Three Attacks on Proof-of-Stake Ethereum

No Author Given

No Institute Given

Abstract. Recently, two attacks were presented against Proof-of-Stake (PoS) Ethereum: one where short-range reorganizations of the underlying consensus chain are used to increase individual validators' profits and delay consensus decisions, and one where adversarial network delay is leveraged to stall consensus decisions indefinitely. We provide refined variants of these attacks, considerably relaxing the requirements on adversarial stake and network timing, and thus rendering the attacks more severe. Combining techniques from both refined attacks, we obtain a third attack which allows an adversary with vanishingly small fraction of stake and no control over network message propagation (assuming instead probabilistic message propagation) to cause even long-range consensus chain reorganizations. Honest-but-rational or ideologically motivated validators could use this attack to increase their profits or stall the protocol, threatening incentive alignment and security of PoS Ethereum. The attack can also lead to destabilization of consensus from congestion in vote processing.

1 Introduction

The Proof-of-Stake (PoS) Ethereum consensus protocol [1,2,4] is constructed by applying the finality gadget Casper FFG [6] on top of the fork choice rule LMD GHOST, a flavor of the Greedy Heaviest-Observed Sub-Tree (GHOST) [19] rule which considers only each participant's most recent vote (Latest Message Driven, LMD). Participants with stake that allows them to vote as part of the protocol are called *validators*. A slightly simplified and analytically more tractable variant of PoS Ethereum is given by the Gasper protocol [7].

Recent works [18,17,16] have presented two attacks on Gasper and PoS Ethereum. The first attack [18] uses short-range reorganizations (*reorgs*) of the blockchain stipulating consensus to delay finality of consensus decisions. Such short-range reorgs also allow validators to increase their earnings from participating in the protocol (*e.g.*, from Maximal Extractable Value, MEV [10]). As a result, honest-but-rational validators will deviate from the protocol, threatening the assumptions underlying the security arguments for it. In the second attack [17,16], the adversary exploits adversarial network delay and strategic voting by a vanishing fraction of adversarial validators to stall the protocol indefinitely.

Our Contributions In this paper we present enhanced variants of the above two attacks [18,17]. First, we reduce the number of validators necessary to launch a

short-range reorg. An adversary who could perform a reorg of k blocks (k-*reorg*) using the old strategy [18] is now able to perform a $(k+1)$-reorg using our new strategy. Second, we considerably relax the network assumption under which the adversary can stall PoS Ethereum using techniques similar to [17,16]: we show that the adversary does not need to exert control over message propagation delays, but that merely stationary *probabilistic* network delay, as is commonly assumed to model networks under normal operation, together with a still vanishingly small (albeit slightly larger than before) fraction of adversarial validators suffices for the adversary to be able to effectively stall the protocol. We then combine techniques from both refined attacks to devise a long-range reorg attack which requires only an extremely small number of adversarial validators and no adversarial (but only probabilistic) network delay.

This third attack is particularly severe for PoS Ethereum for three reasons: 1. Honest-but-rational validators might adopt the strategy as they can use it to increase their payouts from MEV and transaction fees. The resulting protocol deviations destabilize consensus on both the fork choice and the finality gadget level because the blockchain does not grow steadily anymore. 2. Reorgs lead to uncertainty and delay in block confirmation, impacting user experience and quality of service, and undermining users' trust in the protocol. 3. Reorgs can reduce the throughput of the consensus layer to the point where not enough votes can be processed timely, reducing resilience against adversarial validators and jeopardizing proper functioning of PoS Ethereum.

Related Work In both selfish mining [11] and our attacks the adversary withholds blocks to displace honest blocks from the chain. Unlike selfish mining however, our attacks do not lead to an increased block production reward. Undercutting attacks [12] showcase how consensus instability can arise from reorgs incentivized by large variance in block rewards. In fact, this concern will be aggravated by diminishing block rewards in Bitcoin in the future [9]. Time-bandit attacks [10] point out that MEV earned in past blocks can incentivize and subsidize reorgs and other attacks in the future, *e.g.*, for renting hash power or bribing validators.

Outline PoS Ethereum and its network model are reviewed in Section 2. Sections 3 and 4 each first introduce a recent attack and then describe our refined variant thereof. Combining techniques from our refined attacks, we devise a long-range reorg attack in Section 5. We discuss in Section 6 the impact of the presented long-range reorg attack on various aspects of PoS Ethereum.

2 Proof-of-Stake Ethereum: The Gasper Protocol

We provide a concise summary of the PoS Ethereum/Gasper protocol and the network environment it is designed for. The exposition is slightly idealized and streamlined for ease of comprehension. For all details, refer to the paper [7] of Gasper and the PoS Ethereum beacon chain protocol specifications [2,4,1].

2.1 Model

We assume a static pool of N protocol participants (called *validators* or *nodes*), each with unit stake. This corresponds to consensus in a *permissioned* setting. Network communication among validators is synchronous, *i.e.*, network delay is under adversarial control, up to a known delay upper bound Δ. Clocks across nodes are synchronized. This amounts to a *synchronous network* [13]. There is an external shared source of randomness which can be used by the protocol to sample a group (of predetermined size) of validators in a uniform manner without replacement. Validators follow the protocol as prescribed, except for a fraction β which are under adversarial control and can deviate from the protocol in arbitrary and coordinated fashion (*Byzantine faults*).

In its basic version, the state machine replication (SMR) formulation of consensus asks for a protocol that can be run among the N protocol participants to obtain a linear ordering of *transactions* input by the environment to participants, into a shared *ledger* (*i.e.*, to implement an ordering service) with the following security properties:

- *Liveness:* If some honest validator becomes aware of a transaction, then not too long thereafter that transaction will have entered the ledger as output by any honest validator (*i.e.*, 'good things do happen', 'transactions enter the ledger').
- *Safety:* The ledgers output by different honest validators at different points in time are consistent. In other words, it does not happen that a transaction, which has once entered the ledger in some honest validator's view at some time, disappears later (*i.e.*, 'bad things do not happen', 'if a transaction enters the ledger, then it will not leave it').

Given an SMR protocol, we seek to understand for which adversarial fractions β the ledger output by that protocol is both safe and live (and hence *secure*).

2.2 Protocol

Being a composite with the LMD GHOST fork choice rule as the basis and Casper FFG as a finality gadget on top, PoS Ethereum consensus proceeds roughly in two stages and on two time scales.

First, on the smaller time scale where LMD GHOST operates, time proceeds in synchronized slots of duration 2Δ. For each slot, one *block proposer* and a *committee* of W validators is drawn uniformly at random from the N validators. The following LMD GHOST rule is used to determine a canonical block (and its prefix of blocks as a canonical chain) in a node's view in slot t: "Starting at the highest block b_0 'justified' by Casper FFG (see below), sum for each child block b the number of unique (*i.e.*, one per slot and slot's committee member, breaking ties adversarially) valid (*i.e.*, only from earlier than the current slot, and no voting on future blocks) votes for that block and its descendants; count for every validator only its most recently cast vote (LMD). Pick the child block b^* with highest weight (GHOST) (breaking ties adversarially). Recurse ($b_0 \leftarrow b^*$),

until reaching a leaf block. Output that leaf block." At the beginning of each slot, the slot's proposer determines a block using LMD GHOST and extends it with a new proposal. Half way into each slot (*i.e.*, Δ time after the proposal and after the beginning of the slot), the slot's committee members determine a block using LMD GHOST in their view and vote for it (votes are also called *attestations*). (At the same time they also cast a Casper FFG vote, as described later.) An exact confirmation rule of LMD GHOST/Gasper is not specified.

Second, on the larger time scale where Casper FFG operates, time proceeds in epochs comprised of 32 slots. On a high level, Casper FFG is a two-phase traditional propose-and-vote-style Byzantine fault tolerant (BFT) consensus protocol (cast as a blockchain protocol into the chained framework, like Chained HotStuff [21]), except there is no leader in charge of assembling proposals. Instead, the proposals are supposed to be generated consistently across honest nodes by the LMD GHOST fork choice layer. Casper FFG proceeds as follows: Blocks first become justified if a super-majority ($2N/3$) votes 'for them', and subsequently become finalized, roughly when a super-majority votes 'from them' for a subsequent block. The genesis block is justified and finalized by definition. The blocks among which validators cast their votes during an epoch are the so-called epoch boundary blocks, which are those blocks that are leaf blocks after truncating the block tree to only those blocks that came from the previous epoch. Validators vote for the highest epoch boundary block that is consistent with the highest justified block they have observed, which in turn extends the latest finalized block they have observed. Due to the super-majority required to advance a proposal, as well as the two-phase confirmation (called *finalization*), Casper FFG remains safe even under temporary network partition. The confirmation rule on the Casper FFG level is to output the latest finalized block and its prefix.

3 A Refined Reorg Attack

3.1 Motivation

Previous work [18] described a malicious, low-cost reorg attack. In particular, the attack leverages strategic timing of broadcasting blocks and attestations, as opposed to honestly releasing them when supposed to. In a nutshell, in the strategy of [18], an adversarial block proposer in slot n keeps its proposal hidden. The honest block proposer in slot $n+1$ will then propose a competing block. The adversary can now use its committee members' votes from both slots n and $n+1$ to vote for the withheld block of slot n in an attempt to outnumber honest votes on the proposal of slot $n+1$. As a result, blocks proposed by honest validators may end up orphaned, *i.e.*, they are displaced out of the chain chosen by LMD GHOST. In [18] this reorg strategy is part of a bigger scheme to delay consensus.

We show how the attack of [18] can be modified such that the number of adversary validators required is significantly reduced, from a set of size linear in the total number of validators to a constant-size set – indeed for a one-block reorg as little as one adversarial validator is sufficient. Note that similar to [18] the adversarial strategy does not involve any slashable behavior and is therefore

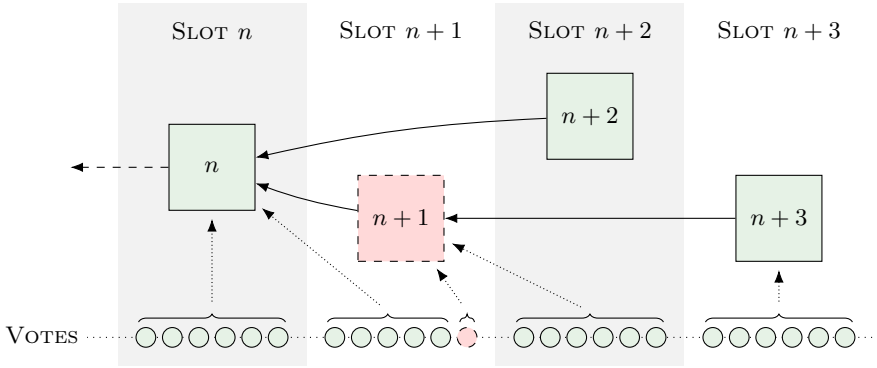


Fig. 1. Example of a one-block reorg attack using the refined strategy: In slot $n+1$ the adversary privately creates block $n+1$ on block n and attests to it. Honest validators of slot $n+1$ do not see any block and thus attest to block n as head of the chain. In the next slot, an honest proposer publishes block $n+2$ building on block n, which is the current head in their view. Simultaneously, the adversary finally publishes block $n+1$ and the attestation voting for block $n+1$. All honest validators of slot $n+2$ attest to block $n+1$ as head of the chain, because it has more weight than block $n+2$. In the next slot block $n+3$ is proposed building on block $n+1$. Block $n+2$ is reorged out.

relatively cheap. In Section 5, we further improve upon this refined reorg attack, combining strategies from both this section and Section 4.

3.2 Refined Reorg Strategy

Consider Figure 1, which shows the adversary being the proposer of slot $n+1$ as well as controlling a committee member in slot $n+1$. We describe the adversarial strategy to perform a 1-reorg:

1. At the beginning of slot $n+1$ the adversary privately creates block $n+1$ on block n and privately attests to it. Honest validators do not see block $n+1$ and so they attest to the previous head of the chain, block n.
2. At the beginning of the next slot, an honest validator proposes block $n+2$. Assuming zero network latency for now, the adversary finally publishes the private block and attestation from slot $n+1$ at the same time as block $n+2$ is released. Honest validators now see both block $n+1$ (and its one attestation) as well as block $n+2$. These blocks are conflicting because they share the same parent, block n. Another result of sharing the same parent is that block $n+1$ inherits all the weight of block n, in particular the honest attestations from slot $n+1$ voting for block n also count in favor of it.
3. Hence, in slot $n+2$ all honest validators vote for block $n+1$ as head of the chain, because it has more weight due to the single adversarial attestation from slot $n+1$.

4. Finally, at the beginning of slot $n+3$, an honest validator proposes block $n+3$ pointing to block $n+2$ as its parent. This effectively orphans block $n+2$ and brings the reorg attack to its conclusion.

The above strategy shows that a block proposer which controls a single committee member of the same slot can successfully perform a 1-reorg. Naturally, the logic of this strategy can be extended to reorg attacks of arbitrary length k. Let the number of honest validators in any given committee be $W_{\text{honest}} \approx (1-\beta)W \leq W$. Then, for a successful reorg attack of length $k > 1$, the proposing adversary needs to control $W_{\text{honest}}(k-1)+1$ validators, since it offsets honest committee members' votes in the first $(k-1)$ slots and uses the above refined attack strategy in the last slot.

The refined reorg attacked described here improves on the strategy proposed in [18] by removing the need for the adversary to compete with the committee of slot $n+k+1$. While the improvement for long-range reorg attacks may not be as significant, short reorg attacks are considerably more feasible using the above refined strategy. In particular, 1-reorg attacks are effectively always possible for large enough parties. With currently 230,000 active validators[1] and 32 slots per epoch, an adversary controlling 200 validators (which amounts to 0.09% of total stake) has a 99.8% chance of being selected block proposer at least once per any given day, and once selected as block proposer in a particular slot controls at least one committee member validator in that slot with probability 99.8%. So with more than 99.6% probability, an adversary with 0.09% of total stake is in a position to execute a 1-reorg for any given day.

We will now relax the assumption of zero network latency. PoS Ethereum's fork choice rule only considers attestations that are at least one slot old [2] (so votes from slot $n+2$ do not count in the fork choice for slot $n+2$). Further, a committee member is supposed to attest if "(a) the validator has received a valid block from the expected block proposer for the assigned slot or (b) one-third of the slot has transpired [...] – whichever comes first"[2] [4]. After block $n+2$ is broadcasted to the network, honest validators immediately attest to it upon reception (unless by that time they see another chain as leading in fork choice). Thus, the adversary must ensure that a majority of validators of slot $n+2$ see block $n+1$ and the adversary's attestation voting for block $n+1$ (from slot $n+1$) before they see block $n+2$, but after block $n+2$ was proposed (to ensure it extends block n). This proves to be a non-trivial but practically feasible issue.

Suppose the adversary controls a number of nodes at different 'locations' in the topology of the peer-to-peer gossip network [3] (these nodes might still be physically collocated). This is possible without greater difficulty because the gossip network has no defenses against such Sybil attacks. Then, some adversarial node will likely receive the new proposal block $n+2$ relatively early on in its dissemination process. The adversary can then release the private block and attestation in a coordinated fashion from all the different locations in the peer-to-peer topology where the adversary controls nodes. Due to the superior number

[1] https://beaconcha.in/validators. Accessed: 2021-10-09

[2] Regarding attestation timing, PoS Ethereum practice slightly deviates from Gasper

of sources of the adversarial block and attestation it is likely that these arrive earlier than the proposal block $n + 2$ at enough (a majority of) honest nodes to ultimately orphan block $n + 2$.

4 A Refined Liveness Attack

4.1 Motivation

Earlier works [8,14,15,16,17] have described balancing-type attacks against variants of the GHOST fork choice rule used in PoS Ethereum as modelled in the Gasper protocol [7]. In particular, the attack described in [17,16] uses adversarial network delay to show that PoS Ethereum is not secure in traditional (partially) synchronous networks. While adversarial network delay (up to some delay bound) is a widely employed assumption in the consensus literature, there is disagreement whether it is appropriate for Internet-scale open-participation consensus. As a result, past attacks are often seen as impractical and have not been mitigated: "Note that this attack does depend on networking assumptions that are highly contrived in practice (the attacker having fine-grained control over latencies of individual validators), [...]" [5]

We show how the attack of [16,17] can be modified and implemented so that an adversary controlling 15% of stake can stall PoS Ethereum *without requiring adversarial network delay*. (For ever larger numbers of validators, ever smaller fractions of adversarial stake suffice.) To this end, we show through experiments that aggregate properties of many individually random message propagation processes (*e.g.*, 'within time T this transmission is received by fraction x of nodes') in real-world Internet-scale peer-to-peer gossip networks [3,20] are sufficiently predictable to give the adversary the required control over how many validators see which adversarial messages when. None of the adversarial actions are slashable protocol violations.

4.2 High-Level Idea

Recall that the *balancing attack* [16,17] consists of two steps: First, adversarial block proposers initiate two competing chains – call them Left and Right. Then, a handful of adversarial votes per slot, released under carefully chosen circumstances, suffice to steer honest validators' votes so as to keep the system in a tie between the two chains and consequently stall consensus.

Assume, w.l.o.g., that when viewing Left and Right with equal number of votes, the protocol's tie-break favors Left over Right. If the adversary manages to deliver a withheld adversarial vote for Right from an earlier slot to roughly one half of honest validators for the current slot i, before validators submit their votes for slot i, while the other half does not receive said vote before casting their votes, then roughly half of honest validators (those who have received the sway vote 'in time') see Right as leading and will vote for it in slot i, while the other half (those who see the sway vote 'late' and hence at the time of voting see a tie which they break in favor of Left) will vote for Left in slot i (see Figure 2).

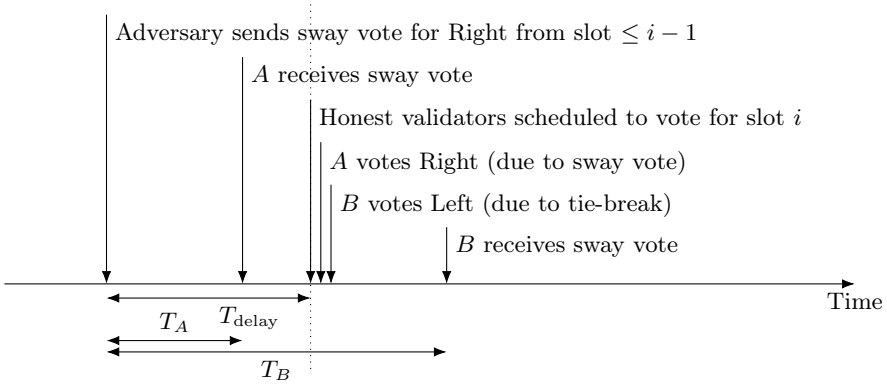


Fig. 2. Assuming a tie between two chains Left and Right, with tie-break favoring Left. The adversary releases a sway vote for Right from a slot $< i$ at time T_{delay} before the point in time at which honest validators vote in slot i according to the protocol. The parameter T_{delay} is chosen such that roughly half of honest validators (such as A) receive the sway vote *before* they submit their vote (and hence vote Right, as Right *now* has more votes *in their view*), and the other half of honest validators (such as B) receive the sway vote *after* they submit their vote for (and hence vote Left, as the tie-break *still* favors Left *in their view*).

Idealizing the above as voting according to a coin flip for each validator, roughly $W_{\text{honest}}/2$ of W_{honest} honest validators per slot would vote Left and Right, respectively, with a gap of $O(\sqrt{W_{\text{honest}}})$ (*cf.* variance of a binomially distributed random variable). So, $O(1/\sqrt{W_{\text{honest}}})$ adversarial fraction of stake would suffice to rebalance the vote to a tie and keep the system in limbo. In Section 4.4 we provide evidence from real-world propagation delay measurements in a replica of Ethereum 2's gossip network [3] to support the hypothesis that the adversary can indeed reliably determine the time T_{delay} it takes for approximately half of nodes to receive a message broadcast by the adversary.

4.3 Detailed Description

First we describe the attack for a given T_{delay}, then we describe how to obtain T_{delay}. Our simulation[3] using the gossip network propagation model obtained in Section 4.4 provides further details.

First, the adversary waits for an opportune epoch to launch the attack. An epoch is opportune if the block proposers in slot 0 and 1 are adversarial (this can be strengthened). Due to the random committee selection in PoS Ethereum, this happens with probability β^2 for any given epoch, so that the adversary needs to wait on average $1/\beta^2$ epochs until it can launch the attack. In the following, assume epoch 0 is opportune. The adversarial proposers of slots 0 and 1 propose conflicting new chains 'Left' and 'Right', respectively. Note that this is not a

[3] Source code: https://anonymous.4open.science/r/gasper-gossip-attack-7816/

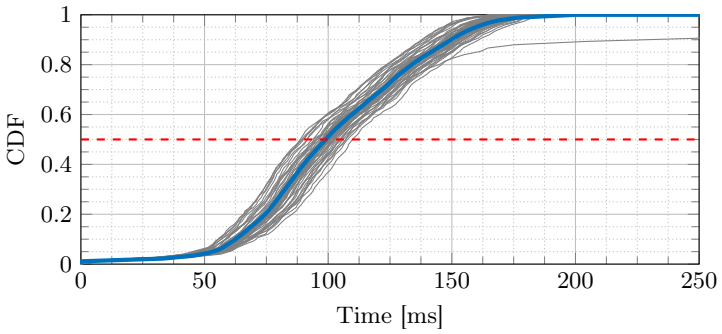


Fig. 3. Fraction of participants in the peer-to-peer gossip network who have received a message broadcast by node 0 at time 0 by the given time (50 sample messages in gray, mean over all samples in blue). Median (dashed red) at $\approx 100\,\text{ms}$.

slashable protocol violation. Both withhold their proposals so that none of slot 0 or 1 honest validators vote for either block. The adversary releases the blocks after slot 1. We assume w.l.o.g. that the tie between Left and Right (recall that no vote has been cast for either so far) is broken in favor of Left.

Time T_{delay} before honest validators in slot 2 vote, the adversary releases a vote for Right from an adversarial committee member of slot 1 (so called *sway vote*, see Figure 2). If T_{delay} is tuned well to the network propagation behavior *at large*, then roughly one half of honest committee members of slot 2 see the sway vote before they cast their vote, and thus view Right as leading (due to the sway vote) and will vote for it; and the other half see the sway vote only after they cast their vote, and thus view Left as leading (due to the tie-break) at the time of voting and will vote for it. Once the adversary has observed the outcome of the vote, which now should be a split up to an $O(\sqrt{W_{\text{honest}}})$ gap, the adversary uses its slot 2 committee members (which stipulates the adversarial fraction $O(1/\sqrt{W_{\text{honest}}})$ required for this attack) as well as slot 0 and 1 committee members to rebalance the vote to a tie. As the tie is restored, the adversary can use the same strategy in the following slot, and so forth.

Note that the adversary can observe the outcome of a vote and learns how many honest committee members saw Left and Right leading, respectively. The adversary can use this information to improve its estimate of T_{delay}. We show in Section 4.4 that the optimal T_{delay} can be reliably localized using grid search.

4.4 Experimental Evaluation

To understand whether the network propagation delay distribution is sufficiently well-behaved for an adversary to reproducibly broadcast messages so that they arrive at roughly half of nodes by a fixed deadline, we replicated the gossip network of Ethereum 2 [3] and measured the network propagation delay of test 'ping' packets from a designated sender to all nodes. The implementation in the Rust

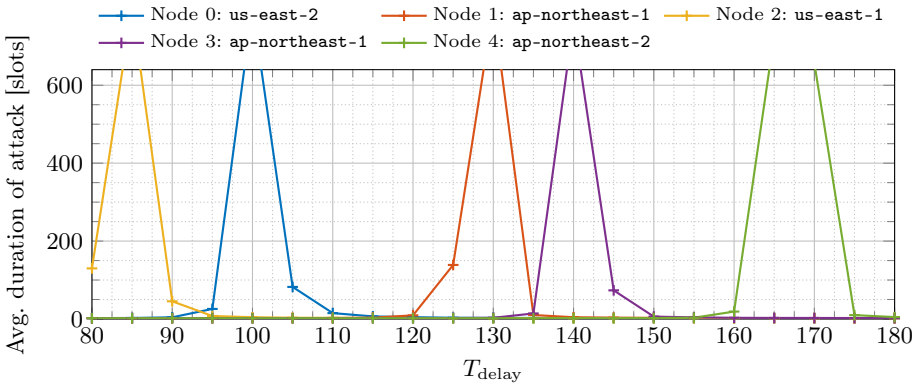


Fig. 4. Using the propagation delay measurements to model network propagation, we simulated our attack for fixed $\beta = 0.15$, varying T_{delay}, and five different positions of the adversary in the network, and plot the resulting average duration of the liveness interruption (cut off at 800 slots horizon). Observe that the peak for node 0 fits well to the median observed in Figure 3. The curves are smooth and allow for easy and reliable localization of the optimal T_{delay}.

programming language used `libp2p`'s `Gossipsub` protocol and implementation, as is used in Ethereum 2 [3].

The gossip network comprised 750 nodes, each on an AWS EC2 `m6g.medium` instance (with 50 instances each in all 15 AWS regions that supported `m6g.medium` as of 21-April-2021). Each node initiated a connection with ten randomly chosen peers. The five nodes with lowest instance ID were designated as senders and continuously broadcasted beacon messages with inter-transmission times uniformly distributed between zero and five seconds over a period of 20 minutes, logging the time when each message was broadcast. All nodes logged the time when a message was first received.

The network propagation delay was determined for each message and each receiving node. The respective CDFs, *i.e.*, what fraction of nodes have received a given message by a certain delay, is plotted as an example for a sample of messages from the first designated sender (node 0) in Figure 3 (together with the average CDF of all messages originating at node 0). (CDFs for the other four designated senders are omitted for brevity here. They show similar behavior, just slightly shifted in time.) It is apparent from the CDFs that depending on the location of the node (nodes 0, 1, 2, 3, 4 happened to be located in `us-east-2`, `ap-northeast-1`, `us-east-1`, `ap-northeast-1`, `ap-northeast-2`, respectively) both geographically as well as within the peer-to-peer network topology, the median of the average CDF varies, but considering messages originating at a fixed sender, the fraction of validators reached by the median of the average CDF is fairly concentrated around 1/2. This suggests that the adversary can indeed determine T_{delay} so that with little dispersion honest validators get split in two halves.

We simulated the attack for $\beta = 0.15, m = 128$, using the network propagation delay samples as a model for random network delay.[4] Assigning the simulated adversary to one of the five designated senders for all of the attack, whenever the adversary broadcasts a sway vote, the propagation delays to the honest committee members of the given slot are sampled (without replacement) from the delays of one randomly drawn message of that designated sender.

To determine the optimal T_{delay}, we performed grid search (with 5 ms step size) and for each T_{delay} simulated ten attacks in opportune epochs and recorded (see Figure 4) how long the adversary was able to stall liveness (terminating at a horizon of 800 slots corresponding to 160 minutes). It is apparent that for the adversary in the position of each of the five designated senders of the measurement experiment, different T_{delay} are optimal. The optimal T_{delay} correspond well with the median of the average CDF (*cf.* Figure 3). As the curves are smooth and have a single distinct peak of width ≈ 5 ms, the adversary can locate the optimal T_{delay} well. In particular, even with T_{delay} approximating the optimal value only up to 10 ms, the adversary can stall liveness for dozens of slots. Recall that none of the adversarial actions are slashable protocol violations, so the adversary can refine T_{delay} iteratively and launch this attack over and over.

5 Reorg Attack Using Probabilistic Network Delay

5.1 Motivation

In Section 3 we describe how an adversary might execute a 1-reorg with only a single adversarial committee member's vote. In Section 4 we show how an adversary can stall consensus and thus delay finality without adversarial control over network delay. By combining ideas from both attacks, we now describe an attack in which the adversary can execute a long-range reorg with vanishingly small stake and without control over network delay.

On a high level, the adversary avoids competing directly with honest validators of $(k-1)$ committees, as done in the reorg attack described in Section 3. Instead, the adversary uses the technique of Section 4 to keep honest committee members split roughly in half by ensuring they have different views on what the current head of the chain is. This way, honest nodes work against each other and maintain a tie which the adversary can tip to their liking at any point using only a few votes.

5.2 Refined Strategy Using Probabilistic Network Delay

Consider Figure 5, in which the adversary is the proposer of slot $n+1$. We describe the strategy where the adversary executes a 2-reorg and analyze how many validators the adversary needs to control, depending on our assumption on the adversary's control over the network:

[4] Source code: https://anonymous.4open.science/r/gasper-gossip-attack-7816/

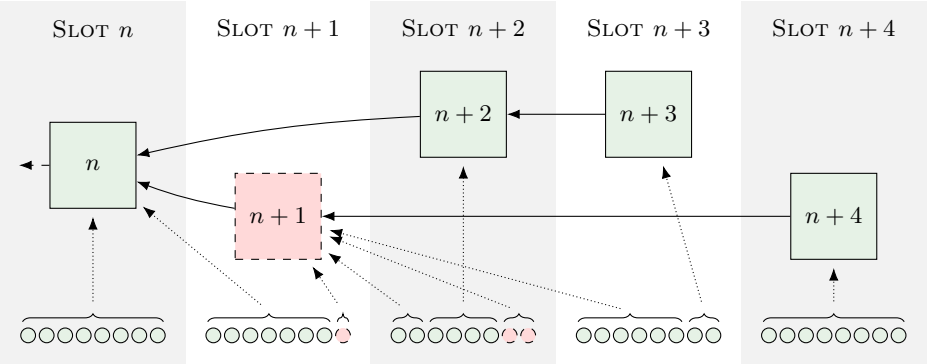


Fig. 5. Example of a 2-reorg combining refined reorgs and balancing strategies: In slot $n+1$ the adversary privately creates block $n+1$ on block n and withholds adversarial votes on it. Honest validators of slot $n+1$ attest to block n. In slot $n+2$, an honest proposer builds block $n+2$ on block n. The adversary releases block $n+1$ and one of the withheld votes in such a way that roughly half of honest committee members vote for blocks $n+1$ and $n+2$, respectively. If the adversary has tight control over network delays, they can effect that block $n+2$ has one more vote than block $n+1$. Without adversarial control of delays, a vanishing fraction of adversarial votes still suffices to rebalance accordingly. In slot $n+3$, the honest proposer views block $n+2$ leading and proposes block $n+3$ off it. The adversary releases two votes voting for block $n+1$ in such a way that a majority of honest committee members vote for block $n+1$, breaking the tie and completing the 2-reorg which orphaned blocks $n+2$ and $n+3$ in slot $n+4$.

1. First, in slot $n+1$ the adversary privately builds block $n+1$ on top of the current head of the chain, block n. Further, the adversary privately votes for block $n+1$ using an attestation from slot $n+1$.
2. In the next slot, the proposer of block $n+2$ builds on block n because they have not seen block $n+1$. Before honest validators in slot $n+2$ attest, the adversary releases block $n+1$, along with the withheld attestation, in such a way that roughly half of honest committee members of slot $n+2$ attest before they see the sway vote (and thus vote for block $n+2$ as the current head), and the other half sees block $n+1$ as leading due to the attestation from slot $n+1$ and thus votes for block $n+1$ as the current head.
   If the adversary has control over the network delay, as assumed in [17,16], then it can target the release of the withheld block and vote such that block $n+2$ accumulates exactly one more attestation than block $n+1$. If network delay is instead probabilistic, as in Section 4, then the adversary needs to spend $O(\sqrt{W_{\text{honest}}})$ adversarial votes to rebalance the gap in votes.
   In the case of a k-reorg, this step is repeated for the first $(k-1)$ slots.
3. Since slot $n+3$ is the last slot of the reorg attack, we use the insight of Section 3 that the adversary does not have to wait for honest votes to take place and rebalance them, but instead can sway validators towards the adversarial chain as soon as the honest proposal for this slot was created. So, in slot $n+3$, the current proposer views block $n+2$ as leading and thus builds

block $n+3$ on it. Finally, the adversary releases two withheld attestations such that a majority of honest committee members of slot $n+3$ views them before attesting. Thus, a majority of validators votes for block $n+1$ as head of the chain. Remember that the fork choice rule only considers attestations at least one slot old.

4. Lastly, in slot $n+4$ the proposer views block $n+1$ as leading and thus builds block $n+4$ on block $n+1$. This completes the 2-reorg and orphans blocks $n+2$ and $n+3$.

For 1-reorg the adversary needs to control a single validator in the same slot they propose their block. For reorg lengths $k > 1$, the number of adversarial validators required depends on the level of control over network delays. If delays are under adversarial control, then $(2k-1)$ adversarial validators suffice for a k-reorg, an amount linear in the reorg length only, but independent of the size of the validator set. If instead network delay is probabilistic rather than under adversarial control, a vanishingly small fraction $O(1/\sqrt{W_{\text{honest}}})$ of adversarial validators suffices to perform the necessary rebalancing to maintain the tie throughout the first $(k-1)$ slots of the k-reorg, leading to an overall requirement of $O(k\sqrt{W_{\text{honest}}})$ adversarial votes. Thus, large stakers can easily execute long-range reorg attacks. To illustrate the severe reduction of attacking conditions, consider the following: Under adversarial network delay, an adversary can perform a 10-reorg by merely controlling 19 validators.

6 Discussion

6.1 Ex Ante Vs Ex Post Reorgs

Typically reorgs refer to an attack in which the adversary observes a block that they subsequently attempt to fork out. We call this an *ex post* reorg attack. The reorg attacks we describe are different in nature. Here, the adversary attempts to fork out a future block that is unknown to the adversary at the start of the attack. We call this an *ex ante* reorg attack.

In an ex post reorg attack, the adversary typically targets a block with abnormally large rewards that the adversary seeks to capture for themselves. In the context of Bitcoin it could be a block that contains transactions paying extraordinary amounts of fees, also referred to as 'whale transactions' [12]. In the context of Ethereum it could be blocks containing large MEV opportunities. Upon observing a lucrative block, the adversary attempts to capture it retrospectively. In PoS Ethereum this proves to be exceptionally difficult for non-majority actors due to the fact that the block the adversary wishes to orphan quickly accrues attestations from committee members in parallel. Each attestation adds weight to the block in question, which in turn is considered by the fork-choice rule LMD GHOST to determine the head of the chain. In short, no technique is known for non-majority adversaries to perform ex post reorg attacks reliably.

In contrast, ex ante reorg attacks are currently very much possible in PoS Ethereum, as this paper shows. The adversary overcomes the 'power of many

parallel attestations' by exploiting LMD GHOST as described in Sections 3 and 5. Intuitively, this is enabled by tricking honest validators into contrary views of the chain such that a handful of adversary validators are sufficient to tip the chain to their favor and thus successfully perform reorgs of sizable length. As a consequence of the different nature of the attack, the adversary's motivations to attack are different. In an ex ante reorg the adversary cannot observe valuable blocks and orphan them ex post, but must find other strategies to extract more value from it than it could from making an honest proposal, one of which is discussed in Section 6.2.

6.2 Reaping Higher Fees and MEV Via The Attack

Maximal Extractable Value (MEV, formerly Miner Extractable Value [10]) represents a third source of profits for block producers, along with the proposer/attester rewards and transaction fees. Common MEV opportunities include arbitraging a trade, frontrunning it to earn greater profits, or tailing liquidation events to buy the collateralized assets backing the defaulting position.

MEV opportunities grow with an increasing amount of pending transactions since more possible transaction order combinations exist. At the same time, the adversary is able to choose from a larger set of pending transactions those earning them the highest fees. More time between blocks then implies weakly more extractable MEV and transaction fees, which in turn implies more profits for the block proposer. The reorg attacks described in this paper can be interpreted as buying the adversary more time to construct their block.

With k-reorgs, it is possible for the malicious proposer to extend their listening period to up to $12k$ seconds (refined reorg strategy from Section 3), the 12 seconds elapsed between the previous block produced and their own slot, as well as $12(k-1)$ more seconds until the next honest block is included in the canonical chain. (The 2Δ duration introduced in Section 2.2 is set to 12 seconds in the PoS Ethereum implementation.) The adversary may even listen to the honest blocks, which will eventually be orphaned, and capture their MEV should they find better opportunities than the adversary themselves. Interestingly, the adversary may simply release their block late, even without attempting a reorg, to increase their listening time and ultimately rewards.

6.3 Reorgs Cause Attestation Overflow

While reorg attacks weakly benefit those who launch them, consensus degradation may be obtained as an unintended side-effect of the reorg.

Validators in a slot committee are distributed among a number of subcommittees. With a target subcommittee size of 128 and currently 230,000 active validators, ≈ 57 subcommittees are formed per slot. In the current implementation of PoS Ethereum, all identical votes from the same subcommittee may be aggregated into one 'summary' vote, lightening the block size. A block may include up to 128 such aggregates. Ideally, with all validators voting correctly and on time, the next block need only feature 57 aggregates, one per subcommittee.

In practice, we observe such a number of large aggregates (summarizing many votes) in the block, with most validators voting identically, along with some aggregates summarizing other votes from validators who may have suffered from latency issues and voted identically, albeit wrongly. Suboptimal packing of the aggregates or adversarial voting behavior may also contribute to filling up the available slots for aggregates in the block. In the case of a reorg, deconfirmed aggregates return into the mempool and need to be included in future blocks. Even for short-range reorgs this can lead to congestion in the sense that many more aggregates wait to be included than there is space available in blocks.

Votes state their view of the current target of the FFG mechanism. A target vote is valid only if it is included in a block no later than 32 slots after the attesting slot. By reorging blocks, an attacker strains the capacity of the chain to include these valid votes. In the worst case, finalization is fully delayed whenever more than $1/3 - \beta$ of valid honest votes do not manage to be included.

6.4 Delaying Finality

Our attacks also enable *a priori* malign actors, perhaps ideologically motivated, to delay and in some cases outright stall consensus decisions. The refined attack of Section 4.2 gives the adversary a tool to do just that, even if the adversary cannot control message propagation delays (which instead are assumed to be probabilistic). Furthermore, in the regime of many validators, a vanishing fraction of adversarial stake suffices to mount the attack.

The attack of Section 5 enables long-range reorgs of the chain constituting consensus. The consequences are two-fold. Readily, transaction confirmation in the LMD GHOST part of the protocol gets delayed. Transactions might enter/leave the LMD GHOST chain multiple times before eventually settling. This causes uncertainty and delay for users who consider a transaction confirmed once it has stabilized in the LMD GHOST chain. Furthermore, the adversary can use reorgs, as proposed in [18], to destabilize epoch boundary blocks. No epoch boundary block might then get the necessary number of FFG votes to become justified, which delays finality by at least an epoch and thus creates delay for users who rely on the finalized ledger.

References

1. Ethereum 2.0 phase 0 – the beacon chain, https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/beacon-chain.md
2. Ethereum 2.0 phase 0 – beacon chain fork choice (2020), https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/fork-choice.md
3. Ethereum 2.0 networking specification (2021), https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/p2p-interface.md
4. Ethereum 2.0 phase 0 – honest validator (2021), https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/validator.md
5. Buterin, V.: Proposal for mitigation against balancing attacks to LMD GHOST (2020), https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation

6. Buterin, V., Griffith, V.: Casper the friendly finality gadget. arXiv:1710.09437 [cs.CR] (2019), https://arxiv.org/abs/1710.09437
7. Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., Zhang, Y.X.: Combining GHOST and Casper. arXiv:2003.03052 [cs.CR] (2020), https://arxiv.org/abs/2003.03052
8. Buterin, V., Stewart, A.: Beacon chain casper mini-spec (comments #17, #19) (2018), https://ethresear.ch/t/beacon-chain-casper-mini-spec/2760/17
9. Carlsten, M., Kalodner, H., Weinberg, S.M., Narayanan, A.: On the instability of Bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 154–167 (2016)
10. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. arXiv:1904.05234 [cs.CR] (2019), https://arxiv.org/abs/1904.05234
11. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. Communications of the ACM **61**(7), 95–102 (2018)
12. Liao, K., Katz, J.: Incentivizing blockchain forks via whale transactions. In: International Conference on Financial Cryptography and Data Security. pp. 264–279. Springer (2017)
13. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
14. Nakamura, R.: Analysis of bouncing attack on FFG (2019), https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113
15. Nakamura, R.: Prevention of bouncing attack on FFG (2019), https://ethresear.ch/t/prevention-of-bouncing-attack-on-ffg/6114
16. Neu, J., Tas, E.N., Tse, D.: A balancing attack on Gasper, the current candidate for Eth2's beacon chain (2020), https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079
17. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: Symposium on Security and Privacy. S&P '21, IEEE (2021)
18. Neuder, M., Moroz, D.J., Rao, R., Parkes, D.C.: Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders. arXiv:2102.02247 [cs.CR] (2021), https://arxiv.org/abs/2102.02247
19. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 507–527. Springer (2015)
20. Vyzovitis, D., Napora, Y., McCormick, D., Dias, D., Psaras, Y.: GossipSub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks. arXiv:2007.02754 [cs.NI] (2020), https://arxiv.org/abs/2007.02754
21. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: Symposium on Principles of Distributed Computing. p. 347–356. PODC '19, ACM (2019)