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 future proof-of-stake (PoS) Ethereum. In its current form, the protocol is brittle and intricate to reason about, as evidenced by recent attacks, patching attempts, and Görli testnet reorgs. We present Goldfish, which can be seen as a considerably simplified variant of the current protocol, and prove that it is secure and reorg resilient in synchronous networks with dynamic participation, assuming a majority of the nodes (*validators*) follows the protocol honestly. Furthermore, we show that subsampling validators can improve the communication efficiency of Goldfish, and that Goldfish is composable with finality gadgets and accountability gadgets. The aforementioned properties make Goldfish a credible candidate for a future protocol upgrade of PoS Ethereum, as well as a versatile pedagogical example. Akin to traditional propose-and-vote-style consensus protocols, Goldfish is organized into slots, at the beginning of which a leader proposes a block containing new transactions, and subsequently members of a committee take a vote towards block confirmation. But instead of using quorums, Goldfish is powered by a new mechanism that carefully synchronizes the inclusion and exclusion of votes in honest validators' views.

1 Introduction

The latest message driven (LMD) greedy heaviest observed sub-tree (GHOST) [49, 27] consensus protocol is a key component of the Gasper protocol [8] that is supposed to power proof-of-stake (PoS) Ethereum's beacon chain after 'the Merge'. The initial version specified with Gasper [8] was shown to be broken using the *balancing attack*, first in synchronous and partially synchronous networks with adversarial message delay [42, 38], and later in

The authors are listed alphabetically.

networks with non-adversarial but merely random network delay [43, 39]. In response, *proposer boosting* was added to the protocol [6]. It was subsequently shown that the LMD functionality alone can be exploited to conduct a balancing-type attack despite proposer boosting [44, 41], and that Gasper's LMD GHOST component without LMD would suffer from a so called *avalanche attack* [44, 40]. Again in response, *equivocation discounting* was added to the protocol. Not least because of its complexity, the protocol 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 US dollars.

We present a protocol, nicknamed Goldfish, with the following key properties (the importance of some of these properties is discussed in detail below):

(a) The protocol can be viewed as a small variation of the currently specified and deployed LMD GHOST protocol of the PoS Ethereum beacon chain.

(b) It is *provably secure*, assuming honest majority of *validators* (*i.e.*, nodes with stake), and network synchrony (*i.e.*, adversarial delay, up to a known upper bound Δ).

(c) It can tolerate *dynamic participation* [45], *i.e.*, a large fraction and fluctuating set of simultaneous temporary crash faults among validators.

(d) It is *reorg resilient*, *i.e.*, block proposals by honest validators are guaranteed to make their way into the output ledger, with a prefix known to the block producer at the time of block production.

(e) It supports *subsampling* of validators, to improve communication efficiency and resilience to *adaptive corruption* (cf. *player-replaceability* [19, 12]).

(f) It is *simple*.

(g) It is composable with *finality gadgets and accountability gadgets* such as [7, 42, 47, 43]. The composite can achieve the *ebb-and-flow consensus formulation* [42] desired of PoS Ethereum's beacon chain.

As a result, Goldfish can serve the following purposes:

(a) The protocol can serve as a drop-in replacement for LMD GHOST in the PoS Ethereum beacon chain protocol. Due to its similarity to LMD GHOST, it is a credible candidate for a future upgrade of PoS Ethereum consensus, requiring relatively small implementation changes, and thus presents a go-to option for the Ethereum community, should problems with the current protocol become acute.

(b) Unlike earlier negative results (attacks) on variants of LMD GHOST as is part of the PoS Ethereum beacon chain, Goldfish is the first positive result (security proof) for a close variant, slightly strengthening confidence in this family of protocols.

(c) The protocol is a good pedagogical example for a simple consensus protocol that is secure in synchronous networks under dynamic participation.

Akin to traditional propose-and-vote-style consensus protocols, Goldfish is organized into slots, at the beginning of which a (pseudo-randomly elected) leader proposes a block containing new transactions, and subsequently (pseudo-randomly elected) members of a committee take a vote towards block confirmation. But instead of using quorums, Goldfish is based on two key techniques, *vote buffering* and *vote expiry*, to carefully synchronize honest validators' views, and which might be of independent interest:

(a) *Vote buffering* (also known as *view merge* [3]) first appeared in [23]. In short, buffering of votes received from the network together with carefully timed inclusion of these votes in each validator's local view leads to the property that in periods with honest leader all honest validators vote in favor of the leader's proposal. This leads to reorg resilience, *i.e.*, honest proposals are guaranteed to remain in the canonical chain. Since honest proposals contain fresh transactions and stabilize their prefix, and long streaks of only adversarial proposals are exponentially unlikely, safety and liveness of the protocol follow readily under high participation and without subsampling.

(b) *Vote expiry* (also known as *ephemeral votes*) means that during each time slot only votes from the immediately preceding time slot influence the protocol's behavior.[2] This allows the protocol to support dynamic participation and to subsample small committees of voters per slot from the full set of validators. Furthermore, vote expiry keeps the set of votes small that might affect short-term future actions of honest validators. Thus, only few protocol messages need to be buffered and merged among honest validators' views at any point in time. *Vote expiry is thus a prerequisite for the efficiency/feasibility of vote buffering.*

Inspired by the application requirements for a drop-in replacement of LMD GHOST in the PoS Ethereum beacon chain, Goldfish was designed to achieve the following goals:

- *Secure consensus in synchronous networks under dynamic participation [45] and honest majority:* The protocol is parametric in a security parameter κ and outputs a single ledger at each validator at any point in time. The ledger is *safe* (meaning that ledgers output by two validators at two points in time are one a prefix of the other), except with probability decaying exponentially in κ. The ledger is *live* (meaning that transactions enter the ledgers output by honest validators 'soon' after they were first input to an honest validator), with a confirmation delay determined by the analysis (a linear function of κ), except with probability decaying exponentially in κ. *Safety and liveness constitute security of the consensus protocol.*

- *Composability with finality gadgets and accountability gadgets:* Goldfish is composable with *finality gadgets and accountability gadgets* such as [7, 42, 47, 43]. The resulting

[2] From the alleged forgetfulness of its animal namesake stems the Goldfish protocol's name.

composite protocol (cf. Figure 2) can achieve the *ebb-and-flow consensus formulation* [42, 43] desired of PoS Ethereum's beacon chain (cf. Definition 3).

- *Reorg resilience:* As part of the Goldfish protocol, honest validators every now and then get to be the *leader* and get to propose blocks (bundles of transactions) for inclusion. The protocol is resilient to reorgs, meaning that whenever there is an honest leader, its proposal will eventually make it into the protocol's output ledger, with a prefix ledger that can be determined at the time of block production. *This property is broadly important for incentive alignment,* e.g.*, it reduces the risk of* undercutting *[30, 9],* time-bandit *[14], or* selfish mining *[16] attacks.*

- *Subsampling:* The protocol supports subsampling, meaning that at each slot the protocol can pseudo-randomly select a small group of validators to run the protocol on behalf of the total validator set. *The results in a considerably lower communication overhead.* Furthermore, the selected validators send only a single protocol message. *Thus, the protocol satisfies* player-replaceability *[19, 12] and is secure against adaptive adversaries (which can corrupt validators during protocol execution).*

- *Optimistic fast confirmation:* Under *optimistic* conditions when participation happens to be high and a supermajority of $\frac{3}{4}$ fraction of validators is honest, Goldfish confirms with expected latency independent of κ.

- *Similarity to LMD GHOST:* Goldfish is intentionally simple, and similar to LMD GHOST as currently deployed, offering a credible path to adoption of Goldfish in the short to medium term. For the two key ingredients, vote expiry can be realized entirely with minor changes to the vote accounting logic. Vote buffering becomes practical due to vote expiry. While vote buffering requires slight changes to the temporal structure and validator behavior of the current protocol, Goldfish and the current LMD GHOST are similar 'in spirit' and share their fundamental structure.

Related works For Goldfish, we build on the sleepy model [45] of a synchronous network where the number and identity of actively participating (*awake*) validators can change over time (*dynamic participation*). The first secure consensus protocol for the sleepy model was Nakamoto's seminal *longest chain* (LC) protocol, first for proof-of-work (PoW) with Bitcoin [34, 18], and subsequently for PoS with protocols such as Ouroboros [28, 15, 1] and Sleepy Consensus/Snow White [45, 13, 4]. A drawback of these protocols is that the (expected) confirmation latency scales linearly with the security parameter κ (same for Goldfish's 'normal' confirmation rule). Parallel composition of LC protocol instances was suggested in [2, 17] to overcome the κ-dependence of the confirmation latency. Goldfish has an optimistic fast confirmation rule providing κ-independent latency under high participation. Furthermore, unlike Goldfish, protocols from the LC family are not reorg resilient: every block produced by the adversary can be used to displace one honestly produced block [16].

In contrast, many 'classical' propose-and-vote-style BFT consensus protocols [10, 50, 11] have constant (expected) confirmation latency and are (or can be modified to be) reorg resilient, but do not tolerate dynamic participation. An early consensus protocol of 'classical' flavor for a model with unknown (but *static* rather than dynamic) participation is due to Khanchandani and Wattenhofer [26, 25]. A subsequent protocol of the 'classical' variety [20] supports dynamic participation, but with confirmation latency linear in the security parameter κ. Like Goldfish (and unlike LC), the latency of this protocol is independent of the participation level. Probabilistic security is also overcome in the permissionless PoW setting with omission faults by [46].

A recent work by Momose and Ren [33] presents the first propose-and-vote-style permissioned/PoS protocol that supports dynamic participation with confirmation latency independent of security parameter and level of participation. In [31], the prerequisites for liveness were relaxed, at the expense of reduced adversarial resilience (from $\frac{1}{2}$ down to $\frac{1}{3}$). A key challenge for 'classical' consensus protocols in the sleepy setting is that quorum certificates are no longer transferable between awake honest validators. The works of Momose, Ren, and Malkhi aim to (partially) restore/replace transferability with graded agreement, but otherwise retain the structure of a 'classical' consensus protocol: Multiple stages of voting, with the aim of reaching quorums to progress towards confirmation. Validators keep individual state across the stages using locks, and express discontent with a proposal by abstaining from the vote. In contrast, Goldfish is closer in spirit to LC: A simple main loop in which validators repeatedly express support for the leading tip in their view (by producing blocks in LC, casting votes in Goldfish). Eventually, honest validators converge on what they perceive as the leading tip, which then accumulates endorsements quickly from the majority of honest validators, so that no competing tip will ever be able to supersede it. Unlike the works of Momose, Ren, and Malkhi, Goldfish achieves constant expected confirmation latency only under optimistic conditions of high participation and less than $\frac{1}{4}$ fraction adversarial validators.

Unlike Goldfish, the aforementioned protocols differ substantially from the LMD GHOST component of the current PoS Ethereum protocol. Furthermore, they are considerably more involved (*e.g.*, in number of stages) than LMD GHOST or Goldfish. Thus, adoption of these protocols for PoS Ethereum would require substantial design and engineering effort.

Highway [23] employs some of the techniques also found in Goldfish and in the PoS Ethereum beacon chain. For instance, vote buffering first appeared in [23]. Furthermore, Highway aims to achieve flexible finality on a gradual scale using a monolithic protocol, and does not consider dynamic participation. In contrast, we follow the ebb-and-flow formulation [42] of the beacon chain requirements (and with it adopt the extension of the sleepy model to allow for periods of asynchrony [42]) with a gradual notion of (probabilistic) confirmation for the available full ledger (which is powered by Goldfish with the help of vote buffering), and a binary notion of finality for the accountable final prefix (which is provided by a separate finality/accountability gadget). In particular, we adopt the modular approach described in [7, 42, 47, 43] to designing protocols that satisfy the ebb-and-flow property using finality gadgets and accountability gadgets (Figure 2).

Outline We recapitulate the model of synchronous networks with dynamic participation and asynchronous periods in Section 2, before describing our basic Goldfish protocol in Section 3, and an optimistic fast confirmation rule in Section 4. We analyze the protocol and prove the desired security properties in Section 5, before concluding in Sections 6 and 7 with a comparison of Goldfish to the variant of LMD GHOST currently deployed in PoS Ethereum. There, we also discuss advantages, drawbacks, and implementation aspects of Goldfish, and survey open problems with the current PoS Ethereum protocol.

2 Model and Preliminaries

Notation We let λ and κ denote the security parameters associated with the cryptographic primitives employed by the protocol, and with the Goldfish protocol itself, respectively. We say that an event happens with probability *negligible* in κ, denoted by $\mathrm{negl}(\kappa)$, if its probability, as a function of κ, is $o(1/\kappa^d)$ for all $d > 0$. Similarly, an event happens with *overwhelming* probability if it happens except with probability that is $\mathrm{negl}(\kappa) + \mathrm{negl}(\lambda)$. We write $\mathsf{ch}_1 \preceq \mathsf{ch}_2$ to express that ledger ch_1 is the same as, or a prefix of ch_2. Two blocks b_1 and b_2 are said to be conflicting if for the ledgers ch_1 and ch_2 defined by the prefix of and including these blocks, it holds that $\mathsf{ch}_1 \not\preceq \mathsf{ch}_2$ and $\mathsf{ch}_2 \not\preceq \mathsf{ch}_1$.

Validators There are n validators. Each validator is equipped with a unique cryptographic identity. The public keys are common knowledge (public-key infrastructure, PKI).

Environment and adversary Time is divided into discrete rounds and the validators have synchronized clocks.[3] Validators receive transactions from the environment $\mathcal{Z}$. The adversary $\mathcal{A}$ is a probabilistic poly-time algorithm. It can choose up to f validators to corrupt, hereafter called the *adversarial* validators. The internal state of the adversarial validators is handed over to $\mathcal{A}$, which can subsequently make them deviate from the protocol in an arbitrary and coordinated fashion (*Byzantine* fault). Adversarial validators constitute at most β fraction of the awake validators (see *sleepiness* below) at any given round, where $\beta < 1/2 - \epsilon$ for a constant ϵ. The adversary is *adaptive*: it can choose which validators to corrupt after the randomness of the protocol is drawn and during the protocol execution.

Sleepiness The adversary can decide for each round which honest validator is *awake* or *asleep* at that round. The sleeping schedule of the validators is *static*: the adversary chooses the schedule before the randomness of the protocol is drawn. Asleep validators do not execute the protocol (cf. *temporary crash faults*). When an honest validator wakes up, it receives all the messages it should have received while asleep. Adversarial validators are always awake. At any point in time, the majority of the awake validators is honest, and the number of validators awake at any given round is bounded below by a constant n_0.

[3] Bounded clock offsets can be captured by the network delay.

Network Honest validators can *broadcast* messages to communicate with each other. Messages arrive with an adversarially determined delay that can differ for each recipient. In a *synchronous network*, the delay is upper-bounded by a constant Δ rounds, and Δ is known to the protocol. Upon receiving a message, an honest validator re-broadcasts it, to ensure receipt by every honest validator within Δ time.

A *partially synchronous network in the sleepy model* [42] is characterized by a global stabilization time (GST), a global awake time (GAT), and a synchronous network delay upper-bound Δ. GST and GAT are constants chosen by the adversary, unknown to the honest validators, and can be causal functions of the protocol randomness. Before GST, message delays are arbitrarily adversarial. After GST, the network is synchronous, with delay upper-bound Δ. Similarly, before GAT, the adversary can set any sleeping schedule for the validators, subject to the honest majority constraint and requirement of having at least n_0 awake validators at any given time. After GAT, all honest validators are awake.

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, represents the confirmation delay for transactions. In our analysis, we consider a finite time horizon T_{hor} that is polynomial in κ. We denote a consensus protocol's output chain, *i.e.*, the Goldfish ledger, in the view of a validator i at round t by ch_t^i.

Definition 1 (Security)**.** Let T_{conf} be a polynomial function of the security parameter κ. We say that a state machine replication protocol Π that outputs a ledger ch is *secure after time* T, and has transaction confirmation time T_{conf}, if ch satisfies:

- **Safety:** For any two rounds $t, t' \geq T$, and any two honest validators i and j (possibly $i = j$) awake at rounds t and t' respectively, either $\mathsf{ch}_t^i \preceq \mathsf{ch}_{t'}^j$ or $\mathsf{ch}_{t'}^j \preceq \mathsf{ch}_t^i$.
- **Liveness:** If a transaction is received by an awake honest validator at some round $t \geq T$, then for any round $t' \geq t + T_{\mathsf{conf}}$ and any honest validator i awake at round t', the transaction will be included in $\mathsf{ch}_{t'}^i$.

The protocol satisfies f-*safety* (f-*liveness*) if it satisfies safety (liveness) as long as the total number of adversarial validators stays below f for all rounds. Similarly, the protocol satisfies $1/2$-*safety* ($1/2$-*liveness*) if it satisfies safety (liveness) as long as the fraction of adversarial validators among the awake ones stay below $1/2 - \epsilon$ for all rounds.

Accountable safety Accountable safety means that in the event of a safety violation, one can provably identify adversarial validators that have violated the protocol. Formally, in case of a safety violation for a protocol with accountable safety resilience $f > 0$, one can, after collecting evidence from sufficiently many honest validators, generate a cryptographic proof that irrefutably identifies f adversarial validators as protocol violators [24, 43]. By definition, the proof does not falsely accuse any honest validator, except with negligible probability. Accountable safety provides a stronger, *trust-minimizing* notion of safety, by making validators accountable for their actions.

Verifiable random function A verifiable random function (VRF) [32] is used for leader election and subsampling of the validators within the Goldfish protocol.

Definition 2 (from [15])**.** A function family $F_{(.)}\colon \{0,1\}^{a(\lambda)} \to \{0,1\}^{b(\lambda)}$ is a family of VRFs if there exist PPT algorithms $(\mathsf{VRF.Generate}, \mathsf{VRF.Prove}, \mathsf{VRF.Verify})$ such that (i) $\mathsf{VRF.Generate}$ outputs a pair of keys $(\mathsf{vpk}, \mathsf{vsk})$, (ii) $\mathsf{VRF.Prove}_{\mathsf{vsk}}(x)$ computes $(F_{\mathsf{vsk}}(x), \pi_{\mathsf{vsk}}(x))$, where π_{vsk} is the proof of correctness, and (iii) $\mathsf{VRF.Verify}_{\mathsf{vpk}}(x, \sigma, \pi)$ verifies that $\sigma = F_{\mathsf{vpk}}(x)$ using the proof π. Formally,

1. **Uniqueness:** Given $\sigma_1 \neq \sigma_2$, no values $(\mathsf{vpk}, x, \sigma_1, \sigma_2, \pi_1, \pi_2)$ can satisfy $\mathsf{VRF.Verify}_{\mathsf{vpk}}(x, \sigma_1, \pi_1) = \mathsf{VRF.Verify}_{\mathsf{vpk}}(x, \sigma_2, \pi_2) = 1$.
2. **Provability:** If $(\sigma, \pi) = \mathsf{VRF.Prove}_{\mathsf{vsk}}(x)$, then $\mathsf{VRF.Verify}_{\mathsf{vpk}}(x, \sigma, \pi) = 1$.
3. **Pseudorandomness:** For any PPT adversary $\mathcal{A} = (A_E, A_J)$, which runs for a total of $s(\lambda)$ steps when its first input is 1^λ, and does not query $\mathsf{VRF.Prove}$ on x,

$$\Pr\left[b = b' \,\middle|\, \begin{array}{c} (\mathsf{vpk},\mathsf{vsk}) \leftarrow \mathsf{VRF.Generate}; \\ (x,\pi) \leftarrow A_E^{\mathrm{PROVE}}(\mathsf{vpk}); \\ y_0 \leftarrow F_{\mathsf{vsk}}(x);\ y_1 \leftarrow \{0,1\}^{b(\lambda)}; \\ b \leftarrow \{0,1\};\ b' \leftarrow A_J^{\mathrm{PROVE}}(y_b,\pi) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

The ebb-and-flow formulation As Goldfish outputs a *dynamically available* ledger (*i.e.*, liveness under dynamic participation), by the availability-accountability dilemma [43], 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 [29]. However, 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 Goldfish [43]. Denoting the output of 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*:

Definition 3 (Ebb-and-flow formulation [42, 43])**.**

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

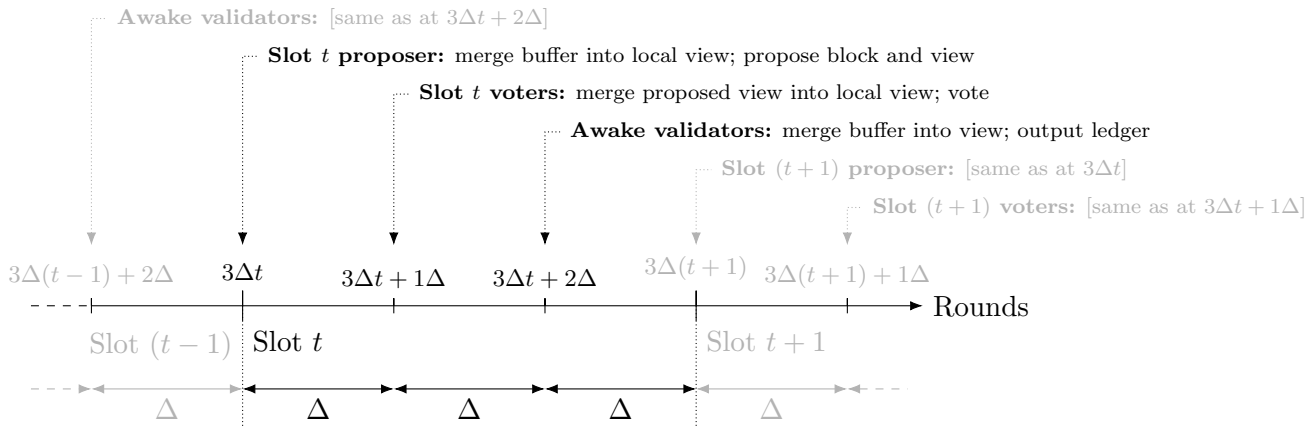

Figure 1: Throughout the execution, honest validators buffer received votes and merge them into their view only as explicitly specified. Goldfish has time slots of 3Δ rounds. Each time slot has a proposer and a committee of validators (voters). At the start of each slot, the proposer merges buffered votes into their local view, determines their canonical chain, and proposes and broadcasts a block extending it, together with their local view. One-thirds into each slot, each voter merges the proposed view into their local view, determines their canonical chain, and casts a vote on the tip. Two-thirds into the slot, all awake validators merge their buffers into their local view, and output a ledger according to GHOST-Eph.

The accountable, final prefix ledger ch_{acc} can experience liveness violations before GST or GAT, due to lack of communication among sufficiently many honest validators (ch_{acc} remains safe throughout); however, ch_{acc} catches up with the available ledger ch_{ava} (which can experience safety violations before GST, but remains live throughout) after validators wake up and the network turns synchronous. This ebb-and-flow behavior lends the formulation its name.

3 Protocol

We first describe the Goldfish protocol that is being proposed as a drop-in replacement for LMD GHOST in PoS Ethereum's beacon chain. We then describe how Goldfish can be securely integrated with accountability and finality gadgets (*e.g.*, Casper FFG).

3.1 The Goldfish Protocol

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

Algorithm 1 Algorithm executed by a validator $\mathcal{P}$ with signature public key pk, secret key sk, VRF public key vpk and secret key vsk. The variables $\mathcal{V}$ and $\mathcal{B}$ correspond to the view and the buffer of the validator, respectively, and are initialized with $\emptyset$. The notation '**at**' signifies blocking any further execution until the local clock of the validator reaches round s. Upon this event, the code snippet associated with the '**at**' block is executed. The notation **upon** $(\ldots)$ signifies an interrupt that is triggered when the message $(\ldots)$ is received over the network. The function $\textsc{Broadcast}\ (\ldots)$ sends the specified message to the peer-to-peer network for it to be gossiped. The function $\text{GHOST-Eph}(\mathcal{V}, t)$ outputs the block returned by the GHOST-Eph fork-choice rule executed using the votes for slot t in view $\mathcal{V}$. It is defined by Algorithm 2. The function $\textsc{NewBlock}(b)$ creates a new block that points at the specified block b as its parent. The notation $\mathsf{ch}^{\mathcal{P}}$ denotes the confirmed Goldfish chain in $\mathcal{P}$'s view at the *current* time specified by the clock.

$\mathcal{B}, \mathcal{V}, t \leftarrow \emptyset, \emptyset, 0$
for $t = 0, 1, 2, \ldots$ **do**
  at $3\Delta t$ ▷ PROPOSAL
    $\sigma_{t,L}, \pi_{t,L} \leftarrow \mathsf{VRF.Prove}_{\mathsf{vsk}}(t \,\|\, \text{'L'})$
    if $\sigma_{t,L} < \mathsf{thr}$ **then** ▷ $\mathcal{P}$ checks if it is eligible to propose in slot t
      $B \leftarrow \text{GHOST-Eph}(\mathcal{V} \cup \mathcal{B}, t-1)$ ▷ $\mathcal{P}$ *merges* its buffer and view
      $B_{\text{new}} \leftarrow \textsc{NewBlock}(B)$
      $\textsc{Broadcast}\ (\text{Proposal}, t, B_{\text{new}}, \mathcal{V} \cup \mathcal{B}, \sigma_{t,L}, \pi_{t,L})_{\mathcal{P}.\mathsf{sk}}$ ▷ $\mathcal{P}$ broadcasts its proposal
    end if
  end at
  at $3\Delta t + \Delta$ ▷ VOTE
    $\mathcal{B}_t \leftarrow \{(B^*, \mathcal{W}^*, \sigma^*) \mid (B^*, t^*, \mathcal{W}^*, \sigma^*) \in \mathcal{B} \wedge t^* = t\}$ ▷ $\mathcal{P}$ filters for proposals from slot t
    $(B^*, \mathcal{W}^*, \sigma^*) \leftarrow \arg\min_{(\sigma, B, \mathcal{W}) \in \mathcal{B}_t}(\sigma)$ ▷ $\mathcal{P}$ identifies the leader of slot t and its proposal
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{W}^* \cup \{B^*\}$ ▷ $\mathcal{P}$ *merges* its buffer and that of the leader into its view
    $\sigma_{t,V}, \pi_{t,V} \leftarrow \mathsf{VRF.Prove}_{\mathsf{vsk}}(t \,\|\, \text{'V'})$
    if $\sigma_{t,V} < \mathsf{thr}$ **then** ▷ $\mathcal{P}$ checks if it is eligible to vote at slot t
      $B \leftarrow \text{GHOST-Eph}(\mathcal{V}, t-1)$
      $\textsc{Broadcast}\ (\text{Vote}, t, B, \sigma_{t,V}, \pi_{t,V})_{\mathcal{P}.\mathsf{sk}}$ ▷ $\mathcal{P}$ broadcasts its vote
    end if
  end at
  at $3\Delta t + 2\Delta$ ▷ CONFIRMATION
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{B}$ ▷ $\mathcal{P}$ *merges* its buffer into its view
    $B \leftarrow \text{GHOST-Eph}(\mathcal{V}, t)$ ▷ $\mathcal{P}$ identifies the canonical GHOST-Eph chain for slot t
    $\mathsf{ch}^{\mathcal{P}} \leftarrow B$'s κ-deep prefix in terms of slots ▷ Goldfish ledger confirmed for slot t
  end at
end for
upon $(\text{Proposal}, t^*, B^*, \mathcal{W}^*, \sigma^*, \pi^*)_{\mathcal{P}.\mathsf{sk}}$ ▷ Executed upon receiving a proposal
  if $t^* = t \wedge \sigma^* < \mathsf{thr} \wedge \mathsf{VRF.Verify}_{\mathsf{vpk}}(t \,\|\, \text{'L'}, \sigma^*, \pi^*)$ **then** ▷ Check if the proposer was eligible
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\sigma^*, t, B^*, \mathcal{W}^*)\}$
  end if
end upon
upon $(\text{Vote}, t^*, B^*, \sigma^*, \pi^*)_{\mathcal{P}.\mathsf{sk}}$ ▷ Executed upon receiving a vote
  if $t^* = t \wedge \sigma^* < \mathsf{thr} \wedge \mathsf{VRF.Verify}_{\mathsf{vpk}}(t \,\|\, \text{'V'}, \sigma^*, \pi^*)$ **then** ▷ Check if the voter was eligible
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\mathcal{P}.\mathsf{pk}, \text{vote}, t, B^*)\}$
  end if
end upon

Algorithm 2 GHOST-Eph variant of GHOST fork-choice rule. The function GetChildren($\mathcal{V}, B$) returns the children of the specified block B that are in the view $\mathcal{V}$. GetNumVotes($\mathcal{V}, B, t$) returns the number of votes sent for slot t and for the blocks within the view $\mathcal{V}$ that lie in the subtree rooted at B.

```
1: function GHOST-Eph(V,t)
2:     B ← B_genesis
3:     while True do
4:         ch ← GetChildren(V, B)
5:         if ch = ∅ then
6:             return B
7:         end if
8:         ▷ Choosing the heaviest subtree with the most validators that have cast a vote for slot t
9:         B ← arg max_{B'∈ch}(GetNumVotes(V, B', t))
10:    end while
11:    return B
12: end function
13: function GetNumVotes(V,B,t)
14:    n ← 0
15:    S ← ∅
16:    for (P.pk, vote, t*, B*) ∈ V do
17:        ▷ Counting only the votes for slot t among the votes in the view V
18:        if P.pk ∉ S ∧ t* = t ∧ B ⪯ B* then
19:            S ← S ∪ {P.pk}
20:            n ← n + 1
21:        end if
22:    end for
23:    return n
24: end function
```

Vote buffering At any time, each validator stores the consensus messages received from the network in its *buffer* $\mathcal{B}$. The buffer is distinct from the validator's *view* $\mathcal{V}$, the ever-growing set of messages used to make consensus decisions. Buffered messages are admitted to the view (*merged*) only at specific points in time which are explicitly stated in the protocol description: $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{B}$.

Subsampling Every message broadcast as part of the protocol is accompanied with a VRF output σ and the associated proof π. Messages with valid VRF proofs, whose VRF output is below a predetermined threshold thr, are added to the buffer upon reception (lines 28 and 33 of Algorithm 1). Otherwise, the messages are ignored.

Vote expiry To determine the canonical chain, validators use the GHOST-Eph fork-choice function with ephemeral votes (cf. Algorithm 2). The function takes a view $\mathcal{V}$ and slot t as input, and finds the canonical GHOST-Eph chain determined by the votes within $\mathcal{V}$ *that were cast for slot* t. More specifically, starting at the genesis block, the function iterates over a sequence of blocks from $\mathcal{V}$, 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 its subtree. This continues until it reaches a leaf of the blocktree, and outputs a complete chain from leaf to root. The fork-choice rule ignores votes from before slot t in its decision (votes are *ephemeral*), lending GHOST-Eph its name.

The complete protocol The major events that happen during a slot are shown on Figure 1. At the beginning of each slot t, *i.e.*, round $3\Delta t$, each awake honest validator $\mathcal{P}$ with VRF secret key vsk locally checks the VRF evaluation $\sigma = F_{\mathsf{vsk}}(s \,\|\, \text{`L'})$ on the string $t \,\|\, \text{`L'}$ to detect if it is *eligible to propose* a block for slot t (line 5 of Algorithm 1). If $\sigma < \mathsf{thr}$, $\mathcal{P}$ identifies the tip of its canonical GHOST-Eph chain using the slot $t-1$ votes in its view, and broadcasts a proposal message containing a new block extending the tip and the union of its view and buffer, $\mathcal{V} \cup \mathcal{B}$ (lines 6 and 8 of Algorithm 1).

At slot $3\Delta t + \Delta$, among the proposal messages received for slot t, each honest awake validator $\mathcal{P}$ selects the one with the minimum VRF output, and accepts the block contained in the message as the proposal block for slot t (line 13 of Algorithm 1). It also *merges* its view with that of the proposal message (line 14 of Algorithm 1). Then, with its VRF secret key vsk, $\mathcal{P}$ checks the VRF evaluation $\sigma = F_{\mathsf{vsk}}(t \,\|\, \text{`V'})$ on the string $t \,\|\, \text{`V'}$ to detect if it is *eligible to vote* for a block at slot t (line 16 of Algorithm 1). If that is the case, $\mathcal{P}$ identifies the new tip of its canonical GHOST-Eph chain using the slot $t-1$ votes in its updated view, and broadcasts a slot t vote for this tip (line 17 and 18 of Algorithm 1).

At slot $3\Delta t + 2\Delta$, each honest awake validator $\mathcal{P}$ merges its buffer $\mathcal{B}$ containing the votes received over the period $(3\Delta t + \Delta, 3\Delta t + 2\Delta]$ with its view $\mathcal{V}$ (line 22 of Algorithm 1). It then identifies the new tip of its canonical GHOST-Eph chain using the slot t votes in its updated view, takes the prefix of this chain corresponding to blocks from slots $\leq t - \kappa$, and

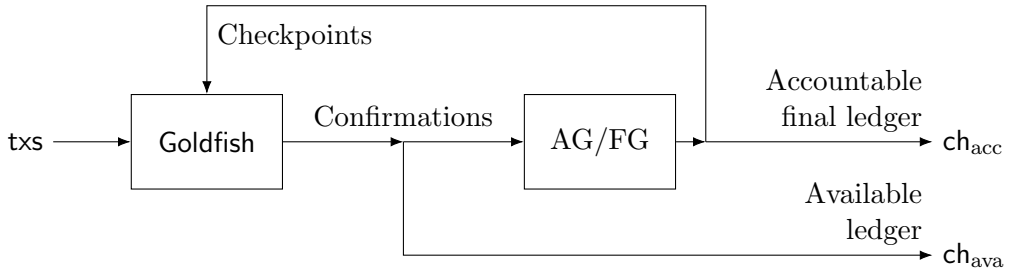

Figure 2: An accountability/finality gadget (AG/FG; also called *overlay*) checkpoints decisions of the dynamically available protocol Goldfish (also called *underlay*). A feedback loop ensures that Goldfish respects earlier checkpoint decisions. This construction satisfies the design goal of PoS Ethereum, to produce an available full ledger that remains live under dynamic participation of validators, and a prefix ledger that remains accountably safe under network partition [42, 43].

outputs this confirmed prefix as the Goldfish ledger[4] (lines 23 and 24 of Algorithm 1).

At the start of the next slot, at round $3\Delta t + 3\Delta$, the buffer of any honest validator contains all the votes received by honest validators in the period $(3\Delta t + \Delta, 3\Delta t + 2\Delta]$, *i.e.*, all the votes which they have merged into their view at round $3\Delta t + 2\Delta$. In particular, the proposal message of an honest leader includes all such votes, ensuring that the view in an honest leader's proposal message is a superset of any honest validator's views at round $3\Delta(t+1)+\Delta$. This fact enables the honest leader to guarantee the inclusion of its proposal in the canonical GHOST-Eph chain thereafter, and is crucial for the security and reorg resilience of the protocol. Security and reorg resilience of the Goldfish protocol are proven in Section 5.1.

Validator replaceability Goldfish supports validator replaceability [19, 12]: due to aggressive subsampling, once a validator takes an action within the protocol, it does not play any further role (at least for a long time). Moreover, validators use key-evolving signatures [15, 21], and honest validators erase old keys. Player replaceability, coupled with pseudorandomness of the VRF, and key evolution, ensures that a validator cannot affect the protocol execution after it has taken an action, enabling security against an adaptive adversary.

3.2 Goldfish with Accountability Gadgets

For the composition of Goldfish with accountability and finality gadgets, we follow the construction in [43] (Figure 2). The accountability gadget of [43] can be applied on top of any dynamically available blockchain protocol modified to respect past *checkpoint* decisions (cf. Algorithm 3). For accountable safety, the gadget must be instantiated with

[4]The *ledger* of transactions can be obtained using the order imposed on the blocks by the chain, together with the order each block imposes on its transactions.

Algorithm 3 The GHOST-Eph fork-choice rule modified to respect the latest checkpoint. Besides the view of the blockchain and the Goldfish votes, it takes as input the latest checkpoint last in the view of the validator. For GetNumVotes, see Algorithm 2.

function GHOST-Eph($\mathcal{V}$, t, last)
  $B \leftarrow$ last ▷ Start the fork-choice rule from the latest checkpoint
  while True **do**
    ch $\leftarrow$ GetChildren($\mathcal{V}$, B)
    if ch $= \emptyset$ **then**
      return B
    end if
    $B \leftarrow \arg\max_{B' \in \text{ch}}(\text{GetNumVotes}(\mathcal{V}, B', t))$
  end while
  return B
end function

Algorithm 4 Interaction between Goldfish and the accountability gadget in a validator $\mathcal{P}$'s view. Goldfish as specified by Algorithm 1 uses the latest checkpoint last in the view of the validator within the modified GHOST-Eph fork-choice rule as specified by Algorithm 3. It outputs the available chain $\mathsf{ch}^{\mathcal{P}}_{\text{ava}}$. The function RunAccGadget starts a new iteration of the accountability gadget, during which the validator $\mathcal{P}$ inputs its available chain ch_{ava} to the gadget. The notation $\mathsf{ch}^{\mathcal{P}}_{\text{ava}}$ denotes the available chain, which is the confirmed Goldfish chain, in $\mathcal{P}$'s view at the *current* time specified by the clock. Similarly, $\mathsf{ch}^{\mathcal{P}}_{\text{acc}}$ denotes the accountable chain at the *current* time specified by the clock.

last $\leftarrow B_{\text{genesis}}$
for $t = 0, 1, 2, \ldots$ **do**
  at tT_{cp}
    ckpt $\leftarrow$ RunAccGadget($\mathsf{ch}^{\mathcal{P}}_{\text{ava}}$)
    if ckpt $\neq \perp$ **then**
      last $\leftarrow$ ckpt
    end if
  end at
  at t
    $\mathsf{ch}^{\mathcal{P}}_{\text{acc}} \leftarrow$ last
  end at
end for

an accountably-safe BFT protocol such as Streamlet, Tendermint, or HotStuff [11, 5, 50]. In our composition, we assume that the accountability gadget is instantiated with a BFT protocol that satisfies an accountable safety resilience of $n/3$ given n validators.

The full protocol proceeds in checkpointing iterations of T_{cp} rounds, where T_{cp} is determined by the gadget as a sufficiently large function of the network delay Δ and the confirmation parameter κ (cf. Algorithm 4). At each iteration, the accountability gadget requires the active participation of *all* awake validators as opposed to Goldfish that works with a subsampled set of validators. Once the iteration starts, *all* awake validators input the Goldfish ledger as determined by Goldfish in their view to the accountability gadget (line 4 of Algorithm 4). Subsequently, they execute the protocol specified by the gadget, after which each validator either outputs a checkpoint decision on a block, or an empty checkpoint. If the checkpoint is not empty, the fork-choice rule of Goldfish is modified to respect the checkpoint from the latest iteration in the validator's view (line 6 of Algorithm 4). Formally, the modified GHOST-Eph rule starts its iteration over the blocks at the last checkpointed block rather than genesis (line 2 of Algorithm 3). At any given round, the available ledger $\mathsf{ch}_{\mathrm{ava}}$ of the validator is defined as the (confirmed) output ledger of the Goldfish protocol with the modified GHOST-Eph fork-choice rule in the validator's view (line 24 of Algorithm 1). Similarly, at any round, the accountable, final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$ of the validator is defined as the prefix of the latest checkpointed block (line 10 of Algorithm 4).

The combination of Goldfish with accountability gadgets is shown to satisfy the ebb-and-flow property in Section 5.2.

4 Optimistic Fast Confirmations

The Goldfish protocol described so far has an advantage over protocols which use blocks as *votes* (*e.g.*, the longest chain protocols [34, 45, 28], GHOST [49]), namely reorg resilience; but its κ-slots deep confirmation rule falls behind many propose-and-vote style protocols that can achieve constant expected latency (*e.g.*, PBFT [10], Tendermint [5], HotStuff [50], Streamlet [11]). By introducing a fast confirmation rule and adding Δ more rounds to a slot, we can indeed achieve much faster confirmation *optimistically*, *i.e.*, under high participation and honest supermajority (Figure 3, Alg. 5). In particular, validators can now confirm honest blocks proposed at the tip of their canonical GHOST-Eph chains within the same slot under optimistic conditions. In contrast, the confirmation rule of Goldfish requires a block to become κ slots deep in the canonical chain. Fast confirmations do not compromise neither on liveness nor safety. At worst, when the optimistic conditions are not satisfied, one reverts back to the slow confirmation rule.

Slot structure Slots are now divided into 4Δ rounds. At each slot t, proposals are broadcast at the beginning of the slot, namely at round $4\Delta t$. Similarly, the buffer is merged with the proposed view and the votes are broadcast at round $4\Delta t + \Delta$. Validators

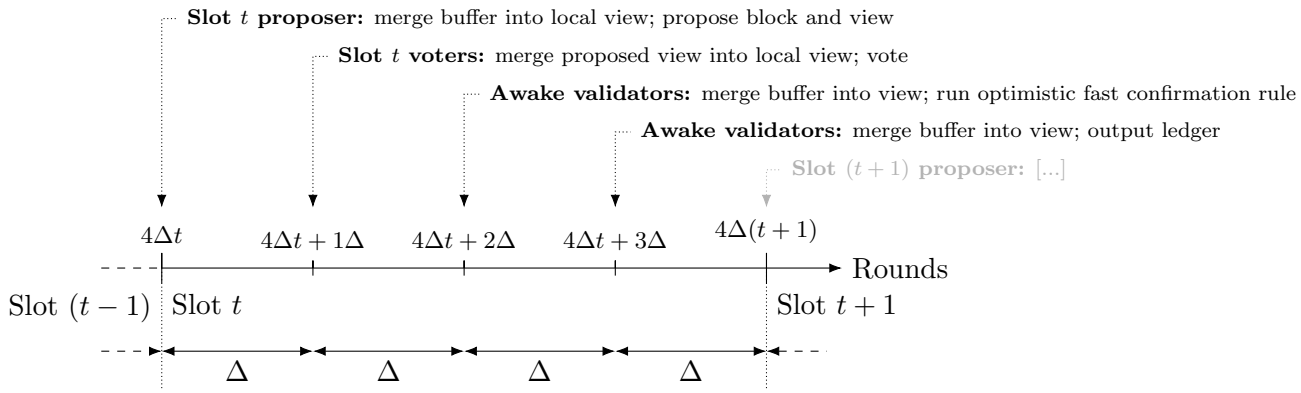

Figure 3: To enable optimistic fast confirmations, Goldfish's time slots are extended to 4Δ rounds (cf. Figure 1). At the start of each slot, the proposer merges buffered votes into their local view, determines their canonical chain, and proposes and broadcasts a block extending it, together with their local view. One-fourth into each slot, each voter merges the proposed view into their local view, determines their canonical chain, and casts a vote on the tip. Two-fourths into the slot, all awake validators merge their buffers into their local view and run the optimistic fast confirmation rule.Three-fourths into the slot, all awake validators again merge their buffers into their local view, and output a ledger according to GHOST-Eph.

Algorithm 5 Algorithm executed by a validator $\mathcal{P}$ with public key pk, private key sk, public VRF key vpk and the secret VRF key vsk using both the (optimistic) fast and slow confirmation rules. The variables and functions presented in the algorithm are defined in the caption of Alg 1. The variable $\mathsf{ch}_{\text{fast}}$ keeps track of the previously fast-confirmed blocks in the view of the validator, whereas the variable $\mathsf{ch}_{\text{slow}}$ keeps track of the previously k-slots deep blocks in the validator's canonical GHOST-Eph chain.

$\mathcal{B}, \mathcal{V} \leftarrow \emptyset, \emptyset$
for $t = 0, 1, 2, \ldots$ **do**
  at $4\Delta t$ ▷ PROPOSAL
    $\sigma_{t,L}, \pi_{t,L} \leftarrow \mathsf{VRF.Prove}_{\mathsf{vsk}}(t \,\|\, \text{'L'})$
    if $\sigma_{t,L} < \mathsf{thr}$ **then** ▷ $\mathcal{P}$ checks if it is eligible to propose of slot t
      $B \leftarrow \text{GHOST-Eph}(\mathcal{V} \cup \mathcal{B}, t-1)$ ▷ $\mathcal{P}$ *merges* its buffer and view
      $B_{\text{new}} \leftarrow \text{NewBlock}(B')$
      Broadcast $(\text{Proposal}, t, B_{\text{new}}, \mathcal{V} \cup \mathcal{B}, \sigma_{t,L}, \pi_{t,L})_{\mathcal{P}.\mathsf{sk}}$ ▷ $\mathcal{P}$ broadcasts its proposal
    end if
  end at
  at $4\Delta t + \Delta$ ▷ VOTE
    $\mathcal{B}_t \leftarrow \{(B^*, \mathcal{W}^*, \sigma^*) \mid (B^*, t^*, \mathcal{W}^*, \sigma^*) \in \mathcal{B} \wedge t^* = t\}$ ▷ $\mathcal{P}$ filters for proposals from slot t
    $(B^*, \mathcal{W}^*, \sigma^*) \leftarrow \arg\min_{(\sigma, B, \mathcal{W}) \in \mathcal{B}_t}(\sigma)$ ▷ $\mathcal{P}$ identifies the leader of slot t and its proposal
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{W}^* \cup \{B^*\}$ ▷ $\mathcal{P}$ *merges* its buffer with that of the leader
    $\sigma_{t,V}, \pi_{t,V} \leftarrow \mathsf{VRF.Prove}_{\mathsf{vsk}}(t \,\|\, \text{'V'})$
    if $\sigma_{t,V} < \mathsf{thr}$ **then** ▷ $\mathcal{P}$ checks if it is eligible to vote at slot t
      $B \leftarrow \text{GHOST-Eph}(\mathcal{V}, t-1)$
      Broadcast $(\text{Vote}, t, B, \sigma_{t,V}, \pi_{t,V})_{\mathcal{P}.\mathsf{sk}}$ ▷ $\mathcal{P}$ broadcasts its vote
    end if
  end at
  at $4\Delta t + 2\Delta$ ▷ FAST CONFIRMATION
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{B}$
    if $\exists \mathcal{V}' \subseteq \mathcal{V} : |\mathcal{V}'| \geq (\frac{3}{4} + \epsilon) n \frac{\mathsf{thr}}{2^{b(\lambda)}} \wedge$ ($\mathcal{V}'$ is a set of unique slot t votes on B^*) **then**
      $\mathsf{ch}_{\text{fast}} \leftarrow B^*$ ▷ Fast confirmation
    end if
  end at
  at $4\Delta t + 3\Delta$ ▷ CONFIRMATION
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{B}$ ▷ $\mathcal{P}$ *merges* its buffer and view
    $B \leftarrow \text{GHOST-Eph}(\mathcal{V}, t)$ ▷ $\mathcal{P}$ identifies the canonical GHOST-Eph chain for slot t
    $\mathsf{ch}_{\text{slow}} \leftarrow B$'s κ deep prefix in terms of slots ▷ Slow confirmation
    ▷ *Outputting the longer chains, the validator does not roll-back old fast confirmed blocks*
    $\mathsf{ch}^{\mathcal{P}} \leftarrow \arg\max_{\mathsf{ch} \in \{\mathsf{ch}_{\text{slow}}, \mathsf{ch}_{\text{fast}}\}}(|\mathsf{ch}|)$ ▷ Goldfish ledger confirmed for slot t
  end at
end for

add the observed votes to their buffers over the period $(4\Delta t+\Delta, 4\Delta t+2\Delta]$, and merge the buffers with their views at round $4\Delta t + 2\Delta$ to run fast confirmation. They subsequently continue to add votes to their buffers, and once again merge their buffers and views at round $4\Delta t + 3\Delta$, prior to outputting the (confirmed) Goldfish ledger. Thus, the new slot structure maps exactly to the old one, except that the middle portion between the rounds votes are broadcast and the Goldfish ledger is output is twice as long. The reason for the extra Δ rounds is to enable validators to fast confirm blocks at round $4\Delta t + 2\Delta$ and to guarantee that all honest validators see the fast confirming votes by the time their views are updated for the last time in the given slot at round $4\Delta t + 3\Delta$. This way, once a block is fast confirmed by an honest validator, it eventually enters the Goldfish ledger in the view of all honest validators.

Fast confirmation rule At round $4\Delta t + 2\Delta$, a validator merges its buffer and view, *i.e.* sets $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{B}$, then marks a block b proposed at the same slot t as fast confirmed if $\mathcal{V}$ contains a subset $\mathcal{V}'$ of slot t votes by distinct validators for b such that $|\mathcal{V}'| \geq (\frac{3}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$, *i.e.*, if more than 3/4 of the eligible voters of slot t voted for b (l. 24, Alg. 5). In this case, b and its prefix are output as the Goldfish ledger at round $4\Delta t + 3\Delta$ (or as the available ledger ch_{ava}, when Goldfish is composed with an accountability/finality gadget).

If no block is fast confirmed 2Δ rounds into a slot in the view of a validator, the validator uses the κ-slots-deep confirmation rule, *i.e.*, the slow confirmation rule, at round $4\Delta t+3\Delta$ to output the Goldfish ledger. However, validators do not roll back their ledgers: if the validator has previously fast confirmed a block within the last κ slots, it continues to output that block.

The security of the protocol with the fast confirmation rule is proven in Section 5.3.

Composition with the 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 slow confirmation rule 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 ch_{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 ch_{ava}, and does not affect the time for a block to enter the accountable, final prefix ledger ch_{acc}.

Reducing the latency of ch_{acc} under optimistic conditions is an important direction for future research. At a high level, an idea is to propose fast confirmed blocks as part of the finality/accountability gadget, and to broaden the fast confirmation rule so as to guarantee that an honest validator fast confirming a block at slot t implies that all honest validators fast confirm it as well at slot $t+1$, under optimistic assumptions on participation, regardless

of whether the leader is malicious. For instance, a simple idea is that, at slot t, a validator fast confirms b, *not necessarily proposed at slot* t, if its view $\mathcal{V}$, after merging the buffer at $4\Delta t + 2\Delta$, contains a subset $\mathcal{V}'$ of slot t votes by distinct validators for b *or descendants of* b (*i.e.*, for the subtree rooted at b) such that $|\mathcal{V}'| \geq (\frac{3}{4} + \epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$. It is then easy to reuse existing results such as Lemma 8, lightly modified, to obtain the aforementioned property. Showing that this property suffices for the desired ebb-and-flow property is future work.

5 Analysis

5.1 Analysis of Goldfish

In the following analysis, we consider a synchronous network in the sleepy model as described in Section 2. We also assume that the fraction of adversarial validators among the awake validators at any round is upper bounded by $\beta < \frac{1}{2} - \epsilon$, and that there are at least $n_0 = \Theta(\kappa)$ awake validators at any round (cf. Section 2).

Lemma 1. *Consider a slot t where all honest validators receive the same valid slot t proposal with the minimum VRF output less than* thr*, and the proposal was broadcast by an honest validator $\mathcal{P}_L$. Then, all honest validators eligible to vote at slot t vote for the proposed block b at slot t.*

Proof. By the *provability* property of the VRF (cf. Section 2), every message, *i.e.*, proposal or vote, broadcast by an honest validator passes the validation check (ll. 28 and 33, Alg. 1). As the honest validators do not send messages with a VRF output exceeding thr, we can thereafter assume that every message sent by an honest validator is added to the buffer by every other honest validator upon reception (ll. 29 and 34, Alg. 1).

By synchrony, any slot $t-1$ vote received by an honest validator by round $3\Delta(t-1)+2\Delta$ will be received by $\mathcal{P}_L$ by round $3\Delta(t-1)+3\Delta = 3\Delta t$. Let $\mathcal{W}$ denote the set of all consensus messages observed by $\mathcal{P}_L$ by round $3\Delta t$, *i.e.*, the union of its buffer and view at that round. By the protocol rules, it is included within the proposal message broadcast by $\mathcal{P}_L$ at round $3\Delta t$ (l. 8, Alg. 1). As the honest validators froze their views at slot $3\Delta(t-1)+2\Delta$ (*i.e.*, they have not merged their buffers into their views since), the view $\mathcal{V}$ of any honest validators at any round in $[3\Delta(t-1) + 2\Delta, 3\Delta t + \Delta]$ is a subset of $\mathcal{W}$: $\mathcal{V} \subseteq \mathcal{W}$.

At round $3\Delta t$, $\mathcal{P}_L$ identifies the tip of the canonical GHOST-Eph chain using the slot $t-1$ votes within $\mathcal{W}$ (l. 6, Alg. 1), and attaches its proposal b to this tip, *i.e.*, b extends GHOST-Eph$(\mathcal{W}, t-1)$. At round $3\Delta t + \Delta$, upon receiving $\mathcal{P}_L$'s proposal and identifying it as the proposal with the minimum VRF output, each honest validator merges its view $\mathcal{V}$ with $\mathcal{W}$: $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{W} \cup \{b\} = \mathcal{W} \cup \{b\}$ as $\mathcal{V} \subseteq \mathcal{W}$ (l. 14, Alg. 1). Since all honest validators now have the same view $\mathcal{W} \cup \{b\}$ and b's parent was output by GHOST-Eph$(\mathcal{W}, t-1)$, running the fork-choice rule using the slot $t-1$ votes within $\mathcal{W} \cup \{b\}$, each honest validator identifies b as the tip of the canonical GHOST-Eph chain, *i.e.*, $b \leftarrow$ GHOST-Eph$(\mathcal{W} \cup \{b\}, t-1)$ (l. 17, Alg. 1), and subsequently broadcasts a vote for b at round $3\Delta t + \Delta$ (l. 18, Alg. 1). □

Lemma 2. *Consider a slot t, where all honest validators vote for blocks in the subtree rooted at some block b. Suppose at every slot, there are more honest validators eligible to vote than adversarial validators. Then, at all slots $t' > t$, all honest validators continue to vote for blocks in the subtree rooted at b. Moreover, for all slots $t' \geq t$, block b and its prefix are in the canonical GHOST-Eph chain adopted by any honest validator (l. 23, Alg. 1).*

Proof. For contradiction, suppose there exists a slot after t, where an honest eligible validator does not vote for a block within the subtree rooted at b at that slot. Let $t' > t$ be the first such slot and $\mathcal{P}_L$ be one of the honest validators eligible to vote at slot t' that does not vote for a block within the subtree rooted at b. By definition of t', every honest validator eligible to vote at slot $t'-1$ must have sent a slot $t'-1$ vote for a block in the subtree rooted at b at round $3\Delta(t'-1)+\Delta$, and all of these votes must have been merged into the view $\mathcal{V}$ of $\mathcal{P}_L$ at round $3\Delta(t'-1)+2\Delta$ (l. 22, Alg. 1). As there are more honest validators eligible to vote at slot $t'-1$ than adversarial ones, at any round in $[3\Delta(t'-1)+2\Delta, 3\Delta t'+\Delta]$, the view of $\mathcal{P}_L$ contains more slot $t'-1$ votes by unique validators for the blocks in the subtree rooted at b than for any subtree conflicting with them. Consequently, upon invoking the GHOST-Eph fork-choice rule at round $3\Delta t'+\Delta$ (l. 17, Alg. 1), $\mathcal{P}_L$ observes more slot $t'-1$ votes in $\mathcal{V}$ for the subtrees rooted at b or its ancestors than for any subtree conflicting with them, at every iteration of the fork choice (l. 9, Alg. 2). Thus, $\mathcal{P}_L$ obtains a block that extends b, and votes for it (l. 18, Alg. 1), which is a contradiction. This implies that at all slots $s \geq t$, all honest validators eligible to vote at slot s broadcast a slot s vote for a block within the subtree rooted at b.

Finally, since at all slots $s \geq t$, each eligible honest validator votes for a block within the subtree rooted at b, upon invoking the GHOST-Eph fork-choice rule at round $3\Delta s + 2\Delta$ (l. 23, Alg. 1), all honest validators observe more slot s votes for the subtrees rooted at b or its ancestors than any other block conflicting with b, at every iteration of the fork choice (l. 9, Alg. 2). Thus, for all slots $t' \geq t$, block b and its prefix are in the canonical GHOST-Eph chain adopted by any honest validator (l. 23, Alg. 1). □

Lemma 3. *Suppose the time horizon T_{hor} of the protocol execution satisfies $T_{\mathsf{hor}} = \mathrm{poly}(\kappa)$. Then, with overwhelming probability, every slot has more honest validators eligible to vote than adversarial validators.[5] Similarly, with overwhelming probability, all slot intervals of length κ have at least one slot where the proposal with the minimum VRF output has output value less than thr, is the same in the view of all honest validators, and was broadcast by an honest validator.*

Proof. By the *pseudorandomness* property of the VRF, for any given slot s and validator

[5] For concreteness, the Ethereum validator set has over $400,000$ validators as of 5-Sept-2022. Suppose we subsample with $\frac{\mathsf{thr}}{2^{b(\kappa)}} = \frac{1}{32}$, *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.

$\mathcal{P}_i$, with overwhelming probability,

$$\begin{aligned}
\Pr[F_{\mathsf{vsk}_i}(s \,\|\, \text{`V'}) < \mathsf{thr}] &\in \left[\frac{\mathsf{thr}}{2^{b(\lambda)}} - \mathrm{negl}(\lambda), \frac{\mathsf{thr}}{2^{b(\lambda)}} + \mathrm{negl}(\lambda)\right] \\
\Pr[F_{\mathsf{vsk}_i}(s \,\|\, \text{`L'}) < \mathsf{thr}] &\in \left[\frac{\mathsf{thr}}{2^{b(\lambda)}} - \mathrm{negl}(\lambda), \frac{\mathsf{thr}}{2^{b(\lambda)}} + \mathrm{negl}(\lambda)\right] \\
\forall i \neq j: \quad \Pr[F_{\mathsf{vsk}_i}(s \,\|\, \text{`L'}) < F_{\mathsf{vsk}_j}(s \,\|\, \text{`L'})] &= \left[\frac{1}{2} - \mathrm{negl}(\lambda), \frac{1}{2} + \mathrm{negl}(\lambda)\right].
\end{aligned}$$

Let $n_t \geq n_0 = \Theta(\kappa)$ denote the number of awake validators at round $3\Delta t + \Delta$. By the *uniqueness* property of the VRF, with overwhelming probability, each validator obtains a single verifiable VRF output for voting at any given slot t (l. 15, Alg. 1). Since $\beta < \frac{1}{2} - \epsilon$, with overwhelming probability, the expected number of honest voters with the unique VRF output below thr is larger than $(\frac{1}{2} + \epsilon) n_t (\frac{\mathsf{thr}}{2^{b(\lambda)}} - \mathrm{negl}(\lambda))$, whereas the expected number of adversarial voters with the unique VRF output below thr is smaller than $(\frac{1}{2} - \epsilon) n_t (\frac{\mathsf{thr}}{2^{b(\lambda)}} + \mathrm{negl}(\lambda))$. Let H_t and A_t denote the number of awake honest and adversarial voters, respectively, with VRF output below thr at round $3\Delta t + \Delta$. By a Chernoff bound,

$$\begin{aligned}
\Pr\left[H_t < \frac{1}{2} n_t \frac{\mathsf{thr}}{2^{b(\lambda)}}\right] &\leq e^{-\frac{\epsilon^2}{1+2\epsilon} n_t \frac{\mathsf{thr}}{2^{b(\lambda)}} + \mathrm{negl}(\lambda)} \\
\Pr\left[A_t > \frac{1}{2} n_t \frac{\mathsf{thr}}{2^{b(\lambda)}}\right] &\leq e^{-\frac{\epsilon^2}{1+3\epsilon} n_t \frac{\mathsf{thr}}{2^{b(\lambda)}} + \mathrm{negl}(\lambda)}.
\end{aligned}$$

Thus, at any given slot t, the number of awake honest voters with VRF output below thr exceeds that of such adversarial voters except with probability $2\exp\left(-\frac{\epsilon^2}{1+3\epsilon} n_0 \frac{\mathsf{thr}}{2^{b(\lambda)}} + \mathrm{negl}(\lambda)\right)$. By a union bound, every slot s has more honest voters $\mathcal{P}_i$ with VRF outputs $F_{\mathsf{vsk}_i}(s \,\|\, \text{`V'}) < \mathsf{thr}$ than such adversarial voters (and more than $\frac{1}{2} n_0 \frac{\mathsf{thr}}{2^{b(\lambda)}}$ such honest voters), except with probability

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

since $n_0 = \Theta(\kappa)$ and $T_{\mathsf{hor}} = \Theta(\kappa)$. By the same reasoning, with overwhelming probability, every slot s has more honest validators $\mathcal{P}_i$ with VRF outputs $F_{\mathsf{vsk}_i}(s \,\|\, \text{`L'}) < \mathsf{thr}$, *i.e.*, potential honest leaders, than potential adversarial leaders.

Finally, for any given slot s, each validator awake at round $3\Delta s$ obtains the minimum VRF output at that round with equal probability up to terms negligible in λ.[6] Consequently, among the awake validators $\mathcal{P}_i$ with VRF outputs $F_{\mathsf{vsk}_i}(s \,\|\, \text{`L'}) < \mathsf{thr}$ (of which the honest validators constitute at least $\frac{1}{2} n_0 \frac{\mathsf{thr}}{2^{b(\lambda)}}$ with overwhelming probability), an honest validator has the minimum VRF output for leader election with probability larger than $1/2 - \mathrm{negl}(\kappa) - \mathrm{negl}(\lambda)$. Then, taking a fixed $t \geq \kappa$, the probability that no awake honest validator has the minimum VRF output for leader election during the slots $[t-\kappa, t]$ is upper

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

bounded by $2^{-\kappa} + \text{negl}(\kappa) + \text{negl}(\lambda)$. Union bounding over all T_{hor} many such intervals, we find that with overwhelming probability all slot intervals of length κ have at least one slot where the proposal with the minimum VRF output is produced by an awake honest validator. In this case, the proposal message with the minimum VRF output less than thr becomes the same for all honest validators. □

Theorem 1 (Reorg resilience)**.** *Suppose the validator that has the proposal with the minimum VRF output within a slot is an honest validator, and the VRF output is less than* thr*. Then, the block proposed by that validator enters and stays in the canonical GHOST-Eph chain adopted by any honest validator at all future slots.*

Proof. Consider a slot t where all honest validators receive the same valid slot t proposal with the minimum VRF output less than thr, and the proposal was broadcast by an honest validator. By Lemma 3, with overwhelming probability, every slot s has more honest validators eligible to vote than adversarial validators. Then, by Lemmas 1 and 2, at all slots $t' \geq t$, the proposed block b and its prefix are in the canonical GHOST-Eph chain adopted by any honest validator (l. 23, Alg. 1). □

Theorem 2 (Security)**.** *Goldfish satisfies T_{conf}-security for $T_{\mathsf{conf}} = \kappa$ slots with overwhelming probability.*

Proof. By Lemma 3, with overwhelming probability, all slot intervals of length κ have at least one slot where the proposal with the minimum VRF output less than thr is the same for all honest validators, and was broadcast by an honest validator. By the same lemma, with overwhelming probability, every slot s has more honest validators eligible to vote than adversarial validators. Then, by Theorem 1, with overwhelming probability, all slot intervals of length κ have a slot, where a block proposed by an honest validator enters and stays in the canonical GHOST-Eph chain adopted by any honest validator (l. 23, Alg. 1) at all future slots. Hence, any transaction input by the environment at some slot t is included in an honest proposal within the interval $[t, t + \kappa]$, which subsequently enters and stays in the canonical GHOST-Eph chain adopted by any honest validator at all future slots. Thus, the protocol satisfies liveness with parameter $T_{\mathsf{conf}} = \kappa$ slots.

We next prove safety by contradiction. Suppose there exist two honest validators $\mathcal{P}_1$ and $\mathcal{P}_2$, and two slots t_1 and $t_2 \geq t_1$ such that $\mathsf{ch}_{t_1}^{\mathcal{P}_1} \not\preceq \mathsf{ch}_{t_2}^{\mathcal{P}_2}$ and $\mathsf{ch}_{t_2}^{\mathcal{P}_2} \not\preceq \mathsf{ch}_{t_1}^{\mathcal{P}_1}$. By the observations above, with overwhelming probability, there exists a block b proposed by an honest validator at some slot $t_b \in [t_1 - \kappa, t_1]$ such that b is in the canonical GHOST-Eph chain adopted by any honest validator at all slots $t \geq t_b$. As $t_b \geq t_1 - \kappa$, $\mathsf{ch}_{t_1}^{\mathcal{P}_1} \preceq b$ by the confirmation rule of $\mathsf{Goldfish}$ (l. 24, Alg. 1). Similarly, if $t_b \geq t_2 - \kappa$, $\mathsf{ch}_{t_2}^{\mathcal{P}_2} \preceq b$, otherwise, $b \preceq \mathsf{ch}_{t_2}^{\mathcal{P}_2}$. However, both cases imply that $\mathsf{ch}_{t_1}^{\mathcal{P}_1} \preceq \mathsf{ch}_{t_2}^{\mathcal{P}_2}$ or $\mathsf{ch}_{t_2}^{\mathcal{P}_2} \preceq \mathsf{ch}_{t_1}^{\mathcal{P}_1}$, a contradiction. □

5.2 Analysis of Goldfish with Accountability Gadgets

In the following analysis, unless stated otherwise, we consider a *partially synchronous* network in the sleepy model as described in Section 2. Subsequent proofs extensively refer

to the details of the accountability gadgets introduced in [43]. The internal mechanisms of these gadgets are described in [43, Section 4]. To distinguish the votes cast by validators as part of the accountability gadget protocol from those broadcast within Goldfish, we will refer to the former as gadget votes. Similarly, to distinguish the leader of an accountability gadget iteration from the slot leader of Goldfish, we will refer to the former as the iteration leader. We refer the reader to [43] for the accountability gadget specific definitions of the timeout parameter T_{to} and the delay T_{bft}. We set T_{cp}, the time gap between iterations of the accountability gadget, to be $3\Delta(\kappa+1)+T_{\text{to}}+T_{\mathsf{bft}}$ (this is necessary for proving the ebb-and-flow property as will be evident in the following proofs).

Our goal in this section is to prove the ebb-and-flow property for Goldfish combined with accountability gadgets (Figure 2). Towards this goal, we first show that ch_{ava} remains secure under synchrony in the sleepy model.

Lemma 4 (Safety and liveness of ch_{ava} under synchrony)**.** *Under the synchronous network in the sleepy model, the available ledger* ch_{ava} *provides* $1/2$*-safety and* $1/2$*-liveness (at all times) with overwhelming probability.*

Proof. Proof works by induction on the checkpointed blocks b_n, ordered by their iterations $n \in \mathbb{N}$. (Note that in practice there might not be checkpointed blocks in this setting, except those checkpointed with adversarial help.)

Induction Hypothesis: Suppose b_n was observed to be checkpointed by an honest validator for the first time at round t_n. Then, every honest validator awake at round t_n-1 observes b_n as part of its canonical GHOST-Eph chain at round t_n-1. Moreover, b_n stays as part of the canonical GHOST-Eph chains of honest validators at future rounds.

Base Case: We know from Theorem 2 that the Goldfish protocol is safe and live with latency $3\Delta\kappa$ with overwhelming probability under the original GHOST-Eph fork-choice rule of Goldfish (Alg. 2) when $\beta < \frac{1}{2}$. As $T_{\mathsf{cp}} > 3\Delta(\kappa+1)$, it holds that $t_1 \geq 3\Delta\kappa$. Then, for any slot s such that $3\Delta s + 2\Delta < t_1$, if a block was proposed at a slot less than or equal to $s-\kappa$, and it is observed to be on the canonical GHOST-Eph chain at round $3\Delta s+2\Delta$ by an honest validator, then that block stays on the canonical GHOST-Eph chains held by all honest validators at subsequent rounds until at least round t_1. Since there are over $\frac{2n}{3} > \frac{n}{2}$ accepting gadget votes for b_1 in the view of an honest validator at round $t_1 > 3\Delta s + 2\Delta$, there must be at least one honest validator that sent an accepting gadget vote for b_1. As honest validators send accepting gadget votes, among the blocks input to the gadget, only for those that are at least κ slots deep in their canonical GHOST-Eph chains, b_1 must have been observed to be at least κ slots deep on the canonical GHOST-Eph chain of at least one honest validator at some round $3\Delta s + 2\Delta$, where $3\Delta s + 2\Delta < t_1$. Consequently, every honest validator awake at a round $t_1 - 1$ observes b_1 as part of its canonical GHOST-Eph chain at round $t_n - 1$.

Finally, as $T_{\mathsf{cp}} > 3\Delta(\kappa+1)$ T_{to}, we know that by the time the first honest validator sends a gadget vote for any b_n, $n > 1$, it must have observed b_1 as checkpointed. As honest validators do not send gadget votes for subsequent blocks that conflict with earlier checkpoints, and since no block can become checkpointed without receiving a gadget vote

from at least one honest validator (as $\beta < 1/2$), b_1 must be in the prefix of b_n. Hence, once b_1 is checkpointed, it stays as part of the canonical GHOST-Eph chains of all honest validators at all future rounds.

Inductive Step: By the induction hypothesis, all honest validators awake at round $t_{n-1} - 1$ observe b_{n-1} as part of their canonical GHOST-Eph chains at round $t_{n-1} - 1$, and b_{n-1} stays in the canonical chains in subsequent slots. This implies that the GHOST-Eph fork-choice rule goes through the same block b_{n-1} for all honest validators at all slots larger than or equal to $t_{n-1} - 1$ (cf. l. 2, Alg. 3). Hence, we can treat b_{n-1} as a *new genesis block* and use the same argument presented in the base case to conclude the inductive step.

Finally, the induction hypothesis implies that new checkpoints can only appear in the common prefix of the honest validators' canonical GHOST-Eph chains, and thus, do not affect the security of the Goldfish protocol as shown by Theorem 2. □

We next demonstrate the liveness of ch_{acc} after $\max(\mathsf{GST}, \mathsf{GAT})$. Recall that the accountability gadget is instantiated with a BFT protocol denoted by Π_{bft} that has an accountable safety resilience of $n/3$.

Proposition 1 (Proposition 2 of [43])**.** Π_{bft} *satisfies* $n/3$*-liveness after* $\max(\mathsf{GST}, \mathsf{GAT})$ *with an upper bound* $T_{\mathsf{bft}} < \infty$ *on the transaction confirmation time.*

Proposition 2 (Proposition 3 of [43])**.** *Suppose a block from iteration* c *was checkpointed in the view of an honest validator at round* t*. Then, every honest validator enters iteration* $c + 1$ *by round* $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$.

Proof. Suppose a block from iteration c was checkpointed in the view of an honest validator $\mathcal{P}$ at round t. Then, there are over $2n/3$ gadget votes that accept the block from iteration c on $\mathsf{LOG}^t_{\text{bft},\mathcal{P}}$, the output ledger of Π_{bft} in the view of validator $\mathcal{P}$ at round t. Note that all gadget votes and BFT protocol messages observed by $\mathcal{P}$ by round t are delivered to all other honest validators by round $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$. Hence, via accountable safety of Π_{bft}, for any honest validator $\mathcal{P}'$, the ledger $\mathsf{LOG}_{\text{bft}}$ in $\mathcal{P}$'s view at round t is the same as or a prefix of the ledger observed by $\mathcal{P}'$ at slot $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$. Thus, for any honest validator $\mathcal{P}'$, there are over $2n/3$ gadget votes at round $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$ on $\mathsf{LOG}_{\text{bft}}$ accepting the same block from iteration c. This implies that every honest validator enters iteration $c + 1$ by round $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$. □

Lemma 5 (Liveness of ch_{acc}, analogue of Theorem 4 of [43])**.** *Suppose* ch_{ava} *is secure (safe and live) after some round* $T_{\mathsf{heal}} \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta + T_{\text{cp}}$*, and the number of adversarial validators is less than* $n/3$ *at all rounds. Then,* ch_{acc} *satisfies* $n/3$*-liveness after round* T_{heal} *with transaction confirmation time* $T_{\mathsf{conf}} = \Theta(\kappa^2)$ *with overwhelming probability.*

Proof. Via Proposition 1, we know that $\mathsf{LOG}_{\text{bft}}$ satisfies liveness 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 was checkpointed in the view of some honest validator before $\max(\mathsf{GAT}, \mathsf{GST})$. (Let $c' = 0$ and the block b the genesis block

if there does not exist such an iteration.) By Proposition 2, all honest validators enter iteration $c'+1$ by round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta$. Then, all honest validators observe a block proposed for iteration $c'+1$ by round $\max(\mathsf{GAT}, \mathsf{GST})+\Delta+T_{\text{cp}}$. Thus, as the validators skip past iterations where no block was checkpointed, entrance times of the honest validators to subsequent iterations become synchronized by round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\text{cp}}$: If an honest validator enters an iteration $c > c'$ at round $t \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\text{cp}}$, every honest validator enters iteration c' by round $t + \Delta$.

Suppose iteration $c > c'$ has an honest iteration leader $\mathrm{L}^{(c)}$, which sends a proposal $\hat{b}_c$ at round $t > T_{\mathsf{heal}}+T_{\text{cp}}$. Note that $\hat{b}_c$ is received by every honest validator by round $t+\Delta$. Since the entrance times of the validators are synchronized by $T_{\mathsf{heal}} \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta + T_{\text{cp}}$, every honest validator sends a gadget vote by slot $t + \Delta$. Now, as ch_{ava} is secure after slot T_{heal} by assumption (which will later be proven by Lemma 7), $\hat{b}_c$ is on all of the checkpoint-respecting canonical GHOST-Eph chains held by the honest validators. Moreover, as we will argue in the paragraph below, $\hat{b}_c$ extends all of the checkpoints seen by the honest validators by round $t + \Delta$. (Honest validators see the same checkpoints from iterations preceding c due to synchrony.) Consequently, at iteration c, every honest validator sends a gadget vote accepting $\hat{b}_c$ by round $t + \Delta$, all of which appear within $\mathsf{LOG}_{\text{bft}}$ in the view of every honest validator by round $t+\Delta+T_{\mathsf{bft}}$. Thus, $\hat{b}_c$ becomes checkpointed in the view of every honest validator by round $t + \Delta + T_{\mathsf{bft}}$. (Here, we assume that T_{to} was chosen large enough for $T_{\text{to}} > \Delta + T_{\mathsf{bft}}$ to hold.)

Note that $\mathrm{L}^{(c)}$ waits for $T_{\text{cp}} \geq 3\Delta\kappa$ slots before broadcasting $\hat{b}_c$ after observing the last checkpointed block before iteration c. Since $t - T_{\text{cp}} > T_{\mathsf{heal}}$, by the safety and liveness of ch_{ava} after T (Lemma 7), the checkpoint-respecting canonical GHOST-Eph chain of $\mathrm{L}^{(c)}$ at round t contains at least one honest block between $\hat{b}_c$ and the last checkpointed block on it from before iteration c. (As a corollary, $\mathrm{L}^{(c)}$ extends all of the previous checkpoints seen by itself and all other honest validators.) This implies that the prefix of $\hat{b}_c$ contains at least one fresh honest block that enters ch_{acc} by round $t + \Delta + T_{\mathsf{bft}}$.

Next, we show that an adversarial leader cannot make an iteration last longer than $\Delta+T_{\text{to}}+T_{\mathsf{bft}}$ for any honest validator. Indeed, if an honest validator $\mathcal{P}$ enters an iteration c at round t, by round $t+\Delta+T_{\text{to}}$, either it sees a block become checkpointed for iteration c, or every honest validator sends a gadget vote that rejects $\hat{b}_c$ or, in the absence of an iteration proposal, rejects checkpointing any block for the iteration. In the first case, every honest validator sees a block checkpointed for iteration c by round at most $t+2\Delta+T_{\text{to}}$. In the second case, rejecting gadget votes from over $2n/3 > n/3$ validators appear in $\mathsf{LOG}_{\text{bft}}$ in the view of every honest validator by round at most $t + \Delta + T_{\text{to}} + T_{\mathsf{bft}}$. Hence, a new checkpoint, potentially $\perp$, is outputted in the view of every honest validator by round $t + \Delta + T_{\text{to}} + T_{\mathsf{bft}}$.

Finally, we observe that except with probability $(1/3)^{\kappa}$, there exists a checkpointing iteration with an honest leader within κ consecutive iterations. Since an iteration lasts at most $\max(\Delta + T_{\text{to}} + T_{\mathsf{bft}}, \Delta + T_{\mathsf{bft}} + T_{\text{cp}}) \leq \Delta + T_{\text{to}} + T_{\mathsf{bft}} + T_{\text{cp}} = \Theta(\kappa)$ rounds, and a new checkpoint containing a fresh honest block in its prefix appears when an iteration has an honest leader, any transaction received by an honest validator at round t appears within

ch_{acc} in the view of every honest validator by round at most $t+\kappa(\Delta+T_{\text{to}}+T_{\mathsf{bft}}+T_{\text{cp}})$, with overwhelming probability. 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 with overwhelming probability, ch_{acc} satisfies liveness after T_{heal} with a transaction confirmation time $T_{\mathsf{conf}} = \Theta(\kappa^2)$. □

The latency expression $T_{\mathsf{conf}} = \Theta(\kappa^2)$ stated in Lemma 5 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 $T_{\mathsf{cp}} = \Theta(\kappa)$ 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 leader except with probability $\text{negl}(\kappa)$. The accountability gadget protocol asks honest validators to wait for $T_{\mathsf{cp}} = \Theta(\kappa)$ slots in between iterations to ensure the security of the protocol, reasons for which will be evident in the proof of Lemma 7.

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 $T_{\mathsf{cp}} = \Theta(\kappa)$ slots in between the accountability gadget iterations. Here, waiting for T_{cp} 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.

Lemma 5 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_{cp}:

Proposition 3 (Gap property, analogue of Proposition 4 of [43])**.** *Given any round interval of size T_{cp}, no more than a single block can be checkpointed in the interval.*

Proof of Proposition 3 follows from the fact that upon observing a new checkpoint that is not $\perp$ for an iteration, honest validators wait for T_{cp} rounds before sending gadget votes for the proposal of the next iteration.

As in [43] and [47], we state that a block checkpointed at iteration c and round $t > \max(\mathsf{GST}, \mathsf{GAT})$ in the view of an honest validator $\mathcal{P}$ is T_{rec}-recent if it has been part of the checkpoint-respecting canonical GHOST-Eph chain of some honest validator $\mathcal{P}'$ at some round within $[t - T_{\text{rec}}, t]$. Then, we can express the recency property as follows:

Lemma 6 (Recency property, analogue of Lemma 1 of [43])**.** *Every checkpointed block proposed after* $\max(\mathsf{GST}, \mathsf{GAT})$ *is T_{rec}-recent for $T_{\text{rec}} = \Delta + T_{\text{to}} + T_{\mathsf{bft}}$.*

Proof. We have seen in the proof of Lemma 5 that if a block b proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is checkpointed in the view of an honest validator at round t, it should have been proposed after round $t-(\Delta+T_{\text{to}}+T_{\mathsf{bft}})$. Thus, over $2n/3$ validators must have sent accepting gadget

votes for it by round t. Let $\mathcal{P}$ denote such an honest validator. Note that $\mathcal{P}$ would vote for b only after it sees the proposal for iteration c, *i.e.*, after round $t - T_{\text{rec}} = t - (\Delta + T_{\text{to}} + T_{\mathsf{bft}})$. Hence, b should have been on the checkpoint-respecting canonical GHOST-Eph chain of validator $\mathcal{P}$ at some round within $[t - T_{\text{rec}}, t]$. This concludes the proof that every checkpoint block proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is T_{rec}-recent. □

Lemma 7 (Healing property, analogue of Theorem 5 of [43])**.** *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 ch_{ava} regains safety and liveness (*i.e.*, is secure) after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\mathsf{cp}} + 3\Delta(\kappa + 1)$.*

Proof. By [43, Theorem 3], ch_{acc} provides accountable safety with resilience $n/3$ except with probability $\text{negl}(\lambda)$ under partial synchrony in the sleepy model. Hence, as the number of adversarial validators is less than $n/3$ at all rounds, with overwhelming probability, no two checkpoints observed by honest validators can conflict with each other.

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.) By Proposition 2, all honest validators enter iteration $c + 1$ by round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta$. Then, all honest validators observe a block proposed for iteration $c + 1$ by round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}}$. Thus, entrance time of the honest validators to subsequent iterations have become synchronized by round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}}$: If an honest validator enters an iteration $c' > c$ at round $t \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}}$, every honest validator enters iteration c' by round $t + \Delta$. Similarly, if a block from iteration c' is first checkpointed in the view of an honest validator at some round after $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}}$, then it is checkpointed in the view of all honest validators within Δ rounds.

Let c' be the first iteration such that the first honest validator to enter iteration c' enters it after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}}$ (*e.g.*, at some round t such that $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}} < t < \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\mathsf{cp}}$). Then, all honest validators enter iteration c' and agree on the last checkpointed block within Δ rounds. Subsequently, the honest validators wait for $T_{\mathsf{cp}} > 3\Delta\kappa + T_{\text{rec}}$ rounds before casting any gadget vote for a proposal of iteration c', during which no block can be checkpointed for iteration c' (Proposition 3, gap property).

After round t (which is larger than GST), 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 $[t + \Delta, t + T_{\mathsf{cp}}]$, by the lack of conflicting checkpoints, the GHOST-Eph fork-choice rule starts at the same last checkpointed block for all honest validators during the interval (cf. l. 2, Alg. 3). Hence, Theorem 1 as well as Lemma 3 hold for the slots during the interval $[t, t + T_{\mathsf{cp}}]$. Then, with overwhelming probability, the slot interval of length κ starting at round t contains a slot t', where a block b' proposed by an honest validator enters and stays in the canonical GHOST-Eph chain adopted by any honest validator until at least a new block is checkpointed in the view of an honest validator.

(Note that if the new checkpoint conflicts with b', it might not be in the canonical GHOST-Eph chains of honest validators later on. However, we will next show that b' indeed stays in the canonical chains.)

By Lemma 6 (recency property), the next block checkpointed in the view of an honest validator (which happens earliest at some iteration c'' and round t'' such that $t'' \geq t + T_{\mathsf{cp}} > t + 3\Delta\kappa + T_{\text{rec}}$ by Proposition 3, the gap property) must have been part of the checkpoint-respecting canonical GHOST-Eph chain of some honest validator $\mathcal{P}$ at some round within $[t'' - T_{\text{rec}}, t'']$, where $t'' - T_{\text{rec}} \geq t + 3\Delta\kappa \geq t'$. Hence, the new checkpoint must include b' in its prefix, implying that b' stays in the canonical GHOST-Eph chain adopted by any honest validator indefinitely. Thus, after slot t', the canonical GHOST-Eph chain adopted by any honest validator goes through block b'.

Finally, we extend the above argument to future checkpoints via an inductive argument. Let b_n be the sequence of checkpoints, ordered by their iteration numbers $c_n \geq c'$, starting at $c_1 = c''$. The rounds t_n, at which the blocks b_n are first checkpointed in the view of an honest validator satisfy the relation $t_{n+1} \geq t_n + T_{\mathsf{cp}}$ and $t_1 = t''$. Via an inductive argument and the reasoning above, in each interval $[t_n, t_{n+1} - T_{\text{rec}}]$, there exists a slot with an honest proposal b that enters and stays in the canonical GHOST-Eph chain of every honest validator at all future rounds, with overwhelming probability. Hence, for a sufficiently large confirmation time $\kappa \geq T_{\mathsf{cp}}$, these honest blocks imply the safety and liveness of the Goldfish protocol after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\mathsf{cp}} + 3\Delta(\kappa + 1)$. □

The reorg resilience property holds for the honest blocks proposed during the intervals $[t_n, t_{n+1} - T_{\text{rec}}]$ as all honest validators agree on the latest checkpoint during this interval.

Theorem 3 (Ebb-and-flow property)**.** *Goldfish combined with accountability gadgets satisfies the ebb-and-flow property given by Definition 3.*

Proof. 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 [43, Theorem 3], ch_{acc} provides accountable safety with resilience $n/3$ except with probability $\text{negl}(\lambda)$ under partial synchrony in the sleepy model. By Lemma 7, under the same model, the available ledger ch_{acc} regains safety and liveness after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + 2T_{\mathsf{cp}} + 3\Delta(\kappa + 1)$. Using this fact and Lemma 5, we can state that ch_{acc} satisfies liveness after round $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathsf{cp}} + 3\Delta(\kappa + 1)$ with transaction confirmation time $T_{\mathsf{conf}} = \Theta(\kappa^2)$ with overwhelming probability.

Finally, the property **P2** follows from Lemma 4, and **Prefix** follows by construction of the ledgers ch_{acc} and ch_{ava}. This concludes the proof of the ebb-and-flow property. □

5.3 Analysis of the Optimistic Fast Confirmation Rule

In the following analysis, we consider a synchronous network in the sleepy model as described in Section 2. We also assume that the fraction of adversarial validators among the awake validators at any round is upper bounded by $\beta < \frac{1}{2} - \epsilon$. Recall that the total number of validators is n (cf. Section 2).

Proposition 4. *Suppose* $T_{\mathsf{hor}} = \mathrm{poly}(\kappa)$. *Then, with overwhelming probability, every slot has at most* $(1+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ *validators that are eligible to vote.*

Proof follows from a Chernoff bound.

Lemma 8. *If an honest validator* $\mathcal{P}$ *fast confirms a block* b *at slot* t*, all honest validators vote for a descendant of* b *at slot* $t+1$*, unless the view of some honest validator, when voting at slot* $t+1$*, contains votes by at least* $\frac{1}{2}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ *equivocating validators eligible to vote at slot* t *(*i.e.*, eligible slot* t *validators that broadcast slot* t *votes for two different blocks). When the latter is not the case,* b *is in the canonical chain of every honest validator at all slots after slot* t*.*

Proof. The validator $\mathcal{P}$ broadcasts b and over $(\frac{3}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes for it at round $4\Delta t + 2\Delta$. Upon merging its buffer with its view, each honest validator observes these votes in its view at round $4\Delta t + 3\Delta$, and continues to observe them when voting at slot $t+1$. If the view of an honest validator, when voting at slot $t+1$, contains votes by less than $\frac{1}{2}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ equivocating validators eligible at slot t, then this view contains more than $(\frac{1}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes for b by non-equivocating validators. Moreover, it contains at most $\frac{1}{4}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes by non-equivocating validators for the subtree rooted at any block conflicting with b. This is due to Proposition 4, which implies that there can be at most $\frac{1}{4}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ eligible validators that are different from the validators that voted for b. Therefore, at round $4\Delta(t+1)+\Delta$, the view of any honest validator contains more than $(\frac{3}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ slot t votes by unique validators for block b, and at most $\frac{3}{4}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ slot t votes by unique validators for any subtree rooted at a block conflicting with b. As a result, the GHOST-Eph fork-choice rule run using the slot t votes at round $4\Delta(t+1)+\Delta$ outputs b or a descendant of b for all honest validators, implying that they all vote for b or one of its descendants at slot $t+1$. Finally, Lemmas 2 and 3 together imply the desired result. □

Theorem 4. *Goldfish with fast confirmations satisfies safety with overwhelming probability if* $\beta < \frac{1}{2} - \epsilon$.

Proof. If an honest validator fast confirms a block b at slot t, and no honest view, when voting at slot $t+1$, contains votes by $\frac{1}{2}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ or more equivocating slot t validators, then b is in the canonical GHOST-Eph chain of every honest validator at all slots larger than t by Lemma 8. Therefore, b is in the κ-slots-deep prefix of the canonical GHOST-Eph chains of all honest validators at slot $t+\kappa$. As $\beta < \frac{1}{2} - \epsilon$, with overwhelming probability, there are less than $\frac{1}{2}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ adversarial validators eligible to vote at any slot, *i.e.*, less than $\frac{1}{2}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ equivocating validators at any slot since equivocation is a protocol violation. Hence, any block fast confirmed by an honest validator will eventually become confirmed by the (slow) confirmation rule of all honest validators. In this light, as $\beta < \frac{1}{2} - \epsilon$, Theorem 2 implies the safety of the protocol. □

We automatically get liveness from the liveness of the slow, κ-slots-deep confirmations if $\beta < \frac{1}{2} - \epsilon$, since fast confirmation is not needed for a block to be confirmed. Nevertheless,

we prove that an honest proposal is immediately fast confirmed within the same slot by the awake honest validators, if there are over $(\frac{3}{4}+2\epsilon)n$ awake, honest validators at the given slot, implying the liveness of fast-confirmations under optimistic conditions.

Theorem 5. *Goldfish with fast confirmations satisfies liveness with $T_{\mathsf{conf}} = \Theta(\kappa)$ with overwhelming probability, if $\beta < \frac{1}{2} - \epsilon$.*

Consider a slot t, where there are $(\frac{3}{4}+2\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ eligible validators that are honest and stay awake throughout the slot. Suppose they receive the same valid slot t proposal with the minimum VRF output less than thr, broadcast by an honest validator. Then, at any such slot t, all awake honest validators fast confirm the proposal at slot t.

Proof. Proof of liveness follows from Theorems 4 and 2.

For the second part of the proof, by Lemma 1, all eligible and awake honest validators vote for the honest proposal. Then, any honest view at round $4\Delta t + 2\Delta$ contains at least $(\frac{3}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes (by Chernoff bound) for the proposal, implying that all awake honest validators fast confirm b at slot t. □

6 Comparison With Current PoS Ethereum

Goldfish is a simple, provably secure, dynamically available and reorg resilient protocol which minimally differs from the LMD GHOST component of the current implementation of the Gasper protocol, responsible for the consensus of Ethereum's beacon chain. Gasper has so far defied formal security analysis even in the simpler, full participation setting, not least because of its complexity. Moreover, it is not reorg resilient even in that setting, and it is not dynamically available. We first analyze these shortcomings and their origins in various components of the protocol, then discuss incorporating Goldfish into Gasper. Finally, we discuss weaknesses of Goldfish under temporary asynchrony.

6.1 Limitations of Gasper

Interaction of LMD GHOST and Casper FFG The combination of Goldfish with the accountability gadget in Section 3 follows the generic construction of [43], 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 [8], is ad-hoc and complicated to reason about. Firstly, it is known to be susceptible to a *bouncing attack* [35]. 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 (Theorem 7) 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 [36], akin to the balancing attacks [42]. It is perhaps possible to resolve these through the use of techniques akin to vote buffering, to avoid the adversary being able to consistently split views. Even then, it is not at all clear that the bouncing attack cannot still be executed by exploiting other aspects of the complex interaction of LMD GHOST and Casper FFG. One such aspect is the fact that the FFG votes at any *Ethereum epoch* point at the last 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.) Another one is that 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.

On-chain inclusion of consensus messages Another peculiarity of Gasper is that the inclusion of consensus messages (FFG votes) into blocks is crucial to the consensus process itself. In particular, its hybrid fork-choice rule filters out all branches whose state (at the tip) has not justified the latest justified checkpoint, meaning that either not enough FFG votes have been included, or that a state transition processing them has not yet occurred. This rule makes the protocol even harder to reason about and formally analyze, and also introduces attack vectors similar to those already mentioned for the bouncing attack, *i.e.*, related to the adversary obtaining private justifications. Future work is required to carefully analyze its role in the protocol, and whether it can be removed from it.

Stale votes in LMD GHOST Without vote expiry, the votes of honest asleep validators can, in certain conditions of low participation, be weaponized by an adversary controlling a small fraction of the validator set to execute an arbitrarily long reorg. Consider for example a validator set of size $2n+1$, and a partition of the validator set in three sets, V_1, V_2, V_3, with $|V_1| = |V_2| = n$ and $|V_3| = 1$. The validators in V_1, V_2 are all honest, while the one in V_3 is adversarial. After GST, the latest messages of validators from V_1 and V_3 vote for a block A, whereas those from V_2 vote for a conflicting block B, so A is canonical. The validators in V_2 are now asleep, and stay asleep for the remainder of this run of the protocol. The awake honest validators, *i.e.*, V_1, keep voting for descendants of A for as long as the adversary does not change their vote. After waiting arbitrarily long, the adversarial validator can vote for B, resulting in all awake honest validators experiencing a reorg of all blocks confirmed after GST. The protocol is then clearly not dynamically available.

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.

Reorg resilience Even in the setting of full participation, 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 [48]. Without subsampling, LMD GHOST is indeed reorg resilient in the full participation setting, *if proposer boost is replaced by vote buffering.* In fact, Theorem 1 obtains reorg resilience as a consequence of two properties, Lemmas 1 and 2, respectively the property that all honest 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 $\beta \geq \frac{1}{3}$, even in the full participation setting and without subsampling, because those two properties are in conflict for such β. The first property only holds if boost is higher than β, *i.e.*, the amount of adversarial votes which might be withheld in an ex ante reorg attempt. On the other hand, the second property only holds if boost plus β is less than $1 - \beta$, because otherwise an adversarial proposer can make use of boost to conclude an ex post reorg. Therefore, we can only have reorg resilience for $\beta < \frac{1}{3}$ (by setting boost to $\frac{1}{3}$).

6.2 Bringing Goldfish to Gasper

Replacing LMD GHOST with Goldfish LMD GHOST could be replaced by Goldfish in Gasper [8], with only minor changes needed. Firstly, multiple potential leaders are elected through VRFs instead of a single leader through RANDAO. This results in additional bandwidth consumption due to multiple proposals being propagated, but helps maintain the confirmation time as a constant as participation drops. Moreover, the election process via VRFs is not biasable by the participants, which is not the case with RANDAO [22], and it automatically provides privacy to the leader before they reveal themselves, protecting them from targeted denial-of-service (DoS) attacks. It is not clear whether VRFs can also be utilized for subsampling, because the functioning of the beacon chain heavily relies on the aggregation of signatures, in order to support a large validator set. Augmenting the votes with VRF proofs is not compatible with aggregation, since the latter requires all messages to be the same (compatibility would require an aggregation scheme for VRF outputs). Nonetheless, RANDAO could still be used for subsampling, though its biasability would affect the tolerable adversarial fraction of validators. Some care has to also be taken to make attestation aggregation compatible with vote buffering.

Combination of Goldfish and FFG The protocol resulting from replacing LMD GHOST with Goldfish in Gasper still does not satisfy all the properties we want, and which we have proved for the combination of Goldfish with an accountability gadget, when following the construction of [43]. In particular, it does not escape the negative result from Section 6.1,

due to the bouncing attack under low participation. Since Casper FFG is not a complete protocol, as it lacks a message schedule for proposals and votes, more work is needed to understand if it is possible to use it as the BFT protocol in the aforementioned construction.

An alternative approach, and perhaps easier to integrate in the protocol if successful, is to try to achieve security by adapting the construction to the protocol, rather than trying to use its black box approach. A simple modification is to stipulate that the honest validators cast their FFG votes only for the blocks that are confirmed by Goldfish, *i.e.*, the blocks that are part of their available ledgers $\mathsf{ch}_{\mathrm{ava}}$. As already mentioned, this is different from the current PoS Ethereum specification for Casper FFG, where the FFG votes at any *Ethereum epoch* point at the last epoch boundary block of that epoch, regardless of its confirmation status by the underlying LMD GHOST rule. Unfortunately, this is not sufficient to guarantee the security of the accountable, final prefix ledger $\mathsf{ch}_{\mathrm{acc}}$ outputted as the prefix of the finalized Casper FFG checkpoints, again due to the bouncing attack. To avoid the latter, and inspired by the aforementioned construction, a second modification is to start the iteration of the hybrid fork-choice from the latest *finalized* checkpoint, rather than the latest justified. The question of whether this is sufficient to ensure security is left for future work.

6.3 Brittleness of Goldfish under temporary asynchrony

LMD GHOST without subsampling, and with vote buffering instead of proposer boost, *i.e.*, similar to the protocol from [23], has the property that extended periods of asynchrony, or many asleep honest validators, cannot jeopardize the safety of blocks that all honest validators have seen as canonical. More precisely, if all honest validators have voted for a block and those votes are known to all of them, the block cannot be reorged, even if at some later point all honest validators are temporarily asleep. This is the key intuition behind the notion of finality of [23]. Longest chain protocols also have some resilience to temporary disruptions of this kind, as blocks accrue safety during periods of synchrony and honest majority of awake validators, because blocks deeper in the canonical chain are safer. Therefore, many honest validators being temporarily asleep, or not being able to communicate due to asynchrony, cannot break the safety of relatively deep confirmations. On the other hand, blocks in Goldfish do not accrue safety against asynchrony as time goes on, due to vote expiry. In fact, vote expiry after one slot means that Goldfish cannot tolerate a single slot in which all honest validators are asleep or in which they cannot hear from the other honest validators due to adversarially induced network delay, making the long term safety very much reliant on the functioning of the accountability gadget.

7 Equivocation Discounting to Mitigate Spamming

Goldfish deals with equivocating votes simply by accepting all of them, but counting at most one per subtree (lines 9, 18, 19 of Algorithm 2). This does not give any additional fork-choice power to an equivocating validator, and it does not allow for irreconcilable

splits of honest validators' views, which would be the case with a naive first-seen (or last-seen) approach. Instead, it guarantees that honest validators can always end up with the same view, in particular through the vote buffering mechanism, and that their fork-choice outputs agree when they do. Nonetheless, 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. This attack vector is particularly pernicious in Goldfish, because of its heightened vulnerability to increased network latency, as discussed in Section 6.3.

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, and that there is no immediate recovery, because the same validator can continue producing equivocating attestations in subsequent slots as well. It is perhaps possible to mitigate this attack vector without breaking the strong properties of vote buffering with approaches similar to those of [37], *i.e.*, by introducing more refined rules for downloading and forwarding consensus messages. The approach we take is instead to introduce *equivocation discounting*. This general technique is already present in the current implementation of PoS 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 heavily 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.

Equivocation discounting

(a) *Fork-choice discounting:* When running the GHOST-Eph fork-choice at slot t, only count slot $t-1$ votes from eligible validators for which your view contains a single vote for slot $t-1$, *i.e.*, 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 slot, and at most two votes per *eligible* validator.

(c) *Validity condition for proposals:* A proposal message which contains more than two votes for some validator is invalid, and so is one which contains any vote from a validator which is not eligible, *i.e.*, with a VRF output $\geq \mathsf{thr}$.

Discounting equivocations from the fork-choice preserves the property that there cannot be irreconcilable splits of validator views, because all that is needed for convergence is agreement on the list of equivocators from the previous slot, which in turn only needs all views to have compatible equivocation evidence, *i.e.*, pairs of equivocating votes for the same list of equivocating validators. 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 view be a superset of honest validators' views. 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.*, Lemma 1 still holds. Note that the leader still only needs to include the union of its view and buffer, $\mathcal{V} \cup \mathcal{B}$, in the proposal message, because following the download rule guarantees that it will contain exactly all 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 VRF output is honest, as in Lemma 1, and all honest validators voting together implies that the proposal is never reorged, as in Lemma 2. Note that the latter is not affected by equivocation discounting, because it relies on the votes of honest validators, which do not equivocate. From these two properties, we obtain reorg resilience as in Theorem 1, 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 Theorem 5, since equivocation discounting plays no role in it. For safety, the key ingredient is Lemma 8, from which Theorem 4 follows unchanged. We thus prove Lemma 8 here for Goldfish with equivocation discounting, by making a very small modification to the argument:

Proof of Lemma 8 for Goldfish with equivocation discounting. The validator $\mathcal{P}$ broadcasts b and the $(\frac{3}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes for it at round $4\Delta t + 2\Delta$, where n is the total number of validators (cf. Section 2). Upon merging its buffer with its view, each honest validator observes these votes in its view at round $4\Delta t + 3\Delta$, and continues to observe them when voting at slot $t+1$. If the view of an honest validator, when voting at slot $t+1$, contains votes by less than $\frac{1}{2}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ equivocating validators eligible at slot t, then this view contains more than $(\frac{1}{4}+\epsilon)n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes for b by non-equivocating validators. Moreover, it contains at most $\frac{1}{4}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ votes by non-equivocating validators for any subtree rooted at a block conflicting with b. This is due to Proposition 4, which implies that there can be at most $\frac{1}{4}n\frac{\mathsf{thr}}{2^{b(\lambda)}}$ slot t eligible validators that are different from the validators that voted for b. Since it no longer counts votes from equivocating validators, the GHOST-Eph fork-choice rule run using the slot t votes at round $4\Delta(t+1)+\Delta$ outputs b or a descendant of b for all honest validators, implying that they all vote for b or one of its descendants at slot $t+1$. Finally, Lemmas 2 and 3 together imply the desired result. □

Acknowledgment

We thank Aditya Asgaonkar, Carl Beekhuizen, Vitalik Buterin, Justin Drake, Dankrad Feist, Sreeram Kannan, Georgios Konstantopoulos, Barnabé Monnot, Dan Robinson, Danny Ryan, and Caspar Schwarz-Schilling 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] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. "Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability". In: *Conference on Computer and Communications Security*. CCS '18. ACM. 2018, pp. 913–930.

[2] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. "Prism: Deconstructing the Blockchain to Approach Physical Limits". In: *CCS*. ACM, 2019, pp. 585–602.

[3] Carl Beekhuizen, Caspar Schwarz-Schilling, and Francesco D'Amato. *Change fork choice rule to mitigate balancing and reorging attacks*. Ethereum Research. Oct. 28, 2021. URL: `https://ethresear.ch/t/change-fork-choice-rule-to-mitigate-balancing-and-reorging-attacks/11127` (visited on 06/28/2022).

[4] Iddo Bentov, Rafael Pass, and Elaine Shi. "Snow White: Provably Secure Proofs of Stake". In: *IACR Cryptol. ePrint Arch.* (2016), p. 919.

[5] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The latest gossip on BFT consensus*. 2018. arXiv: `1807.04938 [cs.DC]`.

[6] Vitalik Buterin. *Proposal for mitigation against balancing attacks to LMD GHOST*. 2020. URL: `https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation` (visited on 04/20/2021).

[7] Vitalik Buterin and Virgil Griffith. "Casper the Friendly Finality Gadget". In: *arXiv:1710.09437 [cs.CR]* (2019). URL: `https://arxiv.org/abs/1710.09437`.

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

[9] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. "On the instability of Bitcoin without the block reward". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 154–167.

[10] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Symposium on Operating Systems Design and Implementation*. OSDI '99. USENIX Association, 1999, pp. 173–186.

[11] Benjamin Y. Chan and Elaine Shi. "Streamlet: Textbook Streamlined Blockchains". In: *Advances in Financial Technologies*. AFT '20. ACM, 2020, pp. 1–11.

[12] Jing Chen and Silvio Micali. "Algorand: A secure and efficient distributed ledger". In: *Theor. Comput. Sci.* 777 (2019), pp. 155–183.

[13] Phil Daian, Rafael Pass, and Elaine Shi. "Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake". In: *Financial Cryptography and Data Security*. FC '19. Springer, 2019, pp. 23–41. ISBN: 978-3-030-32101-7.

[14] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges". In: *arXiv:1904.05234 [cs.CR]* (2019). URL: `https://arxiv.org/abs/1904.05234`.

[15] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain". In: *EUROCRYPT 2018*. Springer, 2018, pp. 66–98.

[16] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *Communications of the ACM* 61.7 (2018), pp. 95–102.

[17] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. "Parallel Chains: Improving Throughput and Latency of Blockchain Protocols via Parallel Composition". In: *IACR Cryptol. ePrint Arch.* (2018), p. 1119.

[18] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin backbone protocol: Analysis and applications". In: *EUROCRYPT 2015*. Springer. 2015, pp. 281–310.

[19] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *SOSP*. ACM, 2017, pp. 51–68.

[20] Vipul Goyal, Hanjun Li, and Justin Raizes. "Instant Block Confirmation in the Sleepy Model". In: *Financial Cryptography (2)*. Vol. 12675. Lecture Notes in Computer Science. Springer, 2021, pp. 65–83.

[21] Gene Itkis and Leonid Reyzin. "Forward-Secure Signatures with Optimal Signing and Verifying". In: *CRYPTO*. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 332–354.

[22] George Kadianakis. *Whisk: A practical shuffle-based SSLE protocol for Ethereum*. Ethereum Research. Jan. 13, 2022. URL: `https://ethresear.ch/t/whisk-a-practical-shuffle-based-ssle-protocol-for-ethereum/11763` (visited on 09/05/2022).

[23] Daniel Kane, Andreas Fackler, Adam Gagol, and Damian Straszak. "Highway: Efficient Consensus with Flexible Finality". In: *CoRR* abs/2101.02159 (2021).

[24] Sreeram Kannan, Kartik Nayak, Peiyao Sheng, Pramod Viswanath, and Gerui Wang. *BFT Protocol Forensics*. 2020. arXiv: 2010.06785.

[25] Pankaj Khanchandani and Roger Wattenhofer. "Brief Announcement: Byzantine Agreement with Unknown Participants and Failures". In: *PODC*. ACM, 2020, pp. 178–180.

[26] Pankaj Khanchandani and Roger Wattenhofer. "Byzantine Agreement with Unknown Participants and Failures". In: *IPDPS*. IEEE, 2021, pp. 952–961.

[27] Aggelos Kiayias and Giorgos Panagiotakos. "On Trees, Chains and Fast Transactions in the Blockchain". In: *LATINCRYPT*. Vol. 11368. Lecture Notes in Computer Science. Springer, 2017, pp. 327–351.

[28] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *CRYPTO 2017*. Springer. 2017, pp. 357–388.

[29] Andrew Lewis-Pye and Tim Roughgarden. "Resource Pools and the CAP Theorem". In: *arXiv:2006.10698* (2020). URL: https://arxiv.org/abs/2006.10698.

[30] Kevin Liao and Jonathan Katz. "Incentivizing blockchain forks via whale transactions". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 264–279.

[31] Dahlia Malkhi, Atsuki Momose, and Ling Ren. *Instant Finality in Byzantine Generals With Unknown and Dynamic Participation*. Chainlink Labs Research. Aug. 28, 2022. URL: https://blog.chain.link/instant-finality-in-byzantine-generals-with-unknown-and-dynamic-participation/ (visited on 09/05/2022).

[32] Silvio Micali, Michael Rabin, and Salil Vadhan. "Verifiable random functions". In: *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE. 1999, pp. 120–130.

[33] Atsuki Momose and Ling Ren. "Constant Latency in Sleepy Consensus". In: *CCS*. Forthcoming. ACM, 2022.

[34] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008.

[35] Ryuya Nakamura. *Analysis of bouncing attack on FFG*. Ethereum Research. Sept. 8, 2019. URL: https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113 (visited on 08/18/2020).

[36] Ryuya Nakamura. *Prevention of bouncing attack on FFG*. Ethereum Research. Sept. 8, 2019. URL: https://ethresear.ch/t/prevention-of-bouncing-attack-on-ffg/6114 (visited on 08/18/2020).

[37] Joachim Neu, Srivatsan Sridhar, Lei Yang, David Tse, and Mohammad Alizadeh. "Longest Chain Consensus Under Bandwidth Constraint". In: *AFT*. Forthcoming. ACM, 2022. URL: https://arxiv.org/abs/2111.12332.

[38] Joachim Neu, Ertem Nusret Tas, and David Tse. *A balancing attack on Gasper, the current candidate for Eth2's beacon chain*. Ethereum Research. Oct. 20, 2020. URL: https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079 (visited on 04/22/2021).

[39] Joachim Neu, Ertem Nusret Tas, and David Tse. *Attacking Gasper without adversarial network delay*. Ethereum Research. July 25, 2021. URL: https://ethresear.ch/t/attacking-gasper-without-adversarial-network-delay/10187 (visited on 09/02/2021).

[40] Joachim Neu, Ertem Nusret Tas, and David Tse. *Avalanche Attack on Proof-of-Stake GHOST*. Ethereum Research. Jan. 24, 2022. URL: https://ethresear.ch/t/avalanche-attack-on-proof-of-stake-ghost/11854 (visited on 01/24/2022).

[41] Joachim Neu, Ertem Nusret Tas, and David Tse. *Balancing Attack: LMD Edition*. Ethereum Research. Jan. 24, 2022. URL: https://ethresear.ch/t/balancing-attack-lmd-edition/11853 (visited on 01/24/2022).

[42] Joachim Neu, Ertem Nusret Tas, and David Tse. "Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma". In: *IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 446–465.

[43] Joachim Neu, Ertem Nusret Tas, and David Tse. "The Availability-Accountability Dilemma and its Resolution via Accountability Gadgets". In: *International Conference on Financial Cryptography and Data Security*. FC '22. May 2022. URL: https://arxiv.org/abs/2105.06075.

[44] Joachim Neu, Ertem Nusret Tas, and David Tse. "Two More Attacks On Proof-of-Stake GHOST/Ethereum". In: *ConsensusDay@CCS*. Forthcoming. ACM, Nov. 2022. URL: https://arxiv.org/abs/2203.01315.

[45] Rafael Pass and Elaine Shi. "The Sleepy Model of Consensus". In: *ASIACRYPT (2)*. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 380–409.

[46] Youer Pu, Lorenzo Alvisi, and Ittay Eyal. "Safe Permissionless Consensus". In: *IACR Cryptol. ePrint Arch.* (2022), p. 796.

[47] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. "Blockchain CAP Theorem Allows User-Dependent Adaptivity and Finality". In: *Financial Cryptography and Data Security*. FC '21. 2021.

[48] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. "Three Attacks on Proof-of-Stake Ethereum". In: *International Conference on Financial Cryptography and Data Security*. FC '22. Forthcoming. 2022. URL: https://arxiv.org/abs/2110.10086.

[49] Yonatan Sompolinsky and Aviv Zohar. "Secure High-Rate Transaction Processing in Bitcoin". In: *Financial Cryptography*. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 507–527.

[50] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Symposium on Principles of Distributed Computing*. PODC '19. ACM, 2019, pp. 347–356.