Available Attestation: Towards a Reorg-Resilient Solution for Ethereum Proof-of-Stake

Mingfei Zhang
Shandong University
mingfei.zh@outlook.com

Rujia Li*
Tsinghua University
rujia@tsinghua.edu.cn

Xueqian Lu
Independent Researcher
xueqian.lu@bitheart.org

Sisi Duan*†
Tsinghua University
duansisi@tsinghua.edu.cn

Abstract

Ethereum transitioned from Proof-of-Work consensus to Proof-of-Stake (PoS) consensus in September 2022. While this upgrade brings significant improvements (e.g., lower energy costs and higher throughput), it also introduces new vulnerabilities. One notable example is the so-called malicious *reorganization attack*. Malicious reorganization denotes an attack in which the Byzantine faulty validators intentionally manipulate the canonical chain so the blocks by honest validators are discarded. By doing so, the faulty validators can gain benefits such as higher rewards, lower chain quality, or even posing a liveness threat to the system.

In this work, we show that the majority of the known attacks on Ethereum PoS are some form of reorganization attacks. In practice, most of these attacks can be launched even if the network is synchronous (there exists a known upper bound for message transmission and processing). Different from existing studies that mitigate the attacks in an ad-hoc way, we take a systematic approach and provide an elegant yet efficient solution to reorganization attacks. Our solution is provably secure such that no reorganization attacks can be launched in a synchronous network. In a partially synchronous network, our approach achieves the conventional safety and liveness properties of the consensus protocol. Our evaluation results show that our solution is resilient to five types of reorganization attacks and also highly efficient.

1 Introduction

Ethereum, a leading blockchain platform, transitioned to Ethereum 2.0 in September 2022. It now uses a Proof-of-Stake (PoS) consensus mechanism called Gasper [6]. Gasper integrates two protocols: Casper the Friendly Finality Gadget (FFG) [5], a protocol ensuring the finality of transactions; a modified version of the Greedy Heaviest-Observed Sub-Tree (HLMD GHOST) for *selecting* the canonical chain. Namely, every honest validator in the system only proposes new blocks that extend its canonical chain and votes for blocks on its canonical chain. Eventually, one chain will be finalized according to FFG, so the system achieves safety (i.e., no double spending) and liveness (i.e., transactions submitted to the system are eventually finalized).

Malicious reorganization attack (reorg attack for short) [27, 28], denotes an attack in which the proposed blocks by honest validators are *re-organized*. In particular, Byzantine validators *manipulate* the canonical chain such that blocks by honest validators will be considered invalid and eventually discarded (i.e., *orphaned*). This concept is also closely related to the notion of selfish mining, first known as an attack on Bitcoin [19]. Namely, selfish mining is one kind of reorganization attack. We illustrate the reorg attack by Neuder et al. in Figure 1. Consider that block b_0 is proposed by an honest validator. In the next *slot*[1], a block b_1 is proposed by a Byzantine validator v_i and all Byzantine validators vote for b_1. Block b_1 is released when the next validator proposes a block b_2 extending b_0. After b_1 is released, block b_1 becomes the canonical chain as the chain led by b_1 is heavier (with more votes and a higher *weight*). The block b_2 is then considered orphaned and discarded. Reorganization attacks typically do not aim to attack the safety or liveness of the system. Instead, Byzantine validators may gain additional benefits such as higher rewards than honest validators.

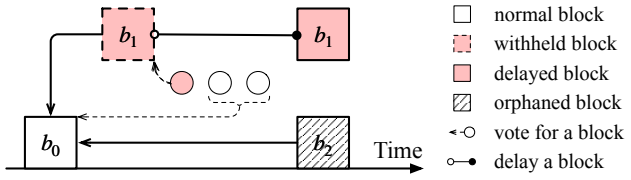


Figure 1: The reorg attack found by Neuder et al. [28].

* Corresponding author.
† Sisi is also with Zhongguancun Laboratory, Shandong Institute of Blockchains, BNRist, and State Key Laboratory of Cryptography and Digital Economy Security.

[1] Each slot lasts for a fixed period of time and only one randomly selected validator is allowed to propose a block.

attack type	scheme	timing assumption	mitigation	
			solution	limitation
changing block weight	ex-ante reorg [28, 33, 34]	synchrony	proposer boosting v1 [31]	cause sandwich reorg
	balancing attack [25, 26, 34]			
	sandwich reorg [12]		proposer boosting v2 [38]*	cannot fully prevent
filtering block tree	bouncing attack [24, 29]	partial synchrony†	safe-slots [2]	cannot fully prevent
	unrealized justification reorg [1]	synchrony	Capella upgrade [20]	cause staircase attack
	justification withholding reorg [30, 32]			
	staircase attack [41]		Deneb upgrade [9]	cannot fully prevent

* Proposer boosting parameter decreases from 0.7 to 0.4.
† The attack is conducted after the network is synchronous.

Table 1: Comparison of known malicious reorganization attacks against Ethereum PoS and their mitigation solutions by Ethereum.

We find that the majority of known effective attacks on Ethereum PoS belong to reorganization attacks, although they emphasize different types of adversarial strategies. According to how the canonical chain is manipulated by the adversary, we classify known attacks into two categories: attacks from *changing block weight* and attacks from *filtering block tree*. The attacks from changing block weight refer to the strategy where Byzantine validators modify the *weight* (informally, block weight is related to the number of votes for the block) of their proposed blocks to make their fork eventually become the canonical chain. The reorg attack [28] shown above is one example. Meanwhile, the attacks from filtering block tree do not change the block weight. Instead, the attacks make honest validators prune the canonical chain. This is often achieved by changing the *state* of honest validators. We summarize these malicious reorganization attacks in Table 1.

In response to the vulnerabilities, mitigation approaches are proposed from both academia and industry. They are often designed in an ad-hoc way, addressing one issue at a time. Without formal proof, the mitigation approaches may create new issues. For instance, to mitigate the ex-ante reorg attack and balancing attack [25], Ethereum implements the *proposer boosting* mechanism [4, 31]. By temporarily adjusting the weight of the block in the current slot, the forks created by the adversary will not become the canonical chain. However, the mitigation approach introduces new issues. A so-called *sandwich reorg attack* [12] was later proposed, exploiting proposer boosting to create a reorg attack. The sandwich reorg attack is a variant of ex-ante reorg attacks where two Byzantine proposers collude to make the blocks by honest validators orphaned. Additionally, many known mitigation solutions lack formal analysis or introduce additional assumptions, e.g., by assuming that the ratio of stake controlled by the adversary is no more than 20% [12].

Therefore, an open research question is:

Does there exist a provably secure and efficient solution that is resilient to reorg attacks in Ethereum PoS?

What can be solved and what cannot be solved? Ethereum PoS assumes a partially synchronous network [18], i.e., the network might be temporarily asynchronous (there does not exist an upper bound on message transmission and processing) but after an unknown Global Stabilization Time (GST), the network becomes synchronous (there exists a known upper bound Δ). When the network is temporarily asynchronous, the reorg attack can not be solved. To see why, consider the example shown in Figure 1. One can never differentiate whether v_i is slow or faulty so it is unavoidable that the chain led by b_1 becomes the canonical chain when the network is asynchronous[2]. Accordingly, it is only possible to provide a provably secure solution that prevents reorg attacks in a synchronous network. Our solution achieves reorg-resilience only during periods of network synchrony. We argue that studying a reorg-resilient solution in a synchronous network is already valuable for two reasons (cf. Section 5). First, most known reorg attacks can be launched even if the network is synchronous, as summarized in Table 1. Second, it is likely that most of the time, the network is synchronous. According to the statistics of Ethereum, more than 99% of the blocks are received *on time*[3]. Therefore, we study a provably secure reorg-resilient solution in a synchronous network. When the network is partially synchronous, our approach achieves the standard *safety* and *liveness* properties for Byzantine fault-tolerant consensus protocol, which are even stronger than the vanilla Ethereum PoS protocol. Namely, the vanilla Ethereum PoS protocol achieves safety, *plausible liveness*, and *probabilistic liveness* [5] and the liveness guarantees are weaker than the conventional liveness notion.

Our approach. We propose a mechanism called *available attestation (AA)*. AA is inspired by the concept of *weak*

[2] In the asynchronous network, there exists a *network scheduler* (i.e., the adversary) that can manipulate the delay of the messages, even between honest validators. However, the messages from an honest sender will eventually be received by an honest receiver.

[3] Date source (accessed in Aug 2024): https://explorer.rated.network/network.

certificate in conventional Byzantine fault-tolerant protocols [17, 21]. Namely, in a group of validators among which at most f is Byzantine faulty, a weak certificate consists of $f+1$ *votes* (i.e., attestations in the notation by Ethereum), proving that at least one honest validator has voted for some block so the block is *available*. We borrow this concept to Ethereum PoS and define blocks with at least one-third votes as *stable* blocks. This guarantees that each stable block is received by at least honest validators. One-third is the minimum number that can achieve the goal. We modify the HLMD GHOST rules such that validators prioritize their votes to the chain with the most number of stable blocks. Informally, this allows one to identify the *longest* chain already observed by some honest validators. In a synchronous network, the chain is likely the chain observed by *all* validators! Accordingly, reorg attacks can be fully prevented when the network becomes synchronous (i.e., after GST). We provide a formal proof of our protocol. Additionally, our solution does not introduce any new attack surface. This is because we do not add any new message types or modify the workflow of the protocol.

One interesting fact is that our solution that requires validators to keep track of one-third of attestations is aligned with the *honest reorg* [37] mechanism currently implemented by Ethereum. Honest reorg is designed to prevent the proposer from delaying its block. In particular, each block needs to receive at least 20% attestations to be considered *valid*. Our solution requires this threshold to be at least 33.3%. Note that honest reorg does not mitigate malicious reorganizations while our approach is reorg resilient.

We implement our AA mechanism using *Prysm*, one of the most popular Ethereum PoS implementations. Our evaluation results show that our solution introduces negligible overhead to the latency and throughput of Ethereum PoS. We also implement a set of known reorg attacks and show that our solution can successfully defend all these attacks in practice, despite the fact that our solution is provably secure.

Summary of our contributions. We summarize our main contributions as follows.

- We provide a classification of reorg attacks and classify known attacks into two types: attacks based on changing block weight (Attack-I) and attacks based on filtering block tree (Attack-II). We show that almost all known attacks against Ethereum PoS belong to reorg attacks (Section 4).
- We introduce available attestation (AA), a provably secure mechanism that is resilient to any reorg attacks for Ethereum PoS when the network is synchronous. To fully instantiate the AA mechanism, our solution slightly modifies the data structure of the blocks, introduces a communication-efficient forwarding rule, and replaces the HLMD GHOST rule with the longest chain rule (Section 6).
- We present a rigorous security analysis of our modified protocol, demonstrating that it is resilient to reorganizations under synchronous networks and maintains both safety and liveness in partially synchronous networks (Section 7).
- We implement our solution using the Prysm codebase. Our evaluation results on up to 16,384 validators show that our solution can effectively mitigate all known reorg attacks. Meanwhile, our solution introduces low overhead to the performance of the system, achieving almost identical throughput and latency as the vanilla protocol (Section 8).

2 Related Work

Mitigation by Ethereum. We review the mitigation solutions for attacks on Ethereum. This part can be viewed as a detailed discussion for Table 1. Readers who are not familiar with the notions of Ethereum and the attacks against Ethereum may refer to Section 3 and Section 4 for details.

Proposer boosting [4, 31, 38] is designed to mitigate balancing attacks [25] and ex-ante reorg attacks. Proposal boosting (v1) assigns a *temporary* additional weight (70% of the total stake of the current committee) for the block proposed in the current slot. This will make the block in the current epoch have a higher weight and not be orphaned. The mitigation has some limitations. First, it only mitigates the balancing attack and ex-ante reorg attack. Second, it causes a new reorg attack called *sandwich reorg attack* [12]. Accordingly, the additional weight is adjusted to 40% (denoted as proposal boosting v2).

Safe-slots [2] is a mitigation solution for the bouncing attacks [24]. The bouncing attack is a liveness attack against Ethereum PoS (see Section 4 for details). Safe-slots defines a new parameter called SAFE_SLOTS_TO_UPDATE_JUSTIFIED. It only allows validators to update the last justified checkpoint in the first SAFE_SLOTS_TO_UPDATE_JUSTIFIED slots of an epoch. However, it was later found that *bouncing attacks* can still be conducted [29].

Capella upgrade [20] consists of the mitigation for unrealized justification reorg attacks [1] and justification withholding reorg attacks [30]. It modifies the *filtering rule* in fork choice HLMD GHOST: any chain that includes enough attestations in the previous epoch will not be pruned in the fork choice. The mitigation suffers from a reorg attack called *staircase attack* [41].

Deneb upgrade [9] consists of the mitigation solution for staircase attacks. It further modifies the filtering rule: a chain will not be filtered if the difference between the epoch of its newest justified checkpoint and the current epoch is no more than two epochs. The mitigation can not fully prevent the staircase attacks. It only decreases the probability of repeating staircase attacks to mitigate the effect of the attack.

Additional mitigation solutions to the reorganization attacks of Ethereum. The concept of reorg resilience was first mentioned in Goldfish [10]. Goldfish provides a reorg-resilient solution that is provably secure in the synchronous network. The Recent Latest Message Driven GHOST (RLMD-GHOST) protocol [13] relaxes the timing

assumption to the partially synchronous network. However, the solution requires *all* validators to vote in every slot, making it not directly compatible with the Ethereum PoS workflow. Meanwhile, the Single Slot Finality protocol [14] and the 3-Slot-Finality protocol [11] both provide reorg-resilient solutions for PoS but require *all* validators to vote. In contrast, our solution preserves the workflow of Ethereum and does not require all validators to vote. We also make the same assumption of the partially synchronous network.

Weak quorum certificates (QC). Weak QC (also called a weak certificate) denotes *votes* from $f+1$ validators. The concept originates from conventional Byzantine fault-tolerant (BFT) protocols [8, 17, 21]. Informally, as f is the maximum number of Byzantine validators, a weak QC proves that "something right has already been done". In Star [17] and Autobahn [21], each validator proposes a block and collects weak QC. After a weak QC is formed, the QC proves that some honest validator has previously received the proposal so the proposal is *available*. As mentioned in the introduction, our available attestation mechanism is motivated by the weak certificate. However, our approach is fundamentally different from conventional BFT protocols. Unlike conventional BFT protocols that often rely on the transferability of the certificate to prove something, our approach does not require validators to send the certificates to other validators.

Other malicious reorganization attacks. Malicious reorganization attacks are found in protocols beyond Ethereum PoS. Selfish mining [19] is arguably the first malicious reorganization attack. In the attack, the adversary withholds a chain and eventually the chain from honest miners is re-organized. Selfish mining is found in Proof-of-Stake (PoS) protocols as well [3, 27]. For instance, Brown-Cohen et al. demonstrate that the "longest-chain" variants of PoS protocols are vulnerable to malicious reorg attacks [3]. However, it was mentioned that the result does not cover Ethereum PoS.

3 Review of Ethereum Proof-of-Stake Protocol

We review the Gasper protocol used by Ethereum 2.0. Our notations largely follow from the Ethereum whitepaper, official document, and previous works [6, 36, 41].

3.1 Model and Notations

Network assumption. Ethereum PoS assumes that the network is *partially synchronous* [18]. In particular, there exists an unknown Global Stabilization Time (GST). After GST, the network is synchronous, i.e., there exists a known upper bound Δ for message transmission and processing.

Validator. Any node that participates in the consensus protocol is a *validator*. To become a validator, each user needs to first deposit some tokens to join the system. Without loss of generality, we assume that the total number of validators n is fixed, denoted as $\{v_1, v_2, \cdots, v_n\}$. Validators are either honest or Byzantine. The Byzantine validators can deviate from the specification protocol arbitrarily. Ethereum assumes that the weight of each validator's vote is related to the account balance (i.e., the stake). To simplify our description, we assume each validator's stake is normalized to *one unit* [6]. Under this assumption, let f be the number of faulty validators, we have $f < n/3$.

Time. Time is divided into *epochs* and each epoch includes 32 *slots*. Each slot lasts for 12 seconds. Each validator is assigned to one slot in an epoch randomly.

Roles of the validators. There are three roles for the validators: *proposer*, *attestor*, and *aggregator*. A proposer generates a block. An attestor *votes* for the blocks and the votes are called attestations. Finally, an aggregator aggregates the attestations. Besides, there is a concept called *committees*. In particular, validators are divided into 32 committees, one for each slot. Each member of the committee is an attestor of the slot. The proposer, attestor, and committees are randomly sampled according to RANDAO[4]. We assume that the roles of all validators are selected pseudorandomly and all validators can validate the roles of other validators. Furthermore, each validator can simultaneously have multiple roles.

In practice, each committee is further divided into *subnets* and each aggregator only aggregates the signatures in the same subnet [36]. We omit the details in this paper without changing the correctness of the system.

Block and checkpoint. A block b consists of four fields: the slot number, the hash of the *parent* block, a set of attestations, and a batch of transactions. The blocks each validator receives form a tree $\mathcal{T}$, rooted at the *genesis block*. A *chain* c is defined as the unique path from the *genesis block* to a specific *leaf block*. A checkpoint block is denoted as a pair (b, e), where b is a block and e is the epoch number. In the paper, we use the block instead. There is only one checkpoint block in each epoch. By default, the block proposed in the first slot of an epoch is the checkpoint block. If a validator does not receive the block from the proposer in the first slot, the most recent block from the previous epoch is considered the checkpoint.

Attestation and aggregated attestation. An *attestation* is a vote by an attestor, denoted as *att*. Each *att* consists of the slot number, hashes of *source* and *target* checkpoints, and the hash of *head* block. The slot number implies the time when the attestation is created. The *source* and *target* are used for finality. The *source* is the last *justified* checkpoint (to be described shortly) and the *target* is the last checkpoint block received by the validator. The *head* field is selected by the HLMD GHOST rule, which is the leaf block of the canonical chain. We say *att* is an attestation for the block in the *head* field or an attestation for the checkpoint block in the *target* field, without any ambiguity.

[4]RANDAO: https://github.com/randao/randao

The attestations that share identical *source*, *target*, and *head* can be aggregated into a single *aggregated attestation*. If attestation is a vote for the block b in the *head* field, it is also considered a vote for all the blocks led by b.

Finality. A checkpoint block cp might be *justified* and *finalized*. Informally, if a block is finalized, its order will never be reversed. Checkpoint cp is *justified* after two-thirds of attestations with cp as *target* are included in the chain. If a checkpoint that extends the justified block cp is also justified, cp is finalized. When cp is finalized, any blocks on the chain led by cp are finalized. A block can be justified or finalized according to the Casper FFG protocol.

Fork choice. HLMD GHOST, a variant of GHOST [35], denotes the rules for validators to select the canonical chain. Based on the block tree $\mathcal{T}$, HLMD GHOST starts from the last justified checkpoint and outputs one leaf block, denoted as *head*. *head* is used to identify the canonical chain. HLMD GHOST has a filtering mechanism: any chain that does not justify the last justified checkpoint is discarded. For the branches that all extend the last justified checkpoint, the output depends on the *weight* of blocks. In particular, the weight of a block b is computed as the cumulative stake of validators who have voted for the subtree rooted at b. The subtree with the largest weight is the *heaviest subtree*. HLMD GHOST recursively chooses the heaviest subtree and outputs the leaf block.

Security properties. The Ethereum PoS protocol satisfies the following properties.

- **(Safety)** If an honest validator finalizes a chain led by block b, another honest validator finalizes a chain led by block b', and the two chains have the same length, $b = b'$.
- **(Liveness)** The length of the finalized chain eventually grows for all honest validators.

3.2 The Ethereum PoS Protocol

We summarize the workflow of the vanilla Ethereum PoS protocol in Figure 3. Each validator maintains two local parameters: the block tree $\mathcal{T}$ and an attestation pool (all the received attestations) $\mathcal{P}$.

The function ForkChoice is used for calculating the leaf block of the canonical chain. It recursively selects the *heaviest subtree* and outputs the leaf block, as mentioned above. In case of a tie, it chooses the block according to the alphabetical order (lines 26-35 in Figure 3).

Each slot t has three phases, as illustrated in each slot in Figure 2. Recall that each slot has $N/32$ randomly selected validators as the committee. In each slot, one validator in the committee is selected as the proposer. All validators in the committee are attestors. A fixed number of validators are aggregators. Let T be the time slot t begins, and each slot proceeds as follows.

- (Time T) (lines 1-7 in Figure 3) The proposer v_i sends a message $(\text{Propose}, t, v_i, H(p), atts, txs)$ to all validators, where p is the output of HLMD GHOST, *atts* is a set of attestations, and *txs* is a batch of transactions from its queue of pending transactions. The attestations in *atts* are attestations for the canonical chain but have not been included in the canonical chain yet. The proposal is also called a *proposed block b*.
- (Time $T+\Delta$) (lines 8-14 in Figure 3) Each attestor sends an attestation $(\text{Attest}, t, v_i, H(h), H(s), H(c))$ to all validators in the committee, where h is the output of HLMD GHOST, s is v_i's last justified checkpoint, and c represents the most recent checkpoint.
- (Time $T+2\Delta$) (lines 15-19 in Figure 3) After receiving the attestations, each aggregator aggregates matching attestations it has received so far from slot t. The aggregated attestation is then sent to all validators in the system.

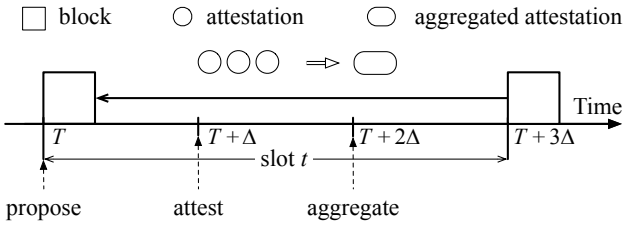


Figure 2: Slot t of the Ethereum PoS protocol (Figure 3).

Upon receiving a proposed block b, each validator v_i adds b in its block tree and then checks whether b is a new checkpoint. If so, v_i updates the *justified* checkpoint and the last checkpoint (lines 20-23 in Figure 3).

Note that we use the notation Δ for our description. As defined in Section 3.1, Δ is the known upper bound for message processing and transmission in a synchronous network. Ethereum assumes a partially synchronous network. Our use of Δ above can be properly interpreted as the fact that the slot duration matches 3Δ after GST. We note that a recent consensus protocol [23] makes the same assumption about a partially synchronous network.

4 Classification of Malicious Reorganization Attacks against Ethereum PoS

As mentioned in the introduction, we claim that almost all known attacks against Ethereum PoS are some form of reorg attacks. We thus classify known reorg attacks according to how the adversary manipulates the canonical chain into two types: attacks from *changing block weight* (Attack-I, Section 4.1) and attacks from *filtering block tree* (Attack-II, Section 4.2).

4.1 Attack-I: Modifying the Weight

In attack-I, the adversary attests attestations that change the weight of some branch to affect the selection of HLMD

Ethereum PoS Protocol for validator v_i.

global parameter: slot counter t
local parameters: block tree $\mathcal{T}$, attestation pool $\mathcal{P}$.

```
01 upon a slot t start
02   as the proposer for slot t
03     let p be the output of ForkChoice
04     obtain a set of newly received attestations atts from P
05     obtain a batch of transactions txs
06     create block b =(Propose, t, v_i, H(p), atts, txs)
07     send b to all validators
08 upon Δ seconds of slot t
09   as the attestor for slot t
10     let h be the output of ForkChoice
11     let s be the last justified checkpoint
12     let c be the last checkpoint
13     create attestation att =(Attest, t, v_i, H(h), H(s), H(c))
14     send att to committee
15 upon 2Δ seconds of slot t
16   as the aggregator for slot t
17     let Atts be the attestations of slot t in P
18     aggregate Atts as message agg with type (Aggregate)
19     send agg to all validators
20 upon receiving block b =(Propose, t', v_j, H(p), atts, txs)
21   add block b into T
       ▷ Exploited by Attack-II
22   if block b is a checkpoint
23     update checkpoint status in the chain led by parent of b
24 upon receiving message m with type (Attest) & (Aggregate)
25   add message m into P
26 function ForkChoice
27   let root be the last justified checkpoint
28   let result ← root
29   while true
30     for all children in result that are not pruned
         ▷ Exploited by Attack-I
31       choose the child b such that b is the root of heaviest subtree
32       break a tie alphabetically
33     result ← b
34     if result is a leaf block
35       return result
```

Figure 3: Ethereum PoS protocol. $H()$ denotes the hash function.

GHOST [12, 25, 26, 28, 33, 34]. This is usually achieved by delaying blocks and attestations and carefully voting for the blocks of the adversary. There are two types of attacks: (1) the adversary tries to make the weight of a branch higher than others [12, 28, 33, 34]; (2) the adversary tries to balance the weight of two branches [25, 26, 34].

Type (1). The ex-ante reorg attack [28, 33, 34] mentioned in the introduction belongs to type (1), in which the weight by the adversary becomes heavier after the withheld block is released. Another example is the sandwich reorg attack [12]. The sandwich reorg attack exploits the *proposer boosting* mechanism [31] (also see Section 2) to make an orphaned chain heavier than the canonical chain. As illustrated in Figure 4, the attack is an extension of the ex-ante reorg attack and requires another Byzantine validator v_l to collude with v_i (the proposer of b_1). Here, v_l is a valid proposer after the slot for b_2. At the end of slot $t+2$, block b_2 is heavier than block b_1 as b_2 receives more attestations. The chain led by block b_2 is the canonical chain and block b_1 is orphaned. When v_l proposes a block b_3, v_l sets b_1 as the parent block although it is supposed to set b_2 as the parent block. Due to the proposer boosting mechanism, the weight of b_3 is 70% weight of the total validators in a committee (i.e., proposer boosting weight). Meanwhile, the weight of block b_2 is the weight of honest validators in a committee. As the weight of honest validators is less than 70% weight of the total validators in a committee, the branch led by b_3 is the canonical chain, and block b_2 by honest validator is orphaned.

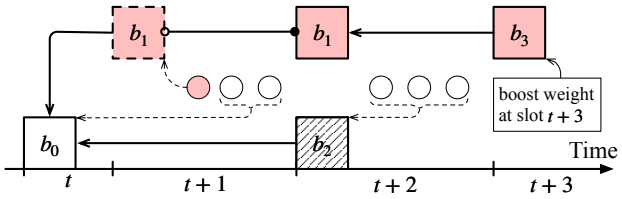


Figure 4: The sandwich reorg attack [12].

Type (2). An example of the type (2) attack is the balancing attack [25, 26, 34]. The idea is to always make two chains have the same weight so neither chain can eventually be finalized, posing a liveness threat to the system. Specifically, the adversary first waits for a situation where the proposers in two consecutive slots are Byzantine. As illustrated in Figure 5, block b_0 is proposed by an honest validator. The first Byzantine validator v_i withholds its block b_1 in slot $t+1$. The second validator v_j releases both b_1 and b_2 (v_j's block) when v_j is the proposer in slot $t+2$. Both b_1 and b_2 set b_0 as the parent block. Next, the adversary splits honest validators into two groups of the same size, V_1 and V_2, and makes each group vote for one chain only. To achieve this goal, the adversary withholds their attestations a_1 (with b_1 as *head*) and a_2 (with b_2 as *head*) and releases a_1, a_2 only to V_1, V_2, respectively. After validators in V_1 receive a_1, they vote for b_1 as b_1 is *heavier*. Similarly, validators in V_2 vote for b_2. Thus, after slot $t+2$ ends, both chains have the same number of attestations and have the same weight. The attack can be launched continuously so no blocks can be finalized on-chain.

4.2 Attack-II: Filtering the Branch

In attack-II, the adversary proposes blocks that make honest validators filter some branches (with blocks proposed by honest validators) from their block tree [1, 24, 29, 30, 32, 41]. This is often achieved by *manipulating* the justified blocks. In par-

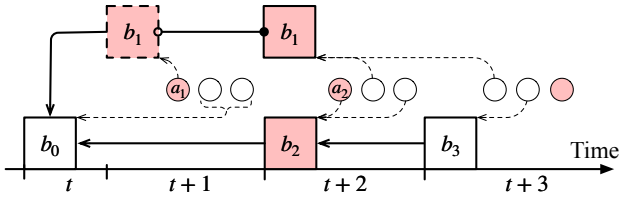


Figure 5: Balancing attack [25].

ticular, the adversary proposes a chain with a new checkpoint that updates the last justified checkpoint (see lines 22&23 specified in Figure 3). While the canonical chain does not update the justified checkpoint, it will be filtered by fork choice (see filtering mechanism in Section 3.1). Below, we describe four examples.

Unrealized justification reorg attack. The unrealized justification reorg attacks [1] aim to justify a checkpoint in a chain earlier than in the canonical chain. To launch the attack, the adversary creates a branch by proposing a checkpoint that extends an *older* block. After the adversarial branch is released, a new checkpoint is justified while the chain observed by honest validators does not justify any new checkpoint. In this way, the chain observed by honest validators is filtered according to HLMD GHOST. As illustrated in Figure 6, consider that honest validators maintain a consistent view of the canonical chain in epoch e and the checkpoint block is cp_0. Block b_1 is the first block that includes enough attestations to justify cp_0. In epoch $e+1$, a Byzantine validator proposes a checkpoint cp_1. Instead of setting the last block in epoch e (i.e., b_3) as the parent block, the Byzantine validator sets b_1 as the parent block of cp_1. This creates a new chain c_2 that conflicts with chain c_1 led by block b_3. After honest validators receive cp_1, they update the checkpoint status (line 22 of Figure 3). Thus, checkpoint cp_0 is justified in the chain c_2. After that, cp_0 becomes the last justified checkpoint, and chain c_1 is filtered in HLMD GHOST. To ensure that cp_1 can justify cp_0, the adversary carefully chooses the slot number of b_1 as the $\lceil 32 \times \frac{2}{3} \rceil = 22$th slot of epoch e. Accordingly, at most $32 - 22 = 10$ blocks can be orphaned in the attack.

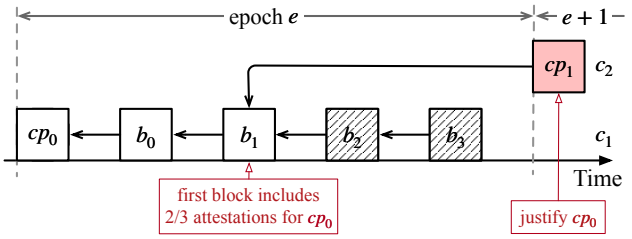


Figure 6: Unrealized justification reorg attack [1].

Justification withholding reorg attack. Justification withholding reorg attacks [30, 32] aim to prevent the chain observed by honest validators from justifying a checkpoint. Instead, the withheld blocks form a chain that justifies the last checkpoint. The attack requires the last few proposers in an epoch to be Byzantine. As illustrated in Figure 7, let b_0 be the last block proposed by an honest validator in epoch e. The Byzantine validators withhold their blocks after block b_0 in epoch e. As a result, the chain led by b_0 does not include enough attestations to justify the checkpoint block cp_0. In epoch $e+1$, as blocks after b_0 are withheld in epoch e, an honest validator proposes a new checkpoint cp_1 that extends b_0. Although honest validators receive cp_1, they can not justify a new checkpoint since fewer than two-thirds of attestations are included in the chain led by b_0 (line 22&23 in Figure 3). Right before epoch $e+1$ ends, the Byzantine validators release the withheld block b_1 and propose a new checkpoint cp_2 that extends b_1. The chain led by b_1 includes two-thirds of attestations for cp_0. After receiving cp_2, the chain led by cp_2 justifies checkpoint cp_0. The chain led by b_3 will be orphaned. As the checkpoint cp_0 is justified at the end of epoch $e+1$, up to 32 blocks proposed by honest validators in epoch $e+1$ will be reorganized.

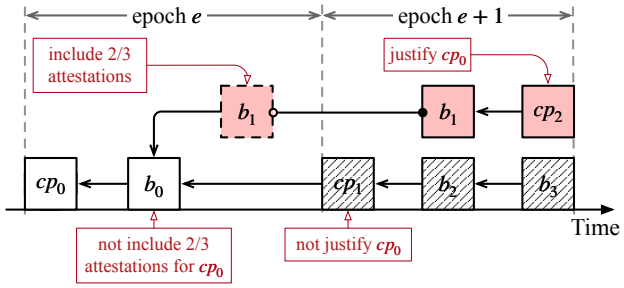


Figure 7: Justification withholding reorg attack [30].

Bouncing attack. The bouncing attack [24, 29] aims to make the last justified checkpoint switch between two chains, and neither chain can be finalized. The attack exploits the fact that chains conflicting with the last justified checkpoint are pruned in HLMD GHOST. As shown in Figure 8, the attack assumes that there are two chains, c_1 and c_2, after GST. Among two chains, chain c_1 led by checkpoint block cp_1 is the canonical chain observed by honest validators. Another chain c_2 is led by checkpoint block cp_2. Both checkpoints cp_1 and cp_2 extend the last justified checkpoint cp_0. Here, checkpoint cp_2 is called *justifiable*, i.e., it can be justified after attestations from the adversary are released [24]. In epoch $e+2$, honest validators extend the canonical chain c_1. The checkpoint of chain c_1 in epoch $e+2$ is cp'_1. After one-third of attestations vote for checkpoint cp'_1 in epoch $e+2$, the adversary justifies the *justifiable* checkpoint cp_2 in chain c_2. Checkpoint cp_2 is the last justified checkpoint. As chain c_1 conflicts with cp_2, chain c_1 is filtered in HLMD GHOST and chain c_2 becomes the canonical chain. The honest validators start to extend chain c_2. The checkpoint cp'_1 becomes the new *justifiable* checkpoint. The adversary then can repeat the above strategies and

make the last justified checkpoint switch between two chains. In this way, neither chain can be finalized, posing a liveness threat to the system.

Note that the attack can only be launched when the network is temporarily asynchronous. After the attack is successfully launched, the attack can be repeated, even after GST.

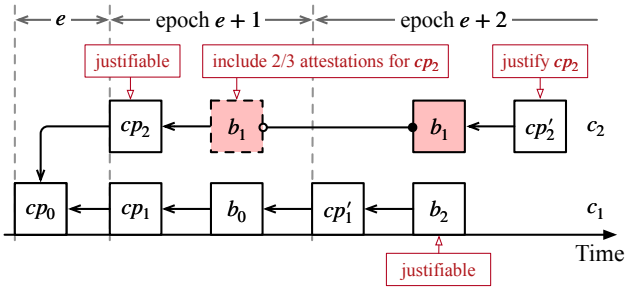


Figure 8: Bouncing attack [24].

Staircase attack. Staircase attack [41] aims to make honest validators suffer from penalties even if they strictly follow the protocol in a synchronous network after Capella upgrade [20] (also see Section 2). In staircase attacks, Byzantine validators withhold their attestations to prevent the canonical chain from justifying the last checkpoint. As illustrated in Figure 9, in epoch e, the Byzantine validators withhold their attestations to prevent the honest validators from justifying the checkpoint cp_0. The attestations from Byzantine validators are included in a withheld block b_1, a block proposed by a Byzantine validator. Before the middle of epoch $e+1$, all honest validators extend the chain led by block b_0. After the middle of epoch $e+1$, the withheld block b_1 is released. The last justified checkpoint is updated as only the withheld chain includes two-thirds of attestations for checkpoint block cp_0. The chain led by cp_1 is filtered in HLMD GHOST and the chain led by b_1 becomes the canonical chain. In this attack, the attestations from honest validators in the first half of epoch $e+1$ are discarded, as the chain they vote for is not later finalized. These honest validators suffer from *penalties* [36] according to the protocol. It was shown that by controlling 29.6% of the total stake, the attack can be conducted in every epoch so eventually all honest validators suffer from no incentive rewards.

4.3 Summary

One notable reason why malicious reorganization is so "easy" is that one can not validate whether a block b proposed by a validator is *correct*, e.g., b indeed extends the canonical chain and is released on time. Since one can never differentiate a slow but honest validator and a Byzantine validator in a partially synchronous network, building a reorg-resilient solution is thus valuable. Luckily, we show that reorg resilience can be achieved when the network is synchronous. When the network is temporarily asynchronous, our approach can still achieve the safety and liveness properties.

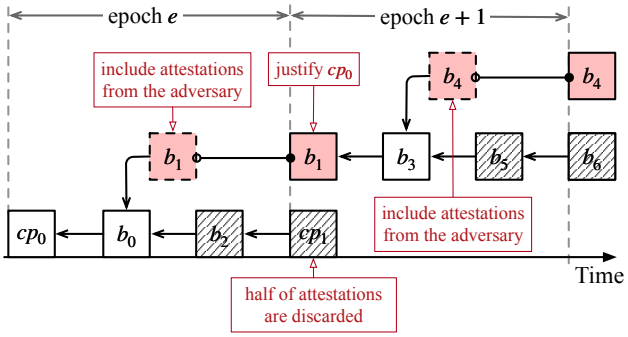


Figure 9: Staircase attack [41].

5 Overview of our Approach

In this section, we provide an overview of our approach. We begin with the definition of reorg resilience and then provide an overview of our technical contributions.

Reorg resilience. We define the property as follows.

- **(Reorg resilience)** If an honest proposer proposes a block b in the synchronous network, b will eventually be finalized.

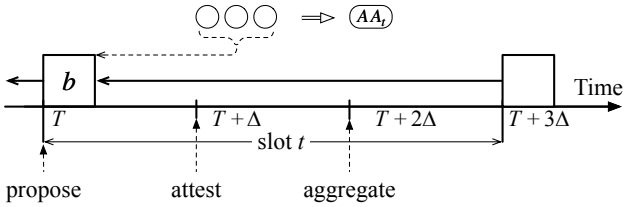
The notion of reorg resilience is not new. Goldfish [10] first studied reorg resilience and provided a synchronous protocol that is proved to be reorg resilient. The insight made by the paper is that conventional Byzantine fault-tolerant (BFT) protocols [8, 16, 22, 39, 40] do not suffer from the reorg issues. Namely, most conventional BFT protocols require the validators to collect a sufficiently large fraction of attestations (i.e., a Byzantine quorum of votes [7]) before proceeding to the next "phase". Any validator that proposes a new block is considered valid only if its parent block has accumulated a sufficiently large fraction of attestations. Accordingly, Goldfish provided a *hybrid* of PoS and conventional Byzantine fault-tolerant (BFT) protocol to achieve reorg resilience. The limitation, also as pointed out by follow-up work RLMD-GHOST [13], is that Goldfish cannot be directly adapted to Ethereum since it is provably secure only in the synchronous network while Ethereum assumes a partially synchronous network. RLMD-GHOST later provided a solution in the partially synchronous model. The drawback of RLMD-GHOST is that it requires *all* validators to vote in every slot, making it impractical for Ethereum PoS. Namely, Ethereum already has over one million validators[5]. It is too expensive to support all-to-all communication in such a large-scale network.

Our approach in a nutshell. We provide a lightweight yet efficient solution for Ethereum PoS which is: 1) reorg resilient

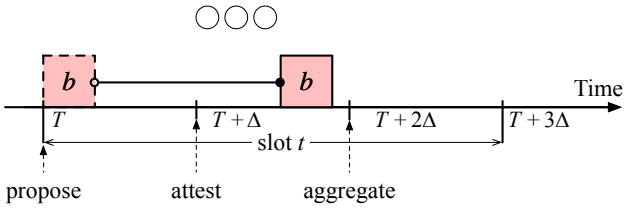
[5]Date source (accessed in Aug 2024): https://www.beaconcha.in/.

in a synchronous network; 2) safe and live in a partially synchronous network; and 3) easy to implement and deploy. As motivated in the introduction, we introduce available attestation (AA), an approach inspired by conventional BFT protocols from *weak quorum* of attestations [17, 21]. Namely, consider a system with N validators, among which at most f are faulty. If $f+1$ validators vote for a block b in slot t, at least one honest validator has validated b. The $f+1$ attestations become a *proof* of the availability of b. We use AA_t to denote a block with $f+1$ attestations in slot t.

Put the concept of weak quorums in the context of Ethereum PoS, we define AA as a mechanism that checks *whether a block receives matching attestations from at least one-third of validators*[6]. In particular, if b is a block for slot t, its child must include attestations for b from at least one-third of validators in slot t. We use the notion of *stable block* to describe this scenario. Having such a AA rule is useful in a synchronous network already. We show an example in Figure 10a. The proposer v_i is honest and it proposes block b. At time $T+\Delta$, all honest validators receive b and will send an attestation by setting the *head* field as b. At time $T+2\Delta$, all honest aggregators receive the attestations, aggregate them, and send them to all validators. At time $T+3\Delta$ (the end of slot t), all honest validators receive them, so AA_t is formed. In contrast, if the proposer v_i is Byzantine (Figure 10b) and does not propose *on time* or does not extend the canonical chain, AA_t will not be formed.



(a) In a synchronous network, a block b proposed by an honest validator always receives an AA_t.



(b) If a block b is delayed, AA_t cannot be formed.

Figure 10: Motivation of available attestation.

Remark. Our solution is reorg-resilient only when the network becomes synchronous. We argue that providing such a solution is already meaningful. First, as mentioned in the introduction, it is impossible to distinguish faulty validators from slow validators, so reorg resilience cannot be achieved. Second, most of the time, the Ethereum network works synchronously. According to Ethereum network statistics[7], the average *participation rate* reaches 99%, where all the messages are received on time. In particular, the average participation rate is calculated over the entire operational period of Ethereum 2.0 since September 2022. During this period, the lowest participation rate was 96.3% (on May 12, 2023) and the standard deviation has been low. As summarized in Table 1, most malicious reorganization attacks on Ethereum are launched when the network is synchronous. Therefore, our solution is already meaningful. Finally, our solution is both safe and live under the partially synchronous model.

6 Our Modified PoS Protocol

We are now ready to present our modified PoS protocol. In this section, we first introduce the notations and the definitions and then show the workflow in detail.

6.1 Notations and Definitions

Recall that the validators are divided into 32 disjoint committees randomly. Since the committees are sampled pseudorandomly, the fraction of Byzantine validators in each committee follows a binomial distribution. Namely, if n is large enough, each committee has at most one-third of Byzantine validators with an overwhelming probability. While we provide detailed analysis in Section 9, we simplify the description. In particular, we use ϑ to denote the number of Byzantine validators in each committee and p as the desirable failure probability (i.e., the probability that the number of Byzantine validators in a committee is greater than ϑ).

Definition 1 (*stable* block)**.** *A block b proposed in slot t is a stable block, if b includes at least $\vartheta+1$ attestations for b' in slot $t-1$, where b' is the parent block of b.*

Definition 2 (*unstable* block)**.** *Block b is an unstable block if b is not a stable block.*

Definition 3 (*stable* chain)**.** *The chain c is a stable chain, if the leaf block of c is a stable block.*

Remark. We do not introduce the concept of *weight* in this paper. In Ethereum, the weight of an attestation denotes the fraction of the stake of the attestor. Accordingly, a block b is stable if the weight of attestations included in b (for the parent block b') is higher than ϑ', where ϑ' is the fraction of stake owned by Byzantine validators.

[6] For simplicity, we assume that all validators attest in every slot in this section.

[7] Data source (accessed August 2024): https://beaconcha.in/charts/participation_rate.

Modified Ethereum PoS protocol for validator v_i.

global parameter: slot counter t.
local parameters: block tree $\mathcal{T}$, attestation pool $\mathcal{P}$.

```
01 upon a slot t start
02   as the proposer for slot t
03     if AA_{t-1} exist
04       let p be AA_{t-1} block (randomly if two)
05     else let p be the output of ForkChoice
06     obtain a set of newly received attestations atts from P
07     obtain a batch of transactions txs
08     let u be the latest unstable block (None if no such block exists)
09     create a forwarding info fi =(Info, t, v_i, H(p), H(atts,txs))
10     create block b =(Propose, fi, atts, txs, H(u))
11     send b to all validators
12 upon Δ seconds of slot t
13   as the attestor for slot t
14     let h be the output of ForkChoice
15     let s be the last justified checkpoint
16     let c be the the last checkpoint
17     create message att =(Attest, t, v_i, H(h), H(s), H(c), h.fi)
18     send att to committee
19 upon 2Δ seconds of slot t
20   as the aggregator for slot t
21     let Atts be the attestations of slot t in P
22     aggregate Atts as message agg with type (Aggregate)
23     send agg to all validators
24 upon receiving block b = (Propose, fi, atts, txs)
25   add block b into T
26   if b includes at least ϑ+1 attestations in slot t'-1 for b.parent
27     set b as stable
28     if b is a checkpoint
29       update checkpoint status in the chain led by parent of b
30   else set b as unstable
31 upon receiving message m with type (Attest) & (Aggregate)
32   add message m into P
33 function ForkChoice
34   let root be the last justified checkpoint
35   let C be set of stable chains extending root that are not filtered
36   let result ← nil
37   for c in C
38     if c is longer than result
39       result ← c
40     else if c is as long as result
       and leaf block of c is later than leaf block of result
41       result ← c
42   return result
```

Figure 11: The modified protocol workflow. Changes on top of Figure 3 are highlighted using blue. $H()$ denotes the hash function.

6.2 Protocol Workflow

We show the workflow of the modified Ethereum PoS protocol in Figure 11. The changes on top of the protocol in Figure 3 are highlighted in blue.

- If a stable block p with AA_{t-1} exists, v_i sets the *parent* field as p. If two AA_{t-1} blocks exist, v_i breaks the tie by randomly selecting one. Otherwise, v_i uses the output of function ForkChoice as the parent (lines 3-5 in Figure 11). Note that there exists a case where a Byzantine validator v_j proposes two conflicting blocks in slot t and both blocks form AA_{t-1}. Upon receiving such blocks, v_i still randomly selects one block. The proposer of the two blocks will eventually be slashed.
- Upon receiving a proposed block b in slot t', validator v_i includes b in its block tree $\mathcal{T}$ and checks whether b is a stable block, i.e., b includes at least $\vartheta+1$ attestations in slot $t'-1$ (the *head* of the attestations is the parent of b). If so, v_i processes the stable block. Otherwise, block b is an unstable block (lines 24-30 in Figure 11).
- We replace the HLMD GHOST with the *longest chain* as the fork choice rule. Our longest chain fork choice rule is very simple: it outputs the head of the chain with the most stable blocks. In case of a tie, the longest chain fork choice rule chooses the chain such that the leaf block has the largest slot number. (lines 33-42 in Figure 11).
- We introduce a *forwarding* rule for the block proposals. The forwarding rule modifies the data structure of the block proposal and the attestation. In particular, proposer v_i creates a digital signature for $(t, v_i, H(p), H(atts, txs))$, where $H(p)$ is the hash of the parent block, $H(atts, txs)$ is the hash of the attestations and proposed transactions. The tuple with the signature is called *forwarding info* fi (line 9 in Figure 11). When a proposer sends a proposed block, it sends (Propose, $fi, atts, txs$) to all validators (line 10 in Figure 11). Meanwhile, upon receiving a proposed block, each attestor includes fi in its attestation and sends it to the committee (line 17 in Figure 11). If validator v_i receives valid forwarding info fi (with digital signatures from some proposer) but has not received the block yet, it obtains the original block proposal from other validators. Then, v_i processes the block according to the protocol.
- We additionally have an u field in a block b. Each proposer sets its u field as the hash of an unstable block. If there is no such unstable block, set u as *None*. The transactions in the unstable blocks should not conflict with the transactions in the longest chain. The transactions in u can also be finalized once block b (that includes u) is finalized (line 8 in Figure 11).

The longest chain fork choice rule. We provide an example of our new longest chain fork choice rule in Figure 12. The tree root is the last justified checkpoint. Three chains extend the root, namely c_1, c_2, and c_3 (line 35 in Figure 11). Our rule selects the chain with the most number of stable blocks, i.e., the chain led by block b_2 (lines 37-40 in Figure 11).

Note that unstable blocks are not considered in our new fork choice rule. When the network is synchronous, we can in fact directly ignore unstable blocks. To deal with network

asynchrony, we still need to consider unstable blocks. In our protocol, unstable blocks can be included in the blocks if the transactions in unstable blocks do not conflict (i.e., b_4 will in the u field of some block in c_2 (lines 8 in Figure 11)).

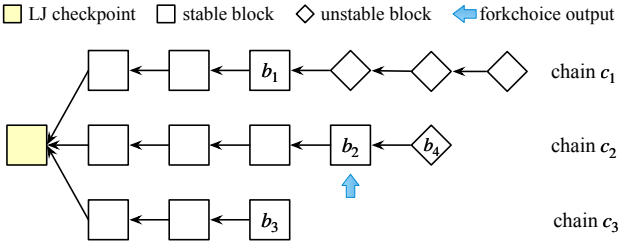


Figure 12: The longest chain fork choice. LJ checkpoint is an abbreviation for the last justified checkpoint.

The forwarding rule. We use the forwarding rule to prevent Byzantine validators from partial withholding blocks. Assume that a block b from the adversary is only released to honest validators v_i in slot t. v_i will release attestations including the forwarding info of b. All honest validators receive the forwarding info by the end of slot t. Even if some honest validators may not have received b directly, they still add b in their block tree (see Lemma 1 for details).

Sketch of correctness. While we provide security analysis in Section 7 and Appendix A, we briefly discuss why our modified protocol: 1) achieves reorg resilience when the network is synchronous; and 2) achieves standard safety and liveness properties in a partially synchronous network.

The reason why our approach is reorg resilient is that the AA mechanism *prevents* Byzantine validators from creating conflicting branches. As summarized in Section 4, there are two strategies for Byzantine validators: (1) Byzantine validators directly propose a block conflicting with the canonical chain and (2) Byzantine validators propose a block that extends the canonical chain and delays releasing the block. None of the strategies work anymore after AA is used. For the first type, only the leaf blocks in the block tree can be the output of the fork choice rule. If a Byzantine validator tries to create a block b_1 that extends a block b_0 that is not a leaf block, b_0 will receive no attestations from honest validators. Thus, block b_1 is an unstable block. For the second type, our approach ensures that if Byzantine validators withhold at least two stable blocks, one of them must have already been observed by *all* honest validators (see Lemma 2 for details).

Now it becomes clear why we use the *longest chain* rule to replace the HLMD GHOST rule. Informally, as HLMD GHOST rule determines the canonical chain based on the weight of the blocks and the weight is determined by the number of attestations, the HLMD GHOST rule cannot prevent the adversaries from withholding their attestations.

Our modified protocol can achieve the safety and liveness properties of the consensus protocol. Safety still holds since we do not modify Casper, the finality gadget protocol. Liveness is achieved after GST mainly because our protocol is reorg resilient in a synchronous network. As all honest validators consider the blocks proposed by honest validators the longest chain, their attestations will be considered valid by all honest validators so eventually some block is finalized.

Overhead. Our approach does not introduce additional computation and only introduces little overhead for communication. In particular, the only change that will affect the communication is the forwarding rule, where each attestor includes the forwarding information in its attestations. The forwarding information includes a slot number, an identifier of the validator, and two hashes. The length of the forwarding information is thus the same as each attestation.

7 Security Analysis

Theorem 1. *The modified Ethereum PoS protocol satisfies reorg resilience in the synchronous model and satisfies safety and liveness properties in the partially synchronous model.*

Due to space constraints, we prove reorg resilience in this section and discuss safety and liveness in Appendix A.

Lemma 1. *When the network is synchronous, if an honest attestor votes for a block b in slot t, by the end of slot t, all honest validators receive b.*

Proof. Let T be the time slot t begins. Assume that an honest attestor v_i votes for b at $T+\Delta$. According to the protocol, v_i forwards the message to all committee members in slot t. At time $T+2\Delta$, any correct aggregator receives the forwarding info fi for b and sends fi to all validators. All honest validators receive fi by time $T+3\Delta$. Accordingly, at the end of slot t, all honest validators receive b. □

Lemma 2. *If the network is synchronous and two consecutive stable blocks are withheld by Byzantine validators, at least one of them must have been received by all honest validators before they are released.*

Proof. Towards a contradiction, consider two blocks b_1 and b_2 that are withheld by two Byzantine validators v_i (for slot t) and v_j (for slot k), respectively. Furthermore, b_2 extends b_1. Since b_2 is a stable block, b_2 consists of $\vartheta+1$ attestations for block b_1. At least one honest validator votes for b_1. According to Lemma 1, all honest validators receive b_1 by the end of slot $k-1$. □

Lemma 3. *When the network is synchronous, no branch can justify a new checkpoint earlier than the longest chain of any honest validator.*

Proof. Towards a contradiction, assume that, in slot t, there exists a chain c_0 that justifies a new checkpoint cp_0 while the longest chain of any honest validator does not, i.e., chain

c_0 conflicts with the longest chain of all honest validators. According to the protocol, to justify checkpoint cp_0 in slot t in chain c_0, a new checkpoint cp_1 is proposed in slot t in chain c_0 (line 28 in Figure 11).

We prove that c_0 is not the longest chain of any honest validator in slot $t-1$. We denote the longest chain as chain c_1. As cp_1 is the latest block and c_0 is not the longest chain of any honest validator in slot t, c_0 is shorter than chain c_1 in slot t. By Lemma 2, two consecutive blocks can not both be withheld as at least one of them must have been received by all honest validators. Therefore, before slot $t-1$ ends, at most one block b_0 is withheld in chain c_1. In slot t, b_0 may be released in c_1. As cp_1 is released in c_0 in slot t, c_1 is still longer than c_0 in slot $t-1$. Therefore, c_0 can not be the longest chain of any honest validators in slot $t-1$ and no honest validator votes for blocks in c_0 in slot $t-1$, a contradiction with the assumption that cp_1 is a stable block in slot t that justifies a new checkpoint on c_0. □

Lemma 4. *When the network is synchronous, a stable block proposed by an honest proposer in a slot t is the output of the longest chain rule of any honest validator by the end of slot t.*

Proof. Let b be the block proposed by the honest proposer in slot t. The proof consists of two parts. First, by the end of slot t, all honest validators receive b and add b to their block tree. Second, by the end of slot t, the chain led by b is the longest chain for *all* honest validators.

We begin with the first part. Let T be the time slot t begins. As the network is synchronous, by time $T+\Delta$, all honest attestors receive b and send their attestations. By Lemma 1, all honest validators receive b at the end of slot t.

We now prove the second part. Towards a contradiction, suppose that the chain led by b is not the longest chain of all honest validators. There are three cases: 1) there exists a longer chain c' before slot t ends; 2) there exists a chain c' before slot t ends and c' has the same length as the chain led by b; 3) there exists a chain c' that justifies some checkpoint that is higher than the last justified checkpoint of c. In the first case, for c' to be longer, its last two blocks must have been withheld by the adversary before slot t, a contradiction with Lemma 2. In the second case, as the slot number of block b is larger than the leaf block of chain c', block b is the output of the longest chain according to our protocol (line 37 in Figure 11). Finally, the third case is a violation of Lemma 3.

The lemma thus holds. □

Theorem 2. *When the network is synchronous, once an honest validator proposes a stable block b, all subsequent stable blocks extend this chain. A block by the Byzantine validators that does not extend b will not be considered valid by honest validators.*

Proof. Let t be the slot when an honest validator proposes a stable block b on chain c. We prove the lemma by induction on k, where k is the number of slots after t and $k > 0$.

(1) Base case ($k = 1$): By Lemma 4, b will become the head of the longest chain for all honest validators. All honest validators will vote for b by the end of slot t. If the proposer of slot $t+k$ is honest, it proposes a block that extends b. If the proposer of slot $t+k$ is Byzantine and proposes a block that does not extend b, the block will not be considered valid by an honest validator as honest attestors only vote for a block that extends its longest chain.

(2) Induction step: By the induction hypothesis, all blocks from slot $t+1$ to $t+k$ extend chain c. Following a similar argument as above, the theorem holds.

□

Theorem 3. *When the network is synchronous, the finalized chain is the prefix of the longest chain of any honest validator.*

Proof. Towards a contradiction, we assume there exists a finalized checkpoint block cp that conflicts with the longest chain by some honest validators. For a checkpoint to be finalized, cp must have previously been justified. When checkpoint cp was justified, at least $f+1$ honest validators vote for cp (by setting cp as *target*). This implies that $f+1$ honest validators, at the time of voting (e.g., slot t), consider the chain containing block cp the longest chain. Let the latest head of the chain by any such honest validator be b'. According to Lemma 1, all honest validators receive b' by the end of slot t. It is then not difficult to see that all honest validators consider the chain led by b' the longest chain by Theorem 2. □

We are now ready to prove reorg resilience (i.e., Theorem 1).

Proof. We first show that any block from an honest validator will be included in the longest chain of any honest validator. Suppose an honest proposer proposes a stable block b in slot t. According to Lemma 4, b is the output of the longest chain by any honest validator at the beginning of slot $t+1$. In addition, according to Theorem 2, after b is proposed, any withheld blocks will never become the longest chain by any honest validators. Therefore, b is always in the longest chain at slot $t' > t$. Finally, Theorem 3 shows that the longest chain by any honest validator does not conflict with the finalized chain. Therefore, block b will be finalized. □

8 Implementation and Evaluation

Implementation. We implement our modification on top of *Prysm*[8], one of the most widely adopted Ethereum 2.0 beacon chain implementations written in Golang. The Prysm version is v5.0 (the newest version as of Aug 2024). We have made our code available[9]. Our codebase modifies around 1,000

[8] Prysm: https://github.com/prysmaticlabs/prysm

[9] Implementation of our modified protocol and the attacks: https://zenodo.org/records/14760370

LOC. Additionally, we implement approximately 5,000 LOC for evaluation.

Evaluation. We evaluate the performance of our protocols on Amazon EC2 using one virtual machine. We use m5.xlarge instances for our evaluation. The m5.xlarge instance has four virtual CPUs and 16GB of memory. We deploy our protocols in the LAN setting to provide better synchronous network conditions for our evaluation. We use 16,384 validators for both the vanilla Ethereum PoS protocol and our modified protocol. Among these validators, 5,461 validators are Byzantine validators. In addition, we set ϑ as 234 to achieve reorg resilience with probability $1 - 10^{-9}$ (the reason why we choose the value can be found in Section 9).

Our evaluation seeks to answer two questions. First, is our approach reorg resilient in practice (despite the fact that our approach is already provably secure)? Second, how much overhead does our approach introduce compared to the vanilla protocol? To answer the first question, we implement five reorg attacks shown in Section 4 and calculate the number of reorganized blocks by honest validators of both protocols. To answer the second question, we evaluate the throughput and latency of our approach and compare it with vanilla Prysm. For throughput, we focus on the failure-free case with no Byzantine validators. For latency, we focus on the computation time (e.g., block generation and attestation generation) in both protocols. The overhead created by our modifications to the protocol is reflected in the throughput.

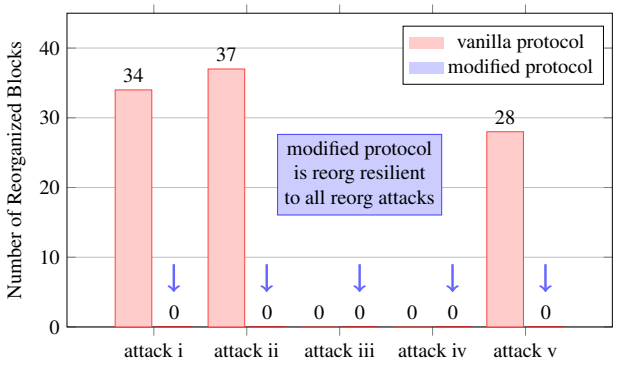


Figure 13: The number of blocks from honest validators that are reorganized out of 500 blocks. Attacks i, ii, iii, iv, and v represent the ex-ante reorg attack, sandwich reorg attack, unrealized justification reorg attack, justification withholding reorg attack, and staircase attack, respectively.

Resilience to reorganization attacks. We implement five reorg attacks, including ex-ante reorg attack, sandwich reorg attack, unrealized justification reorg attack, justification withholding reorg attack, and staircase attack. Although Ethereum has implemented mitigation for most of these attacks, we can slightly modify the attack strategies to make ex-ante reorg attacks and staircase attacks successful. The modified attack strategies for these attacks can be found in Appendix B.

We conduct each experiment for approximately two and a half hours and set the number of blocks from honest validators as 500. In all experiments, we randomly sample around 33.3% Byzantine validators. We then measure the number of reorganized blocks from honest validators. As shown in Figure 13, our modified protocol is reorg resilient to all reorg attacks as the number of blocks from honest validators reorganized in all experiments is zero. In contrast, the vanilla protocol can prevent the unrealized justification reorg attack and the justification withholding reorg attack. However, ex-ante reorg attacks, sandwich reorg attacks, and staircase attacks can still be conducted in the vanilla protocol.

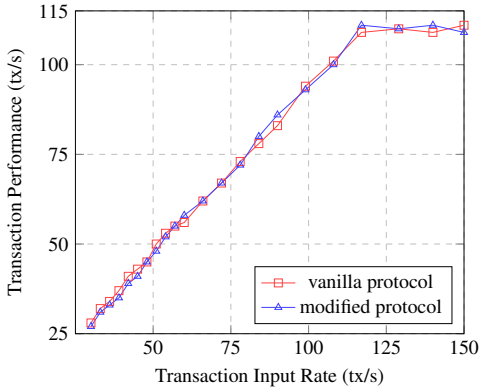


Figure 14: The throughput of the vanilla Ethereum PoS protocol and our modified Ethereum PoS protocol in the failure-free case.

Performance. We show the throughput of our approach in Figure 14. We set the *gas* consumption of each transaction to be 21,000 gwei and we use the standard "transfer" transactions in our evaluation. As transaction volume increases, the throughput in both vanilla Ethereum PoS protocol and modified Ethereum PoS protocol increases, with a peak throughput of around 110 tx/s. This is aligned with the gas limitation of a block (by default 30,000,000). In addition, the throughput of our modified protocol is almost identical to the vanilla protocol.

test items	vanilla	modified	overhead
block generation	6.44ms	6.61ms	-2.6%
block verification	16.49ms	18.28ms	-10.9%
attestation generation	17.39us	17.74us	-2%
attestation verification	17.07us	19.99us	-14.4%

Table 2: Latency breakdown of the vanilla Ethereum PoS protocol and the modified Ethereum PoS protocol.

Latency. We measure the latency breakdown of our approach and compare it with the vanilla protocol. As each slot lasts for a fixed duration, we alternatively assess the latency for

block generation, block validation (the time it takes to verify a received block), attestation generation, and attestation verification (the time it takes to verify an attestation). As shown in Table 2, our approach introduces minimum overhead for the computation of the functions.

Note that we only test our system with 16,384 validators. There may exist some problems when migrating our modified protocol to Ethereum, as Ethereum now has over one million validators. We leave it as future work.

9 Analysis of the Concrete Probability of Achieving Reorg Resilience

So far, we assume that each committee has at most ϑ Byzantine validators so $\vartheta+1$ attestations form a AA. In this section, we analyze the concrete probability.

As shown in Section 3, each committee is composed of $n/32$ randomly selected validators. The distribution of the number of Byzantine validators in each committee can then be modeled as a binomial distribution.

Let n_c be the size of a committee and $f' = f/32$. Our goal is to determine the probability that a randomly selected committee contains no more than ϑ Byzantine validators.

Theorem 4. *Let X be the random variable representing the number of Byzantine validators in a committee. For a large n, X follows a binomial distribution approximated by a normal distribution: $X \sim N(\mu, \sigma^2)$, where $\mu = f'$ and $\sigma^2 = f'(1 - f/n)$*

Proof. The selection of Byzantine validators for a committee follows a binomial distribution with parameters n_c and $p = f/n$. For large n, this binomial distribution can be approximated by a normal distribution with mean $\mu = n_c p = f/32 = f'$ and variance $\sigma^2 = n_c p(1-p) = f'(1-f/n)$, according to the Central Limit Theorem [15]. □

Using this distribution, we can calculate the probability that a committee contains more than ϑ Byzantine validators:

$$P(X > \vartheta) = 1 - \Phi(\frac{\vartheta - \mu}{\sigma}),$$

where Φ is the cumulative distribution function of the normal distribution, $\mu = f'$, and $\sigma = \sqrt{f'(1-f/n)}$.

The key insight is that we can adjust ϑ based on the desired probability $P(X > \vartheta)$ and the total number of validators n. Let the desired failure probability be p. Then, we want to find ϑ such that:

$$P(X > \vartheta) = p.$$

Solving this equation, we get:

$$\vartheta = \lfloor \mu + \sigma \cdot \Phi^{-1}(1-p) \rfloor,$$

where Φ^{-1} is the inverse of the cumulative distribution function of the normal distribution. This formula allows us to calculate the appropriate ϑ for different values of n and desired probabilities p.

n \ p	10^{-6}	10^{-7}	10^{-8}	10^{-9}
2^{14}	0.4316	0.4414	0.4492	0.4570
2^{16}	0.3828	0.3872	0.3916	0.3955
2^{18}	0.3580	0.3603	0.3625	0.3645
2^{20}	0.3457	0.3468	0.3479	0.3489

Table 3: The ratio of ϑ to n_c for different n and p values.

In Table 3, we show some concrete examples of the ϑ/n_c with specific n and the desirable failure probability p. In the parameter setting of Ethereum, the minimum number of validators is 2^{14}. Ethereum now has approximately 2^{20} validators. The maximum desirable probability 10^{-6} means that the protocol fails once every 10^6 slots, i.e., 138 days. If the desirable failure probability is 10^{-9}, malicious reorganization occurs once every 380 years. We can then draw some conclusions below.

(1) For a fixed n, if a lower p is tolerable (i.e., as we require a lower probability of having too many Byzantine validators), the ratio ϑ/n_c increases so we need to set up a larger ϑ in our approach.
(2) For a fixed p, as n increases, the ϑ/n_c is closer to f/n. In this way, we can simply set ϑ as one-third of the committee size.
(3) For a small n and a smaller p (e.g., $p = 10^{-19}$), the value of ϑ/n_c may exceed 0.5. In such cases, reorg resilience is not achieved with overwhelming probability. To mitigate this risk, careful parameter selection is required.

10 Conclusion

Malicious reorganization attacks are known to be a primary threat to the Ethereum Proof-of-Stake (PoS) consensus protocol. In this work, we propose the first provably secure and lightweight solution to achieve reorg resilience. Central to our approach is an available attestation (AA) mechanism, which ensures that validators only vote for a chain with blocks observed by more than one-third of validators. Our evaluation results show that our approach is reorg resilient and does not degrade the performance of the system at all.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under 2022YFB2701700, a research grant from Ethereum Foundation under FY24-1529, the National Natural Science Foundation of China under 92267203, Beijing Natural Science Foundation under M23015, China Postdoctoral Science Foundation under 2023M741949, and Tsinghua Shuimu Scholar.

11 Ethics Considerations and Open Science Policy Compliance

In the section, we show the commitment to ethical research practices and open science principles for our paper on a modified Ethereum protocol to address malicious reorganization attacks. We have carefully considered the ethical implications of our research and strive to conduct our work responsibly.

11.1 Research Ethics Considerations

We are committed to complying with all relevant research ethics considerations. In particular, we are committed to the following principles.

- **Stakeholder Consideration**: We have carefully considered the potential impacts on various stakeholders, including Ethereum users, validators, developers, and the broader cryptocurrency ecosystem.
- **Vulnerability Disclosure**: Our research does not uncover new vulnerabilities but analyzes known malicious reorganization attacks. We will ensure any discussion of vulnerabilities is responsible and does not provide additional exploit information.
- **Live System Testing**: We have not conducted experiments on the live Ethereum network. All experiments were performed using the local testnet.
- **Beneficence**: Our research aims to improve the security and stability of the Ethereum network, providing a net positive benefit to the community. We have carefully weighed potential negative outcomes against these benefits.
- **Respect for Persons**: Our research does not involve human subjects or personal data. We respect the work of other researchers and properly cite all relevant prior work.
- **Justice**: We strive to ensure our proposed modifications do not disproportionately impact or disadvantage any particular group of Ethereum users or validators.
- **Respect for Law and Public Interest**: Our research complies with all applicable laws and regulations. We have considered the broader societal implications of more secure blockchain systems.
- **Potential Dual Use**: We acknowledge that improvements in blockchain security could potentially be used by both legitimate and illicit actors. We focus our work on the reorg resilience and avoid providing any information that could be misused.
- **Team Member Wellbeing**: Our research did not expose team members to harmful or disturbing content. We ensured all team members were comfortable with the nature and scope of the research.
- **Institutional Review**: While our research did not require formal Institutional Review Board (IRB) approval, we have consulted with experts and followed established best practices in blockchain and security research.

We are committed to addressing any ethical concerns that may arise during the review process or after publication. We welcome feedback from the research community on these ethical considerations and are prepared to make adjustments to our approach if necessary.

11.2 Compliance with Open Science Policy

We are committed to the principles of open science, ensuring transparency, reproducibility, and accessibility throughout the research process. We adhere to the following practices:

- **Data Availability**: All experiment data used in our analysis will be made publicly available in a repository upon publication.
- **Code Availability**: Any code for the implementation of our modified protocol and the attacks will be open-sourced under MIT License and shared via Zenodo[10].
- **Reproducibility**: We will provide detailed documentation on our methodology to enable other researchers to reproduce our results.
- **Preprint**: We intend to post a preprint of our paper on ePrint[11] prior to publication to facilitate early feedback and dissemination.

References

[1] Aditya Asgaonkar. Unrealized justification reorgs. https://notes.ethereum.org/@adiasg/unrealized-justification. (accessed in Aug 2024).

[2] Bounce attack resistance #1465. https://github.com/ethereum/consensus-specs/pull/1465. (accessed in Aug 2024).

[3] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In *EC*, pages 459–473, 2019.

[4] Vitalik Buterin. Proposal for mitigation against balancing attacks to lmd ghost. https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation. (accessed in Aug 2024).

[5] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[6] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. *arXiv preprint arXiv:2003.03052*, 2020.

[7] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *TOCS*, 20(4):398–461, 2002.

[9] Confirmation rule prerequisite - fork choice filter change. https://github.com/ethereum/consensus-specs/pull/3431. (accessed in Aug 2024).

[10] Zenodo: https://zenodo.org/

[11] ePrint: https://eprint.iacr.org/

[10] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. Goldfish: No more attacks on ethereum?! In *FC*, 2024.

[11] Francesco D'Amato, Roberto Saltini, Thanh-Hai Tran, and Luca Zanolini. 3-slot-finality protocol for ethereum. *arXiv preprint arXiv:2411.00558*, 2024.

[12] Francesco D'Amato and Caspar Schwarz-Schilling. Proposer boost considerations. https://notes.ethereum.org/@casparschwa/H1T0k7b85. (accessed in Aug 2024).

[13] Francesco D'Amato and Luca Zanolini. Recent latest message driven ghost: Balancing dynamic availability with asynchrony resilience. *arXiv preprint arXiv:2302.11326*, 2023.

[14] Francesco D'Amato and Luca Zanolini. A simple single slot finality protocol for ethereum. In *ESORICS Workshops*, 2023.

[15] Jay L Devore. *Probability and Statistics for Engineering and the Sciences, 8th edition*. Duxbury Press, 2011.

[16] Sisi Duan and Haibin Zhang. Foundations of dynamic bft. In *SP*, pages 1317–1334, 2022.

[17] Sisi Duan, Haibin Zhang, Xiao Sui, Baohan Huang, Changchun Mu, Gang Di, and Xiaoyun Wang. Dashing and star: Byzantine fault tolerance from weak certificates. In *Eurosys*, 2024.

[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *JACM*, 1988.

[19] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[20] Fork choice upgrade. https://github.com/ethereum/consensus-specs/pull/3290. (accessed in Aug 2024).

[21] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn : Seamless high speed bft. In *SOSP*, 2024.

[22] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO*, pages 451–480, 2020.

[23] Dahlia Malkhi and Kartik Nayak. Extended abstract: Hotstuff-2: Optimal two-phase responsive bft. *IACR Cryptol. ePrint Arch*, page 397, 2023.

[24] Ryuya Nakamura. Analysis of bouncing attack on ffg. https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113. (accessed in Aug 2024).

[25] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *SP*, pages 446–465, 2021.

[26] Joachim Neu, Ertem Nusret Tas, and David Tse. Two more attacks on proof-of-stake ghost/ethereum. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 43–52, 2022.

[27] Michael Neuder, Daniel J. Moroz, Rithvik Rao, and David C. Parkes. Defending against malicious reorgs in tezos proof-of-stake. *AFT*, 2020.

[28] Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders. *arXiv preprint arXiv:2102.02247*, 2021.

[29] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake under scrutiny. In *SAC*, pages 212–221, 2023.

[30] Potuz. Justification widtholding attacks. https://hackmd.io/o9tGPQL2Q4iH3Mg7Mma9wQ. (accessed in Aug 2024).

[31] Proposer lmd score boosting #2730. https://github.com/ethereum/consensus-specs/pull/2730. (accessed in Aug 2024).

[32] Danny Ryan. Epoch reorg. https://notes.ethereum.org/VH_B3kEVQFav4roEgYuCjA. (accessed in Aug 2024).

[33] Roozbeh Sarenche, Ertem Nusret Tas, Barnabe Monnot, Caspar Schwarz-Schilling, and Bart Preneel. Breaking the balance of power: Commitment attacks on ethereum's reward mechanism. *arXiv preprint arXiv:2407.19479*, 2024.

[34] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *FC*, pages 560–576, 2022.

[35] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *FC*, pages 507–527, 2015.

[36] Ethereum proof-of-stake consensus specifications. https://github.com/ethereum/consensus-specs. (accessed in Aug 2024).

[37] Michael Sproul. Allow honest validators to reorg late blocks. https://github.com/ethereum/consensus-specs/pull/3034. (accessed in Aug 2024).

[38] Update proposer_score_boost to 40 percent #2895. https://github.com/ethereum/consensus-specs/pull/2895. (accessed in Aug 2024).

[39] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC*, 2019.

[40] Qianyu Yu, Giuliano Losa, and Xuechao Wang. Tetrabft: Reducing latency of unauthenticated, responsive bft consensus. *arXiv preprint arXiv:2405.02615*, 2024.

[41] Mingfei Zhang, Rujia Li, and Sisi Duan. Max attestation matters: Making honest parties lose their incentives in ethereum pos. In *USENIX Security*, 2024.

A Safety and Liveness Proof

In this section, we provide a formal safety and liveness proof for our modified protocol, as presented in Section 7.

A.1 Safety

Our modified protocol does not modify the Casper FFG [5] in vanilla Ethereum PoS protocol. Thus, our protocol can achieve the same safety guarantee as Casper FFG. In this section, we prove the correctness of the protocol using the notions used in Casper FFG.

We first introduce the slashing conditions defined in Casper FFG. According to these conditions, any validators found in violation will be slashed, meaning their entire deposits are forfeited and subsequently removed from the system. Casper FFG operates under the

assumption that the fraction of validators susceptible to slashing is less than one-third of the total validators, a condition referred to as *1/3-slashable* [6].

We now formalize the slashing condition, justified checkpoint, and finalized checkpoint. To give a formal definition, we expand the attestation notion as (s,t,h), where s, t, and h represent the *source* checkpoint, the *target* checkpoint, and *head* block, respectively. As we only consider the checkpoints in attestations, we only consider (s,t) as an attestation. In addition, we use $\mathsf{ep}(cp)$ to represent the epoch of the checkpoint cp.

There are two slashing conditions:

- **(I-Double Voting)** A validator can not publish two distinct votes for the same target epoch. Formally, for any two attestations (s_1,t_1) and (s_2,t_2), $\mathsf{ep}(t_1) \neq \mathsf{ep}(t_2)$.
- **(II-Surround Voting)** A validator can not vote within the span of its other votes. Formally, for any two attestation (s_1,t_1) and (s_2,t_2), $\mathsf{ep}(s_1) < \mathsf{ep}(s_2) < \mathsf{ep}(t_2) < \mathsf{ep}(t_1)$ can not hold.

Definition 4 (Justified Checkpoint)**.** *A checkpoint cp is justified if at least two-thirds of validators attest (s,cp), where s is a justified checkpoint.*

Definition 5 (Finalized Checkpoint)**.** *A checkpoint cp is a finalized checkpoint if (1) cp is a justified checkpoint, (2) at least two-thirds of validators attest (cp,t), and (3) the consecutive checkpoints between cp and t are all justified.*

Lemma 5. *If 1/3-slashable holds, each justified checkpoint cp has a unique epoch number $\mathsf{ep}(cp)$.*

Proof. Towards a contradiction, assume that two justified checkpoints cp_1 and cp_2 have the same epoch number e. At least $n-f$ validators have sent attestations by setting *target* as cp_1. Meanwhile, at least $n-f$ validators have sent attestations by setting *target* as cp_2. Since there are at most f Byzantine validators, at least one validator has sent both an attestation with cp_1 as *target* and an attestation with cp_2 as *target*, a contradiction. □

Theorem 5 (Safety)**.** *If 1/3-slashable holds, two conflicting checkpoints can not both be finalized.*

Proof. Towards a contradiction, assume that there exist two conflicting finalized checkpoints, cp_1 and cp_2. WLOG, we assume that $\mathsf{ep}(cp_1) < \mathsf{ep}(cp_2)$ ($\mathsf{ep}(cp_1) \neq \mathsf{ep}(cp_2)$ by Lemma 5). As checkpoint cp_1 is finalized, at least two-thirds of validators have sent an attestation for (cp_1,cp_1') (where cp_1 is *source* and cp_1' is *target*). By Definition 5, as cp_1 is finalized, the consecutive checkpoints between cp_1 and cp_1' are all justified. By Lemma 5, $\mathsf{ep}(cp_1') < \mathsf{ep}(cp_2)$. As checkpoint cp_2 is finalized, checkpoint cp_2 is justified, at least $n-f$ validators have sent an attestation (s,cp_2) (where s is *source* and cp_2 is *target*). By Lemma 5, $\mathsf{ep}(s) < \mathsf{ep}(cp_1)$. As $\mathsf{ep}(cp_1) < \mathsf{ep}(cp_1')$, at least one honest validator has sent an attestation (cp_1,cp_1') and an attestation (s,cp_2) such that $\mathsf{ep}(s) < \mathsf{ep}(cp_1) < \mathsf{ep}(cp_1') < \mathsf{ep}(cp_2)$, a contradiction. □

A.2 Liveness

The vanilla Ethereum PoS protocol achieves plausible liveness and probabilistic liveness [5]. The two properties are not very formally defined. Informally, plausible liveness means that it is always possible to finalize a block and some blocks always extend the finalized chain. Probabilistic liveness means that it is possible that some block is finalized. Notably, there is a gap between plausible liveness/probabilistic liveness and the conventional liveness property, also as mentioned by the Gasper paper [5].

We now show that our modified protocol achieves the conventional liveness property.

Lemma 6. *If the network is asynchronous and a block b is AA_t in a slot t, all honest validators receive b after GST.*

Proof. Assume that block b is proposed by validator v_i. As block b is AA_t, an honest validator v_j receives b and votes for b in slot t. According to the protocol, v_j forwards its attestation $att = (\textsc{Attest}, t, v_i, H(b), H(s), H(c), b.fi)$ to all validators, where s and t are two checkpoints, and $b.fi$ is the forwarding information for block b. After reaching GST, all honest validators receive att. The forwarding info $b.fi$ is also received by all honest validators. Accordingly, all honest validators receive b after GST. □

Lemma 7. *After GST, if two consecutive stable blocks are withheld by Byzantine validators, at least one of them must have been received by all honest validators before they are released.*

Proof. The proof is similar to that of Lemma 2, by replacing Lemma 1 with Lemma 6 in the argument. □

Lemma 8. *After GST, no branch can justify a new checkpoint earlier than the longest chain of any honest validator.*

Proof. The proof is similar to that of Lemma 3, by replacing Lemma 2 with Lemma 7 in the argument. □

Theorem 6 (Liveness)**.** *The length of the finalized chain eventually grows for all honest validators.*

Proof. We first show that once an honest validator proposes a block b after GST, all honest validators will set the chain led by b as the longest chain. Let b be proposed in slot t. The proof consists of two parts. First, by the end of slot t, all honest validators receive b and add b to their block tree. Second, block b is the longest chain for *all* honest validators.

We begin with the first part. As the network is synchronous, let T be the time slot t begins. By time $T+\Delta$, all honest attestors receive b and send their attestations. By time $T+2\Delta$, all honest aggregators receive the attestations and then aggregate the attestations. By time $T+3\Delta$ (i.e., the end of slot t), all honest validators receive the attestations. Accordingly, all honest validators receive block b.

We now prove the second part. Towards a contradiction, suppose that the chain led by b is not the longest chain by all honest validators. There are two cases: 1) there exists a longer chain c' before slot t ends; 2) there exists a chain c' before slot t ends and c' has the same length as the chain led by b. In the first case, for c' to be longer, its last two blocks must have been withheld by the adversary before slot t, a contradiction with Lemma 7. In the second case, the slot number of block b must be larger than the leaf block of chain c' since block b is the latest block. Therefore, block b^* is the output of the longest chain of all honest validators according to our protocol (line 37 in Figure 11). So this case is impossible.

By Lemma 8, block b is not filtered by the fork choice. Thus, all honest validators propose blocks and attest attestations on the chain led by b. By Theorem 1, blocks will eventually be finalized. The finalized chain grows. □

B Adversary Strategy

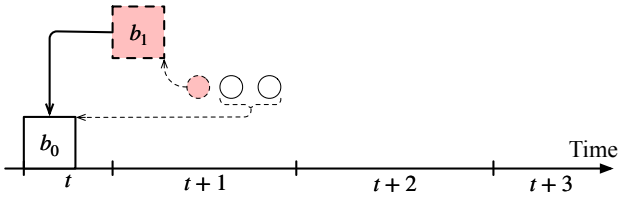
In this section, we provide the attack strategies of our modified ex-ante reorg attack and staircase attack, so the mitigation provided by Ethereum does not work anymore. We implement these strategies in our implementation (as mentioned in Section 8).

B.1 Ex-ante Reorg Attack

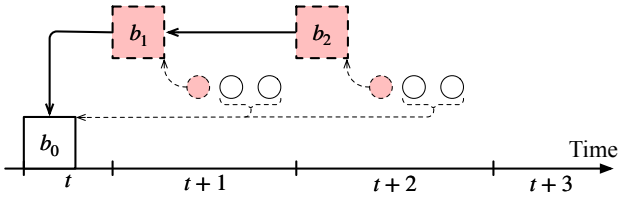
The modified ex-ante reorg attack assumes that the proposers in two consecutive slots are Byzantine validators. Block b_0 is proposed by an honest validator in slot t. The strategies are summarized below (as illustrated in Figure 15):

(1) (slot $t+1$) As shown in Figure 15a, Byzantine proposer v_i withholds its block b_1. Block b_1 extends block b_0. All Byzantine attestors in slot $t+1$ vote for b_1 and withhold their attestations.
(2) (slot $t+2$) As shown in Figure 15b, Byzantine proposer v_j withholds its block b_2. Block b_2 extends block b_1. All Byzantine attestors in slot $t+2$ vote for b_2 and withhold their attestations.
(3) (slot $t+3$) As shown in Figure 15c, after an honest validator v_l proposes its block b_3, Byzantine proposers v_i and v_j release blocks b_1 and b_2. Meanwhile, Byzantine attestors in slots $t+1$ and $t+2$ release their withheld attestations.

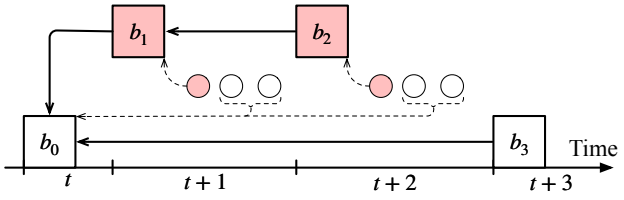
After the attack is conducted, block b_3 is orphaned.



(a) Step 1: Byzantine validators withhold b_1 and their attestations in slot $t+1$.



(b) Step 2: Byzantine validators withhold b_2 and their attestations in slot $t+2$.



(c) Step 3: Byzantine validators release all withheld messages.

Figure 15: The modified ex-ante reorg attack.

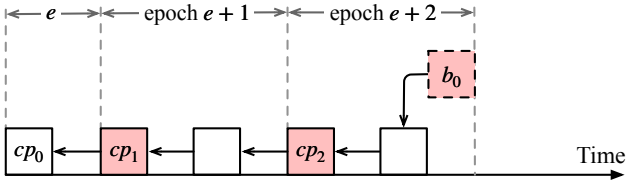
B.2 Staircase Attack

The modified staircase attack assumes that the proposers in the first slots of two consecutive epochs are Byzantine validators. Checkpoint cp_0 is the last justified checkpoint in epoch e. The strategies are summarized below (as illustrated in Figure 16):

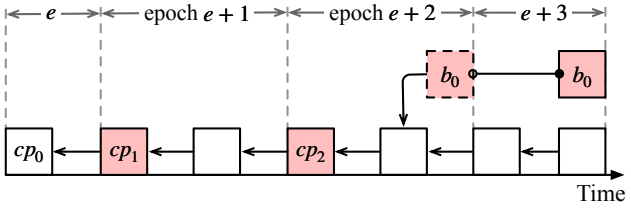
(1) (epoch $e+1$) As shown in Figure 16a, Byzantine proposer v_i delays its checkpoint block cp_1 for one slot. Checkpoint cp_1 extends the canonical chain. All Byzantine attestors in epoch $e+1$ withhold their attestations.
(2) (epoch $e+2$) As shown in Figure 16b, Byzantine proposer v_j delays its checkpoint block cp_2 for one slot. Checkpoint cp_2 extends the canonical chain. All Byzantine attestors in epoch $e+2$ withhold their attestations. The *last* Byzantine proposer v_l (all proposers in the rest of epoch $e+2$ are honest) extends the canonical chain and withholds its block b_0. Block b_0 includes all attestations from Byzantine attestors in epoch $e+2$.
(3) (epoch $e+3$) As shown in Figure 16c, all Byzantine attestors in epoch $e+3$ withhold their attestations. Just before the epoch $e+3$ ends, Byzantine proposer v_l releases its block b_0.

After the attack is conducted, all blocks from honest validators in epoch $e+3$ are orphaned.

(a) Step 1: Byzantine validators delay checkpoint cp_1 for one slot. All Byzantine attestors in epoch $e+1$ withhold their attestations.



(b) Step 2: Byzantine validators delay checkpoint cp_2 for one slot and withhold their attestations in epoch $e+2$. The attestations are included in block b_0. Block b_0 is withheld by Byzantine validators.



(c) Step 3: Byzantine validators withhold their attestations in epoch $e+3$. Block b_0 is released at the end of epoch $e+3$.

Figure 16: The modified staircase attack.