Block Reorgs Mitigation in Ethereum Proof-of-Stake

Dimitar Stefanov, Stoycho Nenov

10. june 2022

10. semester CS-IT10

AALBORG UNIVERSITET

Department of Computer Science
Aalborg University
www.aau.dk

Title:
Block Reorgs Mitigation in Ethereum Proof-of-Stake

Theme:
Mitigating a vulnerability known as 'Block Reorgs' in the context of Ethereum's Proof-of-Stake

Semester:
Spring semester 2022 CS-IT10

Participants:
Dimitar Stefanov
Stoycho Nenov

Supervisor:
Michele Albano
Brian Nielsen

Character count: 93262

Page count: 58

Project period:
1. february 2022 - 10. june 2022

Abstract:

This report is written by a group of Computer Science (IT) students on their 10th semester. The objective of the project is to propose a mitigation technique to greatly reduce the probability of a dishonest validator creating a selfish fork with which to force the network to orphan blocks from the canonical chain in a blockchain network such as the Ethereum one. Initially we begin with providing information on Beacon Chain (Ethereum 2.0) then we present a solution that was previously proposed and rejected. Afterwards we present the new proposal together with mathematics to showcase the extent of mitigation followed by simulation results and finally reflection on the results, justification based on previous rejection of a solution, and what impact on the network is anticipated.

Preface

This report is written by a group of Computer Science (IT) students on their 10th semester of the master's program in Computer Science (IT) at Aalborg University (AAU). The purpose of this report is present a mitigation technique for block reorgs vulnerability on the Beacon Chain network.

Reading instructions

In the report, source references appear according to the Vancouver method. This means that [1] refers to source number one in the bibliography. In the bibliography, Internet pages are indicated with the author, title and a link to the document.

Figures and tables are numbered according to the chapter and number in the row. That is, in chapter two, a reference is made to figure one with the text: Figure 2.1.

The number of characters is counted using the CountAnything program, where the content, which is not included in the number of characters, is deleted from the PDF file before the count.

Aalborg University - Department of Computer Science, 10. june 2022

Contents

Summary

In this project we analyze a vulnerability called block reorgs for Beacon chain (Ethereum 2.0) that has previously been investigated by [1] and a solution was proposed by them, which was later rejected. In our work we showcase an alternative solution but instead of fixing the problem we provide a mitigation technique that reduces the probability of the issue occurring. In the project we provide mathematical analysis and simulations on the probabilities of a dishonest validator being able successfully carry out the attack within an epoch before and after the mitigation technique. We also analyze different aspects of the network that are or could be affected by our proposal such as: availability, partition tolerance, decentralization etc.

Our proposal extends Beacon chain by introducing a secondary block proposer that will create a block that would be used if the primary block proposer's block is not observed in time by the network. The impact of the secondary validator extends to increasing the availability of the network by reducing the amount of missed blocks, which also increases the amount of transactions processed and reduces the gas fees slightly. Another aspect of improvement is the increased difficulty of maintaining a private fork as after the mitigation implementation a dishonest validator would have to own both nodes selected as block proposer in order to be able to maintain the private fork. By using our solution the difficulty of maintaining such fork is increased quadratically for each slot that is included in the fork.

For 20% of the total stake validator we have achieved a reduction from 10% per epoch to only 0.1% for a length 1 attack. Length 2 and lower attacks yield extremely low numbers based on the mathematical analysis and we did not observe any length 2 attacks using the mitigation technique from 10 000 080 epochs, while before the mitigation 7735 epochs were affected which is about 0.078% chance of occurring. For lower stake validators the likeliness of being able to carry out the attack is practically negligible even for a length 1 attacks.

For 30% of the total stake validator we have achieved reduction of the probabilities of length 1 attack from 90% to 20% to occur within an epoch, which considering the amount of 26 billion USD for having 30% of the current stake, is a huge improvement. For length 2 attacks the reduction is from 44% to 0.02% which is already rather low.

1 Introduction

Cryptocurrencies and blockchain technology have emerged in the past decade as the most promising type of projects in terms of investments. Cryptocurrencies allow small investors to play a part in an enormous economy but many big investors are afraid to invest big amounts not only because of the high volatility of such markets but also because of vulnerabilities in the technologies. In this project we describe a present vulnerability, block reorgs, in the Ethereum protocol. We also present a solution that was proposed by the people who researched the vulnerability [1] that was presented initially in [2], the reason why it was rejected and afterwards a solution that fixes the issue together with analytics providing information on how efficient the solution is.

1.1 Initial Motivation

Initially we knew about the paper of Michael Neuder, Daniel J. Moroz, Rithvik Rao, David C. Parkes [1] and we found out their proposal for block reorgs was rejected. After the rejection of their proposal we decided to propose an alternative solution to strengthen the Ethereum network. But in order to present a viable solution it should introduce little or no overhead to the current state. As a metrics to check up with we consider availability, partition tolerance, performance and bandwidth as these are core elements of any blockchain network. Any improvement typically results in some overhead so we are going to limit ourselves to having the same amount of messages passed between peers, roughly the same amount of data within the messages and also keep the computational requirements low so that we do not slow down the network and not to increase the hardware requirements for the validators. Based on these constraints we want to make the probability of this issue to practically 0 so that a malicious actor with a high stake would not be incentivised to censor the network. Based on our understanding of the problem we can derive the following initial problem statement:

How can we resolve the block reorgs vulnerability without significant additional overhead on the network?

2 | Background

In this chapter we provide the necessary information in order to understand thoroughly block reorgs vulnerability. First we begin with terminology and explaining different ethereum components that are important for the attack, then we continue with explaining the attack and what a dishonest validator can get out of it and finally we show related solution for the problem that was rejected in the past.

2.1 Blockchain terminology

Term	Definition
Blockchain	A linked list where every element is a block linked to the previous block. Represents the state of the network in the form of a distributed database.
Block	The basic element in a blockchain. It is where the data is stored. This includes transactions, hash of the previous block and metadata.
Transaction	An operation that changes the state of the blockchain.
Public blockchain	Anyone can join a public blockchain.
Private blockchain	Only selected people can join a private blockchain.
Permissioned blockchain	A blockchain with permission management (roles) e.g. admin, user etc.
Permissionless blockchain	A blockchain where all users have the same rights.
Proof of Work	A consensus algorithm where computational effort is used to prove a user's legitimate involvement.
Proof of Stake	A consensus algorithm where a stake, usually in the form of a cryptocurrency, is used to prove a user's legitimate involvement.
Token	The currency of a blockchain.
Smart Contract	Code that sets conditions which when met allow for certain transactions to happen.
Miner/validator	A user responsible for producing/validating a block with a number of transactions in it. Usually rewarded with tokens for doing so.
Minting	The process of producing a token, increasing the total supply in circulation.
Weight of attestation	The number of tokens that are staked by the validator who creates the attestation
Weight of block	The sum of tokens that all validators whose attestation is included in the block

2.2 Ethereum network

In this section we describe the necessary components and terminology of the ethereum network (Beacon Chain) in order to present our work on block reorgs. In this section we will explain

Tokens, Proof of Stake (PoS), validators and their duties, epochs, slots, slashings, justification and finalization of epochs, attestations, fork choice rule, effective balance and memory pools.

2.2.1 Network Overview

The Ethereum network is a public permissionless blockchain, see **Section 2.1**. It currently has the second highest market cap in the cryptocurrency market. Its token is Ether (ETH). The network currently supports both proof of work and proof of stake. It is expected that eventually the network will run entirely on proof of stake.

Ethereum supports smart contracts, which makes it practically useful and creates many potential applications in the real world. Because of the smart contracts there are many other blockchains and software products built and/or based on Ethereum. The Ethereum network was built with the purpose of developing decentralized applications rather than only supporting transacting and minting tokens. This sets it apart dramatically from Bitcoin.

2.2.2 Tokens

Ethereum network is using its own native token called ether and is abbreviated as ETH. The ETH is divisible into wei.

$$1ETH = 10^{18}wei$$

Wei is the smallest amount the ether is divisible into but its monetary value is negligible. For that reason in most cases gwei is used because based on the current price of ethereum it has a very small monetary value, a lot less than a cent.

$$1Gwei = 10^{9}wei$$

Usually gwei are used to pay the cost for executing code on the validators' machines, this cost is called gas fee.

2.2.3 Proof of Stake (PoS)

In distributed computing, one of the main problems to be resolved is that all machines agree on the same result at the end of execution. In public blockchains a major concern is that, since anybody can join the network, anybody can act maliciously and attempt to sabotage the network. In order to mitigate this, consensus algorithms like PoS are designed such that actors in the network that act honestly are being incentivized to do so, typically with a monetary value, and

actors that act maliciously are being punished. In the Ethereum PoS protocol the entities that ensure consensus are called validators and in order to become one, 32 ethers (defined in Ethereum protocol) have to be deposited to the staking contract, which is a known address. Ethers are typically obtained through exchanges.

When a validator performs their duty correctly their balance increases so does their voting power within the network proportionally, where maximum voting power is achieved at 32 ethers. Effectively when a validator starts a node, they have maximum voting power. When they do not perform correctly, then their deposit is reduced. There are different degrees of punishments which are discussed in the slashing conditions section.

Another major concern in distributed computing is how we measure time. As when an actor (later referred in this report as a node) joins the network, they typically do not have an aligned time as the other nodes. In Ethereum the time of the nodes is being synchronized over time when they participate in the network by seeing messages from other nodes, either arriving too early, or arriving too late. This also aligns the latency in the network a bit.

In the Ethereum network time is managed through epochs which are subdivided into 32 slots. Each slot contains at most 64 committees, each consisting of at least 111 [3] and at most 2048 validators. Then a randomly selected validator from a committee is assigned to create a block. This is further discussed in the Validator duties section and time specifications are described in the Epochs and Slots section.

2.2.4 Validator duties

A validator is a node that has staked 32 ETH tokens and actively participates in the consensus protocol. During an epoch the whole validator set is participating but since they are divided into committees, certain committees are used for a single slot, therefore a validator participates only once within a single epoch. Validator responsibilities vary between epochs as each validator is given one of the three responsibilities an epoch ahead of time: Block creator; Attester, Aggregator.

When a validator is selected to be a block creator, or also referred to as block proposer, they are expected to create a block at the beginning of a slot and broadcast the block to the other nodes within the network. The blocks contain transactions with monetary value that were submitted ahead of time and also aggregated attestations (see **subsection 2.2.8**) typically generated from the second and third stage of the previous slot (see **subsection 2.2.5**). Once the epoch becomes finalized (see **subsection 2.2.7**) then all transactions that happened in each of the blocks within it are considered final.

When a validator is selected to be an attester, they are expected to broadcast an attestation after

1/3 of the slot time has passed or once they have received the block created by the block proposer for the current slot, whichever happens first. An attestation can be broadcast at a later stage, but the reward gets reduced based on the delay counted by the number of slots using the following formula:

$$reward = \frac{base\_reward}{inclusion\_delay}$$

base_reward is a variable based on the total and effective balances of the validator and *inclusion_delay* is the delay in terms of block count. There are different rewards within a block and the distribution of those is represented in the following table.

Sync committee	2/64
Timely head	14/64
Timely target	26/64
Timely source	14/64
Block creator	8/64

Despite the fact that only 8/64 of the block reward goes to the block creator it is the highest reward a validator is getting because it is currently not split between multiple validators while the other 54/64 is being split among all attesters and the last 2/64 of the reward is split between 512 validators.

When a validator is selected as an aggregator their responsibility is to collect all attestations of a committee. All validators from the committee that broadcast an attestation with the same data are aggregated to a single attestation that has a weight equal to the sum of all attestations it is an aggregate of. The higher the weight of the aggregated attestation the higher the chance it will be included in the next block. There will be only a single aggregated attestation if all attesters vote on the same data, but there can be more if there are attesters who see the network in a different way. In other words, there is an aggregated attestation for each unique piece of attestation data. The weight of an aggregated attestation is not derived from the number of votes behind it, but rather from the sum of the effective balance of the attesters (see **subsection 2.2.10**).

2.2.5 Epochs and Slots

As already briefly mentioned earlier, epochs and slots are the necessary data structure for the Ethereum network to track time. Epochs are a collection of 32 slots and each slot may contain a block. The slot duration is 12 seconds and it is subdivided into 3 stages of 4 seconds each: block creation and propagation; attestations and propagation; aggregation of attestations.

In the first stage, the block proposer creates a block and broadcasts it to the rest of the network.

In the second stage attesters will post an attestation either when they receive the proposed block or when the first stage is over (4 seconds have elapsed). In the last stage aggregators will simply aggregate signatures of the attesters voting for the same data and will produce an aggregate signature.

2.2.6 Slashings

Slashings are the way to disincentivize malicious activity in the network. They are a way to reduce the malicious nodes' balance significantly and reduce their voting power. There are two slashing mechanisms: proposer slashing; attester slashing; and a network state that can also be considered as a slashing mechanism - inactivity leak state or also called 1/3 slashable network.

Proposer slashing is imposed when a block proposer proposes at least two blocks on at least two forks of the chain.

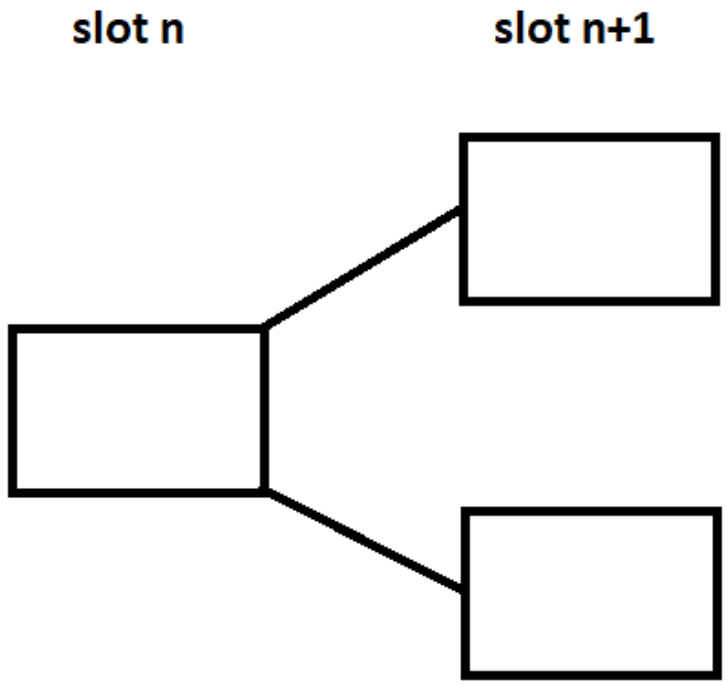

Figure 2.1: Block proposer at slot n+1 produces two blocks at the same slot, ultimately leading to *proposer slashing*

In **Figure 2.1** we can see a violation, since the slot n+1 has an appointed validator to be the block proposer, there cannot be two blocks for the same slot on two forks unless the validator created the block on both.

Attester slashing is imposed when an attester posts an attestation on at least two forks of the chain. Effectively proposer slashings and attester slashings are a way to disincentivize nothing-at-stake problem [4]. Attester slashing can also occur for creating an incorrect attestation where the source Epoch Boundary Block (EBB) is after the target EBB. (see **subsection 2.2.7**)

The inactivity leak state is when 1/3 of the balance of validators is inactive and 4 consecutive epochs have not been finalized. When the network enters inactivity leak state then the following changes take place:

1. Attesters receive no rewards for attesting correctly
2. Block proposer and aggregator rewards are unchanged
3. Inactive attesters receive a penalty, which quadratically grows with each epoch until either an epoch is finalized or they are ejected from the network, which happens when the attester has less than 16 ETH in their balance

Another side of the same problem could be that there are two epochs on two forks that are both finalized. That is only possible when 1/3 of the attester set is attester slashable. This is the case because in order to finalize an epoch a supermajority (66.67%) is required, therefore at least 1/3 of the attesters, based on the weight of the stake they have, have posted attestations on both forks for the same slot.

2.2.7 Justification and Finalization

Justification and finalization refer to epochs. When an epoch is finalized it means that the transactions processed by its blocks are immutable unless the network is 1/3 slashable (see **subsection 2.2.6**).

The first block in an epoch is called epoch boundary block, slots after the first in an epoch generate regular blocks.

Figure 2.2: Regular epoch boundary block

EBBs are checkpoints that validators of the current epoch vote for and when they get 2/3+1 weight of attestations, later referred to as supermajority, the epoch containing the EBB as source is finalized while EBB as the target becomes justified. The EBB is the point when validator balances are updated, committees are shuffled, slashings and validators exits are processed. In case the EBB was not created, because a validator was offline or because they were validating on a private fork then the last block from the previous epoch is "borrowed" as the EBB for the new epoch, see **Figure 2.3**.

Figure 2.3: Missed epoch boundary block

During an epoch validators broadcast attestations that both contain FFG vote and LMD-GHOST vote (see **subsection 2.2.9**). The FFG vote is used to finalize epochs while the LMD-GHOST vote is used to add weight to the block so that LMD-GHOST fork choice rule can lead to the latest block. Furthermore by attesting to the latest block being the head of the chain, the validator also confirms that the data within the block is correct.

2.2.8 Attestations

An attestation is the way validators in the network confirm that a block is valid and support the current data. An attestation includes information for the slot it is for, index of the committee that is voting, the head of the chain according to LMD-GHOST, source EBB and target EBB. In order to incentivize validators to create and broadcast the attestation on time the reward is reduced over time. If 1/3 of the attestations are sufficiently delayed, then a supermajority is not achieved and the epoch might not get timely finalized and all transactions within it as well. If an epoch is finalized and an attestation in the following epoch is included after the supermajority for the previous epoch was achieved, the reward gets even further reduced.

An aggregated attestation is an attestation that is agreed upon by the committee. The way validators agree on attestation is by simply broadcasting the same attestation data. Typically, in the network an attestation will be broadcast before the validator has seen any other attestations. Afterwards, all the attestations with the same attestation data by the same committee will be aggregated into a single attestation, which increases the chances of that attestation being included

in the next block. If an aggregated attestation does not include many members of the committee, it might not be included in the next block, thus reducing the reward earned by the attesters. An attestation will not be included in the block if 128 attestations are already included in the current block as 128 is the maximum number of attestations per block.

There are scenarios in which an attestation for slot N could be confirming block at slot N-1, if the block at slot N was not observed in the first stage of the slot. A block that has not been created for a slot is called a missed block. Missed blocks typically happen when the block proposer is not online, therefore does not propose the block.

2.2.9 Latest Message Driven Greedy Heaviest Observed SubTree (LMD-GHOST)

LMD-GHOST is the algorithm that the ethereum protocol adopts to determine which is the head of the chain, the latest block. Based on the latest set of attestations the head can be determined by following the highest attestation weight at each conflicting fork. In the following figure we can see 2 forks of the chain and we can also observe the weight of each fork. LMD-GHOST is the fork choice rule for ethereum and it will always select the canonical chain to be the one with the highest weight of blocks.

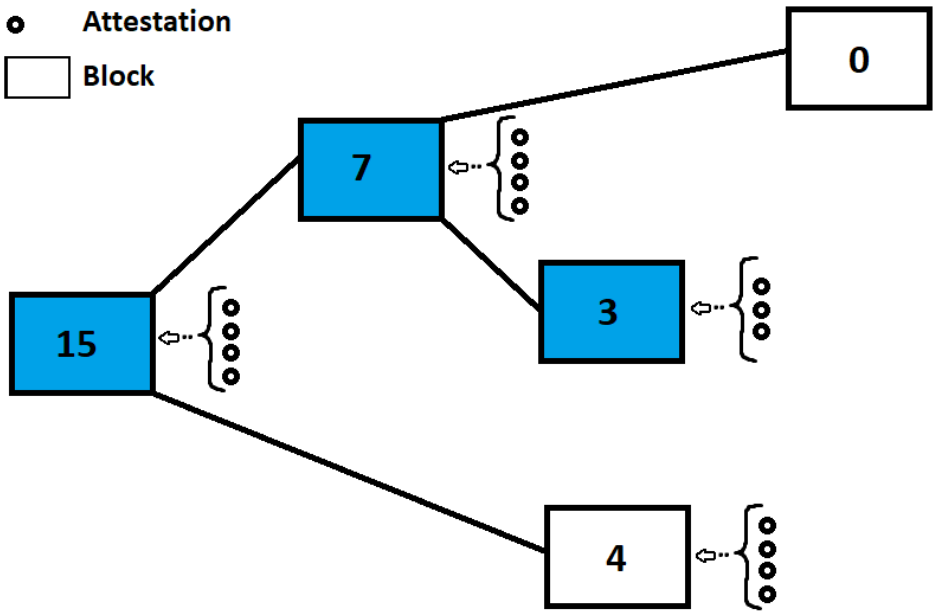

Figure 2.4: Each attestation on a block that is a child accumulates weights also for the parent thus the weight of 15 for the first block. In the current scenario the canonical chain would be **15-7-3**, indicated with blue blocks. The weight of attestation is the sum of ether staked by all committees aggregated attestations and not the number of attestations.

2.2.10 Effective balance

Incentives within the network are determined based on the balance of the validator. More specifically the importance of a validator is based on the effective balance they have. If a validator has less than 16 ETH tokens they will be ejected from the validators set, alternatively if a validator has more than 32 ETH their effective balance will be 32 ETH. The effective balance is rounded to the lower whole number. For example if a validator has 31.75ETH their effective balance is 31 ETH. [5]

2.2.11 Memory pool

The memory pool is the place where nodes record transactions that are to be processed. When a block proposer is to generate a block, they request nodes for the transactions that will bring the highest value and include them in the block. When there are more transactions recorded by the nodes per block than the ones included in blocks, the fees go up and when there are more transactions being processed within a block than the transactions added to the memory pool during a block, the fees go down.

2.3 Reorg Attack

In this section we are explaining the attack step by step and then we present an example scanario how the attack can be carried out.

2.3.1 Overview

A block reorganization attack is done when the attacker privately creates and attests to a block and delays its release to the network (selfish mining), so that a number of later blocks are orphaned. In Ethereum PoS, when attesters do not see a block in the current slot, they attest to the latest block they see.

For example, the time for slot n comes and attesters see block n in it, so they attest to it as being the head of the chain they see. Now suppose slot n+1 comes up, but no block is proposed. So the attesters from n+1 vote for block n being the head of the chain as there is no block in n+1 (this is what happens when a block is missed or intentionally held back, there is no way the honest attester can differentiate between the two).

An attacker with a significant stake (for the most part of this document we consider a 30% stake)

of the network can take advantage of this. When the attacker is selected as a block proposer, they can withhold their block and not share it with the rest of the network.

1. Attacker creates a block at slot n+1 and does not broadcast it to other validators
2. Honest validators have not observed block so they attest to that block n is the head of the chain, while dishonest validators attest to that n+1 is the head of the chain
3. Honest validator creates block at slot n+2
4. Honest validators attest to block at slot n+2 being the head of the chain while dishonest validators ignore the block and attest to block at slot n+1 being the head of the chain

The attacker can now disclose their n+1 block to the public. This is now a fork situation where on one chain we have a missed n+1 block and on the other (attacker's) we have a missed n+2 block. Since the attacker has the attestations of n+1 and n+2 while the honest attestations are only from n+2, it is possible that the attestations of the attacker voting for n+1 are more than the honest attesters voting for n+2. This is simply because the honest attesters from n+1 voted for n and n is the root of both forks. If the attacker has more attestations, LMD-GHOST will select the attacker's chain being the correct one, effectively orphaning the valid n+2 block.

In practice, it can be difficult to gather enough malicious attestations as an attacker by just withholding one block. Usually it is necessary for the attacker to be selected as block proposer at least twice in a row, so they can withhold 2 or more blocks, which results in their private fork having more attestations to put against a future block created by an honest validator. The reason behind this is very simple - with each withheld block, the attacker gets more and more attestations on their chain, while the honest attestations go to the parent node of both forks, so they are not conflicting with the attacker.

Another worthwhile observation is that the more attestations the attacker can gather, the longer the reorg attack they can execute. In other words, they can wait to release their attestations until a number of blocks (n) are created by the honest validators resulting in 'n' blocks being orphaned. Counting 'n' starts from the first honest block and continues until the attacker decides to propagate their fork through the network. We refer to this as an n-length reorg attack. The higher n is, the harder and less likely it is to successfully execute such an attack, as the attacker needs to be selected as a block proposer a higher number of times consecutively. [1]

2.3.2 Example of the attack

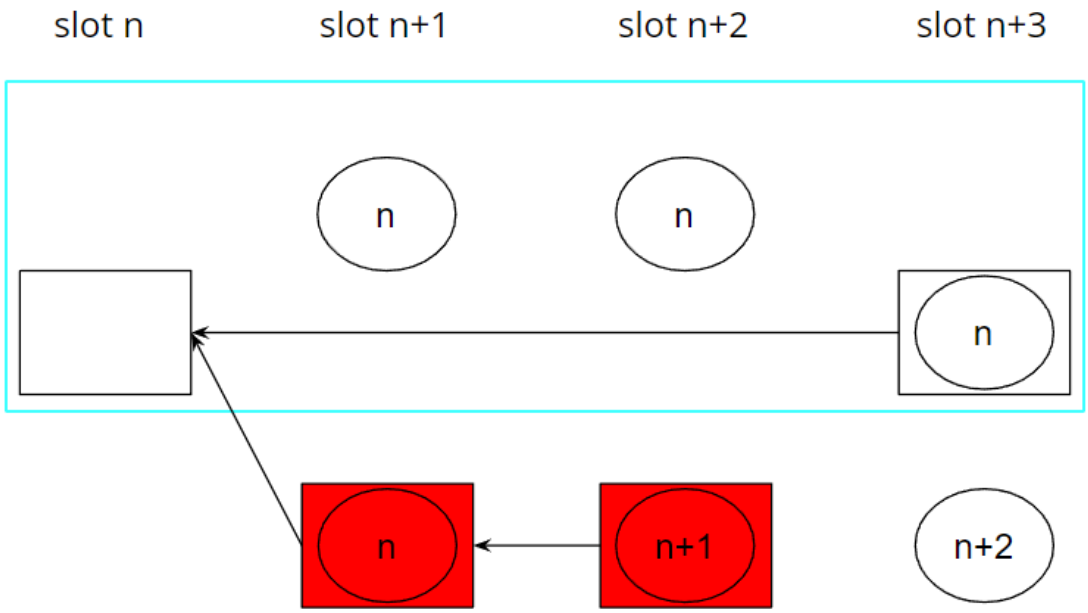

Figure 2.5: In the figure the circles represent the attestations that validators generate, the rectangles are blocks, the blue rectangle is the view of the honest network and the red rectangles form the private fork of the dishonest validator. (Note that attestations from the honest network are also included in the private fork and the attestation n+2 is also included in the block at slot n+3)

The block at slot n is the forking point where the attack begins. At slot n+1 the malicious actor creates a private block (marked as red in **Figure 2.5**) and includes attestations for n (note that attestations generated at slot n are included the earliest at slot n+1). At slot n+2 the honest validators have not observed block at slot n+1 so they attest to block n being the head of the chain, while the malicious actor includes the attestations generated at slot n+1 and creates attestations for the current block, at slot n+2. Then an honest validator creates the block at slot n+3. This block includes the attestations generated by the honest network (attestations for slot n+1 and n+2) together with the attestations for block n+2, even though they have not observed it yet. Then the malicious actor releases the private fork to the honest network before the block at slot n+4 is to be generated and the block creator analyzes the weight of both forks.

Assuming that at each slot the dishonest validator (with all the nodes) has a weight of 30% (0.3) and the honest validators have 70% (0.7) at each slot when the time comes for block at slot n+4 to be generated, the block proposer sees the formerly private fork has a weight of 0.9 (0.3 from slot n+1 + 0.3 from slot n+2 + 0.3 from slot n+3) while the previously honest fork has only 0.7 (n+3) so the parent of the block at slot n+4 becomes the block at slot n+2. The reason why we do not take into account the weight of block n is because both forks accept it as a parent so block

at slot n does not conflict with either fork as it is included in both of them.

2.4 Reorg Attack in practice

Now that we have described what a reorg attack is, it is time to explain how it works in the real world. We will focus on probabilities and frequencies of different lengths of reorg attacks. First, let us discuss the motivation behind such an attack. While some cyber attacks are done just to assert dominance as a hacker, show a weakness in a system or simply cause pain and harm, the majority of attacks have some selfish incentive behind them, more often than not in the form of money. According to [6], reorg length 1 attacks can be executed on a daily basis with just as little as 200 nodes in the network.

One of the consequences of the attack is to delay the finality of the network. Assuming a 30% attacker, the attacker needs to be block proposer for the first two blocks of an epoch - the EBB and the block after. (e.g. EBB is block 32 and the block after is block 33) to successfully delay the finality of the previous epoch and the justification of the current one. The strategy for the attacker is to privately propose the EBB - block 32. Then they privately propose block 33 and cast a single attestation to block 32 as head of the chain and EBB. Meanwhile for slot 32 and 33, the honest validators attest that block 31 is the EBB, as 32 is missing. After the attestations in 33 are cast but before 34 is proposed, the attacker releases their 2 blocks to the public and holds all their attestations but one. From block 34 to the end of the epoch the attesters will select block 32 as the EBB. However, the older attestations are just enough so that supermajority cannot be achieved on block 32 as EBB resulting in the current epoch being unjustified and the previous one unfinalized. This can be observed in the following calculations: There are 32 blocks in an epoch, therefore there are 1/32 = 3.125% attesters in each slot. Therefore two slots have 2*3.125% = 6.25% of the attestations. 30% of those belong to the attacker, which leaves 6.25%*0.7 = 4.375% of the honest attesters to be in these 2 slots. These 4.375% of attesters vote on block 31 being EBB. Which obviously is not enough for supermajority. The attacker holds their 30% of attestations for the entire epoch. This means that at most 100%-30%-4.375%=65.625% at most can vote on block 32 being EBB. Which is also not enough for supermajority, therefore the current epoch cannot be justified, which results in the previous epoch not being finalized.

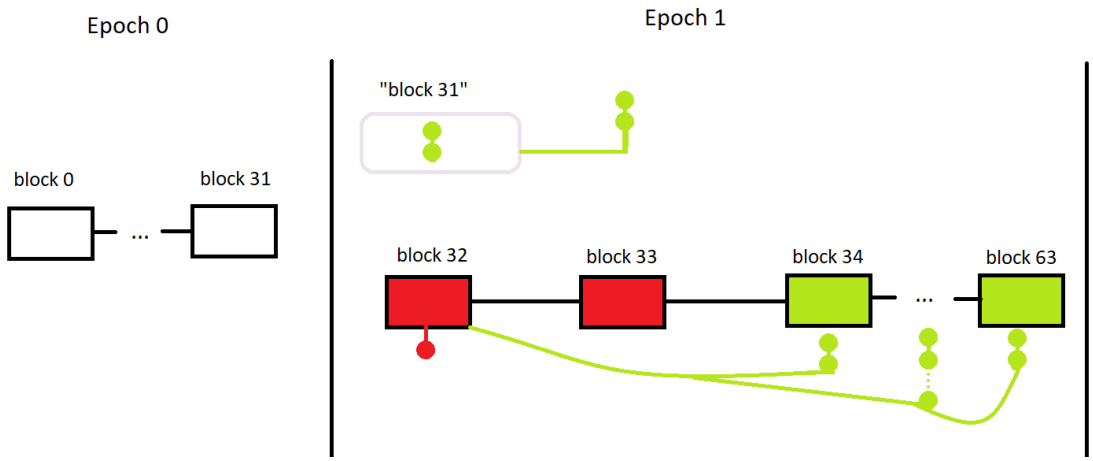

Figure 2.6: The attestations in this figure illustrate votes for the target EBB rather than the head of the chain.
Greed - honest attesters
Red - dishonest attesters

In reality, an attacker is unlikely to be financially incentivised to perform a finalization delay attack, especially with 30% stake. If epochs are often not finalized, the network will be considered by people generally unreliable due to these delays. Then it is likely that the price of the coin of the network will drop, which is definitely not in the financial interest of an attacker. This leads us to the conclusion that this attack is not very practical, despite being able to be executed with high probability.

Another aspect of this attack is the possibility to 'farm' high fee transactions. This can be achieved by orphaning a block with well-paid transactions. Then these transactions are added to the memory pool. If the attacker is the proposer for the next block, they can include the highest cost transactions in their block, increasing their rewards at the expense of others. This can be further extended to distribute attestations of the honest network within the fork, maximizing the gain in a block with high transaction fees, since the block reward as seen in **Subsection 2.2.4** is distributed by fractions and then split between the validators that participate in the block.

2.5 Existing Proposal

In this section we explain existing proposal that resolves the block reorgs vulnerability followed by advantages and disadvantages of the solution.

2.5.1 Proposal

Michael Neuder, Daniel J. Moroz, Rithvik Rao, David C. Parkes have previously worked on a proposal for resolving the block reorgs vulnerability by modifying the fork choice rule (see **Subsection 2.2.9**) [1]. The essence of their proposal is to change the LMD-GHOST vote of the attestation from voting for the head of the chain to instead vote for a pair consisting of the head of the chain and the slot the vote is for.

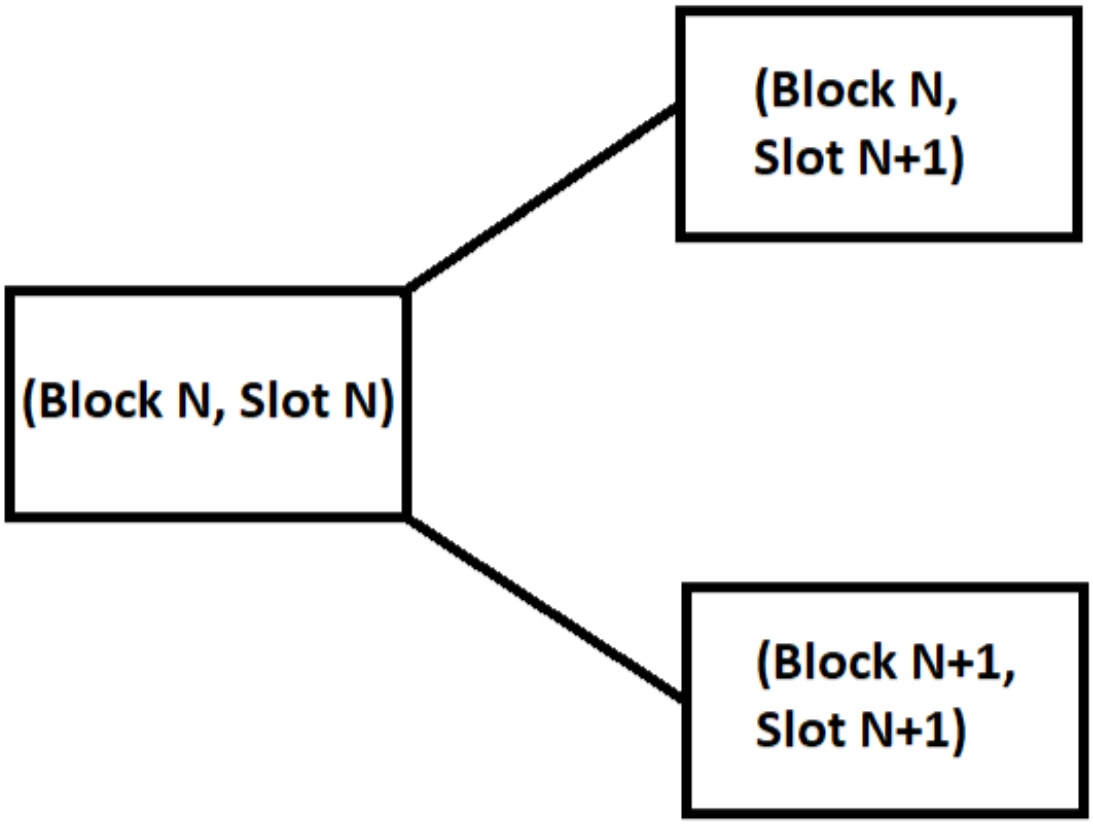

Figure 2.7: LMD-GHOST Attestation consists of a pair block and slot

In **Figure 2.7** we can observe that at slot N the attested block was N. In the next slot attesters that have not observed block N+1 would attest with (Block N, Slot N+1) effectively changing the behavior of the network such that this attestation would be conflicting with the fork where block N+1 exists.

2.5.2 Advantages

The primary advantage of this solution is that block reorgs are practically impossible to achieve because at each slot if a block was not observed in the first stage (SECONDS_PER_SLOT / 3), in the current configuration that is 4 seconds) then all validators would attest to a blank block. By doing this a malicious actor would not have nearly as close chances of achieving block reorgs as they would not be able to mine selfishly and bring more attestations than the network has. The only way in this solution to achieve block reorgs is when the committee shuffling process results in such a state, where for 2 consecutive blocks the sum of honest validators is less than the sum of dishonest ones, with a distribution of validators 7:3 honest to dishonest validators with 10000 draws at each slot for committees the probability of this occurring is:

$$p \approx 4.7 * 10^{-23}$$

With a rate of 2628000 blocks per year, equal to 82125 epochs, this probability is considerably 0.

The finality delay attack is also resolved by disallowing a selfish fork to mislead 3.3% of the network into casting invalid FFG vote.

2.5.3 Disadvantages

There was one enormous drawback to the proposal which was also the reason why it was rejected. The drawback was that under poor network conditions, where the blocks could not propagate in the network under (SECONDS_PER_SLOT / 3), then the network would stall. This reason was enough to reject the proposal because it would have been 1B dollars to obtain 30% of staking power at the time when it was proposed. If we assume that 30% of validators need to be malicious, then, currently, it would cost 10B dollars. In the pull request it was argued that it would be a lot cheaper to carry out a long-term DOS attack at the network level. Even though (SECONDS_PER_SLOT / 3) is enough time even in poorer conditions, a concern was expressed based on the support for ethereum 1 chain, which currently imposes a 2 second delay that significantly reduces the time a block on the beacon chain would have to propagate. [7]

3 Problem Statement

In this chapter we summarize the findings from the background chapter and based on those we refine the initial problem statement and finally we present the direction of work and necessary steps in order to prove that we have resolved the vulnerability.

3.1 Summary of analysis

3.1.1 Impact of block reorgs

A block reorganization attack allows a malicious user to cause blocks to be orphaned by proposing blocks privately on a competing fork and releasing them with a delay in such a way so the malicious user's block is the one chosen by the attesters as head of the chain.

The reorg attack can be used to delay the finality of the network. Since some transactions will take longer to be finalized, the network is overall slower. If this happens often, it can lead to decreased trust in the network, which in itself means the coin will devalue as well.

Another aspect of this attack is that by canceling blocks, the transactions in them need to be processed at a later block, which increases the number of transactions in the memory pool, leading to an increase in the transaction fees.

3.1.2 Key metrics of the attack

After researching the problem, we realized there are two ways a block reorgs attack can occur: favorable distribution of nodes within the committees; or consecutively getting nodes to be block creators in order to perform a private fork.

Favorable distribution for the dishonest validator is when within consecutive slots they are selected at the first slot of the sequence to be also block creator and can delay this single block

but attest to it. Since they have the majority of the nodes within the committees they can later publish the block and force the network to orphan honest blocks because the weight of the stake they have in these committees is greater than the one the honest validators have.

Consecutively being selected as a block creator allows the dishonest validator to perform a private fork of indefinite length until an honest node is selected to create the block at subsequent slots. Then depending on the length of the private fork a length 'n' attack can be carried out.

We have noticed that the favorable distribution for the dishonest validator has extremely low probability of occurring even with a 30% stake because of the 64 committees per slot and each has about 160 validators. For simplicity, the essential part is that approximately 11000 validators are selected per slot in the current state of the chain and with such a great number of draws the distribution is following the ratio 3:7 dishonest:honest validators per slot. On the other hand, we analyzed the private fork option and it resulted in very high probabilities. A length 5 attack can be carried out daily with 30% stake [1].

3.2 Refining problem statement

The solution proposed in [7] has been an approach to solve the vulnerability with the private fork scenario, which is very elegantly extending FFG [8] but unfortunately due to the comparison of very low likeliness of a validator obtaining 30% of the stake and the drawback of stalling the network in case of network delays it was rejected. For this reason instead of trying to resolve the vulnerability we decided to work in the direction of reducing the probability of it occurring. We can now reformulate our problem statement into the following:

How can we mitigate the block reorgs vulnerability without significant additional overhead on the network?

3.3 Direction of work

In order to prove our solution is actually mitigating the chance of block reorgs occurring we have to do a mathematical analysis on how likely it is to create a sequence of blocks that will result in the desired length attack. We then have to do a mathematical analysis on the probabilities of the same length attacks when our solution is in place and compare it against the current state of Ethereum. From this comparison we will learn whether it is meaningful to implement our mitigation or not.

To confirm the results from our mathematical analysis we shall also run simulations on this attack for specific lengths. Again, we will run simulations with and without our mitigation in place and compare the two.

The results from the simulation should be similar to the theoretical expectations from the mathematical analysis. This can also serve the purpose of a double check, in case our mathematical analysis has problems we did not foresee. Should there be such potential issues, they should appear in the form of discrepancies between the simulations and the mathematical analysis. It also works in the opposite direction, if there are mistakes in the simulation code, its result should not align with the results from our mathematical analysis. In short, if we get different results from theory and practice, we have a problem.

After we have proven that our solution actually fulfills the requirement of reducing the probability of block reorgs occurring we will have to also go back to our constraints from before and analyze the extent to which we have affected the core elements of the blockchain. Afterwards, we will argue how reasonable the changes are and what side effects we expect to occur.

4 | Proposed Solution

In this chapter we present our solution proposal that greatly reduces the chance of dishonest validators with a high stake to succeed in block reorgs attack. Afterwards, we present mathematical analysis of the mitigation strategy and simulation results that confirm that the vulnerability is mitigated to the expected extent and finally we present what are the disadvantages of the solution.

4.1 Solution Proposal

Our solution extends the ethereum chain by electing at every slot one more block proposer. The secondary block proposer will perform the exact same duties a regular block proposer does but when the attesting phase comes the validators will have 3 options:

1. If no blocks proposals are received in the first 4 seconds then attest to the block before.
2. If only secondary proposer block is received, attest to it.
3. If only the primary block or both blocks are received, attest to the primary.

In order for the malicious actor to carry out the attack successfully, they would have to own the nodes that are selected to be block proposer for 2 consecutive blocks, considering 30% stake, in order to remove one honest block, length 1 reorg, from the canonical chain. The reason why they need only two consecutive blocks is because during those 2 blocks the honest validators will attest to the block that is before the fork occurred so they are not conflicting while the malicious actor has attested to their own blocks. In the third block all honest validators attest to the honest block, while all other attesters vote for the selfish fork.

Thus, statistically, the dishonest validator will have higher weight of attestations: 30% of committees per slot for 3 slots, while the honest network statistically only has 70% of the committee members for the last slot and none for the first two slots because honest validators attested on

the block before the fork. Therefore they are not conflicting with the selfish fork. By adding a secondary validator to act as block proposer we reduce the chances of the attack being possible exponentially.

Using our proposal we reduce the chances of a malicious actor getting consecutive blocks without a conflicting fork by randomly selecting two validators from the validators pool, where 70% are honest validators, which results in only 0.81% chance of the malicious actor having both validators to be theirs at both slots, which is reduced from 9%. These probabilities are for the dishonest validator achieving the prerequisite for a length 1 reorg at specific two slots, 2 out of 2 slots. To present the impact of this on the overall network we will present in the next section the mathematical analysis and the probabilities of different length attacks occurring within an epoch.

4.2 Mathematical analysis for 30% stake attacker

To present the degree this technique mitigates the probability of having a block reorg within an epoch we can look at **Table 4.1**.

Length of attack	Before mitigation	After mitigation
1	0.9008	0.2022
2	0.4369	0.0185
3	0.1423	0.0016
4	0.01192	0.000011
5	0.003278	9.1769e-7
6	0.00089	7.4746e-8
7	6.7886e-5	5.1008e-10
8	1.7885e-5	4.01920-11
9	4.6215e-6	3.1031e-12

Table 4.1: Table representing the probability of attack of specific length occurring at least once in an epoch

From the data we can reach to the conclusion that despite having 30% of the stake, which is worth approximately 26 billion USD, the dishonest validator will only have about 20% chance to carry out a length 1 attack within an epoch to the contrary to about 90% before the mitigation. It is also reasonable to assume if a person could afford 30% of the stake, they could also afford 33 1/3%.

To simplify the problem we are going to map the problem. The validators can be considered as balls of two colors, red and black, for dishonest and honest validators respectively. In order to

achieve a length n attack we will need consecutive selection of validators, which in the new problem is drawing consecutively red balls. Finally, the last detail is the epoch constraint, therefore we will have to draw 32 balls and a concise formulation of the problem is the following:

There is a bag with 320 000 balls, 96 000 of which are red (R) and 224 000 are black (B). What is the probability of having a sequence of "n" consecutive red balls considering we draw 32 times?

To calculate "n" we can use **Formula 4.1**, where "x" is the length of attack and "n" is the amount of private blocks needed by the dishonest validator to carry out the attack. In the aforementioned mapped problem we can think of the number of consecutive red balls that we need to draw. When n is not an integer, it will be rounded to the closest greater integer than n.

$$n \geq \frac{0.7 * x}{0.3} - x \tag{4.1}$$

After we have calculated "n" we can define **Algorithm 1**, which accepts 4 parameters: k - the number of red balls we want to draw in a row; n - the total number of draws; R - the number of red balls; B - the number of black balls. The algorithm consists of 3 main sections:

1. Line 2-4: validates the input
2. Line 8-10: calculates the probability of the whole sequence to start with the sequence of red balls
3. Line 12-23: calculates the probability of the sequence being preceded by a black ball but not preceded by a sequence of red balls of length "k"

Line 8-10 covers the case where we have the language described in **Formula 4.2**, visualized in **Figure 4.1** and Line 12-23 covers the case where we have the language shown in **Formula 4.3**, visualized in **Figure 4.2**.

$$\{R^k X^* | X \in \{R, B\} \wedge \mathbf{k} \text{ is a constant}\} \tag{4.2}$$

Figure 4.1: RRB, which is part of the language described by **Formula 4.2**

$$\{Y^*BR^kX^* | X \in \{R,B\} \wedge \mathbf{k} \text{ is a constant} \wedge Y \in \{R,B\} \wedge R^k \nsubseteq Y\} \tag{4.3}$$

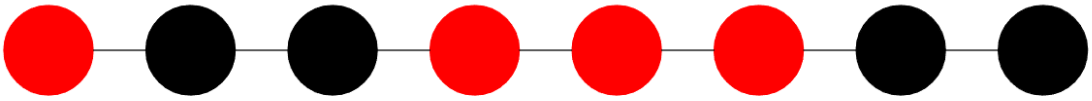

Figure 4.2: RBBRRRBB, which is part of the language described by **Formula 4.3**, shows an example of length 2 attack for a dishonest validator with 30% stake

To prepare the mathematical algorithm for calculating the probability of the attack occurring considering the mitigation strategy we will have to modify the problem slightly. Instead of drawing a ball each time we will draw a pair of balls to represent the primary and secondary validator

There is a bag with 320 000 balls, 96 000 of which are red (R) and 224 000 are black (B). What is the probability of having a sequence of "n" consecutive red pairs of balls considering we draw 32 times a pair?

To adapt **Algorithm 1** we will have to change the following:

1. Line 9: we will have to change this to adapt to the change that now we are drawing 2 balls at this slot instead of 1.
2. Line 10: Since we are drawing a pair now we will have to remove 2 balls from the bag.
3. Line 15: This time because we work in pairs it does not matter if we have a red ball or black ball preceding the sequence of red balls, instead we need a pair that is not consisting of two red balls and therefore instead of calculating and summing the probability of the 3 other pairs to occur we will simply calculate the probability of two red balls being drawn consecutively and subtract the probability from 1, which will yield the same result.
4. Line 16-17: We will also have to modify what we take out of the bag so we will remove 2 black balls and 1 red ball from the bag as the pair has 3 states: RB, BR,BB which statistically will draw 2:1 B:R balls and because we work with whole numbers, that are also rather big (96 000 red, 224 000 black), we are more interested in the ratio thus we will not remove 1.33(3) black and 0.66(6) red balls.
5. Line 18: Similarly to Line 9 will need to be adapted for drawing a pair.

Line 8-10 covers the case where we have the language described in **Formula 4.4** and Line 12-23 covers the case where we have the language shown in **Formula 4.5**

$$\{R^{2k}X^* | X \in \{R, B\} \wedge \mathbf{k} \text{ is a constant}\} \tag{4.4}$$

$$\{Y^*TR^{2k}X^* | X \in \{R, B\} \wedge \mathbf{k} \text{ is a constant} \wedge Y \in \{R, B\} \wedge R^k \nsubseteq Y \wedge T \in \{RB, BB, BR\}\} \tag{4.5}$$

```
Input  : k, n, R, B
Output: probability of having k consecutive red balls in a sequence of n draws
Function name: getProbabilityForSequence
if k > n or k > R or n > (R + B) then
    return 0;
end
define probability = 1;
define r = R;
define b = B;
foreach j in k do
    probability = probability * r / (r + b);
    r -= 1;
end
foreach z in (n-k) do
    r = R;
    b = B;
    define prob = b / (r + b);
    b -= 1;
    foreach j in k do
        prob = prob * r / (r + b);
        r -= 1;
    end
    prob = prob * (1 - getProbabilityForSequence(k, z, r, b));
    probability = probability + prob ;
end
return probability ;
```

Algorithm 1: Algorithm for finding the probability of at least one occurrence of a sequence in a sequence

```
Input  : k, n, R, B
Output: probability of having k consecutive pairs of red balls in a sequence of n draws
Function name: getMitigationProbabilityForSequence
if k > n or k > R or n > (R + B) then
    return 0;
end
define probability = 1;
define r = R;
define b = B;
foreach j in k do
    probability = probability * (r / (r + b)) * ((r - 1) / (r + b));
    r -= 2;
end
foreach z in (n-k) do
    r = R;
    b = B;
    define prob = 1 - (r / (r + b)) * ((r - 1) / (r + b));
    b -= 2;
    r -= 1;
    foreach j in k do
        prob = prob * (r / (r + b)) * ((r - 1) / (r + b));
        r -= 2;
    end
    prob = prob * (1 - getMitigationProbabilityForSequence(k, z, r, b));
    probability = probability + prob ;
end
return probability ;
```

Algorithm 2: Algorithm for finding the probability of at least one occurrence of a sequence of pairs in a sequence

When we consider the results from **Table 4.1** we expect discrepancies when comparing later the results for length 3, length 6 and length 9 attacks. This is because when we calculate the mitigation extent through mathematical analysis, we use the lesser number of blocks needed to carry out an attack. If we assume even distribution then in real scenario we expect that the dishonest validator would ensure all of his nodes to be online. Meanwhile the honest validators are likely to have some nodes that are offline. This is why in a case where the honest and dishonest attesters are even, the dishonest 'win'.

If we look at **Formula 4.1**, when we calculate n to be an integer then there is an equality between the honest and dishonest validators in both forks. In the pessimistic scenario for the attacker we could increase n with 1, and in the optimistic we present the data as in **Table 4.1**. Therefore, we expect the results from the simulation to present more accurate results for those cases because, the simulation is based on actual selection of validators and the distribution of validators within an epoch on each fork can be compared to precision of 1 validator which is insignificant given the fact the network consists of 374133 real validators on the Beacon Chain as of writing this report and 320000 in our simulation. [9]

To visualize the data from **Table 4.1** we have plotted the data in **Figure 4.3** and to present the degree of mitigation we have also transformed the data into a logarithmic scale in **Figure 4.4**

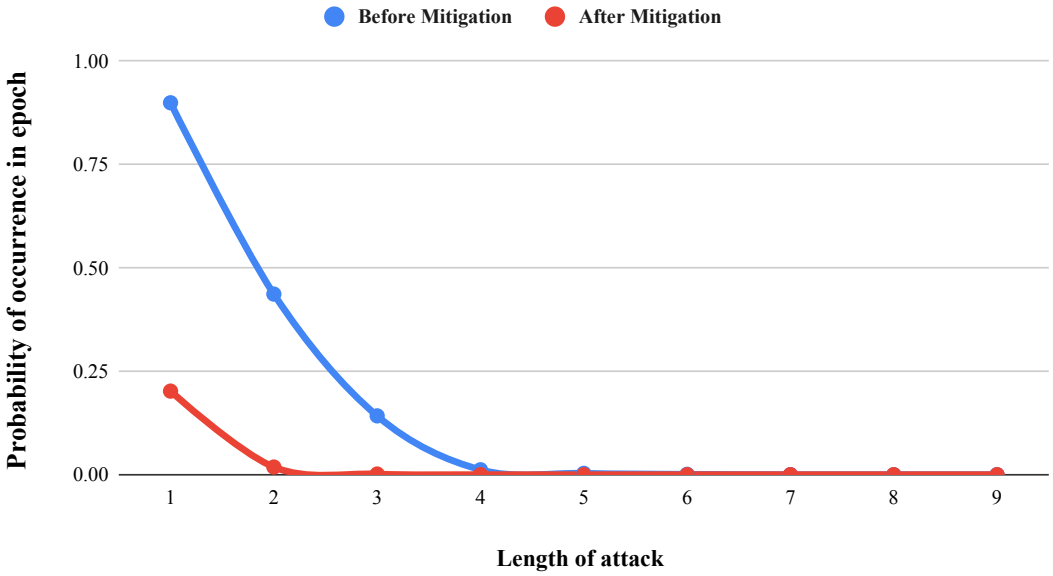

Figure 4.3: Visualization of data from Table 4.1

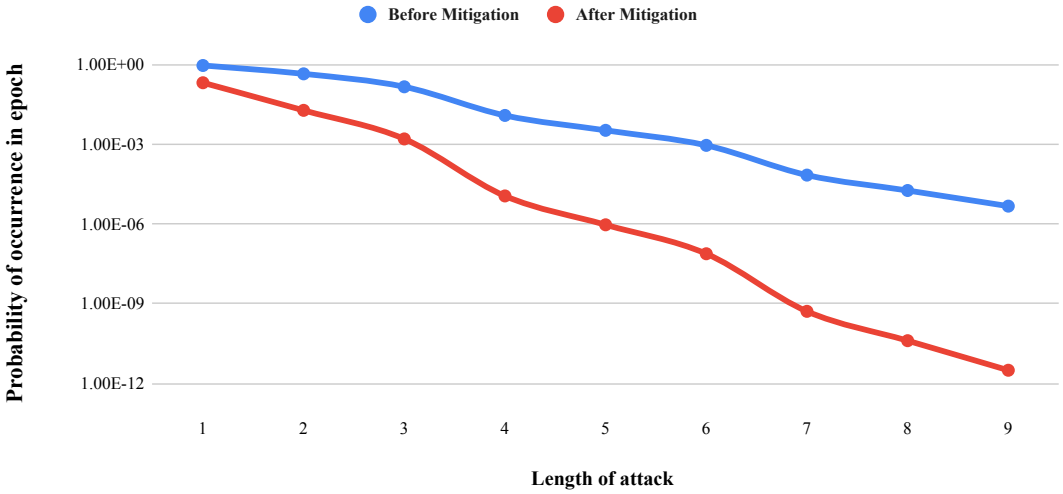

Figure 4.4: Visualization of Table 4.1 on logarithmic scale

4.3 Simulations

In order to verify the results from our mathematical analysis, we have decided to also run simulations, for the specification of the simulation refer to **Section 4.3.1**. The simulations represent accurately how certain aspects of the ethereum network work, which are relevant to our problem. Each simulation run has a number of rounds. One round corresponds to one epoch in Ethereum. So running a simulation of 100 rounds 10 times or running it once with 1000 rounds makes no difference as a round is completely independent of another. What we simulate is the validator distribution/selection process, for the block proposers and attesters. We have two different implementations, one before and one after the mitigation. The only difference is that we draw 2 block proposers and both need to be dishonest for the actual proposer to be considered dishonest. This is due to the fact that if either primary or secondary proposer is honest, they will distribute it to the network instead of withholding it, therefore such a block is not helping the attacker, just the same way when the single proposer in the pre-mitigation implementation is honest.

The block proposer is chosen in the following way - one validator at random from all the validators of a slot. A validator can either be honest or dishonest (belonging to the attacker). We generate a random number between 0 and 1. If it is less than the ratio of dishonest validators to total validators, then the ‘drawn’ validator is considered dishonest. For example, if we have 30 dishonest and 70 honest validators, and we generate a number between 0 and 1, if the number is smaller than 0.3 it is dishonest, therefore a 30% chance to be selected when one has 30% stake.

For every epoch the first validator ‘drawn’ is the block proposer. When a validator is ‘drawn’,

they are removed from the pool of validators of that epoch, just as is the case in the real Ethereum network. We also keep whether the block proposer is honest or dishonest, which we use later. Each slot contains 1/32 of the validators, see **Figure 4.5**, which is also so in real world Ethereum. And also just like in Ethereum, one is a block proposer and the rest are attesters. Similarly, in the implementation after the mitigation there are two proposers and for the honest/dishonest field to be saved as dishonest, we need both proposers to be dishonest. After drawing all the validators for a slot, we save the number of dishonest validators in a property.

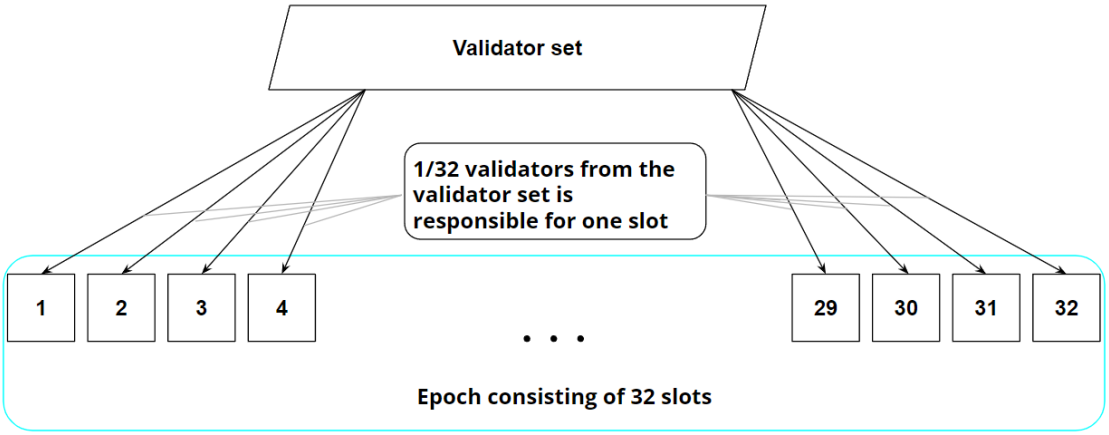

Figure 4.5: Visualization of an epoch. For each slot a subset of 1/32 validators from the whole validator set is assigned, each validator participates exactly once throughout the epoch

In order to look for an attack, we look for blocks proposed by dishonest validators and start counting their attestations. When we see a block proposed by an honest validator, we also start counting the honest attestations. From this point, for every block we add 1 to the length of the attack until the dishonest attestations are less than the honest ones.

Furthermore the number of total validators in our simulation is very close to the real number of validators in ethereum - we have 320 000 validators. Since it is the stake that determines weight, for simplicity we consider each validator has a 32 ETH stake. In reality there are 374,133 validators in the real Ethereum network at the time of writing. Their stake totals at 12,547, 176 ETH. Which results in 33.5 ETH per validator staked. [9] As already mentioned when a validator has more than 32 ETH, their weight is still the same as 32 ETH, while if it is less than 16 the validator is no longer active.

Having this in mind, we believe that the numbers selected for the simulation are very close to reality and that our simulations will paint an accurate picture of what to expect in terms of Reorg Attacks before and after our solution is applied.

Once an epoch is over (we have drawn all validators for all 32 slots), then we check for attacks.

We assume the attacker always acts dishonestly, as they know in advance whether their attack will succeed or not. We also assume that honest validators always act honestly, because they are incentivized to do so.

Since an epoch is a very well established slice of time in the Ethereum network, our simulations intend to show the likelihood of an attack happening within an epoch. For example, if an attacker is the block proposer of the last 2 blocks in an epoch, they can cancel the first block in the following epoch. However, we do not count this as it is outside the epoch boundary. Also, we only count if at least 1 such attack happened in the epoch. This means that if there are 10 reorg one attacks or 1 reorg one attack within a single epoch, we count it as one - we count the number of epochs in which a certain incident happened, rather than the number of incidents happening throughout the life of the network. Another important note is that a length 'n' attack also counts as a length 'n-1' attack. So if we had a length 4 attack in an epoch, it means we also had a length 3 attack in that epoch, which also means we had a length 2 attack in that epoch and so on.

To be able to get a good idea of the frequency of high length reorg attacks and to be able to more accurately see the impact of reorgs and our mitigation, we have decided to run 10 million epochs worth of simulations per scenario. This is equivalent to about 120 years of running the Ethereum network per scenario. In reality we do 10 000 080 rounds per simulation because this number is divisible by 24, which is the number of logical processors on our test system, which in term determines how many threads we can split the simulation in order to do it faster. To put into perspective, an epoch in Ethereum is 384 seconds (32 slots * 12 seconds each). This means there are approximately 9 epochs in an hour, 225 epochs in a day, 1575 epochs in a week, 6750 epochs in a month, 82 125 epochs in a year.

4.3.1 Simulator definition

To complete our definition of the simulator we are going to introduce the algorithm we use. We will first define an algorithm for determining whether we draw a red or black ball - **Algorithm 3**.

Input : R,B
Output: Return 1 if the ball is red (R) or 0 if the ball is black (B)
1 **Function name**: drawBall
2 **return** genRandomNumberBetween0And1 <= R/(R+B) ? 1 : 0 ;

Algorithm 3: Algorithm for determining whether an honest or dishonest validator is selected. Input consists of 2 parameters: R - the amount of red balls (dishonest validators); B - the amount of black balls (honest validators).

Then we will define the next algorithm that is responsible for generating an epoch - **Algorithm**

4. This algorithm generates an array of length 32. Each element in the array represents the block that is generated for that specific slot and contains information of how many red balls were drawn in this block and also whether the block belongs to the red balls. The block will belong to the red balls if the first selected validator for that slot is red.

```
Input  : R,B
Output: Returns an array of 32 objects (1 for each slot), each having 2 properties: red -
        how many red balls were drawn; isOwnedByRed - if the block creator is the
        dishonest validator
1  Function name: generateEpoch
2  define epoch = [];
3  foreach j in 32 do
4      define drawnRedBalls = 0;
5      define isBlockCreatorRed = false;
6      foreach i in (R+B)/32 do
7          if drawBall(R,B) == 1 then
8              if i == 0 then
9                  isBlockCreatorRed = true;
10             end
11             drawnRedBalls = drawnRedBalls + 1;
12             R = R - 1;
13         else
14             B = B - 1;
15         end
16     end
17     epoch.add(red: drawnRedBalls, isOwnedByRed: isBlockCreatorRed)
18 end
19 return epoch;
```

Algorithm 4: Algorithm for generating an array with 32 objects inside each representing a slot and the array represents the epoch. Input consists of 2 parameters: R - the amount of red balls (dishonest validators); B - the amount of black balls (honest validators).

After we have generated the epoch we will need to analyze it for attacks. We do this in **Algorithm 5** and the result is an array where the value at *index* represents how many times attack of length *index + 1* occurred within the epoch.

```
Input  : epoch, R, B
Output: An array where each index represents how many times the attack of length index +
        1 occurred within the epoch
Function name: analyzeEpoch
define localLength = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
foreach (slot, index) in epoch do
    if slot.isOwnedByRed then
        define flag = slot.isOwnedByRed;
        define sumOfRed = slot.red;
        define currentIndex = index;
        while flag and currentIndex != 0 do
            if epoch[currentIndex - 1].isOwndedByRed then
                sumOfRed = sumOfRed + epoch[currentIndex - 1].red;
                currentIndex = currentIndex - 1;
            else
                flag = false;
            end
        end
        define lengthOfAttack = 0;
        define attackFlag = true;
        define attackCurrentIndex = index;
        define sumOfBlack = 0;
        while attackFlag and attackCurrentIndex != 31 do
            sumOfBlack += (R+B)/32 - epoch[attackCurrentIndex + 1].red;
            sumOfRed += epoch[attackCurrentIndex + 1].red;
            if sumOfRed > sumOfBlack then
                lengthOfAttack = lengthOfAttack + 1;
                attackCurrentIndex = attackCurrentIndex + 1;
            else
                attackFlag = false;
            end
            if lengthOfAttack > 0 then
                define attackArray = new Array(lengthOfAttack);
                foreach (_, index) in attackArray do
                    localLength[index] = localLength[index] + 1;
                end
            end
        end
    end
end
return localLength;
```

Algorithm 5: Algorithm for collecting information about the attacks that occurred within the epoch. Input consists of 3 parameters: epoch - an array of 32 objects each representing a slot; R - the amount of red balls (dishonest validators); B - the amount of black balls (honest validators).

Lastly after we have defined the prerequisites we can define the aggregator **Algorithm 6** that will generate epochs and analyze them sequentially and finally aggregate the results, which allows for having parallel runs and combination of the results afterwards. We can do this because in the production Ethereum network the only link between epochs is the randomness, which determines the committees and the roles of the validators for the specific epoch. In our setup we use pseudo randomness to generate a number between 0 and 1 in **Algorithm 3** so we do not have any handover between the epochs. When we aggregate the results we ignore multiple occurrences of a specified length attack within single epoch and count it as one.

```
Input  : rounds, R, B
Output: An array where each index represents how many times the attack of length index +
        1 occurred (If multiple attacks occurred in the same epoch they are counted only
        as one)
1  Function name: simulation
2  define length = [0,0,0,0,0,0,0,0,0,0,0,0,0];
3  foreach _ in rounds do
4      define epoch = generateEpoch(R, B);
5      define attacksArray = analyzeEpoch(epoch, R, B);
6      foreach (number, index) in attacksArray do
7          if (number != 0) then
8              length[index] += 1
9          end
10     end
11 end
12 return length;
```

Algorithm 6: Algorithm for simulations. Input consists of 3 parameters: rounds - the number of epochs to be generated; R - the amount of red balls (dishonest validators); B - the amount of black balls (honest validators).

To accommodate later the secondary validator in the algorithms we modify **Algorithm 4** to account for the secondary validator. To do this we will redefine **Algorithm 4** but when we are drawing the second ball within the epoch if it is black we will set *isBlockCreatorRed* to false as this would mean that the dishonest validator will not be able to create a private fork. In the real scenario if the dishonest validator attempts to create the fork the honest one will propose his block and since the rest of the network would not have seen the primary validator's block they will attest to the secondary. The new definition of the algoritm is presented in **Algorithm 7**

```
Input  : R,B
Output: Returns an array of 32 objects (1 for each slot), each having 2 properties: red -
        how many red balls were drawn; isOwnedByRed - if the block creator is the
        dishonest validator
1  Function name: generateEpoch
2  define epoch = [];
3  foreach j in 32 do
4      define drawnRedBalls = 0;
5      define isBlockCreatorRed = false;
6      foreach i in (R+B)/32 do
7          if drawBall(R,B) == 1 then
8              if i == 0 then
9                  isBlockCreatorRed = true;
10             end
11             drawnRedBalls = drawnRedBalls + 1;
12             R = R - 1;
13         else
14             if i == 1 then
15                 isBlockCreatorRed = false;
16             end
17             B = B - 1;
18         end
19     end
20     epoch.add(red: drawnRedBalls, isOwnedByRed: isBlockCreatorRed)
21 end
22 return epoch;
```

Algorithm 7: Algorithm for generating an array with 32 objects inside each representing a slot and the array represents the epoch. Input consists of 2 parameters: R - the amount of red balls (dishonest validators); B - the amount of black balls (honest validators).

4.3.2 Simulations for 30% stake attackers

Running the simulations for the current implementation of Ethereum gets us the expected results.

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	8998885	89.988	0.00949
2	4372116	43.72	0.0157
3	939388	9.394	0.00923
4	119781	1.198	0.00344
5	33076	0.331	0.00182
6	5805	0.058	0.000762
7	654	0.00654	0.000256
8	187	0.00187	0.000137
9	22	0.00022	0.0000469
10	3	0.00003	0.0000173
11	0	0	0

Table 4.2: Table representing the frequency of epochs containing certain length attacks revealed by the simulations for a 30% stake attack

In **Table 4.2** we can easily see that length 1 attacks are so common, they are basically the "norm" of the ethereum network, and length 2 attacks happen every other epoch. Approximately every hour there is an epoch with a length 3 attack, every day there are almost 3 epochs with a length 4 attack. Epochs with length 5 attacks are a little less common than daily, while epochs having a length 6 attack are a little less common than weekly. Seeing an epoch with a length 7 attack is something rather rare happening approximately once per 10 weeks. About once every 8 months we can observe an epoch with a length 8 attack. While epochs with length 9 and 10 attacks seem to be as rare as every 5-6 and every 40 years respectively, the margin for error there is rather high due to the low number of occurrences in our timespan.

Running the simulations with the applied mitigations also nets results which are close to what we expected.

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	2023538	20.235	0.0127
2	185217	1.852	0.00426
3	8451	0.0845	0.000919
4	109	0.00109	0.000104
5	11	0.00011	0.0000332
6	1	0.00001	0.00001
7	0	0	0

Table 4.3: In this table we see the frequency of epochs containing certain length attacks revealed by the simulations for a 30% stake attack after applying our mitigation

After we apply our mitigation, we get numbers that are quite different from before the mitigation. We make a more detailed comparison below. An epoch containing a length 1 attack is created once every 5 epochs, which is about every half hour. An epoch with a length 2 attack can be observed once every 6 hours. An epoch where a length 3 attack happened at least once is made about every 5 days. A length 4 attack within an epoch is rather rare, happening a little less often than yearly. Length 5 and 6 attacks are very rare and due to the low number of occurrences, they should be considered as having a large margin of error, but even with a lot of subjectivity, they seem to happen once every 11 years and 120 years respectively, while length 7 attacks and longer did not occur at all.

Reorg Length	Percentage of epochs in which the incident occurred (Default)	Percentage of epochs in which the incident occurred (After mitigation)
1	89.988	20.235
2	43.721	1.852
3	9.394	0.0845
4	1.198	0.0011
5	0.331	0.00011
6	0.058	0.00001
7	0.00654	0
8	0.00187	0
9	0.00022	0
10	0.00003	0
11	0	0

Table 4.4: This table shows a comparison between the odds of at least one length n attack happening within an epoch before and after the mitigation is implemented according to the simulations.

From the data in **Table 4.4** you can tell that there is an exponential improvement the longer the length of the attack. For length 1, the improvement of 4.5 times is significant, but the odds of such an attack remain very high at roughly 20%. At length 2 we see a great improvement of nearly 24 times compared against the current state of the network. At length 3 the improvement is 111 times, which is obviously huge - approximately from hourly to every 5 days. From there on the improvements are so great that attackers will probably be discouraged to even attempt it. At length 4 the odds of succeeding with an attack within an epoch are almost 1100 times lower. Something that usually happens about every 8 hours, with our mitigation is reduced to almost a yearly event. At length 5 the improvement is about 3000-fold, but since the occurrence of it after the mitigation is so rare, the margin for error is rather high. From length 6 and beyond the attacks are practically impossible. In the case of length 6, no sane person would obtain 30% of the stake in Ethereum to do something that may or may not happen every 120 years.

4.3.3 Simulations for 20% stake attackers

We will now look into the case where an attacker has 20% of the stake. As in the previous section, we will first look into the frequency of occurrence with the current state of the network. We will then look into the numbers after our fix is applied. This will be followed by a comparison between the two.

We expect that with 20% stake our mitigation will be even more effective. Generally, our expec-

tations are that the lower the stake of an attacker, the exponentially more effective our mitigation is. This is due to the fact that our mitigation is exponentially more effective against longer length attacks, which should affect the probabilities in a similar fashion as the lower stake.

Another reason why we expect the probabilities of success for an attacker to drop is the fact that the lower the stake, the higher the stake of the honest nodes. So in the case of 20%, to carry out a length 1 attack, the attacker needs to be proposer of at least 3 blocks in a row instead of 2. And in the case of 3 blocks, the validators are evenly distributed - 3 private blocks and 1 public block for a total of 0.8 blocks worth of validators against the 0.8 blocks worth of validators from the single public block.

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	1078102	10.781	0.00981
2	7735	0.0773	0.000879
3	62	0.00062	0.0000787
4	0	0	0

Table 4.5: In this table we see the probabilities of an epoch with at least one length n attack occurring (3rd column) as well as how many such epochs there were in total for the simulation run (2nd column). This is without our mitigation.

Right from length 1 we can see that with 20% of the stake, the probability of succeeding with a reorg attack within an epoch is drastically lower, almost 9 times, compared to the same attack with 30% of the stake. This makes it an hourly event rather than the “normal” behaviour of the network.

An epoch containing a length 2 attack is rather uncommon, occurring about once every 6 days. It is obvious that is a huge decrease in frequency compared to the approximately 15-minute occurrences of such epochs.

We had epochs with length 3 attacks around once every 2 years, which is eternity compared against the hourly occurrence with 30% stake.

Finally, out of about 120 years worth of epochs, no length 4 attacks were observed. We can already see that going from 30% to 20% stake has a huge impact on the viability of the attack.

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	9477	0.0948	0.000973
2	0	0	0

Table 4.6: In this table we see the probabilities of an epoch with at least one length n attack occurring (3rd column) as well as how many such epochs there were in total for the simulation run (2nd column). This is after our mitigation is applied.

There is not much to see in **Table 4.6** , an epoch containing length one attacks happens about once every 5 days. Length 2 and higher attacks never occurred over the period of 120 years.

Reorg Length	Percentage of epochs in which the incident occurred (Default)	Percentage of epochs in which the incident occurred (After mitigation)
1	10.781	0.0948
2	0.0773	0
3	0.00062	0
4	0	0

Table 4.7: In this table we show a comparison of the frequency of epochs with certain length attacks before (column 2) and after (column 3) applying the mitigation.

In **Table 4.7** we can easily see that there is over 100 fold improvement with our mitigation in place for length 1 attacks. And after that our solution is so effective that length 2 and higher attacks are practically impossible, as we had 0 occurrences in 120 years of blocks.

4.3.4 Simulations for 15% stake attackers

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	17429	0.1743	0.001319
2	1	0.00001	0.00001
3	0	0	0

Table 4.8: In this table we see the probabilities of an epoch with at least one length n attack occurring (3rd column) as well as how many such epochs there were in total for the simulation run (2nd column). This is without our mitigation.

In **Table 4.8** we can see that epochs with a length 1 attack occur about every 3 days, while length 2 is once every 120 years.

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	1	0.00001	0.00001
2	0	0	0

Table 4.9: In this table we see the probabilities of an epoch with at least one length n attack occurring (3rd column) as well as how many such epochs there were in total for the simulation run (2nd column). This is after our mitigation is applied.

In **Table 4.9** the data from the simulation shows that our mitigation practically eliminates any length of attacks that were possible before its implementation. The single epoch with a successful length 1 attack marks such a rare occurrence that it is not realistic.

Reorg Length	Percentage of epochs in which the incident occurred (Default)	Percentage of epochs in which the incident occurred (After mitigation)
1	0.1743	0.00001
2	0.00001	0
3	0	0

Table 4.10: In this table we show a comparison of the frequency of epochs with certain length attacks before (column 2) and after (column 3) applying the mitigation.

In **Table 4.10** we can easily see that our mitigation is only useful for the length 1 attack, as the length 2 is so rare, it cannot be successfully executed in a practical amount of time.

4.3.5 Simulations for 10% stake attackers

Reorg Length	Number of epochs in which the incident occurred out of 10 000 080 epochs in total	Percentage of epochs in which the incident occurred	Margin of error(for percentage column, 95% confidence)
1	1	0.00001	0.00001
2	0	0	0

Table 4.11: In this table we see the probabilities of an epoch with at least one length n attack occurring (3rd column) as well as how many such epochs there were in total for the simulation run (2nd column). This is without our mitigation.

From the data in **Table 4.11** it is obvious that at 10 % stake, even without our mitigation, the probability of a successful attack is so low, it is negligible - one epoch containing a successful attack in a period of 120 years. Including the high margin for error this becomes 0-2 epochs containing a successful attack every 120 years which is still negligible. For this reason we will not include 'after mitigation' and 'comparison' tables.

An interesting observation that can be made from the simulations is that while our mitigation is exponentially more effective on lower stake percetnages, there's a threshold after which the reorg attack is so improbable that the mitigation is redundant. From the data in the tables above, we can conclude that this threshold is around the 10 percent mark.

4.4 Comparison - Mathematical Analysis vs. Simulation Results

4.4.1 30% Attacker

Reorg Length	Mathematical analysis (%)	Simulation results (%)
1	90.08	89.988
2	43.69	43.72
3	14.23	9.394
4	1.192	1.198
5	0.3278	0.331
6	0.089	0.058
7	6.7886e-3	6.54e-3
8	1.7885e-3	1.87e-3
9	4.6215e-4	2.2e-4

Table 4.12: Comparison for 30% attacker in the default configuration, i.e. before mitigation technique is applied

Comparing the results of the mathematical analysis and the results of the simulation, in **Table 4.12**, we can see the numbers we get are very similar. For length one the difference is only about 0.1 between 90.08 and 89.988, for length 2 it is 0.03 between 43.69 and 43.72 etc.

While this trend continues for most of the lengths, you can easily that there are large differences between our theoretical expectations and the actual results from the simulations. For instance, for length 3, we have a difference of almost 5 and between 14.23 and 9.394 this is a lot. If you take a look at the table, those discrepancies only affect lengths divisible by 3.

There is a very good explanation for that. The length of the attack is the number of blocks where the honest attestations count. And the honest attesters are on average 70%, or 0.7 per block. Thus when the block count is divisible by 3, then 0.7 times the block count is also divisible by 3. And since the dishonest stake is 30% which is always divisible by 3, we will get on average the same number of honest vs dishonest validators.

Let us take the length 3 attack as an example. It requires at least 4 private blocks and 3 public blocks, for a total of 7 blocks. The honest validators get 3 * 0.7 as those are the only public blocks, while the attacker gets 7 * 0.3, as they get their validators from private and public blocks. This results in 2.1 honest vs 2.1 dishonest. In the mathematical analysis, this is a 'win' for the attacker. In the simulations, however, on average 2.1 vs 2.1 means 21000 honest vs 21000 dishonest validators (as one slot has 10000 validators). Due to entropy, sometimes the honest might be e.g. 21005 (vs 20995 dishonest), and similarly, sometimes the dishonest can be e.g. 21002

(vs 20998 honest). Which means the attacker does not always 'win'. Hence, the discrepency versus the mathematical analysis.

Reorg Length	Mathematical analysis (%)	Simulation results (%)
1	20.22	20.235
2	1.85	1.852
3	0.16	0.0845
4	0.0011	0.00109
5	9.1769e-5	1.1e-4
6	7.4746e-6	1.0e-5
7	5.1008e-8	0
8	4.01920-9	0
9	3.1031e-10	0

Table 4.13: Comparison for 30% attacker in in the mitigated configuration, i.e. after mitigation technique is applied

As in the previous comparison, here, in **Table 4.13** ,the probabilities derived from the mathematical analysis and those from the simulations are very close. And in the same way, there is the discrepancy with the numbers divisible by 3, for the same reason - the mathematical analysis favours the attacker in case of equality of the honest and dishonest attesters, while in the simulations, there is always some variation, hence the attacker does not always 'win' the statistical equality.

We going down the table, we can see that for length 7 and beyond, the probability is so low that in the simulated 120 years worth of blocks, we did not get a single occurrence, resulting in an obvious discrepancy between theory and practice.

Overall, the inconsistencies between the mathematical analysis and the simulation are tiny. Exceptions are the cases where the length of the reorg attack is divisible by 3, which is caused by the honest/dishonest validator equality and is not a mistake. This leads us to believe that, beyond reasonable doubt, that the simulation results are correct.

4.4.2 20% Attacker

Reorg Length	Mathematical analysis (%)	Simulation results (%)
1	10.4900	10.7809
2	0.077018	0.07735
3	9.8659e-5	6.2e-4

Table 4.14: Comparison for 20% attacker in the default configuration, i.e. before mitigation technique is applied

The mathematical analysis for 20% attacker is difficult to calculate because using **Formula 4.1** n will will always be an integer, which skews the results in favor for the dishonest validator. To adjust this to more accurate results we calculate the average of optimistic and pessimistic approach which results in the numbers presented in **Table 4.14** and **Table 4.15**. We expect the the actual probability to be slightly over the probability of the mathematical analysis because the average of the two is more inclined towards the pessimistic approach because of the difficulty of obtaining longer chain of blocks.

Reorg Length	Mathematical analysis (%)	Simulation results (%)
1	0.086312	0.094769
2	5.10596e-6	0
3	1.0444e-11	0
4	0	0

Table 4.15: Comparison for 20% attacker in the mitigated configuration, i.e. after mitigation technique is applied

Despite the slight mismatch of the mathematical analysis and simulations we are confident that the numbers from the simulation are accurate and fulfill the expectation that the probability from the mathematical analysis is slightly inaccurate due to the averages we use to calculate for a dishonest validator owning 20% of the total stake.

4.5 Threats against validity

A problem with the simulation could be a bug in the code, which we developed ourselves. But such bug is rather unlikely to be present as the code is simple and the simulation results are aligning closely with the mathematical analysis results. Therefore we can have confidence in the results of the simulation. There is a mismatch between the 30% stake of the dishonest validator for length 3,6 and 9 and, as previously explained in **Section 4.2**, that is because in the mathematical analysis we are biased towards the dishonest validator. But the simulation results are based on the assumption that all validators are online and also the distribution of validators in the simulation uses the whole set of validators for the epoch. Therefore the precision of calculating the length of attack is up to 1 validator. Furthermore, we developed our simulation and mathematical analysis independently and they yield identical values as the ones in [1]

Since we are doing probabilistic simulations there could be a problem with the random number generator that we use in our simulation to determine whether the selected validator is going to be from the malicious or the honest set. This could be a real problem but since we are doing 10 000 selections of validators at each slot, we expect the distribution of validators to be near 30% for the dishonest validator and 70% for the honest validators. Which is the case in the simulation

results, therefore we are convinced that the random number generator is working as intended.

Another threat for the mathematical analysis is that we do not calculate the probability to absolute precision, due to the difficulty of the calculation. Instead we rely on the distribution ratio, which could result in not exact results but still in very close proximity to the actual value.

An assumption we have in the simulation is that the RANDAO mechanism [10] is not influenced by validators as in our simulation we generate epochs independently from one another.

It is important to mentioned that we assume there is zero network latency, which allows all validators to vote on time every time. In the real world, due to latency, there can be cases where honest validators do not vote on time because of this. This can cause successful attacks when otherwise impossible, but this is hard to predict so it is rather risky for the attacker.

5 | Discussion

This chapter aims to evaluate the mitigation technique in relation to the project goals and the previous proposal.

5.1 Empty block proposal | Secondary validator

5.1.1 Resolved problems

The empty block proposal mechanism would make the validators to take one of the following actions:

1. If no block proposal is received in the first 4 seconds then attest to empty block.
2. If the primary block is received, attest to the primary.

Obviously the major flaw is that if there are poor network conditions and validators do not receive the block on time, they would attest to an empty block, therefore no transactions will be processed and the network will be unusable.

In our proposal we do no change the way the current Ethereum works, but we only extend it. In the current setup a validator would take one of the following actions:

1. If no block proposal is received in the first 4 seconds then attest to the head of the chain according the validators view.
2. If the primary block is received, attest to the primary.

We modify this choice to be one of the following:

1. If no blocks proposals are received in the first 4 seconds then attest to the head of the chain according the validators view.
2. If only secondary proposer's block is received, attest to it.
3. If only the primary block or both blocks are received, attest to the primary.

In our extended version even if poor network conditions are in place the network will not stall and network will continue running as expected.

5.1.2 Potential problems

Unfortunately each solution comes with a drawback, in our case such is also related to poor network conditions. In case there is high delay between receiving blocks from the proposers then the network could contain forks where on one the primary block would be selected and on the other the secondary. Of course, this is the mechanism we use to fight against block reorgs attack but it could also become problematic.

Naturally, having an attacker with such high amount of stake is very unlikely. The more likely scenario would be long term DoS on the network, which is also very unlikely but more probable.

If we assume the worst case scenario where the whole propagation of blocks is too slow to fit in the 4 seconds of the block proposing, then we do not degrade the network in any way as in both cases all validators will attest to the last observed on time block. A more problematic case would be a slight delay where both blocks have time to propagate but not through the whole network. This is because then the primary and secondary could collect different amount of attestations and thus creating a fork. This fork would be indistinguishable from the private fork scenario which we are trying to solve. In this case the weight of blocks will be reduced because some validators would have used their vote for the other block. But even in that scenario one of the blocks will be considered valid. The worst outcome could be a hard fork for the network, where both networks would be considered valid, which is already a problem, if there is a big partition of the network, so we are not imposing any new problems.

If such a partition occurs in the network, then the network even without our solution, would lead to a hard fork because blocks would be conflicting. Before applying our mitigation strategy the fork would still conflict because the immediate blocks after the fork would have the same parent therefore they cannot be in the same chain [8], see **Figure 5.1**. Using our mitigation technique such a fork would look more like in **Figure 5.2**, except there could also be missed blocks there as well.

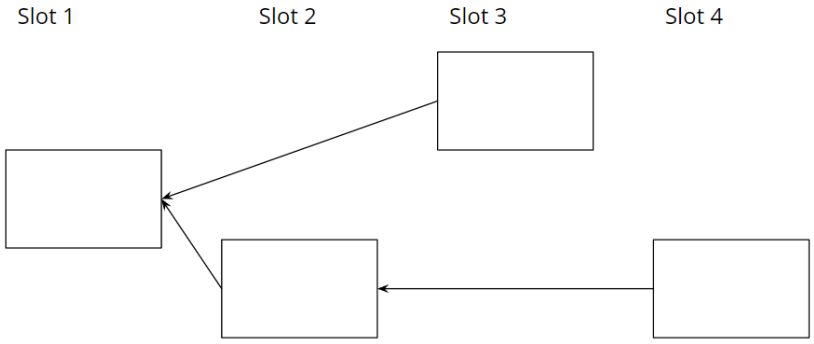

Figure 5.1: Visualization of fork before applying mitigation technique

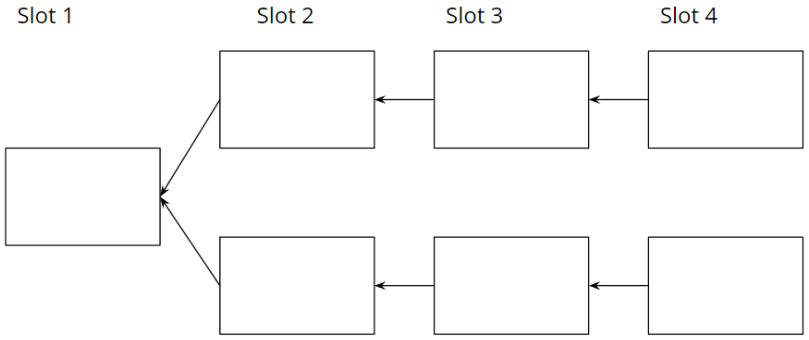

Figure 5.2: Visualization of fork after applying mitigation technique

5.2 Reflection on initial constraints

5.2.1 Overhead

In our solution we do not require more additional resources for the network to be functioning as regular. Our solution only requires one more validator to act as a block proposer instead of acting as an attester. If their hardware would support the regular lifecycle of Ethereum then their hardware supports the node to be a block proposer.

5.2.2 Bandwidth

Our solution increases slightly the bandwidth by requiring one more message to be broadcast to all committee members for that specific slot. Before our solution, if that validator was an

attester then they would only send the message to the respective committee aggregators and the substitute aggregators.

5.2.3 Availability and performance

By using our solution the network will experience the benefit of increased availability, as the number of proposed transactions that are processed will be higher, due to the lack of missed blocks. We have not done simulations on this particular scenario, but it is a logical consequence of having fewer missed blocks. By having more transactions processed, the gas fees will be reduced, which would allow users to use the network more.

5.2.4 Partition tolerance

In relation to partition tolerance we do not have any improvement nor we degrade it. As described in **Subsection 5.1.2** we modify the way the same problem would occur.

5.2.5 Decentralization

Our solution does not directly affect decentralization that much because very few people can spend such an amount of money to be able to carry out this attack. Should someone decide to do so, the centralisation of power will be less severe because we reduce the probability of them being able to carry out the block reorg attack. Indirectly, removing one reason (high probability of success without consequence) why a person could have the incentive to obtain such amount of stake it will help in the direction of decentralization.

5.2.6 Security

With our solution we mitigate the block reorgs attack with the rather insignificant propagation of one more block per slot. By reducing or fixing problems over time the network becomes more secure and less prone to malicious activity.

6 | Conclusion

To conclude, the work documented in this report covers a way to mitigate the Block Reorg attack in the context of Ethereum's Proof-of-Stake. This mitigation was compared against a solution proposed by students at Harvard University. In order to propose this mitigation, a lot of background research had to be done, covering complex topics in the field of blockchain, especially in the Proof-of-Stake consensus algorithm and fork choice rules. Furthermore, to be able to evaluate the effectiveness of our implementation, we had to solve complex probability problems in theory and verify them in the domain of Ethereum via simulations.

Now we can finally answer the question specified in the problem statement - 'How can we mitigate the block reorgs vulnerability without significant additional overhead on the network?'. To answer this question briefly, we can say that this is done by introducing a secondary block proposer to vastly decrease the probability of such private forks being possible, that can be later released to orphan legitimate blocks. We also showed that not only is our mitigation highly effective against the reorg attack, but also the transaction throughput of the network is increased, which leads to cheaper transactions overall.

This task was no small feat. But despite the large number of challenges and setbacks along the way, through high effort, well coordinated teamwork and strong persistence, we managed to achieve a good balance of university and personal life, allowing us to successfully achieve the goal of this project within the target deadline.

Future work

In this project we focused on the mathematical analysis and simulations for our solution to prove that it is efficient. But in order for this solution to reach the Ethereum mainnet we would need to perform additional work.
As a first step, we would need to integrate our solution in the consensus specification project for Ethereum. Afterwards we will need to implement the change for the Beacon Node interface so

that different implementations of the client will support the new messages.
After those two steps are complete then Beacon Nodes will correctly accept messages from secondary block proposers and will attest to the block from the secondary block proposer in case the block from the primary was not observed on time.
Finally, calculations will need to be performed to find out the necessary reward for the secondary block proposer. The calculation will have to account for multiple variables. Such variables are: the necessary funds to award, funds to subtract in case block was not proposed (typically the penalty is equal to the award), are new coins added to the network or the total amount of rewards per block stays the same (if new coins are issued, this could destabilize the network).

Bibliography

[1] Rithvik Rao David C. Parkes Michael Neuder, Daniel J. Moroz. Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders, 2020. URL https://arxiv.org/abs/2102.02247.

[2] Thor Kamphefner Khiem Pham Zhi Qiao Danny Ryan Juhyeok Sin Ying Wang Yan X Zhang Vitalik Buterin, Diego Hernandez. Combining ghost and casper, 2020. URL https://arxiv.org/abs/2003.03052.

[3] Vitalik Buterin. Sharding, 2018. URL http://web.archive.org/web/20190504131341/https://vitalik.ca/files/Ithaca201807_Sharding.pdf.

[4] Ethereum Foundation. Proof of stake, . URL https://eth.wiki/en/faqs/problems.

[5] Ethereum Foundation. Ethereum consensus specification, . URL https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md.

[6] Barnabé Monnot Aditya Asgaonkar Ertem Nusret Tas Caspar Schwarz-Schilling1, Joachim Neu and David Tse. Three attacks on proof-of-stake ethereum. URL https://eprint.iacr.org/2021/1413.pdf.

[7] Rithvik Rao David C. Parkes Aditya Asgaonkar Michael Neuder, Daniel J. Moroz. Three attacks on proof-of-stake ethereum, 2020. URL https://github.com/ethereum/consensus-specs/pull/2197#issuecomment-790585637.

[8] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget, 2019. URL https://arxiv.org/abs/1710.09437v4.

[9] Vitalik Buterin. Ethereum staking, 2022. URL https://ethereum.org/en/staking/.

[10] Vitalik Buterin. Random sampling, 2022. URL https://notes.ethereum.org/@vbuterin/serenity_design_rationale?type=view#Random-sampling.