from iteration c was checkpointed in the view of an honest validator id at round r. Then, there are over $2n/3$ accepting gadget votes for B from iteration c on $\mathsf{LOG}^{r}_{\mathsf{bft},\mathsf{id}}$, the output ledger of the BFT protocol in id's view at round r. All gadget votes and BFT protocol messages observed by id by round r are delivered to all other honest validators by round $\max(\mathsf{GST}, \mathsf{GAT}, r)+\Delta$. Hence, by the safety of the BFT protocol when $f < n/3$, for any honest validator id', the ledger $\mathsf{LOG}^{r}_{\mathsf{bft},\mathsf{id}}$ is the same as or a prefix of the ledger observed by id' at round $\max(\mathsf{GST}, \mathsf{GAT}, r) + \Delta$. Thus, for any honest validator id', there are over $2n/3$ accepting gadget votes for B from iteration c on $\mathsf{LOG}_{\mathrm{bft}}$ at round $\max(\mathsf{GST}, \mathsf{GAT}, r) + \Delta$. This implies every honest validator enters iteration $c + 1$ by round $\max(\mathsf{GST}, \mathsf{GAT}, r) + \Delta$.

Finally, by the reasoning above, all honest validators enter iteration $c'+1$ by round $\max(\mathsf{GAT}, \mathsf{GST})+\Delta$. Thus, entrance time of the honest validators to subsequent iterations have become synchronized by round $\max(\mathsf{GAT}, \mathsf{GST})+\Delta+T_{\mathrm{chkpt}}$: If an honest validator enters an iteration $c'' > c'$ at some round $r \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{chkpt}}$, every honest validator enters iteration c'' by round $r + \Delta$. Similarly, if a block from iteration c'' is first checkpointed in the view of an honest validator at some round after $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{chkpt}}$, then it is checkpointed in the view of all honest validators within Δ rounds. □

Lemma 8 (Liveness of $\mathsf{ch}_{\mathrm{acc}}$, analogue of Thm. 4 of [6])**.** *Consider a $(\frac{1}{3}, 3\Delta)$-compliant execution of Goldfish in the partially synchronous sleepy network model of Sec. II-B. Suppose $\mathsf{ch}_{\mathrm{ava}}$ is secure (safe and live) after some round $T_{\mathsf{heal}} \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta + T_{\mathrm{chkpt}}$. Then, w.o.p., $\mathsf{ch}_{\mathrm{acc}}$ satisfies $n/3$-liveness after round T_{heal} with transaction confirmation time $T_{\mathrm{conf}} = \Theta(\kappa^2)$.*

Proof of Lem. 8 follows from the proof of [6, Thm. 4].

Proof. By Prop. 3, $\mathsf{LOG}_{\mathrm{bft}}$ is live with transaction confirmation time T_{bft} after $\max(\mathsf{GST}, \mathsf{GAT})$, a fact we will use subsequently.

Let c' be the largest iteration such that a block B was checkpointed in the view of some honest validator before $\max(\mathsf{GAT}, \mathsf{GST})$ (Let $c' = 0$ and B be the genesis block if there does not exist such an iteration). Then, by Prop. 4, entrance times of the honest validators to subsequent iterations become synchronized by round $\max(\mathsf{GAT}, \mathsf{GST})+\Delta+T_{\mathrm{chkpt}}$: If an honest validator enters an iteration $c > c'$ at some round $r \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{chkpt}}$, every honest validator enters iteration c by round $r + \Delta$.

Suppose an iteration $c > c'$ has an honest iteration leader $\mathrm{L}^{(c)}$, which sends a checkpoint proposal, denoted by $\hat{b}_c$, at some round $r > T_{\mathsf{heal}} + T_{\mathrm{chkpt}}$. The proposal $\hat{b}_c$ is received by every honest validator by round $r + \Delta$. Since the entrance times of the validators are synchronized by $T_{\mathsf{heal}} \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta + T_{\mathrm{chkpt}}$, every honest validator sends a gadget vote by round $r + \Delta$. By Lem. 10, $\hat{b}_c \preceq B^{\lceil\kappa}$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator after r. Moreover, $\hat{b}_c$ is a descendant all of the checkpoints seen by the honest validators until then. Consequently, at iteration c, every honest validator sends a gadget vote accepting $\hat{b}_c$ by round $r + \Delta$, all of which appear within $\mathsf{LOG}_{\mathrm{bft}}$ in the view of every honest validator by round $r + \Delta + T_{\mathsf{bft}}$. Thus, $\hat{b}_c$ becomes checkpointed in the view of every honest validator by round $r+\Delta+T_{\mathsf{bft}}$. (Here, we assume that T_{tmout} was chosen large enough for $T_{\mathrm{tmout}} > \Delta + T_{\mathsf{bft}}$ to hold.)

Since $r > T_{\mathsf{heal}} + T_{\mathrm{chkpt}}$, by Lem. 10, $\hat{b}_c$ contains at least one honest block since an earlier checkpointed block in its prefix from before iteration c. This implies that the prefix of $\hat{b}_c$ contains at least one fresh honest block that enters $\mathsf{ch}_{\mathrm{acc}}$ by round $r + \Delta + T_{\mathsf{bft}}$.

Next, we show that an adversarial leader cannot make an iteration last longer than $\Delta + T_{\mathrm{tmout}} + T_{\mathsf{bft}}$ for any honest validator after the initial T_{chkpt} period elapsed. Indeed, if an honest validator id enters an iteration c at round $r - T_{\mathrm{chkpt}}$, by round $r + T_{\mathrm{tmout}}$, either it sees a block (potentially $\perp$) become checkpointed for iteration c, or it sends a reject vote for iteration c. In the first case, every honest validator sees a block checkpointed for iteration c by round at most $r + T_{\mathrm{tmout}} + \Delta$. In the second case, rejecting gadget votes from over $2n/3 > n/3$ validators appear in $\mathsf{LOG}_{\mathrm{bft}}$ in the view of every honest validator by round at most $r+T_{\mathrm{tmout}}+\Delta+T_{\mathsf{bft}}$. Hence, a new checkpoint, potentially $\perp$, is output in the view of every honest validator by round $r + T_{\mathrm{tmout}} + \Delta + T_{\mathsf{bft}}$.

Finally, we observe that except with probability $(1/3)^{\kappa}$, there exists a checkpoint iteration with an honest leader within κ consecutive iterations. Since an iteration lasts at most $\max(\Delta + T_{\mathrm{tmout}} + T_{\mathsf{bft}}, \Delta + T_{\mathrm{chkpt}} + T_{\mathsf{bft}}) \leq \Delta + T_{\mathrm{chkpt}} + T_{\mathrm{tmout}} + T_{\mathsf{bft}} = \Theta(\kappa)$ rounds, and a new checkpoint containing a fresh honest block in its prefix appears when an iteration has an honest leader (Lem. 10), w.o.p., any transaction received by an honest validator at round t appears within $\mathsf{ch}_{\mathrm{acc}}$ in the view of every honest validator by round at most $t + \kappa(\Delta + T_{\text{tmout}} + T_{\mathsf{bft}} + T_{\text{chkpt}})$. Hence, via a union bound over the total number of iterations (which is a polynomial in κ), we observe that if ch_{ava} satisfies security after some round T_{heal}, then w.o.p., ch_{acc} satisfies liveness after T_{heal} with a transaction confirmation time $T_{\text{conf}} = \Theta(\kappa^2)$. □

The latency expression $T_{\text{conf}} = \Theta(\kappa^2)$ stated in Lem. 8 is a *worst-case* latency to guarantee that an honest block enters the accountable, final prefix ledger ch_{acc} with overwhelming probability. In the expression, the first κ term comes from the requirement to have $T_{\text{chkpt}} = \Theta(\kappa)$ slots in between the accountability gadget iterations, and the second κ term comes from the fact that it takes $\Theta(\kappa)$ iterations for the accountability gadget to have an honest iteration leader except with probability $\text{negl}(\kappa)$. The accountability gadget protocol asks honest validators to wait for $T_{\text{chkpt}} = \Theta(\kappa)$ slots in between iterations to ensure the security of the protocol, reasons for which will be evident in the proof of Lem. 10.

Unlike the worst-case latency, the expected latency for an honest block to enter ch_{acc} after ch_{ava} regains its security would be $\Theta(\kappa)$ as each checkpointing iteration has an honest leader with probability at least $2/3$. In this context, the latency of $\Theta(\kappa)$ is purely due to the requirement to have $T_{\text{chkpt}} = \Theta(\kappa)$ slots in between the accountability gadget iterations. Here, waiting for T_{chkpt} slots in between iterations guarantees the inclusion of a new honest block in ch_{ava}, which in turn appears in the prefix of the next checkpoint, implying a liveness event whenever there is an honest iteration leader.

Lem. 8 requires the available ledger ch_{ava} to eventually regain security under partial synchrony when there are less than $n/3$ adversarial validators. Towards this goal, we first analyze the gap and recency properties, the core properties that must be satisfied by the accountability gadget for recovery of security of ch_{ava}. The gap property states that the blocks are checkpointed sufficiently apart in time, controlled by the parameter T_{chkpt}:

Proposition 5 (Gap property, analogue of Prop. 4 of [6])**.** Consider a $(\frac{1}{3}, 3\Delta)$-compliant execution of Goldfish in the partially synchronous sleepy network model of Sec. II-B. Given any round interval of size T_{chkpt}, no more than a single block can be checkpointed in the interval in the view of any honest validator.

Proof of Prop. 5 follows from the fact that upon observing a new checkpoint that is not $\perp$ for an iteration, honest validators wait for T_{chkpt} rounds before sending gadget votes for the checkpoint proposal of the next iteration, and there cannot be two conflicting checkpoints for the same iteration in the view of any honest validator.

As in [6] and [14], we state that a block B^* checkpointed at iteration c and round $r > \max(\mathsf{GST}, \mathsf{GAT})$ in the view of an honest validator id is T_{rcnt}-recent if $B^* \preceq B^{\lceil \kappa}$ for B identified in Alg. 1, l. 28 by id' at some round within $[r - T_{\text{rcnt}}, r]$. Then, we can express the recency property as follows:

Lemma 9 (Recency property, analogue of Lem. 1 of [6])**.** *Consider a $(\frac{1}{3}, 3\Delta)$-compliant execution of Goldfish in the partially synchronous sleepy network model of Sec. II-B. Every checkpointed block proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is T_{rcnt}-recent for $T_{\text{rcnt}} = \Delta + T_{\text{tmout}} + T_{\mathsf{bft}}$.*

Proof. By the proof of Lem. 8, if a block B proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is checkpointed in the view of an honest validator at some round r, it should have been proposed after round $r - (\Delta + T_{\text{tmout}} + T_{\mathsf{bft}})$. Moreover, over $2n/3$ validators must have sent accepting gadget votes for B by round r. Let id denote such an honest validator. It would vote for B only after it sees the checkpoint proposal for iteration c, *i.e.*, after round $r - T_{\text{rcnt}} = r - (\Delta + T_{\text{tmout}} + T_{\mathsf{bft}})$, and only if the proposal is confirmed in its view. Hence, B must be κ slots deep in the chain returned at Alg. 1, l. 28 by validator id at some round within $[r - T_{\text{rcnt}}, r]$. This concludes the proof that every checkpointed block proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is T_{rcnt}-recent. □

Lemma 10 (Healing property, analogue of Thm. 5 of [6])**.** *Consider a $(\frac{1}{3}, 3\Delta)$-compliant execution of Goldfish in the partially synchronous sleepy network model of Sec. II-B. Then, ch_{ava} is secure with transaction confirmation time $T_{\text{chkpt}} + T_{\text{tmout}} + T_{\mathsf{bft}} = \Theta(\kappa)$ after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\text{chkpt}}$.*

Moreover, for the iteration proposal $\hat{b}_c$ of an honest iteration leader broadcast at round r, it holds that $\hat{b}_c \preceq B^{\lceil \kappa}$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator after r, and $\hat{b}_c$ contains a fresh honest block that is not in the prefix of any checkpoint from before iteration c.

Proof of Lem. 10 is different from the proof of [6, Thm. 5] since the accountability gadget is applied to a longest chain protocol in [6], whereas it is applied to Goldfish in our case. Therefore, the full proof is presented below.

Proof. By [6, Thm. 3], ch_{acc} provides accountable safety with resilience $n/3$ except with probability $\text{negl}(\lambda)$ in the partially synchronous sleepy network model. As the execution is $(\frac{1}{3}, 3\Delta)$-compliant, w.o.p., no two checkpoints observed by awake honest validators conflict.

Let c be the largest iteration such that a block B was checkpointed in the view of some honest validator before $\max(\mathsf{GAT}, \mathsf{GST})$. (Let $c = 0$ and B be the genesis block if there does not exist such an iteration.) Then, by Prop. 4, if an honest validator enters an iteration $c' > c$ at some round $r \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\text{chkpt}}$, every honest validator enters iteration c by round $r + \Delta$. Let c' be the first iteration such that the first honest validator to enter c' enters it after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\text{chkpt}}$ (*e.g.*, at some round r such that $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\text{chkpt}} < r < \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\text{chkpt}}$). Then, all honest validators enter iteration c' and agree on the last checkpointed block within Δ rounds. Subsequently, the honest validators wait for T_{chkpt} rounds before casting any gadget vote for a checkpoint proposal of iteration c', during which no block can be checkpointed (Prop. 5, gap property).

By Lem. 1, w.o.p., the slot interval of length κ starting after round $r + \Delta$ contains a slot t with an honest leader and proposal P^*. After round $r \geq \mathsf{GST}$, all messages broadcast by honest validators are received by all honest validators within Δ rounds. As honest validators agree on the last checkpointed block during the interval $[r+\Delta, r+T_{\text{chkpt}}]$, by the absence of new checkpoints, the GHOST-Eph fork-choice rule starts at the same last checkpointed block for all honest validators during the interval (Alg. 2, l. 2). Then, by Lem. 1, w.o.p., $P^*.B \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any round after $3\Delta t + 2\Delta$, until at least a new block is checkpointed in the view of an honest validator.

By Lem. 9 (recency property), the next block checkpointed in the view of an honest validator (which happens earliest at some iteration $c'' \geq c'$ and round $r' \geq r + T_{\text{chkpt}}$ by Prop. 5, the gap property) must have been confirmed by some honest validator id at some round within $[r' - T_{\text{rcnt}}, r']$, where $r' - T_{\text{rcnt}} \geq r + 6\Delta\kappa + 4\Delta$. Hence, the new checkpointed block is κ slots deep in the chains identified in Alg. 1, ll. 8, 22, 28 by id, and is a descendant of $P^*.B$. This implies $P^*.B \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any round after $3\Delta t + 2\Delta$ *indefinitely*.

Note that if the iteration leader was honest, for its proposal $\hat{b}_c$ broadcast at some round r'', it holds that $\hat{b}_c \preceq B^{\lceil\kappa}$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator after round r. Moreover, $P^*.B \preceq \hat{b}_c$, implying that honest checkpoint proposals contain fresh honest blocks in their prefixes.

Finally, we extend the above argument to future checkpoints by induction. Let B_n denote the sequence of checkpointed blocks, ordered by their iteration numbers $c_n \geq c'$, $c_1 = c''$. The rounds r_n, at which the blocks B_n are first checkpointed in the view of an honest validator satisfy the relation $r_{n+1} \geq r_n + T_{\text{chkpt}}$ and $r_1 = r''$. Via the inductive assumption and the reasoning above, w.o.p., in each interval $[r_n+\Delta, r_{n+1}-T_{\text{rcnt}}]$, there exists a slot t_n with an honest leader and proposal P_n such that $P_n.B \preceq B$ for any B identified in Alg. 1, ll. 8, 22, 28 by any awake honest validator in any round after $3\Delta t_n + 2\Delta$ *indefinitely*. Hence, for a sufficiently large confirmation time exceeding the maximum possible length of an iteration (*i.e.*, $T_{\text{conf}} \geq T_{\text{chkpt}} + T_{\text{tmout}} + T_{\mathsf{bft}}$), these honest blocks imply the security of the Goldfish protocol after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\text{chkpt}}$. □

Note that Thm. 1 holds for the honest blocks proposed during the intervals $[r_n + \Delta, r_{n+1} - T_{\text{rcnt}}]$ as all honest validators agree on the latest checkpoint during these intervals.

Proof of Thm. 4. We first show the property **P1**, namely, the accountable safety and liveness of the accountable, final prefix ledger ch_{acc} under partial synchrony in the sleepy model. By [6, Thm. 3], ch_{acc} provides accountable safety with resilience $n/3$ except with probability $\text{negl}(\lambda)$ under partial synchrony in the sleepy model. By Lem. 10, under the same model, the available ledger ch_{ava} is secure after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\text{chkpt}}$. Using this fact and Lem. 8, we can state that, w.o.p., ch_{acc} satisfies liveness after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\text{chkpt}}$ with transaction confirmation time $T_{\text{conf}} = \Theta(\kappa^2)$.

Finally, the property **P2** follows from Lem. 7, and **Prefix** follows by construction of the ledgers ch_{acc} and ch_{ava}. This concludes the proof of the ebb-and-flow property. □