ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

An Analysis of the Ethereum Proof of Stake Protocol

Relatore:
Chiar.mo Prof.
Cosimo Laneve

Presentata da:
Sergio Solmonte

Correlatore:
Chiar.ma Dott.ssa
Adele Veschetti

Sessione III
A.A. 2022/2023

Dedicata a Papà.

Sommario

La tesi indaga il protocollo *Gasper* della blockchain *Ethereum*, utilizzando una versione estesa del model checker *Prism* per quanto riguarda la simulazione e la verifica. Gasper rappresenta un significativo passo avanti nell'evoluzione di *Ethereum*, introducendo un nuovo meccanismo di consenso e affrontando le sfide della scalabilità. Questo lavoro mira a modellare e a valutare il protocollo Gasper in modo sperimentale, verificandone la coerenza e la robustezza, alla ricerca di vulnerabilità rispetto a vari attacchi.

Lo studio comprende un'analisi completa della transizione di *Ethereum* dalla Proof of Work (PoW) alla Proof of Stake (PoS), chiarendo le motivazioni e i vantaggi di questo cambiamento fondamentale. Inoltre, approfondisce le caratteristiche uniche di Gasper e i miglioramenti rispetto ai suoi predecessori, come Hybrid Casper, evidenziando il suo ruolo nel raggiungimento di maggiore sicurezza e scalabilità.

Uno degli obiettivi principali della ricerca è la validazione del modello realizzato tramite *Prism+* rispetto ai complessi requisiti di Gasper, garantendo che il modello simulato rifletta accuratamente il comportamento del protocollo. Attraverso test e verifiche, questo studio mira a fornire preziose informazioni sulla sua robustezza e affidabilità.

Inoltre, la tesi esplora potenziali attacchi al protocollo *Gasper* in modo teorico e pratico, valutandone la resilienza rispetto alle strategie avversarie. Identificando vulnerabilità e punti deboli, la ricerca contribuisce a migliorare la sicurezza del protocollo e a guidarne l'ulteriore sviluppo.

Questa tesi offre un esame del protocollo Gasper di Ethereum, combi-

nando l'analisi teorica con la simulazione pratica utilizzando il model checker *Prism+*. Lo studio comprende la transizione da PoW a PoS, i progressi di Gasper, la convalida della coerenza del modello e l'analisi della sicurezza, contribuendo alla continua evoluzione della tecnologia blockchain di Ethereum.

Abstract

The thesis investigates the *Ethereum* blockchain's *Gasper* protocol, employing an extended version of the Prism model checker for simulation and verification. *Gasper* represents a significant leap forward in *Ethereum*'s evolution, introducing a new consensus mechanism and addressing scalability challenges. This research aims to rigorously assess the *Gasper* protocol's adherence to its specifications, while also scrutinizing the model for potential vulnerabilities against various attacks.

The study encompasses a comprehensive analysis of Ethereum's transition from Proof of Work (PoW) to Proof of Stake (PoS), elucidating the motivations and advantages of this fundamental shift. Additionally, it delves into *Gasper*'s unique features and improvements over its predecessors, such as Hybrid Casper, highlighting its role in achieving greater security and scalability.

One primary focus of the study is the validation of the *Prism+* model against the Gasper protocol's intricate requirements, ensuring that the simulated model accurately reflects the protocol's behavior. Through extensive testing and verification, this study aims to provide valuable insights into the protocol's robustness and reliability.

Furthermore, the thesis explores potential attacks on the Gasper protocol, assessing its resilience against adversarial strategies. By identifying vulnerabilities and weaknesses, the research contributes to enhancing the protocol's security and guiding its further development.

This thesis offers a comprehensive examination of Ethereum's *Gasper*

protocol, combining theoretical analysis with practical simulation using the Prism model checker. The study encompasses the transition from PoW to PoS, Gasper's advancements, model coherence validation, and security analysis, contributing to the ongoing evolution of Ethereum's blockchain technology.

Contents

List of Figures

List of Tables

Introduction

The advent of the blockchain revolution has ushered in a new era characterized by decentralized digital systems, and leading this transformative wave is Ethereum, an innovative and groundbreaking platform. This research embarks on a comprehensive and insightful analysis of the Ethereum Proof of Stake (PoS) protocol, delving into its core principles while illuminating key differentiators that set Ethereum apart from its counterpart, Bitcoin.

At the heart of this exploration lies the Prism model checker, an indispensable tool that empowers us to meticulously dissect and understand the intricacies of the Ethereum PoS protocol. The Prism model checker serves as a guiding light, enabling one to navigate through the protocol's intricate web, uncovering its underlying mechanisms and shedding light on its inner workings.

Ethereum and Bitcoin, as the cornerstone cryptocurrencies, share the fundamental tenets of decentralization and unalterable record-keeping. While both embrace these principles, Ethereum's visionary path extends far beyond Bitcoin's focal point on digital currency. Ethereum emerges as a versatile and dynamic platform, not merely a medium for currency exchange, but a robust ecosystem for the execution of smart contracts and the operation of decentralized applications (DApps) that are poised to revolutionize a multitude of industries.

Originating from Vitalik Buterin's visionary ideation in 2013, Ethereum's inception was driven by a mission to transcend the limitations of Bitcoin. This marked the birth of the Ethereum Virtual Machine (EVM). This Turing-

complete virtual apparatus [3] unleashed the potential for autonomous execution of intricate code, catalyzing the evolution of DApps and expanding the horizons of blockchain technology.

A pivotal juncture in Ethereum's evolutionary journey is the shift from the energy-intensive Proof of Work (PoW) consensus mechanism to the more eco-friendly Proof of Stake (PoS) protocol. PoS, unlike PoW, relies on the stake held by validators, ensuring their vested interests are harmonized with the network's equilibrium and creating a deterrent against malevolent actions.

Our journey of exploration traverses through the very architecture and functionality of Ethereum's blockchain unravels the intricate tapestry of smart contracts, and dissects the nuances of the PoS consensus mechanism. Utilizing the Prism model checker for the execution of experiments and analyses facilitates a more comprehensive exploration of the inherent advantages and challenges associated with this pioneering protocol. The objective is to extract profound insights into the transformative potential of Ethereum's PoS, with a focus on discerning its impact within the broader landscape of the blockchain ecosystem.

The Ethereum Proof of Stake protocol, illuminated by the Prism model checker, stands as a linchpin for establishing a sustainable, scalable, and adaptive blockchain framework. A profound comprehension of its underlying mechanics and far-reaching implications is a quintessential requisite for navigating the swiftly evolving landscape of decentralized technologies, poised to leave an indelible imprint on diverse industries worldwide.

Chapter 1

Ethereum

1.1 Blockchain

A blockchain stands as a distributed and decentralized digital ledger, meticulously recording transactions and data across a network of computers [6]. It operates through a series of interconnected blocks, each encompassing verified transactions and referencing the previous block, forming an unbroken chain. This sequential arrangement ensures the sanctity and immutability of the stored data.

At the core of a blockchain lie its pivotal features: decentralization and security. Unlike conventional centralized databases, a blockchain thrives through the collaborative efforts of globally distributed nodes. These nodes work harmoniously to establish consensus on transaction validity and the blockchain's prevailing state. This consensus, driven by mechanisms like proof-of-work or proof-of-stake, curbs deceitful conduct and thwarts any single entity's dominance.

To uphold its security, a blockchain employs cryptographic techniques. Every block boasts a unique cryptographic hash, an intricate mathematical function dependent on the block's data and the prior block's hash. Any alteration to the block's data would instantly modify its hash, unmasking any tampering endeavor. Moreover, the blockchain's extensive distribution across

numerous nodes renders seizing control and manipulating data a monumental challenge for potential attackers.

1.1.1 Ethereum Blockchain

The Ethereum blockchain emerges as a prominent platform for transcending simple transactions. It evolves into a decentralized realm for crafting smart contracts and decentralized applications (Dapps). Smart contracts, akin to self-executing agreements coded with predefined terms, autonomously trigger upon meeting specific conditions, erasing the need for intermediaries and fostering trustworthiness. Ethereum embraces the proof-of-stake consensus mechanism defined by the Gasper protocol specification [13], evolving via the *Ethereum 2.0* or *Eth2* upgrade. In this framework, validators are randomly designated to propose and verify new blocks. Participation as a validator necessitates staking a certain amount of Ether (ETH) as collateral. This incentivizes ethical conduct among validators and shores up the network against malicious actions.

The Ethereum blockchain occupies a crucial role in the realm of decentralized finance (DeFi). DeFi capitalizes on smart contracts to deliver financial services sans traditional intermediaries such as banks. It facilitates activities encompassing lending, borrowing, and trading of diverse digital assets.

Furthermore, Ethereum lends support to non-fungible tokens (NFTs), exclusive digital assets applied in domains like digital art, gaming, and collectibles. NFTs bestow verified ownership of digital assets, elevating their significance in the burgeoning landscape of digital creativity and ownership.

Persisting in its focus on smart contracts and Dapps, the Ethereum blockchain endures as an indispensable infrastructure nurturing innovation and facilitating novel decentralized applications spanning varied industries. The Eth2 upgrade endeavors to augment scalability, security, and energy efficiency, thereby amplifying its prowess as a robust blockchain platform.

Ethereum History

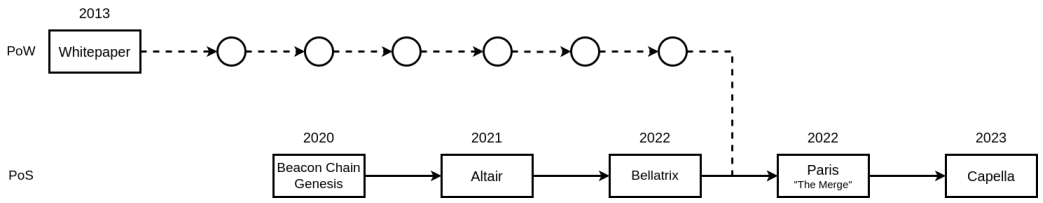

Figure 1.1: History of Ethereum PoS Upgrade.

Figure 1.1 shows all the updates made to Ethereum's Proof of Stake protocol, starting from the publication of *V.Butalik*'s white paper in 2013 [2] which envisaged a blockchain based on Proof of Work protocol.

The first update is called **Beacon Chain Genesis** or Phase 0 and takes place in 2020. With this update, the deposit smart contract was introduced which required at least 16384 deposits of 32 ETH to start. This happened on November 27, meaning the Beacon Chain started producing blocks on December 1, 2020.

The **Altair** upgrade was a key step for the Beacon Chain, being its first planned improvement. It introduced support for *sync committees*, making it easier for light clients to participate. Additionally, penalties for inactive validators were increased as development moved towards The Merge. Unlike previous upgrades relying on variable block numbers in the proof-of-work chain, Altair was the first major upgrade with a specific rollout time. This precision was possible because the Beacon Chain doesn't use proof-of-work but operates on a time-based system with 32 twelve-second *slots* per epoch. This ensured a predictable upgrade at epoch 74,240 when Altair went live.

The **Bellatrix** upgrade, the Beacon Chain's second planned improvement, was geared towards readying the chain for The Merge. It ensured that validator penalties for inactivity and slashable offenses were fully enforced. Bellatrix also implemented an update to the fork choice rules, laying the groundwork for The Merge and the shift from the final proof-of-work block to the inaugural proof-of-stake block.

The **Paris** upgrade took place on September 15, 2022, at block 15537393, immediately activating The Merge transition in the subsequent block. Notably, Paris marked a significant shift by disabling the proof-of-work mining algorithm and its related consensus logic, replacing it with proof-of-stake as its primary mechanism.

The **Capella** upgrade, the third significant enhancement to the Beacon Chain's consensus layer, introduced the capability for staking withdrawals. With this upgrade, stakers who hadn't initially provided withdrawal credentials with their deposit gained the ability to do so, facilitating withdrawals. Additionally, the upgrade incorporated automatic account sweeping functionality. This feature continuously reviews validator accounts, ensuring prompt processing of any available rewards payments or full withdrawals.

Ether

In the Ethereum blockchain realm, Ether (ETH) emerges as the native cryptocurrency, assuming a central role in facilitating fundamental interactions and functions within the network [7].

Foremost, Ether serves as a medium of exchange, empowering users to remunerate transaction fees when executing actions or engaging with smart contracts on the Ethereum platform. Dubbed as *gas* fees, these transaction fees incentivize network participants by compensating them for the computational resources requisite for processing and validating transactions and smart contracts.

Beyond its transactional role, Ether operates as a unit of account within Ethereum dapps, functioning as a standardized reference for pricing, payments, and the valuation of digital services and assets. Its versatile application as a measure of value streamlines transactions and guarantees seamless interactions across the ecosystem.

Over and above its transactional and accounting functions, Ether plays a pivotal role in fortifying the crypto-economic security of the Ethereum network [8]. Validators, the vanguards responsible for securing and verifying

the blockchain, are rewarded with fresh Ether. These rewards function as potent incentives, motivating network participants to adhere to established consensus rules and consequently safeguard the network's integrity.

Ether also serves as valuable collateral in varied DeFi protocols and lending markets. By staking Ether as collateral, participants can unlock loans and financial services, liberating themselves from the clutches of conventional intermediaries. This decentralized approach nurtures trustless interactions and extends financial inclusivity within the Ethereum ecosystem.

Moreover, Ether wields influence in Ethereum's governance and decision-making mechanisms. Token holders exercise voting rights concerning proposed upgrades and alterations to the network's protocol. The voting influence of each token holder corresponds to their Ether holdings, culminating in a democratic governance structure within the decentralized network.

Ether's multifaceted utility within the Ethereum blockchain underscores its significance as a versatile cryptocurrency. Beyond being a mere medium of exchange or a unit of account, Ether propels network security, collateralization, and democratic governance. Its omnipresence fosters innovation and paves the way for an array of financial and non-financial applications within the vibrant decentralized ecosystem.

Transactions

Within the Ethereum blockchain landscape, transactions constitute well-structured processes through which users set events in motion or engage with the network.

The journey of a transaction begins with its originator devising a transaction request, outlining essential particulars such as the sender's address, the recipient's address, a transaction signature, a unique identifier known as a nonce ($nonce$) to sequence transactions, the sum of Ether (ETH) to be transferred, optional input data, and gas-related parameters. Gas assumes a pivotal role in expediting the transaction's execution. It symbolizes the computational effort required by the network to validate and process the

transaction. The sender stipulates the maximum amount of gas units the transaction can consume using the *gasLimit* parameter, alongside specifying the maximum price per gas unit they're willing to pay as a gratuity to validators, as denoted by the *maxPriorityFeePerGas* parameter. The total fee for each gas unit, determined by the *maxFeePerGas*, results from the amalgamation of the *baseFeePerGas* and *maxPriorityFeePerGas*.

The transaction's gas fee (gas_{fee}) can be computed as follows:

$$gas_{fee} = gasLimit \times maxFeePerGas \tag{1.1}$$

With the transaction request dispatched, it traverses the expanse of the Ethereum network, making its way to participating nodes. Validators, who have committed Ether as collateral to partake in the PoS consensus mechanism, bear the mantle of verifying and validating transactions. These validators are chosen at random to suggest new blocks and verify transactions, thereby upholding the authenticity and legality of each transaction.

Collaborating in harmony, validators reach a consensus on the transaction's validity. Upon consensus, the transaction is incorporated into a new block proposed by a validator. This novel block incorporates a unique cryptographic reference linking it to the preceding block, effectively maintaining the unbroken chain of blocks on the Ethereum blockchain.

Upon integration into a block, the transaction becomes an immutable fixture of the blockchain. Any attempt to manipulate data within the block necessitates alterations to subsequent blocks, an immensely intricate endeavor given the cryptographic nature of the blockchain, culminating in heightened security and resistance to tampering.

Following completion, the transaction achieves confirmation on the Ethereum blockchain. The recipient's account balance is updated to mirror the Ether received, while the sender's account balance is decreased by the transferred sum.

Blocks

The process of ensuring a synchronized and agreed-upon history of transactions within the Ethereum network involves the organization of transactions into distinct blocks [15]. Each of these blocks forms a fundamental unit of the blockchain architecture. Unlike a continuous flow of transactions, these blocks create a structured framework that provides order and security to the network.

At its core, the formation of the blockchain involves the arrangement of blocks, with each block comprising a batch of transactions. To maintain the integrity and continuity of the blockchain, each block contains a hash that is cryptographically derived from the data within the block. More notably, each block also includes a reference to the hash of the previous block in the chain, effectively linking all these blocks together. This interlinking mechanism is the cornerstone of the blockchain's resilience against fraud and unauthorized alterations. If any tampering occurs in a block's history, it would trigger a chain reaction, altering the hashes of all subsequent blocks. This tamper-evident feature ensures that any illicit changes are promptly detected by every participant in the blockchain network.

In the Ethereum network, this process is carried out through meticulous orchestration. Transactions are not immediately added to the blockchain; instead, they are grouped together into blocks. This arrangement allows for a more structured consensus mechanism and the verification of multiple transactions in a coherent manner. Dozens or even hundreds of transactions are bundled together into a single block, creating a more organized and manageable framework for validation and synchronization.

A crucial aspect of this orchestration is the concept of time intervals. While transaction requests might occur with high frequency, Ethereum introduces a time gap between block creations. In Ethereum, blocks are generated approximately every twelve seconds. This deliberate pacing provides network participants ample time to reach a consensus on the contents of each block. Even though transaction requests may flood the network in rapid suc-

cession, the blockchain accommodates them by committing and validating these transactions in well-defined intervals.

To maintain the historical record of transactions, Ethereum ensures that blocks are not only linked together but also meticulously ordered. Each new block incorporates a reference to its parent block, reinforcing the chronological sequence of transactions. This orderly arrangement extends to the transactions within each block, contributing to the overall coherence and structure of the blockchain.

The process of assembling these blocks is overseen by a selected validator within the network. Once a validator compiles a block, it is broadcasted to the entire network. Every participating node incorporates this newly generated block into its own copy of the blockchain, thereby maintaining a synchronized history of transactions. Subsequently, a new validator is chosen to construct the next block, and the cycle continues.

This entire block assembly process, as well as the consensus mechanism, is defined by Ethereum's *proof-of-stake* protocol. In this protocol, validators are required to stake a certain amount of Ethereum (32 ETH) as collateral against any potential malicious behavior. This financial commitment incentivizes honest participation and discourages fraudulent activities.

The protocol also introduces a slot-based system, where each slot represents a fixed time interval of twelve seconds. In each slot, a validator is randomly selected to propose a block. This proposed block includes a collection of transactions, and it serves as a new point in the blockchain. Other validators, upon learning about this new block, re-execute the transactions to ensure agreement with the proposed changes.

In the event of conflicting blocks proposed within the same slot, validators employ a fork-choice algorithm to determine the block supported by the most staked ETH. This mechanism ensures a consistent and unified blockchain history, even in the presence of temporary discrepancies.

Every block comprises a rich set of fields that encapsulate crucial information. These fields include data like slot number, proposer's identifier, parent

block's hash, state root, and more. This structured arrangement facilitates the orderly organization of information and the maintenance of a coherent blockchain.

In summary, Ethereum's blockchain operates through the meticulous organization of transactions into blocks, with each block building upon the preceding one. The architecture ensures synchronization, consensus, and order within the network. This disciplined approach, governed by the proof-of-stake protocol, safeguards the integrity of the blockchain's history and paves the way for a decentralized and secure digital ecosystem.

Smart Contracts

In the Ethereum blockchain, smart contracts represent self-executing digital agreements with predefined conditions written in code. They allow for the creation of decentralized applications (dapps) that can operate autonomously without the need for intermediaries [10].

Smart contracts are programmed using Solidity [11], Ethereum's native programming language for writing smart contracts. They consist of functions, data variables, and logic that define the contract's behavior and interactions with the Ethereum network. The functions within a smart contract are similar to methods in traditional programming languages. They perform specific tasks, modify the contract's state, and can be called by external entities or other smart contracts. Data variables store information within the contract, preserving the state of the contract and providing a persistent storage solution on the blockchain. The logic embedded in a smart contract allows it to enforce specific rules, conditions, and validations, enabling the contract to act autonomously and securely.

To use a smart contract, a user needs to deploy it to the Ethereum blockchain. This process involves submitting the smart contract code as a transaction to a specific address on the network. Once deployed, the contract becomes part of the blockchain and is publicly accessible to all participants. Smart contracts execute automatically when triggered by an external entity

or another contract. These triggers are often initiated by sending a transaction to the contract's address, which calls a specific function within the contract. When a smart contract is executed, the logic inside the contract is processed by the Ethereum Virtual Machine (EVM). The EVM ensures that the contract's code is executed consistently across all nodes in the network, achieving consensus on the contract's state changes.

One of the key features of smart contracts is their decentralization [12]. They operate on a distributed network of nodes, ensuring transparency and security without reliance on a central authority. Once deployed, smart contracts are immutable, meaning their code cannot be altered or modified. This immutability guarantees that the contract's rules and behavior cannot be changed, providing a reliable and tamper-resistant mechanism for executing agreements.

Smart contracts have a wide range of applications in various industries. They enable decentralized financial services (DeFi), including lending, borrowing, and decentralized exchanges. They can be used to implement voting systems, supply chain management, identity verification, and more.

1.2 Consensus Mechanism

A consensus mechanism in the context of blockchain refers to the process by which a distributed network of nodes reaches an agreement on the state of the blockchain. It ensures that all participants in the network collectively validate and confirm the correctness of transactions and the order in which they are added to the blockchain. The primary goal of a consensus mechanism is to maintain the integrity and security of the blockchain, preventing double-spending and malicious attacks.

1.2.1 Proof-of-Work (PoW) Consensus

Proof-of-Work is one of the earliest and most widely adopted consensus mechanisms used in blockchain systems, including Bitcoin. In PoW, network

participants, known as miners, compete to solve complex mathematical puzzles. The first miner to find a valid solution for the puzzle is allowed to propose a new block of transactions and add it to the blockchain. This process is resource-intensive and requires significant computational power, making it costly and time-consuming to produce new blocks. As a result, PoW is highly secure and resistant to malicious attacks. However, it consumes a substantial amount of energy, raising concerns about its environmental impact.

1.2.2 Proof-of-Stake (PoS) Consensus

Proof-of-Stake is an alternative consensus mechanism that addresses the energy consumption issues of PoW. In PoS, validators are chosen to create new blocks based on the number of coins they hold and are willing to "stake" as collateral. The more coins a validator is willing to stake, the higher the chances of being selected to propose a block. Validators are economically incentivized to act honestly because they risk losing their staked coins in the event of malicious behavior. PoS is generally more energy-efficient than PoW and promotes a more sustainable blockchain ecosystem.

1.2.3 The Merge

The Ethereum Merge constitutes a significant and momentous upgrade to the Ethereum blockchain, signifying a transition from the traditional proof-of-work (PoW) consensus mechanism to the more contemporary proof-of-stake (PoS) consensus mechanism [25]. This pivotal transition reached its completion on the noteworthy date of September 2022 via the *Paris* update, marking a significant milestone in the evolution of the Ethereum network. The Ethereum Merge is poised to introduce a multitude of advantageous enhancements to the Ethereum blockchain, encompassing:

- Augmented Scalability: PoS networks inherently offer superior scalability compared to their PoW counterparts. This advanced consensus

mechanism facilitates the processing of a higher number of transactions per second, laying the foundation for Ethereum's increased efficiency.

- Elevated Security: PoS networks are renowned for their heightened security when contrasted with PoW networks. They are notably less susceptible to the perils of 51% attacks, thereby bolstering the overall security posture of Ethereum.

- Mitigated Energy Consumption [23]: A salient virtue of PoS networks is their markedly diminished energy consumption in comparison to PoW networks. This reduction in energy utilization is a pivotal stride toward the environmental sustainability of Ethereum.

The realization of The Merge occurred through a structured implementation process, divided into two distinct phases: The Beacon Chain and the ultimate Merge. Commencing with The Beacon Chain, this PoS blockchain was launched in December 2020 via the *Phase 0* update, designed to coexist alongside the pre-existing PoW chain and eventually supplant it. The culmination of this momentous transition transpired during The Merge, which entailed the cessation of the PoW chain and the seamless amalgamation of its functionalities into The Beacon Chain. Validators operating within The Beacon Chain assumed the vital role of securing and validating transactions, effectively superseding the traditional miners.
Moreover, this transition signifies the consummate realization of the Ethereum Merge, commonly referred to as Eth2.0. This momentous transformation is the product of extensive research and development endeavors spanning several years, serving as a testament to Ethereum's commitment to innovation.

The Ethereum Merge stands as an epochal juncture in the trajectory of the blockchain ecosystem. Its transition to the more efficient and sustainable PoS consensus mechanism not only addresses prior limitations but also reaffirms Ethereum's mission to facilitate decentralized applications and smart contracts on a global scale. Beyond its immediate benefits to Ethereum

users, this transition serves as a paradigm for the broader blockchain community, emphasizing the pivotal significance of environmental responsibility and scalability within the dynamic landscape of blockchain technology.

1.3 Hybrid Casper Protocol

The Hybrid Casper Protocol [1] is a consensus mechanism designed to combine the strengths of Proof-of-Work (PoW) and Proof-of-Stake (PoS) to achieve consensus in a blockchain network. It was initially proposed as an upgrade to Ethereum's PoW-based consensus to address its energy consumption issues and improve scalability.

The main idea behind the Hybrid Casper Protocol is to use PoS as the primary consensus mechanism while maintaining a minimal PoW component for security and decentralization. In this hybrid approach, PoW is used to establish checkpoints or finality on the chain, ensuring that blocks are securely committed to the blockchain. Validators in the PoS consensus observe these checkpoints, reducing the need for full PoW validation.

Validators in the Hybrid Casper Protocol are responsible for proposing and validating blocks, and they are required to lock up a certain amount of cryptocurrency (e.g., Ether) as collateral. Validators can be penalized for dishonest behavior by losing their staked coins. This PoS component enhances efficiency and reduces energy consumption compared to traditional PoW.

1.4 Gasper Protocol

The Gasper Protocol [13] is an advanced version of the Ethereum PoS consensus mechanism, introduced in Ethereum 2.0. It aims to further enhance scalability, reduce energy consumption, and improve the security and finality of transactions. In this chapter, the protocol will be introduced in a theoretical manner, encompassing an examination of its core operations, with

a particular emphasis on the practical aspects of block proposal operations. For a more comprehensive and precise understanding of the protocol's architecture, readers are directed to the subsequent chapter, specifically within Section 3.1.

1.4.1 Components of the Gasper Protocol

1. **Beacon Chain**: The Beacon Chain is the PoS-based blockchain that drives the Ethereum network. It acts as the central coordination mechanism and is responsible for selecting validators from the network, organizing them into committees, and proposing blocks. The Beacon Chain also manages *Casper* the Friendly Finality Gadget (FFG) component to ensure the security and finality of transactions.

2. **Validator Committees**: Validators in Ethereum are grouped into committees, which are responsible for voting and finalizing blocks in the GASPER protocol. They are selected using a random process, with each validator having a chance of being selected proportional to its stake considering a maximum value of 32 ETH. Validators with a stake higher than 32ETH will not have greater chances than validators with a stake equal to 32ETH. The committee selection process takes place in two phases: candidate selection, in which all validators are randomly selected to become candidates, and member selection, in which the candidates are selected to become members of the committee in such a way as to represent a uniform distribution of validators in terms of stake.

   Committees are responsible for the following tasks: voting for the proposed block, voting for the last checkpoint, and voting for the last justified block. These three different votes made up an *attestation*.

   The committee selection process is designed to be secure and efficient. It is secure because it ensures that all validators have a chance of being selected for a committee, and it is efficient because it requires only a

relatively small number of validators for each committee. Each slot can have a maximum of 64 committees, where each committee must contain at least 128 validators.

3. **Casper the Friendly Finality Gadget (FFG)**: Casper FFG is a hybrid consensus mechanism that combines PoS with a finality mechanism. It allows the Ethereum network to achieve faster transaction finality, meaning once a block is added to the blockchain, it is considered irreversible. This feature enhances security and prevents the possibility of chain reorganizations.

4. **LMD-Ghost**: The LMD-Ghost algorithm is used as a fork-choice rule, compared to the more classic longest chain used in past protocols. This protocol consists of two parts: Latest Message-Driven and Greedy Heaviest Observed SubTree. This algorithm is based precisely on the number of votes a block receives, instead of considering the longest chain and therefore the one containing a greater number of blocks. More details in Section 1.4.6.

5. **Epochs**: In the Ethereum Proof-of-Stake (PoS) consensus mechanism, epochs play a crucial role in organizing the validator set and facilitating the block proposal and validation process. An epoch is a fixed-duration time interval of 384 seconds during which validators take turns participating in block creation and validation. Each epoch consists of 32 slots of 12 seconds. These epochs help achieve a more efficient and decentralized consensus process. During the e epoch, the process of pseudo-random selection of the validators who will be elected as proposers of the $e + 2$ epoch and the composition of the various committees for the $e + 2$ epoch begins. The committee and proposers' selection process is designed to be randomized and fair, ensuring that no single group of validators dominates the consensus process. The main functions of epochs in the Ethereum PoS consensus are as follows:

- **Validator Rotation**: With each epoch, the validator set is rotated, meaning different sets of validators are chosen to participate in the consensus process. This rotation helps prevent centralization of power and encourages wider participation among network validators.
- **Committee Formation**: Within each epoch, committees are formed to handle specific tasks related to block validation and proposal. These committees are created in a decentralized manner to distribute the workload and enhance security.
- **Block Proposal**: The validators previously elected as proposers have the task of creating blocks containing a set of transactions. Each proposer is designated for a specific slot, so in an epoch there will be a maximum of 32 proposers.
- **Block Validation**: The other validators in the committee then validate the proposed block to ensure that it adheres to the consensus rules and includes valid transactions. Validators actively participate in this process to maintain the integrity of the blockchain.
- **Finality and Consensus**: Once the committee successfully agrees on a block proposal, the consensus algorithm finalizes the block, making it irreversible. This finality ensures that the block is confirmed as part of the canonical blockchain and cannot be changed, enhancing security and reducing the risk of chain reorganizations.

The use of epochs in Ethereum PoS provides several benefits, including more efficient block validation and proposal, increased decentralization, and better resistance against certain attacks. It allows the network to achieve a balance between security, scalability, and decentralization.

6. **Sync Committee**: It consists of 512 validators chosen randomly and refreshed approximately every 27 hours, tasked with signing valid block headers. These committees enable clients to monitor the blockchain

head without requiring access to the complete validator set. The use of a sync committee is introduced in the Altair update.

1.4.2 Block Proposal

Blocks stand as the foundational units within the blockchain framework, embodying discrete parcels of information that traverse between network nodes, achieving consensus and integration into each node's database. The elucidation presented in this section delves into the intricacies of the block proposal process, elucidating the pivotal actors, mechanisms, and stages involved.

The block proposal process intricately orchestrates the introduction of new blocks to the Ethereum blockchain. This process involves randomized selection, careful crafting of checkpoints, rigorous verification, and subsequent integration into the chain. These systematic actions, bolstered by incentives, coalesce to sustain Ethereum's dynamic and secure decentralized ecosystem.

Block Proposer Identification

The critical role of proposing blocks is fulfilled by validator accounts. Validator accounts are under the purview of node operators, who execute validator software within their execution and consensus clients. Validators are required to deposit a minimum of 32 ETH into the deposit contract, signifying their commitment to the network's integrity. It's noteworthy, however, that not every validator is tasked with proposing blocks at all times.

Ethereum's temporal measurement is structured around slots and epochs. Each slot encompasses twelve seconds, and an epoch consists of 32 slots, equivalent to approximately 6.4 minutes. Every slot presents an opportunity for a new block to be introduced to the Ethereum blockchain.

Randomized Selection Mechanism

The selection of a block proposer occurs via a pseudo-random process, which aims to maintain unpredictability while adhering to the principles of consensus. True randomness isn't employed due to its potential conflict with consensus. Instead, Ethereum employs an algorithm called RANDAO. This algorithm incorporates a hash from the prospective block proposer and a continually updated seed, thus achieving the desired element of randomness.

The Ethereum network employs this mechanism to select a specific validator from the entire validator set. It's pertinent to note that validator selection is established two epochs ahead, mitigating certain types of manipulations related to the seed.

The probability of selection isn't uniformly distributed among validators. It is proportionally determined by the effective ETH balance held by each validator. An upper limit of 32 ETH for effective balance ensures uniformity of selection.

The randomness of the proposer selection process is guaranteed by a value associated with each epoch called randa_value or epoch_seed. This value collects the entropy generated in the previous epoch by combining through XOR all the randao_reveal values combined with the signature of the block proposer contained in the various blocks proposed in the previous epoch

Block Creation Process

The designated block proposer is entrusted with broadcasting a digitally signed beacon block. This block is crafted based on the proposer's perception of the most recent chain head, as dictated by their locally-run fork choice algorithm. The fork choice algorithm integrates attestations from the preceding slot and determines the parent block for the newly proposed one.

The process of block creation requires the block proposer to gather data from its local database and its view of the chain. Key data components include:

- **randao_reveal**: A verifiable random value generated by the block proposer, blending its own entropy with the cumulative RANDAO value.
- **eth1_data**: A vote for the deposit contract's status, encompassing the deposit Merkle trie's root and the aggregate count of deposits.
- **graffiti**: An optional field for including a message in the block.
- **proposer_slashings** and **attester_slashings**: Fields containing evidence of slashable offenses committed by certain validators, as perceived by the proposer.
- **deposits** and **voluntary_exits**: Lists of new validator deposits and exit requests known to the proposer through the consensus layer gossip network.
- **sync_aggregate**: A vector indicating validators assigned to a sync committee and participating in data signing.
- The **execution_payload** facilitates the passage of transaction-related information between execution and consensus clients. This block of execution data is nested within the beacon block, reflecting the Ethereum yellow paper's block structure. The **execution_payload** encompasses transactions, and the execution client's execution of these transactions generates an updated state trie, present in the **state-root** field.

These constituent elements are amalgamated into a beacon block, which is then signed by the block proposer and disseminated to its peers. Subsequently, the peers propagate the block, fostering network-wide awareness.

Block Verification and Integration

Upon receiving the proposed block, validators undertake comprehensive verification. This verification entails scrutinizing various aspects, such as the block's parent, corresponding slot, proposer index, the validity of the RANDAO reveal, and absence of slashing instances. The execution payload

is extracted, and the validator's execution client re-executes the transactions within, validating the proposed state alteration. Should the block pass these checks, validators integrate it into their individual canonical chains.

This process of block proposal, verification, and integration perpetuates in successive slots, bolstering Ethereum's continuous and secure operation.

Block Proposer's Reward

The block proposer is duly rewarded for its pivotal contribution. A base reward, contingent on the number of active validators and their effective balances, serves as the foundation. Additional rewards are linked to the inclusion of valid attestations within the block, exemplifying the proposer's collaborative role. The higher the number of validators attesting to the block, the greater the resultant reward. Furthermore, the proposer receives a reward for identifying validators that merit slashing, denoting a commitment to network integrity.

1.4.3 Finality

Finality constitutes an essential attribute inherent to specific blocks within a blockchain, signifying their immutable nature unless subjected to an extraordinary consensus breakdown, accompanied by an attacker's destruction of no less than one-third of the total staked ether. Such finalized blocks encapsulate information that the blockchain unequivocally endorses [19]. The process of achieving finality for a block follows a methodical two-step upgrade path:

First, the *justification* phase necessitates the approval of two-thirds of the total staked ether, securing the block's inclusion in the canonical chain. While this *justified* status enhances resilience against reversion, it remains susceptible to specific exceptional circumstances.

Subsequently, when a subsequent block attains justification upon an already justified block, it attains *finalized* status. This entails an irrevocable

commitment to incorporate the block into the canonical chain. This commitment remains steadfast, save for a rare scenario where an attacker eradicates a substantial amount of ether.

Notably, these block-level advancements are exclusive to epoch-boundary blocks, referred to as *checkpoints*, and are subject solely to these designated blocks. This upgrade process hinges upon pairs of checkpoints and requires the existence of a *supermajority link* connecting them, indicated by two-thirds of staked ether affirming checkpoint B as the legitimate successor of checkpoint A.

This two-thirds consensus requirement for finality significantly hampers malicious attempts to establish an alternative finalized chain, necessitating ownership or manipulation of at least two-thirds of the aggregated staked ether and the infliction of damage exceeding one-third of the total staked ether.

The algorithm overseeing block justification and finalization within Gasper stems from an adapted version of the Casper the Friendly Finality Gadget (Casper-FFG) algorithm [16].

1.4.4 Incentives and Slashing

Validators receive rewards for valid block proposals and validations, augmenting their stake with ether. Conversely, absent or unresponsive validators forego these rewards and may incur minor stake reductions. Yet, specific actions by validators indicative of potential malfeasance trigger severe penalties known as *slashing*, culminating in stake destruction and ejection from the validator network. This process spans 36 days, commencing with an initial penalty of up to 1 ETH on Day 1. More information about rewards and penalties is in Section 3.1.5.

1.4.5 Liveness

Gasper also provides *liveness*, ensuring finalization as long as two-thirds of staked ether remains committed to honest participation, regardless of adversarial activity or other challenges. An *inactivity leak* acts as an additional safeguard, triggered when the chain fails to finalize across four successive epochs. Validators not actively participating experience gradual stake depletion until the majority stake regains two-thirds dominance.

1.4.6 Fork rule

The original fork choice algorithm by Casper-FFG [16], dictating adherence to the chain containing the justified checkpoint with the highest height, has evolved into the more intricate LMD-GHOST algorithm [13]. A fork choice rule becomes pivotal in cases of network asynchrony or equivocation by dishonest block proposers. LMD-GHOST stands for *latest message-driven greedy heaviest observed sub-tree*, selecting the fork with the most substantial weight of attestations as the canonical one. This choice considers only the latest message in cases of multiple messages from a validator. Validators assess each block according to this algorithm before incorporating the weightiest block into their canonical chain.

When encountering forked chains, LMD-GHOST undertakes a meticulous analysis to converge the network towards a unified state. It identifies an *observed sub-tree*, a subset of blocks referenced by validators' attestations, and assesses each fork's cumulative validator weight and its support in the sub-tree. This assessment ensures that the selected chain not only boasts significant stake commitments but also aligns with validators' most current consensus.

Taking into account the illustration presented in Figure 1.2, and employing a standardized weighting scheme for validators, the conventional fork selection algorithm based on chain length alone would have favored the selection of chain B. This preference is attributed to chain B's extended dura-

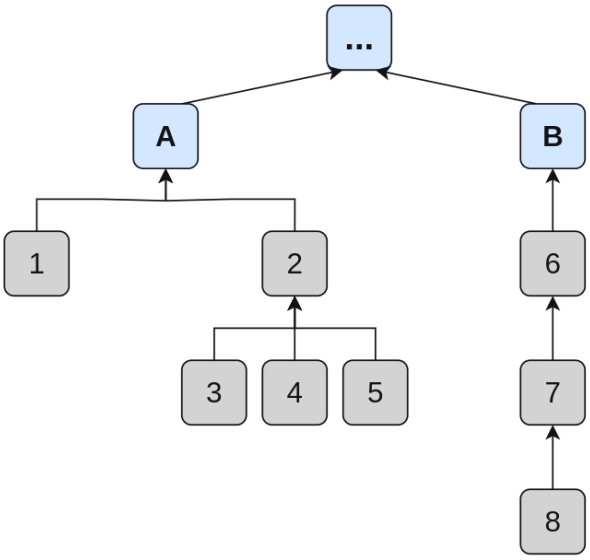

Figure 1.2: LMD-Ghost fork rule example.

tion of 3 blocks, whereas chain A endured for a comparatively shorter span of 2 blocks. Conversely, when employing the LMD-GHOST algorithm, an additional factor must be considered, i.e. the number of votes garnered by each chain. In this context, chain A emerges as the preferred choice, having received 5 votes, whereas chain B secured only 3 votes.

In Listing 1.1 it is possible to see the pseudocode of LMD-GHOST algorithm.

```
function LMD_GHOST(block_tree, attestations):
def find_observed_subtree(attestations, block):
    observed_subtree = {block}
    for attestation in attestations:
        if attestation.references(block):
            observed_subtree.add (attestation.block)
            observed_subtree = find_observed_subtree(attestations,
                attestation.block)
    return observed_subtree

function weight(block):
    return sum([validator.weight for validator in block.validators])

function score(block):
    observed_subtree = find_observed_subtree(attestations, block)
    return weight(block) + sum([weight(attestation.block) for attestation
        in attestations if attestation.block in observed_subtree])

canonical_chain = []

for block in block_tree:
    if block.is_genesis():
        canonical_chain = [block]

for block in block_tree:
    for attestation in attestations:
        if attestation.references(block):
            canonical_chain.append(block)
            break

for block in block_tree:
    if score(block) > score(canonical_chain[-1]):
        canonical_chain.append(block)

return canonical_chain
```

Listing 1.1: LMD-GHOST pseudocode.

1.5 Comparison between Hybrid Casper and Gasper

When comparing the Hybrid Casper Protocol and the Gasper Protocol, it's evident that both aim to achieve consensus in the context of Ethereum's transition to a Proof of Stake (PoS) mechanism. Nevertheless, they exhibit distinctions in their design and operational characteristics. Their respective design choices and operational characteristics lead to distinctions in their energy efficiency, finality, and validator rotation dynamics. These differences are essential considerations when evaluating the suitability of each protocol for Ethereum's future blockchain ecosystem.

1.5.1 Consensus Mechanism

The Hybrid Casper Protocol primarily relies on PoS as its consensus mechanism, supplemented by a minimal PoW component that is used to establish checkpoints or confirm finality within the blockchain. In contrast, the Gasper Protocol is a fully-fledged PoS consensus mechanism, with no reliance on PoW. Gasper leverages both Casper FFG and Casper CBC components to ensure the finality of transactions.

1.5.2 Energy Efficiency

In terms of energy efficiency, both protocols represent significant improvements over traditional PoW-based consensus mechanisms. The Hybrid Casper Protocol derives its energy efficiency from the integration of PoS, effectively reducing the energy consumption associated with mining. Gasper, on the other hand, excels in energy efficiency by exclusively employing PoS, thereby eliminating the energy-intensive PoW mining process.

1.5.3 Finality

Finality, or ensuring that transactions are securely committed to the blockchain, has no code differences between the two protocols, as they both use Casper FFG as the finality gadget. The Hybrid Casper protocol may have longer finalization times due to the inclusion of PoW checkpoints in its consensus mechanism. In contrast, the Gasper protocol achieves faster transaction finality through the implementation of temporal slots for proposing blocks, ensuring a greater number of blocks created with fewer forks.

1.5.4 Validator Rotation

The dynamics of validator rotation also differentiate the two protocols. In the Hybrid Casper Protocol, validator rotation may not be as dynamic when compared to Gasper, primarily because the former retains some PoW mining processes. Gasper introduces efficient validator rotation through the concept of proposers and committees, promoting decentralization and encouraging wider participation among validators.

Chapter 2

Prism Model Checker

The Prism model checker plays a vital role in formal verification, tailored particularly for examining complex probabilistic systems. It serves as a beacon for researchers and engineers, illuminating the path toward confirming critical properties, interpreting performance metrics, and guiding decisions rooted in the foundation of system behavior analysis.

2.1 Key Aspects

Prism's framework is built upon several fundamental tenets:

1. **Flexible Modeling:** Prism introduces a gateway to formalism through its support for Markov chains and Markov decision processes. This versatile framework accommodates both discrete and continuous probabilistic behaviors, empowering modelers to articulate their ideas with elegance.

2. **Mastery of Temporal Logic:** Temporal logic, wielded through Prism, is the language in which properties are expressed. Enriched by probabilistic temporal logic, this sophisticated vernacular enables the articulation of intricate behavioral requirements.

3. **Comprehensive Verification Capabilities:** Prism's arsenal encompasses diverse techniques like model checking, parameter synthesis, and strategy synthesis. It diligently guards properties such as reachability, safety, liveness, and more, upholding the bastion of system integrity.

4. **Unveiling Quantitative Insights:** Prism's magnificence is underscored by its prowess in quantitative analysis. It unveils probabilities of state occurrences, anticipates the temporal horizon for pivotal events, and crafts a myriad of metrics that sculpt the tapestry of system comprehension.

5. **Algorithmic Ingenuity:** Beneath Prism's accomplishments lie formidable algorithms. Symbolic model-checking and probabilistic model-checking algorithms harmonize their efforts, deftly taming the complexities of vast state spaces.

2.2 Constructing Models

Prism facilitates the construction of models through a specialized language. This syntax enables the formulation of states, transitions, probabilistic dynamics, and inquiries about system behavior. Here is an excerpt showcasing a discrete-time Markov chain model within Prism's realm:

```
// Realms of Imagination
const int Possibilities = 3;
module Fantasy
    dream : [0..Possibilities] init 0;

    [] dream=0 -> 0.5: (dream' = 1) + 0.5: (dream' = 2);
    [] dream=1 -> 0.3: (dream' = 0) + 0.7: (dream' = 2);
    [] dream=2 -> 0.6: (dream' = 1) + 0.4: (dream' = 0);

endmodule
```

2.3 Formulating Property Assertions

The art of articulating property assertions within the prism of probabilistic analysis finds its embodiment through the utilization of temporal logic constructs. Much like a virtuoso crafting a harmonious symphony, Prism adeptly composes intricate probabilistic narratives, seamlessly harmonizing the dimensions of probability and time.

To illustrate, contemplate the following assertion: *The probability of ultimately reaching state 2 surpasses the threshold of 0.8.* In the eloquent language of Prism, this conceptual notion metamorphoses into the precise formalization:

$$P \geq 0.8(<> \text{Fantasy.dream} = 2)$$

In this refined expression, Prism captures the essence of the assertion, employing its linguistic elegance to encapsulate the inherent uncertainties and temporal dynamics within a mathematical framework. Such meticulous articulations serve as the cornerstone of accurate and comprehensive analysis, allowing stakeholders to navigate the intricate landscape of probabilities with clarity and precision.

2.4 Applications of Prism

The versatile utility of Prism spans a multitude of disciplines, resonating across various domains with profound impact and significance. This section sheds light on the diverse applications where Prism's capabilities find purposeful expression.

1. **Software Analysis:** Prism assumes the role of a meticulous evaluator, meticulously examining the correctness and dependability of probabilistic algorithms and protocols. Through its rigorous scrutiny, Prism ensures the veracity and robustness of intricate software systems, offering a safeguard against potential errors and vulnerabilities.

2. **Performance Evaluation:** With its sophisticated analytical tools, Prism emerges as a potent instrument for assessing vital metrics related to system efficiency. These metrics encompass pivotal aspects such as response time, throughput, and resource utilization. By wielding its analytical prowess, Prism empowers organizations to fine-tune their systems, enhancing operational efficiency and resource allocation.

3. **Exploration in Biology:** Within the realm of biology, Prism assumes the role of an insightful translator, unraveling the enigmatic behaviors exhibited by biological systems when subjected to the whims of chance. Through probabilistic modeling, Prism aids researchers in comprehending the intricate dynamics of biological phenomena, offering valuable insights into the complex interplay of genetic, molecular, and environmental factors.

4. **Advancements in AI and Robotics:** The influence of Prism extends seamlessly into the domains of artificial intelligence and robotics, playing a pivotal role in ensuring the integrity of probabilistic AI systems and guiding the strategic orchestration of robotic endeavors. By subjecting AI algorithms to rigorous probabilistic analysis, Prism safeguards against unforeseen aberrations, while also contributing to the development of adaptive and reliable robotic strategies.

Chapter 3

Architecture

This experiment encompasses the unveiling of a simulated Ethereum network in operation, with adherence to the Gasper protocol. The formulation and substantiation of the model are executed by means of the Prism model checker, wherein the model is treated as a continuous-time Markov chain. The ensuing portions of this discussion will furnish a broad overview of the Gasper architecture, concurrently offering an exposition of how it is encompassed within the Prism modeling framework. The forthcoming chapter will delve into the procedural details of crafting the model through Prism, including the provision of code snippets and elucidation of each constituent element.

3.1 Gasper architecture

The Ethereum network has embarked on a transformative journey by transitioning from the energy-intensive Proof of Work (PoW) consensus mechanism to the innovative Proof of Stake (PoS) protocol under the Gasper architecture. This architectural shift aims to enhance network efficiency, scalability, and environmental sustainability while maintaining robust security and decentralization.

3.1.1 Overview of Gasper Protocol

The Gasper protocol is the crux of Ethereum's transition to PoS. Named after the famous ghost-catching character, it seeks to address the limitations of PoW and accommodate the intricacies of a PoS mechanism. The protocol is designed to ensure both security and liveness within the network. It achieves this by leveraging a two-layer structure consisting of the Beacon Chain and Shard Chains.

Beacon Chain

The Beacon Chain serves as the backbone of the Ethereum PoS network. It is responsible for managing validators, maintaining consensus, and managing cross-links to Shard Chains. Validators participate in the consensus process by proposing and attesting to blocks on the Beacon Chain. The consensus algorithm, known as the Casper protocol, ensures that validators act honestly by staking their Ether as collateral.

Shard Chains

Shard Chains provide scalability by dividing the network into smaller, interconnected chains. Each Shard Chain processes a subset of transactions and smart contracts, thereby reducing network congestion and increasing throughput. Cross-links from Shard Chains to the Beacon Chain maintain the overall network's coherence.

3.1.2 Validator Participation and Staking

Under the Gasper protocol, validators play a crucial role in securing and validating transactions. Validators are Ethereum addresses that are required to stake a certain amount of Ether as collateral to participate in block validation. This stake serves as a disincentive against malicious behavior, as validators risk losing their staked Ether if they are found to be dishonest or unresponsive.

3.1.3 Committees

A committee constitutes a set of at least 128 validators tasked with validating blocks within each designated time slot, known as a *slot*. Each slot can accommodate a maximum of 64 of these committees. Within each committee, one of the validators takes on the role of aggregator, responsible for assembling the signatures issued by all the other validators in the committee, provided they agree on a certain attestation.

The main role of the committees is to fairly distribute the overall workload of the network, establishing a specific time for each group of validators to present their attestations. This labor division process aims to ensure efficient management of validation activities within the network, allowing each committee to contribute in a synchronized and organized manner to the functioning of the blockchain system.

3.1.4 Attestation

Every 6.4 minutes [17], a validator submits an attestation to the network, specifying a particular slot within the epoch. The purpose of this attestation is to cast a vote in favor of the validator's perspective on the blockchain, particularly concerning the most recently justified block and the initial block of the current epoch, referred to as the source and target checkpoints, respectively. These attestations, collectively contributed by all participating validators, enable the network to establish consensus regarding the state of the blockchain.

An attestation comprises the following key components:

- **aggregation_bits**: A bitlist representing validators, where each position corresponds to the validator index in their committee, indicating whether the validator has signed the data (i.e., whether they are active and in agreement with the block proposer).
- **data**: Detailed information concerning the attestation, as further defined below.

- **signature**: A BLS signature that consolidates the signatures of individual validators.

The initial task for a validator preparing an attestation is to construct the data component. This data includes the following details:

- **slot**: The slot number referenced by the attestation.
- **index**: A numerical identifier denoting the validator's committee membership for a given slot.
- **beacon_block_root**: The root hash of the block observed by the validator at the blockchain's head, which is the result of applying the fork-choice algorithm.
- **source**: A segment of the finality vote indicating the most recently justified block, as perceived by the validators.
- **target**: A segment of the finality vote signifying the first block within the current epoch, according to the validators' viewpoint.

Once the data is assembled, the validator can set the aggregation_bits by flipping the bit corresponding to their validator index from 0 to 1, sign the attestation, and disseminate it across the network.

The transmission of attestation data from each individual validator to the entire network entails substantial overhead. Therefore, these attestations are aggregated within subnets before being distributed more widely. This aggregation encompasses combining signatures, so an attestation broadcast includes consensus data and a single signature formed by merging the signatures of all validators in agreement with the data. This consolidation is verified using aggregation_bits, which provides the index of each validator within their committee, with their ID being specified in the data. In each epoch, one validator within each subnet is designated as the aggregator. The aggregator collects all matching attestations it receives over the gossip

network, recording the sender of each matching attestation in the aggregation_bits. Subsequently, the aggregator broadcasts the attestation aggregate to the broader network.

When a validator is chosen as a block proposer, they compile aggregate attestations from the subnets up to the latest slot for inclusion in the new block.

Attestation Inclusion Lifecycle

The lifecycle of an attestation comprises several stages [17]:

1. **Generation**
2. **Propagation**: a first propagation of the attestation inside the committee.
3. **Aggregation**: the aggregator takes all the valid attestations and combines them into one.
4. **Propagation**: from the aggregator to the entire network.
5. **Inclusion**

These stages collectively define the journey of an attestation within the Ethereum network. Those steps are repeated in every epoch.

3.1.5 Stake, Rewards and Penalties

Individuals who operate nodes and aspire to engage in the validation of blocks and the determination of the blockchain's principal head initiate deposits of ether into a specialized smart contract embedded within the Ethereum network [43]. In reciprocation for their contributions, they receive ether as remuneration for executing validator software. The core function of this software lies in the meticulous evaluation of the legitimacy of newly

acquired blocks through the peer-to-peer network, coupled with the application of the fork-choice algorithm to ascertain the primary block within the chain.

Validators shoulder two paramount responsibilities: primarily, they are entrusted with the task of meticulously examining new blocks and providing their validation (*attesting*) if these blocks adhere to the requisite criteria. Secondarily, when randomly selected from the comprehensive pool of validators, they assume the role of proposing new blocks. Failure to fulfill these obligations within the stipulated timeframe leads to the forfeiture of their entitlement to an ether payout. Additionally, validators may, on occasion, be charged with additional tasks such as signature aggregation and participation in synchronization committees.

In Ethereum's validation ecosystem, incentives are a cornerstone, serving as a mechanism that stimulates and compensates validators for their roles.Validators in Ethereum Gasper receive rewards for specific actions, such as timely votes, block proposals, and active participation in sync committees. The cornerstone of this reward system is the *base_reward* which epitomizes the average compensation a validator can expect under optimal conditions per epoch. The formula for calculating the base_reward is as follows:

$$\text{base\_reward} = \text{effective\_balance} \times \left(\frac{\text{base\_reward\_factor}}{\text{base\_rewards\_per\_epoch} \times \sqrt{\sum(\text{active\_b})}} \right) \tag{3.1}$$

Base Validator's reward Here:

- base_reward_factor stands at 64.
- base_rewards_per_epoch is 4.
- $\sum$(active_b) signifies the cumulative staked ETH across all active validators.

The Formula 3.1 underscores that the base_reward increases with a validator's effective balance while inversely correlating with the number of val-

idators. Consequently, an expanded validator pool elevates the total issuance (following $\sqrt{N}$), while diminishing the base_reward per individual validator (following $1/\sqrt{N}$). These dynamics wield significant influence over the Annual Percentage Rate (APR) for validators.

A validator's total reward derives from a summation of five components [43], each assigned specific weights to gauge its contribution to the total reward. These components comprise:

1. Source vote: Timely voting for the correct source checkpoint.
2. Target vote: Timely voting for the correct target checkpoint.
3. Head vote: Timely voting for the correct head block.
4. Sync committee reward: Active involvement in a sync committee.
5. Proposer reward: Proposing a block within the designated slot.

and the associated weights are:

$$\begin{aligned} \text{TIMELY\_SOURCE\_WEIGHT} &: \text{uint64}(14) \\ \text{TIMELY\_TARGET\_WEIGHT} &: \text{uint64}(26) \\ \text{TIMELY\_HEAD\_WEIGHT} &: \text{uint64}(14) \\ \text{SYNC\_REWARD\_WEIGHT} &: \text{uint64}(2) \\ \text{PROPOSER\_WEIGHT} &: \text{uint64}(8) \end{aligned}$$

The cumulative weights of these components sum to 64, and the reward computation entails the addition of relevant weights divided by 64. An exemplar scenario involves a validator fulfilling all five components, earning a reward equivalent to the base_reward. However, it's important to note that validators typically do not partake as block proposers, hence their maximum reward typically amounts to $\frac{7}{8}$ of the base_reward. Validators uninvolved in block proposal or sync committees can anticipate $\frac{6.75}{8}$ of the base_reward.

In addition to these rewards, there exists an *inclusion_delay_reward* designed to motivate swift attestations. This reward is computed based on the

slot interval between block proposal and attestation, with shorter intervals yielding higher rewards.

Block proposers, on the other hand, reap rewards contingent upon the number of valid attestations they incorporate into their proposed block. Encouragingly, they are prompted to include evidence of misbehavior by other validators, which enhances their rewards.

To incentivize integrity, block proposers who incorporate slashing evidence acquire additional rewards, equivalent to a fraction of the slashed validator's effective balance, divided by 512.

The consequences for failing to cast target and source votes are commensurate with the rewards the attestor would have garnered if they had cast them successfully. In essence, instead of witnessing their balance increase with a reward, an equivalent amount is deducted from their balance. However, penalties differ based on specific actions:

- **Source and Target Vote Penalties**: Validators who miss timely source and target votes incur penalties equivalent to the rewards they would have earned had they voted on time. Essentially, the penalty deducts an equal value from their balance instead of adding a reward.

- **Head Vote**: No penalties exist for missing head votes; they are exclusively rewarded, never penalized.

- **Inclusion Delay Penalty**: There is no direct penalty for inclusion delay; validators not meeting inclusion criteria merely forfeit the associated reward without facing a specific penalty.

- **Block Proposal Penalties**: Validators do not face penalties for failing to propose a block.

These penalty mechanisms encourage validators to act promptly and responsibly within the Ethereum Gasper network. Validators, being integral to network security and performance, are incentivized to adhere rigorously to established rules.

Source	Target	Head	Vote Reward	Penalty	Inclusion Delay	Result
X	X	X	0	3 × b_reward		-3 × b_reward
V	X	X	b_reward × partecipation_rate	2 × b_reward	b_reward × $\frac{7}{8}$	Vote_reward - $\frac{9}{8}$ × b_reward
V	V	X	2 × b_reward × partecipation_rate	b_reward	b_reward × $\frac{7}{8}$	Vote_reward - $\frac{1}{8}$ × b_reward
V	V	V	3 × b_reward × partecipation_rate	0	b_reward × $\frac{7}{8}$	Vote_reward + b_reward × $\frac{7}{8}$

Table 3.1: Rewards and Penalties.

Slashing and Inactivity

The latest actions undertaken by the protocol to ensure good conduct and finality within the network are *slashing* and *inactivity leak*.

Slashing is a stringent measure within the Ethereum Gasper network, resulting in the forceful removal of a validator from the network and the forfeiture of their staked ETH. Validators can be slashed for engaging in dishonest actions related to block proposals or attestations. There are three actions that can lead to slashing:

1. **Double Proposal**: When a validator proposes and signs two different blocks for the same slot.

2. **Surrounding Attestation**: When a validator attests to a block that *surrounds* another block, effectively attempting to alter the blockchain's history.

3. **Double Voting**: When a validator attests to two competing candidates for the same block.

Upon detection of any of these actions, the validator is subjected to slashing. This involves an immediate burn of 1/32 of their staked ETH (with a maximum of 1 ETH). Following the burn, a 36-day removal period begins during which the validator's stake gradually diminishes. At the midpoint of this period (Day 18), an additional penalty is applied. The magnitude of this penalty scales with the total staked ETH of all slashed validators within the 36 days preceding the slashing event. Consequently, if more validators are slashed, the magnitude of the slash increases. The most severe slash could

lead to the forfeiture of the entire effective balance of all slashed validators. Conversely, isolated slashing events result in a relatively minor loss of the validator's stake. This midpoint penalty, contingent on the number of slashed validators, is termed the *correlation penalty*.

Inactivity Leak is an emergency protocol in Ethereum Gasper triggered when the consensus layer goes more than four epochs without finalizing. The primary objective of the inactivity leak is to create conditions conducive to the recovery of finality within the blockchain.

As previously explained, achieving finality necessitates a 2/3 majority of the total staked ETH agreeing on source and target checkpoints. If validators representing more than 1/3 of the total validators either go offline or fail to submit accurate attestations, it becomes impossible for a 2/3 supermajority to finalize checkpoints. In such a scenario, the inactivity leak serves to gradually reduce the stake held by inactive validators until they control less than 1/3 of the total stake. This shift allows the remaining active validators to finalize the chain.

Regardless of the size of the pool of inactive validators, the remaining active validators will ultimately control more than 2/3 of the total stake. The loss of stake serves as a potent incentive for inactive validators to reactivate as swiftly as possible. An instance of the inactivity leak was encountered on the Medalla testnet when fewer than 66% of active validators could reach a consensus on the current blockchain head. The inactivity leak was activated, and finality was reestablished.

The penalty for an inactive validator i is calculated as:

$$p_i = \frac{s_i \times B_i}{INACTIVITY\_SCORE\_BIAS \times INACTIVITY\_QUOTIENT}$$

where the s_i is the validator i inactivity score, B_i is its staked balance, INACTIVITY_SCORE_BIAS is equal to 4 and INACTIVITY_QUOTIENT is equal to 2^{24}.

The inactivity score is updated each epoch following:

- At the end of epoch N, irrespective of the inactivity leak,
  - decreases a validator's score by one when it made a correct and timely target vote in epoch $N-1$, and
  - increases the validator's score by INACTIVITY_SCORE_BIAS (four) otherwise.
- When not in an inactivity leak,
  - decreases every validator's score by INACTIVITY_RECOVERY_RATE (sixteen).

3.1.6 Finality and Liveness

Gasper brings in the idea of *correct-by-construction* to make sure that the PoS network reaches a point of no return, called finality. Finality guarantees that once a block joins the blockchain, it's there for good, making the whole system more secure. The method it uses is called *justification and finalization*, which works through epochs and checkpoints to achieve this unchangeable state.

3.1.7 Benefits of Gasper Architecture

The Ethereum PoS protocol under the Gasper architecture offers several notable benefits:

- **Energy Efficiency:** Unlike PoW, PoS does not require miners to solve complex mathematical puzzles, reducing energy consumption.
- **Scalability:** Shard Chains enable the network to process multiple transactions simultaneously, increasing throughput.
- **Security:** Validators' economic stake serves as collateral, aligning their incentives with network security and reducing the risk of malicious behavior.

- **Decentralization:** PoS encourages wider participation, as the barriers to entry are lower than PoW, enhancing network decentralization.

3.2 Model simulation

Continuous-time Markov Chains (CTMCs) are a fundamental mathematical concept used to model and analyze systems that exhibit probabilistic behavior over time. CTMCs are especially useful in modeling stochastic processes where transitions between states occur continuously and are characterized by rates rather than discrete time steps.

3.2.1 CTMC Basics

A Continuous-Time Markov Chain (CTMC) finds its essence in a collection of states joined by the pathways of transition rates [26]. Each state serves as a snapshot of a system's arrangement, while the transition rates paint a picture of how likely it is to move from one state to another during a specific span of time. This CTMC's actions and dynamics are captured through a series of differential equations, recognized as Kolmogorov's forward equations, which outline its evolution and behaviors.

3.2.2 Prism Model Checker and CTMCs

Prism stands as a highly regarded tool for model checking, playing a pivotal role in simplifying the process of shaping, simulating, and validating probabilistic systems. Its capabilities extend to assisting in the description and examination of Continuous-Time Markov Chains (CTMCs), allowing for a detailed investigation into the traits and actions of systems through both simulation and the rigor of formal verification methods [27].

Model Specification

Within the framework of Prism, a CTMC model unfolds using an approachable language that precisely lays out states, the speeds at which they transform, and the uncertain decisions steering these shifts. Beyond that, this model is equipped with the ability to integrate labels, rewards, and other significant variables, all harmonizing to enable a thorough exploration of its behaviors and characteristics.

Simulation and Analysis

Prism equips users with the ability to simulate CTMC models, allowing them to witness the evolution of a system as time unfolds. This is achieved through the utilization of Monte Carlo simulation techniques, which generate illustrative sample trajectories revealing the transitions between various states. These trajectories provide a wealth of valuable insights into the dynamic character of the system and the diverse spectrum of probable outcomes that can emerge.

3.2.3 Simulating Ethereum PoS Network

To build a simulation of an Ethereum PoS network using Prism and CTMCs, it is possible to craft a model that depicts the diverse states the network can be in, the transitions that occur between these states, and the rates at which these changes happen.

State Representation

Outline distinct states that mirror various setups within the Ethereum PoS network. These states encompass arrangements of validators, conditions of the blockchain, and levels of congestion in the network. For example, they might cover how validators are spread out, what state the blockchain is in, and how busy the network is at different times.

Transition Rates

Explain the speed at which the system transitions between various stages in line with the functioning of the PoS protocol. These rates act as indicators of actions taken by validators, the timing of block proposals, the shifting of epochs, and any noteworthy events occurring within the network.

Property Analysis

Identify the important characteristics one wants to examine, like how many transactions can be processed, how steady the network remains, and how actively validators are engaged. Then, utilize Prism's analysis tools to evaluate the chances of these specific features being met.

3.2.4 Benefits and Limitations

Using CTMCs and Prism to simulate an Ethereum PoS network offers several advantages, including the ability to quantitatively analyze probabilistic behavior, assess network stability, and optimize protocol parameters. However, it's important to acknowledge that CTMC models are simplifications of real-world systems and might not capture all intricacies accurately.

3.2.5 Prism+

Prism+ is a version of the Prism model checker designed to simulate the behavior of a blockchain. This expansion of the model checker allows you to use objects such as blocks and blockchains, implicitly allowing you to resolve issues internal to these objects, such as the presence of forks in blockchains or the automatic updating of the ID regarding blocks created by the same node.

Chapter 4

Implementation

In this chapter, the focus is directed toward the implementation phase of the codebase designed for the Prism model checker. This segment constitutes a pivotal aspect of this work, elucidating the foundational architecture and operational characteristics of the code.

First, the global variables implemented and used in this Ethereum PoS model are shown. Each variable has a fundamental role in the experiments carried out. The use of global variables in Prism allows you to synchronize the execution of different operations by validators, without the obligation to use specific synchronization constructs. Subsequently, the Prism modules created for this implementation are introduced, namely *Validators*, *Network*, *Updater*, *RanDAO*, and *Global*. Each module with its sub-components assumes a crucial role in simulating and scrutinizing the behavior of the system, thereby enhancing the overall robustness and dependability of the implementation.

Furthermore, an examination of the utilization of labels is conducted, as these components play a vital role in simulating specific aspects of the model's behavior, facilitating comprehensive tracking and evaluation. Additionally, scrutiny of global variables employed throughout the code is undertaken, shedding light on their significance and influence on the operational dynamics of the system.

4.1 Global variables

The Listing 4.1 outlines the global variables used in the Prism model checker code for Ethereum Proof of Stake (PoS) simulations.

```
const int EpochSize = 32;
const double rMw=1;
const epochs;
const double T=epochs*EpochSize*(12);
const double rC = 12/(384);
const double rAdd = 1/12.6;
const double r = 1/12.6;
const int K =1;
const int Start .. Fin = 0 .. 8;
const int N = 100;
global ValidatorID : int init -1;
global Voter0 : int init -1;
global Voter1 : int init -1;
global Voter2 : int init -1;
```

Listing 4.1: Global variables initialization.

These variables are fundamental parameters that define the behavior and characteristics of the simulated Ethereum network. Here is a description of each global variable:

- **EpochSize** defines the size of an Ethereum epoch, consistently set at 32 slots.
- **rMw** represents the delay between block creations and propagation, initially configured as 1, equivalent to a 12-second delay for each block creation.
- **epochs** signifies the number of epochs considered, which can be adjusted either dynamically or statically before simulation.
- **T** calculates the overall simulation duration, factoring in the number of epochs, epoch size, and a 12-second block creation delay.

- **rC** indicates the time available to a validator to carry out a vote. Considering that a validator can send his vote in his reference slot, this value is marked as 12/384, i.e. 1/32.
- **rAdd** corresponds to the rate of delay involved in adding a block to the blockchain.
- **r** represents the delay associated with block removal.
- **K** remains a constant with a value of 1, serving as a predefined parameter, notably employed in fork-length simulations.
- All the variables from **Start**=0 and **Fin**=8 are used inside the Validator's modules and are referred to the validator's state.
- The variable **N** represents the value 100, likely indicating a predefined constant related to the simulation. It is used as a maximum value in multiple situations.
- The **ValidatorID** is a global variable used during the RanDAO operations. It refers to the ID of the validator that is chosen as the block proposer.
- The $\mathbf{Voter}_i$ is a global variable used during the RanDAO operations. It refers to the ID of the validators that are selected to vote in a specific slot.

These global variables play a crucial role in defining the parameters and behavior of the Ethereum PoS simulation within the Prism model checker code, allowing for the precise modeling and analysis of Ethereum's PoS consensus mechanism.

4.2 Validator

In this section, only the validator with id=0 is considered, for simplicity in the description, it should be noted that all the validators have the same

structure and the same local variables. So, for Validator1, you will have M1_STATE, b1, B1, c1, and so on.

```
module Validator0

    M0_STATE : [Start, Create, Receive, Move, Vote, Check, Fin]
        init Start;
    b0 : block{genesis,0;genesis,0} ;
    B0 : blockchain [{genesis,0;genesis,0}];
    votes0 : [0..1000] init 0;
    lastFinalized0 : block {genesis,0;genesis,0};
    lastJustified0 : block {genesis,0;genesis,0};
    lastCheck0 : block {genesis,0;genesis,0};
    finalized0 : bool init false;
    justified0 : bool init false;

    [] (M0_STATE=Start)&(validatorID=0) ->
            1 : (b0'=createBlock()) &
                (M0_STATE'=Create) &
        (validatorID'=-1);

     [] (M0_STATE=Start)&(voter_i=0) ->
            1 :       (M0_STATE'=Vote);

    [] (M0_STATE=Start)&!ListIsEmpty()) ->
        rC : (M0_STATE'=Check);

    [] (M0_STATE=Start) ->
            1 : (M0_STATE=Receive);

    [addBlock0] (M0_STATE=Create) ->
        rMw: (B0'=addBlock();

    [voteBlock0] (M0_STATE=Vote) ->
            1 : (getBlockHeigth()) &
                (addBlockSet(b0));

    [] (M0_STATE=Receive) & !isEmpty(Set_0) ->
```

```
                1 : (b_0=receive(Set_0)) &
                    (M0_STATE=Move);

[] (M0_STATE=Receive) & isEmpty(Set_0) ->
                1 : (M0_STATE=Start);

[] (M0_STATE=Check)&(isCoherent(Stakes, Heigth)) ->
          rIst : (numFinBlocks0'=numFinBlocks0 + 1) &
                 (lastFinalized0'=lastJustified0) &
                 (lastJustified0'=lastCheck0) &
                 (B0'=updateChain()) &
                 (M0_STATE'=Fin) &
                 (justified0'=true) &
                 (finalized0'=true);

[] (M0_STATE=Check)&(!isCoherent(Heigth)) ->
 rIst : (removeCheckpoint() &
         (M0_STATE'=Start);

[] (M0_STATE=Check)&(!isCoherent(Stakes)) ->
          rIst : (M0_STATE'=Start);

[finBlock0] (M0_STATE=Fin) ->
                1 : (reset());
```

endmodule

Listing 4.2: Pseudocode of the first Validator with id=0. Each validator has the same implementation, except for the reference ids, which in this case refer to the Validator with id=0, so the next one will have its local variables ending with its own id, i.e. b1, B1 and so on.

The module of each Validator encapsulates the behavior and characteristics of a validator within the context of the simulation. To describe the functioning of a validator, it is necessary to divide the module into two sections: *state variables* and *transitions*.

State Variables

The *Validator0* module defines the following state variables:

- **M0_STATE**: Represents the current state of Validator0, which can be one of the states from *Start* to *Fin*.
- **b0**: Represents a block, it can be the block created by validator0 or another block taken from the chain, created by some other validator.
- **B0**: Represents the view of the blockchain for validator0, initialized with the genesis block.
- **votes0**: A variable representing the number of votes, initialized to 0.
- **lastFinalized0**: Represents the last finalized block, initialized to the genesis block.
- **lastJustified0**: Represents the last justified block, initialized to the genesis block.
- **lastCheck0**: Represents the last checkpoint block, initialized to the genesis block.
- **finalized0**: A boolean variable initialized to false, representing whether a block is finalized.
- **justified0**: A boolean variable initialized to false, representing whether a block is justified.

Transitions and Actions

The *Validator0* module defines the following transitions and actions:

- **M0_STATE=Start & ValidatorID**: Validator0 starts in the *Start* state and if *validatorID* contains the id of the validator, then it means that it is elected as block proposer. It transitions to *Create*, creating a block and resetting the validatorID variable.

- **M0_STATE=Start & VoterID**: The Validator is chosen as *voter* for the current slot. It has to send a vote for the head of the chain (the last proposed block that it sees) and send a vote for the last checkpoint.

- **M0_STATE=Start**: In each slot, a validator is required to keep his personal view of the blockchain up to date. Therefore at all times, the validator is listening to the new blocks proposed, even if he cannot actually vote, given that it is not certain that he has also been elected as a *voter*.

- **addBlock0** Transition (From *Create* State): When Validator0 is in the *Create* state, it can add a new block to the blockchain (*B0*) with a delay represented by **rMw**. This transition adds the previously created block to the main view and notifies the other validators on the committee of this addition, so they can proceed with a vote.

- **voteBlock0** Transition (From *Vote* State): When in the *Vote* state, the validator votes for a block by first considering the height of the checkpoint and then adding it to its view.

- **M0_STATE=Check** Transitions: In the *Check* state, validators check conditions for finalization or justification, updating various variables and potentially transitioning to the *Fin* or *Start* state. A validator first checks if the stakes are greater than 2/3 of the total stakes and next computes the height as done before.

- **finBlock0** Transition (From *Fin* State): If the validator is in the *Fin* state, it resets flags and transitions back to *Start*.

4.3 Updater

The Updater module is used to initialize, track and update the stakes of each validator belonging to the network.

```
module Updater

    Updater_STATE : [0..1] init 0;
    n_epochs : [2..N] init 2;
    tot_stake : [0..50000] init 32 x VALIDATOR_COUNT;
    voti : hash [];
    maxHeight : [0..5000] init 0;
    reward_i : [0..N] init 0;
    stake_i : [0..N] init 32;

    [voteBlock_i] (Updater_STATE=0) ->
        1 : (voti'=addVote(voti,b_i,stake_i));

    [finBlock_i] (Updater_STATE=0)&(coherentHeigth()) ->
        1 : (stake_i'=updateStake(voti,lastFinalized_i,
            (getHeight(lastFinalized_i),stake_i,tot_stake)) &
                ( i+1 to n repetitions );

endmodule
```

Listing 4.3: Pseudocode for Updater module. Here it is implemented a generic code, without using specific IDs. It is necessary to understand that for a correct implementation, it is necessary to repeat the three transitions described for all the validators that will be implemented. For example, if the number of validators is equal to 13, it will be necessary to implement 13 transitions such as voteBlock and finBlock taking care to replace the IDs correctly.

This module manages the traffic of stakes, rewards and, more generally, the updates linked to them. The state variables are:

- **Updater_STATE**, used as a binary flag that can take the value 0 or 1 and represents the actual use of this module, i.e. when it must be used and when not.
- The variable **n_epochs** is used in the phase of updating the stake of the single validator.

- The variable **tot_stake** takes into account all the stakes, therefore initialized to 32 × VALIDATOR_COUNT, as required by the protocol.
- **voti** is a hasMap used to keep track of the votes made to the validators on the proposed blocks and this table follows the principles of the LMD (Last Message Driven) algorithm.
- **maxHeigth** already seen in the previous module, serves to keep track of the maximum height up to that moment of the checkpoints, or more abstractly, of the chain.
- **reward**$_i$ and **stake**$_i$ are the two variables linked to individual validators, which keep track of the rewards received and the current stake of each individual user.

The transitions involved in this module are repeated for each validator, as can be imagined from the pseudocode. The transition **voteBlock**, when the Updater is ready to work, all it does is add a vote inside the hashMap for that specific block passed as input.
Much more complex is the **finBlock** transition which, after a careful analysis of the consistency regarding the height of that block in relation to the other checkpoints or finalized blocks, allows you to update the stake value of the single validator and consequently also that of the variable tot_stakes.

4.4 Network

The Network module manages the entire part relating to the communication between validators, i.e. the addition or elimination of blocks to the main blockchain, considering the addition or removal delays that are foreseen by the protocol used.

```
module Network

    Network_STATE : [0..2] init 0;
```

```
set_i : list [];
lenGlobal_i : [0..N] init 0;

[addBlock_i] (M_STATE=Create) ->
    rAdd : (set_{i+1}'=addBlockSet(b_i)) &
           (set_{i+2}'=addBlockSet(b_i)) &
            (repeat for each vaidator i+N);

[extractBlock_i] (M_i_STATE=Move) ->
        r : (set_i'=removeBlock(b_i));
```

endmodule

Listing 4.4: Pseudocode for Network module. Each validator has its own variables set_i and lenGlobal_i and its own transitions addBlock_i and extractBlock_i, where i is referred to the ID.

The variables used in the Network 4.4 module are **set**, which contains all the blocks currently intended as checkpoints, which can then be justified or finalized, and the variable **lenGlobal** which takes into account the length of each set of the individual validators, acting as a local block counter. This variable has a very important role when calculating the various heights, described in the previous listings.

The transitions in this module are used to manage the step before justifying or finalizing a block. If a block needs to be added, in the **addBlock** transition, then that block will be added to each validator's set, so that everyone can have it in their set. The opposite situation exists in **extractBlock**, which is carried out only in the set of the user who proposed the block, i.e. the validator with the role of proposer, given that it is an operation that is called chronologically before the addition.

4.5 Global

The Global module is used to manage the Fork calculation according to the algorithm described by the Gasper protocol, and all the main operations

carried out for the experiments.

```
module Global

    diff : [0..N] init 0;
    kLength : bool init false;
    finalizationIncrease : bool init false;

    [] (M_i_STATE = Move) | (repeat for each Val) ->
                1 : (diff'=calculateFork(B_i .. B_N));

    [] (diff>0) & (diff=K) ->
                1 :(kLength' = true);

    [] (finCount > oldFinCount) ->
                1 : (finalizationIncrease' = true) &
                (oldFinCount' = finCount);

endmodule
```

Listing 4.5: Pseudocode for Global module.

The pseudocode 4.5 shows the operations carried out to calculate the length of the fork, starting from the **MOVE** state of each validator, calculating its height, then inserted into the **diff** variable. The length of the fork will then be analyzed as shown in the experiments in Chapter 5.

A further analysis is that carried out for *probabilistic liveness*, discussed in Chapter 5, which has the objective of ascertaining the constant increase in the number of finalizations, **finalizationIncrement**, even with delays or attacks on the network.

4.6 RanDAO

A decentralized autonomous organization (DAO) is established to facilitate the participation of any interested entity. Within this framework, random numbers are collaboratively generated by all participating entities. The

process begins by creating a RANDAO contract on the blockchain, a pivotal step that defines the rules of participation. The fundamental procedure for generating a random number unfolds through three distinct phases [31]:

1. Collection of Valid sha3(s): Entities seeking involvement in random number generation must initiate a transaction with contract C during a specified time frame (e.g., a 6-block period or approximately 72 seconds). Alongside this transaction, participants are required to pledge m ETH and submit the result of sha3(s), where s represents the secret number individually chosen by each participant.

2. Collection of Valid s: Subsequent to the conclusion of the first phase, participants who successfully submitted sha3(s) must dispatch another transaction containing their secret number s to contract C within a predetermined time limit. Contract C conducts a validity check on s by executing the sha3 operation and comparing the result with previously committed data. Valid s are retained in the collection of seeds, culminating in the eventual generation of the random number.

3. Random Number Computation, Pledge Refund, and Bonus Allocation: Once all secret numbers have been successfully collected, contract C proceeds to calculate the random number using the function $f(s1, s2, ..., sn)$ The resultant value is recorded within contract C and subsequently distributed to all other contracts that had previously requested the random number.

   Contract C administers the return of pledges to participants from the first phase. Any profit generated is equitably divided among all participants as an additional bonus, with these profits being sourced from fees paid by other contracts utilizing the random number.

Additional Rules

To safeguard against the manipulation of the Random Number Generation (RNG) and to enhance safety and efficiency, contract C enforces sup-

plementary regulations [32]:

- In the first phase, only the first occurrence of sha3(s) is accepted if two or more identical sha3(s) submissions are consecutively received.
- The first phase imposes a minimum participant requirement; failure to accumulate a sufficient number of sha3(s) within the specified time frame results in the failure of RNG at that particular block height.
- If a participant submits sha3(s) that is accepted by contract C, they are obligated to reveal s during the second phase.
  - Failure by the participant to reveal s in the second phase results in the confiscation of the m ETH pledged during the first phase without the possibility of reimbursement.
  - If one or more s remain undisclosed during the second phase, RNG at that block height is deemed unsuccessful. The confiscated ETHs are evenly distributed among other participants who did reveal s in the second phase, with refunds provided for fees paid by other contracts.

Prism code

These operations are segregated into two distinct modules. The initial module, as depicted in Listing 4.6, is responsible for generating, during epoch e, the proposers and the voters for the 32 slots of epoch $e + 2$, considering the indices encompassing all validators within the network. Conversely, the Randao Selection module is tasked with allocating the appropriate proposer and voter for that specific slot by assigning the designated proposer into the global variable $Validator\_ID$ and the voters into $VoterID\_1$, $VoterID\_2$ and $VoterID\_3$.

module Randao

```
for i from 0 to 31:
```

```
        ValidatorB_i: int init -1;
        for j from 0 to 2: Voter_ji: int init -1;
    for i from 0 to N: index_i : int init i;

    [] (state=0)&(randao=true) -> 1 : (state'=1)& for i from 0
       to N:
                              if Stakes[i] < 32 : index_i' = -1;
    [] (state=1) ->
            1: (randao'=false) &(state'=0)& for i from 0 to 31:
                ValidatorB_i'=randomNumber(epoch,i, index_0,...,
                    index_N)&
                Voter_0i'=randomNumber(epoch,i+31, index_0,...,
                    index_N)&
                Voter_1i'=randomNumber(epoch,i+63, index_0,...,
                    index_N)&
                Voter_2i'=randomNumber(epoch,i+95, index_0,...,
                    index_N)

endmodule
```

Listing 4.6: Pseudocode for Randao Module. The constant n refers to the total number of validators who will take part in the next draw.

```
module Randao_Selection

    i : [0..32] init 0;

    [] (i=0)&(randao=false) -> 1 : (validatorID'=ValidatorB0)&(i'
       =i+1);
    for j from 1 to 31:
        [] (i=j) -> slot :
            (validatorID'=ValidatorB_j)&(voterID_0'=Voter_0i)
            &(voterID_1'=Voter_1i)&(voterID_2'=Voter_2i)&(i'=i+1)
                ;
        [] (i>31) -> 1: (i'=0)&(randao'=true);
endmodule
```

Listing 4.7: Pseudocode for Randao Selection Module. The constant n refers to the total number of validators who will take part in the next draw.

The division into these two distinct modules allows for a systematic approach to the process, wherein the first module handles the comprehensive selection of proposers for a future epoch based on validator indices, while the latter module focuses on the precise assignment of a specific proposer for the intended slot. This segregation of functions aids in the efficient and organized execution of the overall process, ensuring the appropriate allocation of validator roles within the epoch structure.

4.7 Labels

The labels employed during these experiments play a pivotal role in Chapter 5. These labels serve as explicit references to the particular conditions and scenarios meticulously examined within this study section. They are instrumental in delineating the specific contexts in which we sought to quantify and assess the probabilities of success or failure.

These labels are, in essence, identifiers for distinct situations and scenarios where it becomes feasible to perform precise calculations or forecasts regarding the likelihood of success or failure. They function as crucial tools in our analytical framework, enabling us to establish a rigorous basis for comparing the model we have constructed with the regulations and guidelines stipulated by the protocol under examination.

By assigning these labels to well-defined conditions and scenarios, we create a structured and systematic approach to evaluate the performance and adherence of our model to the prescribed standards. This systematic comparison allows us to gauge the model's efficacy in predicting outcomes accurately within the given protocol. In essence, these labels act as the cornerstone for a comprehensive and meticulous evaluation, providing the necessary context to assess the model's proficiency in the context of the examined conditions and, ultimately, contributing to a more insightful and robust analysis.

```
label "winner" = (M_i_STATE .. M_n_STATE = Winner);

label "someCreated" = (created_i .. created_n = true );

label "someJustified" = (justified_i .. justified_n = true );

label "someFinalized" = (finalized_i .. finalized_n = true );

label "finInc" = (finalizationIncrease = true);

label "equalLength" = (kLength = true);
```

Listing 4.8: Pseudocode for Labels delcaration. The constant n refers to the total number of validators.

The labels depicted in Listing 4.8 hold significance in the context of a series of experiments conducted. The pseudocode provided therein succinctly delineates the process of label creation, subsequently enabling their utilization in bespoke property formulations during subsequent simulations of the model.

The **someWinner** label has been deliberately conceived for the express purpose of scrutinizing both the efficiency and correctness of the *RanDAO* module. It is predicated upon the attainment of the coveted "Winner" status, a distinction conferred upon an entity when selected as a block proposer within the system.

On the other hand, **someCreated** pertains to the precise instant when a block comes into existence, prior to any form of voting or validation procedures. This label assumes particular significance as it serves as an indispensable component for assessing the ramifications of delays in block creation on subsequent justifications or finalizations within the framework.

Furthermore, **someJustified** and **someFinalized** labels correspond to the principal conditions of finality, encompassing both justification and finalization events. Their integration has been instrumental in fostering an analytical framework aimed at gauging the rates of finalization and the process of achieving consensus within the network.

Lastly, **finInc** and **equalLength** labels are closely aligned with two specific experiments pertaining to probabilistic liveness and fork probability. The former scrutinizes the dynamic progression of finalization events within the network, seeking to ascertain whether there exists a discernible increase in their frequency. The latter, *equalLength*, undertakes the intricate task of evaluating the length of a fork, explicitly examining whether it falls below, exceeds, or equals the parameter passed as an input to the analysis. This operation is done only for reasons of comparison with the previous protocol, given the use of the LMD-Ghost algorithm in Gasper which does not consider the length of the fork, but rather the total weight of the blocks.

Chapter 5

Experiments

In this chapter, experiments will be performed in order to test the simulation produced with the Prism model checker of the Ethereum Gasper protocol. Initially, some basic tests, or *Coherence* tests, will be performed in order to ensure that the simulation works correctly. These tests will also have to reproduce the expected behavior of the Gasper protocol.

Another way to test the model would be to conduct stress tests and comparisons with other protocol simulations, but since this simulation is not connected to the network, it would not be possible to test its ability to handle large numbers of transactions. Since there are no practical simulations of models relating to the Gasper protocol in the literature, the model created is compared with the one created for the previous protocol, i.e. *hybrid casper*.

Subsequently, *Security* tests will be carried out, where the model will be tested, theoretically and practically, against potential attacks.

5.1 Simplifications

Considering that there are no practical simulations of Ethernet PoS in the literature and no data are reported regarding the coherence of the various proposed models, for these experiments some simplifications are adopted, in order to be able to compare it with the previous protocol. For this reason,

block votes and checkpoint votes share the same structure and implementation. Only one type of generic vote is considered, but if this vote is directed to a block with a height multiple of 32, then it is a checkpoint vote, whereas if the block height is not a multiple of 32, then it is a block vote. A further simplification concerns the number of validators created for these experiments. 6, 13 and 16 are considered and could not be increased due to hardware limitations. However, in Figure 5.1 the justification rates for the three groups of validators are shown. Behaving in the same way, it is decided to perform further experiments with a number of validators N equal to 13. The last simplification adopted concerns the incentives and penalties for honest or dishonest verification, which depend on the quantity of ETH deposited. In the simulation, the value of the penalty or reward is approximated to the value that that specific validator could have in the analyzed epochs.

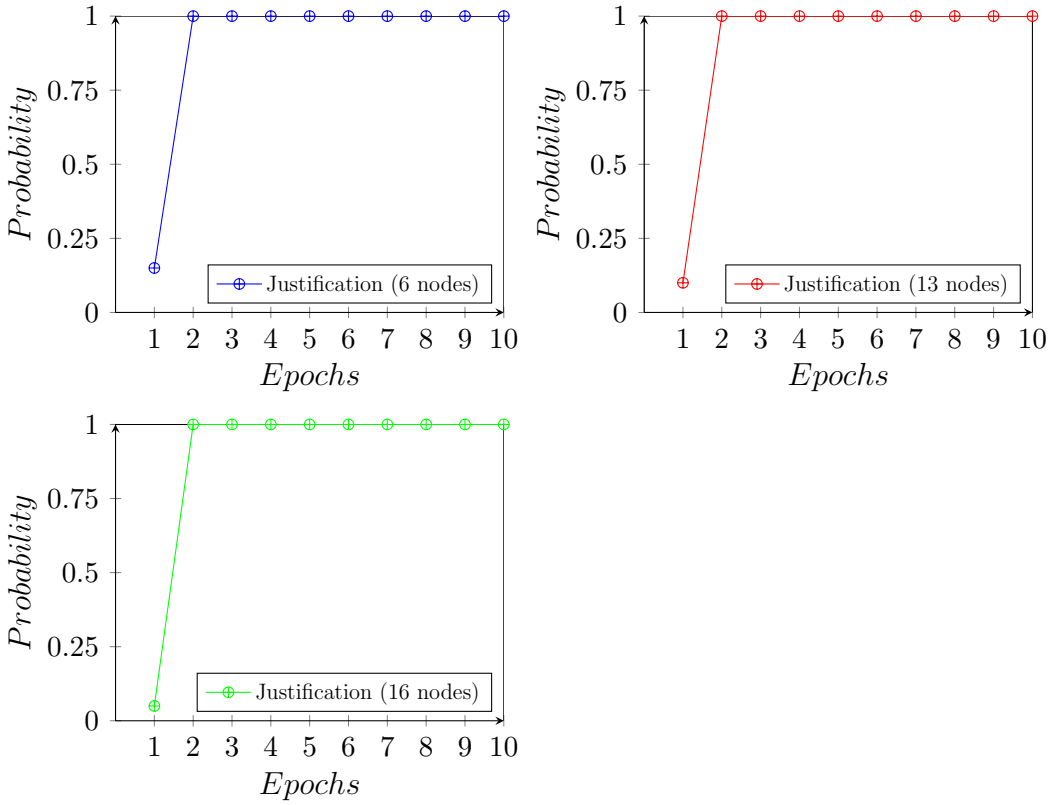

Figure 5.1: Justification rates over 6, 13 and 16 Validators.

5.2 Coherence

In this section, many of the main features of the *Gasper* protocol characterizing the *Ethereum Proof of Stake* network in use since September 2022 are applied and reproduced. The consistency of the model created using Prism Model Checker will be demonstrated, comparing it with the expected results of Ethereum PoS.

These experiments will also be compared with the results obtained from previous experiments performed on the *Hybrid Casper* [1] protocol.

Block Creation

In both Ethereum Proof of Work (PoW) and Ethereum Proof of Stake (PoS) protocols, *block creation* refers to the process of adding new blocks to the blockchain.

Ethereum Proof of Work (PoW) Block Creation: In the PoW consensus mechanism, block creation involves solving complex cryptographic puzzles, known as the *Proof of Work*, to validate transactions and create new blocks. Miners, who are participants in the network, compete to solve these puzzles, and the first miner to find a valid solution gets the right to create and add the new block to the blockchain. This process is computationally intensive and requires significant computational power and energy consumption.

Once a miner finds a valid solution and creates a new block, they broadcast it to the network for verification. Other nodes in the network then validate the block's transactions and the Proof of Work solution. If the block is deemed valid, it is added to the blockchain, and the miner receives a reward in the form of newly minted Ether (the native cryptocurrency of the Ethereum network) and transaction fees.

Ethereum Proof of Stake (PoS) Block Creation: In the PoS consensus mechanism, block creation involves a different approach that does not rely on solving computational puzzles. Instead, validators (also called *forgers*

or *stakers*) are chosen to create new blocks and validate transactions based on the number of coins they have staked as collateral in the network.

Validators lock up a certain amount of cryptocurrency (Ether in the case of Ethereum) as a stake to participate in block creation and validation. The probability of being chosen as a validator to create a new block is proportional to the amount of cryptocurrency they have staked. This means that validators with larger stakes have a higher chance of being selected.

When a validator is selected to create a new block, they assemble the transactions to include in the block and propose it to the network. Other validators then validate the proposed block and its transactions. If the block is deemed valid, it is added to the blockchain, and the validator who proposed the block receives transaction fees as a reward.

In summary, block creation in Ethereum PoW involves solving cryptographic puzzles, while in Ethereum PoS, it relies on the selection of validators based on the amount of cryptocurrency they have staked. Both mechanisms aim to secure the network and ensure consensus on the state of the blockchain. However, PoS is considered more energy-efficient compared to PoW due to the absence of the resource-intensive mining process.

In contrast to the proof-of-work mechanism, where block timing depends on mining difficulty, proof-of-stake operates with a **fixed tempo**. In proof-of-stake Ethereum, time is organized into **slots**, each lasting 12 seconds, and epochs, comprising 32 slots. During each slot, a single validator is randomly chosen to act as the block proposer. This selected validator assumes the responsibility of creating a new block and disseminating it to the other nodes within the network [4].

Block time refers to the time separating blocks. In Ethereum, time is divided up into twelve-second units called *slots*. In each slot, a single validator is selected to propose a block. Assuming all validators are online and fully functional there will be a block in every slot, meaning the block time is 12s. However, occasionally validators might be offline when called to propose a block, meaning slots can sometimes go empty [5].

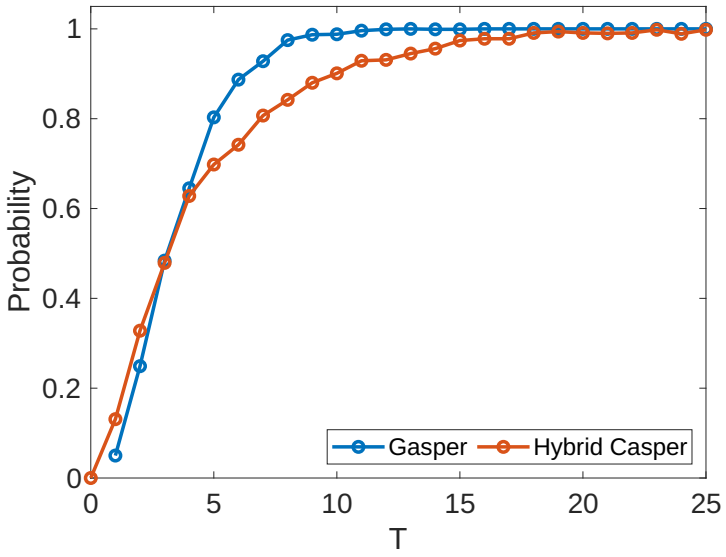

Figure 5.2: Block creation probability over time.

In Figure 5.2 it is depicted the process of block creation over a duration of 25 seconds, comprising two slots, each lasting 12 seconds. Within this timeframe, two blocks are expected to be created, as one block is anticipated to be generated in each slot. The block creation process exhibits a block creation probability (p) greater than 0.8 in the first slot. However, it is essential to consider the possibility of a validator being offline during one slot, resulting in the need to await the subsequent slot for block creation in such circumstances. Consequently, occasional occurrences may arise where validators are offline, leading to the possibility of encountering empty slots without any block creation.

Comparing these results with *Hybrid Casper*, presented in [1], it is clear that the switch from Hybrid Casper to Gasper significantly increased the block creation rate.

Finality: Justification and Finalization

Finality in the Ethereum blockchain refers to a critical property of certain blocks that renders them irreversible, except in extreme scenarios involving

a severe consensus failure and an attacker compromising at least 1/3 of the total staked ether.

To achieve finality, blocks undergo a two-step upgrade procedure. First, for a block to be *justified*, it must obtain approval from two-thirds of the total staked ether, indicating a high level of confidence in its inclusion in the canonical chain. Although justified blocks are unlikely to be reverted, certain exceptional conditions might still allow for reversals.

The second step involves *finalization*, which occurs when another block is justified on top of a justified block. This action commits to including the block in the canonical chain, making it irreversible, except under highly improbable circumstances where an attacker would have to destroy vast amounts of ether, equating to billions of USD in value.

It is important to note that not every block undergoes these upgrades; only *epoch-boundary blocks* are eligible for justification and finalization. These blocks, known as *checkpoints*, are considered during the upgrade process. To trigger the upgrade, a *supermajority link* must exist between two successive checkpoints, meaning that two-thirds of the total staked ether must vote in favor of one checkpoint being the correct descendant of the other. This procedure ensures the integrity and security of the blockchain, enhancing its robustness against potential attacks or malicious behavior.

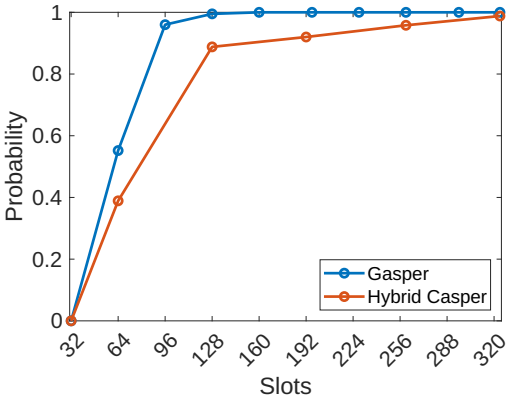

Figure 5.3: Justification.

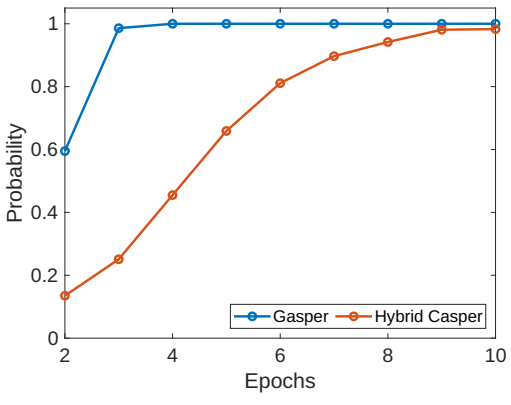

Figure 5.4: Finalization.

In the context of a no-delay and honest network, a block achieves final-

ization after approximately 2 epochs as shown in Figure 5.4. This result also emerges from the fact that two epochs are necessary to justify a block. In theory, the time needed for all validators to vote is one epoch, but considering delays in the network, each validator is assigned a time limit to carry out the vote, called *inclusion delay*. This value allows a validator to perform a vote with at most one epoch of delay, before receiving penalties. However, voting outside their own slot will lead to a reduction in reward.

To clarify the concepts, justification and finalization are determined through the FFG (Friendly Finality Gadget) voting process, wherein all validators contribute one FFG vote per epoch. If a supermajority link is established between the source checkpoint and the target checkpoint, with validators constituting more than 2/3 of the total stake agreeing on the vote, the target checkpoint becomes justified. Subsequently, if a justified checkpoint serves as the source for a supermajority link with the checkpoint of the next epoch as its target, the source checkpoint attains finalization.

The theoretical number of slots required for an epoch to be finalized is 54 slots. This can be observed with an example: Starting at epoch A, the checkpoint will be justified after 22 slots, followed by the justification of epoch B's checkpoint after another 22 slots. This, in turn, finalizes the checkpoint of epoch A. The finalization process occurs within 54 slots, considering no latency, voting conflicts, empty slots, and full participation in voting.

However, it is important to acknowledge that in practical scenarios, validators tend to wait until the end of an epoch to finalize new checkpoints and that we have to respect the Gasper protocol, i.e. consider the checkpoints for the finality, increasing the wait time to 64 slots, or 2 epochs. Considering the possibility of empty slots, we ascertain that in 2 epochs (64 slots), the probability of finalization is approximately 1. Hence, theoretically, if a block is justified in epoch 1, the same block would be finalized in epoch 3, with a high probability ($p_{fin} > 0.9$) accounting for possible forks, empty slots, or network delays.

Considering a delay in the network regarding the diffusion of created

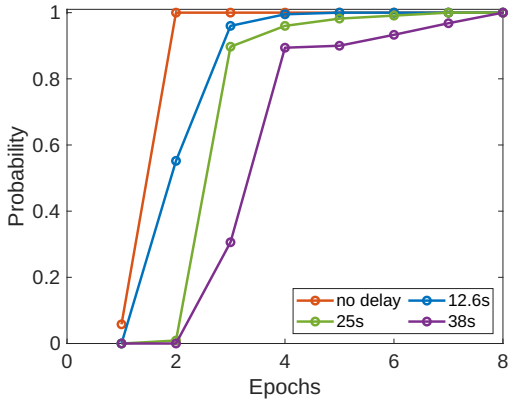

Figure 5.5: Justification with delay.

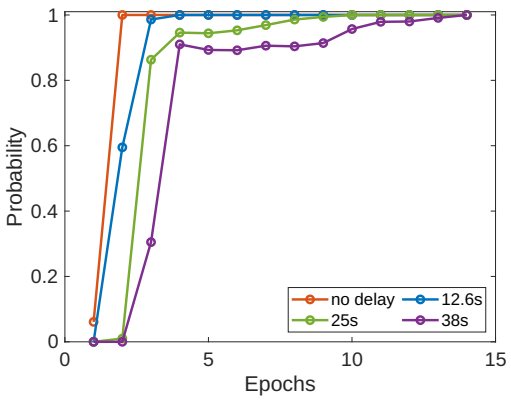

Figure 5.6: Finalization with delay.

blocks or attestations, one obtains clearly different results, as shown in Figure 5.5 and Figure 5.6. The value of the delay was calculated starting from the average delay in the Bitcoin network equal to 12.6 [28].

Comparing these results with those presented in [1], there are no significant differences regarding justification and finalization between the two protocols since both use Casper FFG.

Prism Code

These experiments are done in Prism+ simply considering the labels declared in the previous section. For *Justification* and *Finalization* is:

$$\mathbf{P} = [\mathbf{F} \leq \mathrm{T}\ \text{"someJustified"}]$$
$$\mathbf{P} = [\mathbf{F} \leq \mathrm{T}\ \text{"someFinalized"}]$$

5.3 Fork Probability

A *fork* refers to a divergence in the blockchain, resulting in the creation of alternative chains due to differing validators' decisions. When validators propose competing blocks for the same slot, or when validators reference different blocks in their attestations, a fork occurs. This leads to the emergence of multiple potential chains, each with its own set of blocks and transactions.

The LMD-GHOST (Latest Message-Driven Greedy Heaviest Observed Sub-Tree) fork choice rule operates in response to these forks. It evaluates the cumulative weight of attestations and the referencing of blocks within these competing chains to determine the canonical chain. This canonical chain, selected based on the consensus of validators' choices, becomes the accepted version of the blockchain, ensuring that a single valid state is agreed upon by the network.

We then analyzed how much the maximum length of a fork would be before its resolution, in order to compare the new protocol with the previous one.

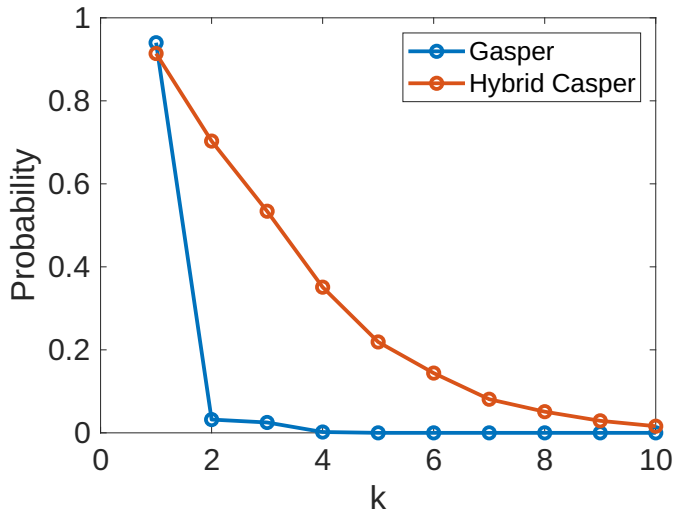

Figure 5.7: Fork of length K in 40 epochs.

As illustrated in Figure 5.7, observable is the trend in N epochs, wherein N was specifically set at 40 epochs for this experiment, and no delay factors were taken into account apart from the *fixed tempo*. Evidently, as the experiment progresses, the likelihood of attaining a continuously elongating fork approaches zero. This observation underscores the algorithm's proficiency in establishing a capped count of forks, thus mitigating potential risks to the network's security, particularly within a context featuring a predominance of honest validators and minimal latency disruptions. Comparing this result

to the one obtained in [1], the probability of forks is decreased following the Gasper protocol. One cause could be the division into specific slots performed in Gasper, where only one validator can propose a block in that slot, which did not happen before, given that when solving the puzzle multiple validators could send their block simultaneously.

Prism Code

This experiment is done in Prism+ simply considering the label declared in the previous section and the variable **K** which increases every set of epochs. For *Forks* is:

$$\mathbf{P} = [\mathbf{F} \leq \mathrm{T}\ \text{"equalLength"}]$$

5.4 Stake Analysis

In Ethereum Gasper, validators partake in a proof-of-stake (PoS) consensus system, committing their ETH holdings to secure the network while earning rewards. The dynamics of their stakes are governed by rewards and penalties [43], both of which significantly impact validator behavior and are fully described in Section 3.1.5.

In this section, an in-depth analysis of the average stakes of validators in the network is conducted. The purpose is to clarify the effectiveness of the Gasper protocol in the context of increasing or decreasing stakes at the peak of individual epochs. Each graphical representation presented here will show the stake value of a single validator or set of validators if they have the same stake on the y axis on an increased scale of $10e^4$, as the rewards they are in *Gwei*, a submeasure of *Ether*. On the x axis, we will indicate the time intervals in Epoch, with the understanding that one slot equals 12 seconds and one epoch equals 32 slots, so one epoch equals 384 seconds. In all experiments, we consider a network consisting of honest validators with few delays ($< 1sec$). In fact, the probability that a validator sends correct attestations is $\frac{4}{5}$ while the probability that it sends incorrect attestations is $\frac{1}{5}$.

It is pertinent to mention that this specific simulation does not incorporate the more punitive aspects of the protocol, such as the inactivity leak and slashing mechanisms.

To calculate the rewards and penalties of Validator i, the following formulas are used regarding rewards for completely correct attestations, penalties for completely incorrect attestations, and rewards for proposing a block:

$$\texttt{reward}_i = (3 \times \texttt{base\_rew}_i \times \texttt{stake}_i) / \sum_{j \in \texttt{V}} \texttt{stake}_j + \ 7/8 \times \texttt{base\_rew}_i$$

$$\texttt{pen}_i = -\ 3 \times \texttt{base\_rew}_i$$

$$\texttt{reward\_bp}_i = \texttt{base\_rew}_i + 1/8 \times \sum_{j \in \texttt{V}} \texttt{reward}_j$$

where *base_reward* is discussed in Section 3.1.5.

The graphical representation in Figure 5.8 illustrates the average stake evolution of a validator throughout 100 epochs, considering a total of 100 thousand validators, each possessing an equivalent stake of 32 ETH. Within this simulation, the base reward is set at 9050 Gwei, resulting in 30600 Gwei for a valid attestation and a reward of 0.113 ETH for proposing a block. In the event of an incorrect attestation, the penalty levied stands at 22010 Gwei, deducted from the validator's stake.

The graphic depiction vividly portrays that across the 100 epochs, the validator assumes the role of a block proposer once, around epoch 50, culminating in a significantly amplified reward compared to other epochs. This singular event of proposing a block resulted in a notably higher reward, delineating the potential impact of such opportunities within the PoS network. This observation provides insights into the dynamics of validator rewards and the intermittent nature of achieving substantial rewards through block proposal within the specified epoch, signifying a key factor influencing the validator's stake and rewards.

The subsequent experiment was conducted to juxtapose the preceding

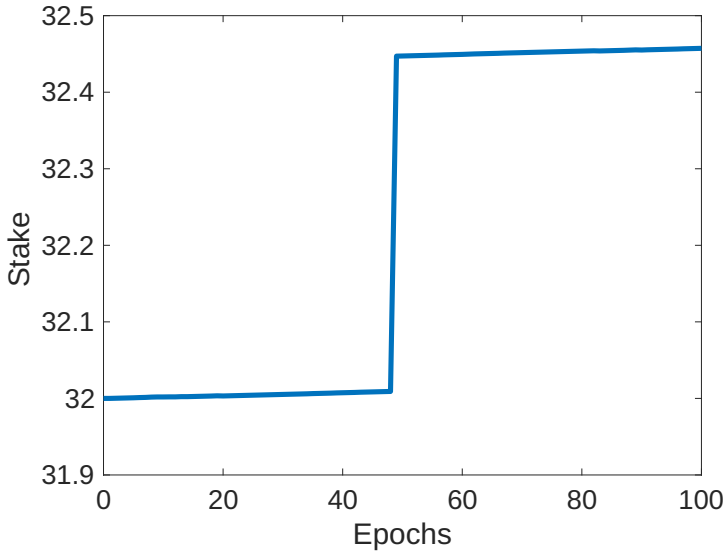

Figure 5.8: 100k validators - Equal stake.

findings within a smaller network, aiming to discern the presence of any consistent trends. Figure 5.9 presents the average stake dynamics of a validator across 100 epochs, considering a total validator count of 50 thousand, each possessing an identical stake of 32 ETH. In this simulation, the base reward stands at 12800 Gwei, translating to 49600 Gwei in the event of a valid attestation, and a reward of 0.310 ETH upon block proposal. In the case of an incorrect attestation, the penalty imposed equals 38400 Gwei.

The graphical representation illustrates that over the course of 100 epochs, the validator assumes the role of a block proposer twice, approximately around epoch 10 and epoch 80. This position entitles them to two significantly larger rewards, ultimately surpassing the limit of 32.5 ETH, which was the pinnacle achieved in the experiment involving 100 thousand validators. This observation suggests that within a smaller network, there exist instances where a validator can surpass the earnings attained within a larger network, potentially signifying an impact on user engagement due to these differing reward dynamics.

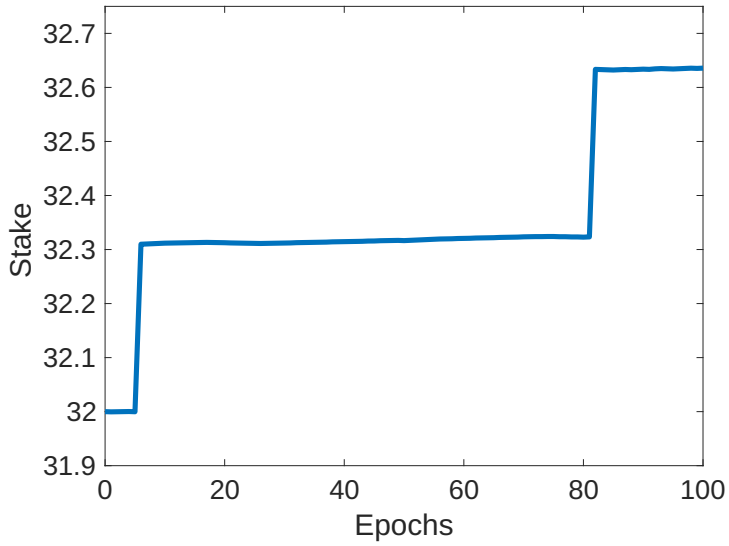

Figure 5.9: 50k validators - Equal stake.

Considerations over Rewards and Penalties

These experimental considerations are undertaken with the primary objective of comprehending the underlying rationales guiding the decisions made within the framework of the Gasper protocol.

The experiment depicted in Figure 5.10 illustrates the performance evaluation of a singular validator possessing a stake higher than that of other validators, specifically set at 50 ETH. The analysis spans over 100 epochs, maintaining the same reward structure for block proposals as previously employed, which predominantly depends on the stakes held by other validators within the network. In this experimental scenario, the base reward allocated to the wealthiest validator stands at 20000 Gwei, while the reward for a valid attestation amounts to 77500 Gwei. The penalty imposed for incorrect attestations is fixed at 38400 Gwei.

This experiment sheds light on the fact that the presence of a single validator with a stake deviating from the average does not inherently translate to a significant enrichment compared to others. This observation emerges from the realization that the stake amount or threshold influencing the selec-

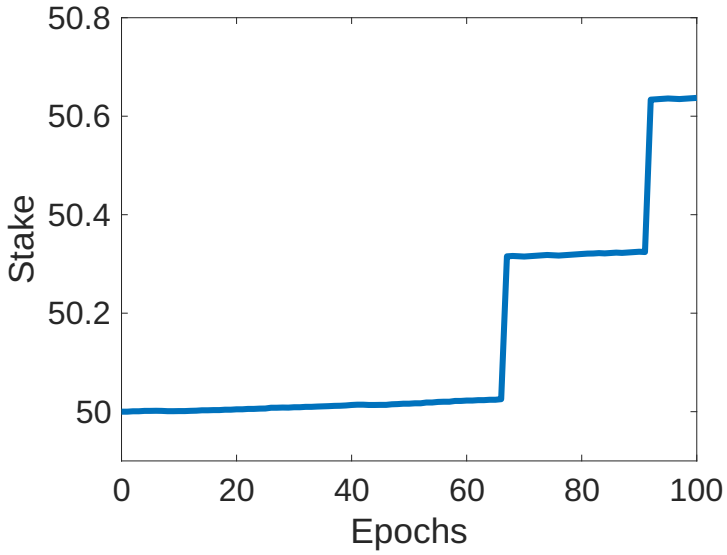

Figure 5.10: 50k validators - One with greater stake.

tion of the validator proposer remains at 32 ETH, indicating that individual stake variations, while influential, may not substantially elevate one validator's rewards beyond a certain point.

The final experiment within this stake analysis, depicted in Figure 5.11, delineates the trajectory of the average stake assuming all validators possess greater wealth, considering a per capita stake of 64 ETH. In this experimental scenario, the base reward stands at 18102 Gwei, translating to 70145 Gwei for valid attestations. The penalty incurred for incorrect attestations amounts to 53306 Gwei. As all validators are assumed to possess a higher stake, the reward for proposing blocks escalates to 0.4384 ETH. Remarkably, within 100 epochs, with a single validator proposing two blocks, the threshold of earning 1 ETH in total was attained. This experimental insight allows us to infer that as the average total stake value surges - in this instance, doubling the anticipated amount required to become a validator and enter the network - the gain for an individual validator experiences a substantial upsurge. This phenomenon leads to a considerable infusion of ETH within the network.

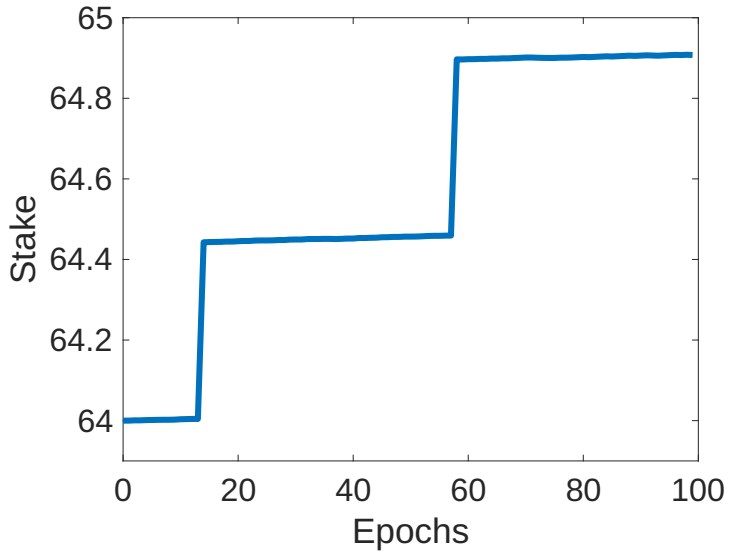

Figure 5.11: 50k validators - All with greater stake.

Prism Code

These stake analyses mainly involved the Updater module, responsible for managing votes, rewards and penalties. In this case, simulations are performed considering that with 100k validators with a stake equal to 32ETH, the probability of becoming a proposer is 1/100000. Same thing for the 50k validators. In this case the stake has been scaled to 32000, considering that the rewards are in *Gwei*. As can be seen from the Listing 5.1, the probability of sending a correct attestation is $\frac{4}{5}$ while the probability of receiving a penalty for an incorrect attestation is $\frac{1}{5}$.

```
//all validators have 32 ETH (100k validators)
[] (Updater_STATE=0) -> 1/100000 : (stake0'=stake0+1130);
[voteBlock] (Updater_STATE=0) -> 4/5 : (stake0'=stake0+3);
[voteBlock] (Updater_STATE=0) -> 1/5 : (stake0'=stake0-2);

//all validators have 32 ETH (50k validators)
[] (Updater_STATE=0) -> 1/50000 : (stake0'=stake0+3100);
[voteBlock] (Updater_STATE=0) -> 4/5 : (stake0'=stake0+5);
[voteBlock] (Updater_STATE=0) -> 1/5 : (stake0'=stake0-4);
```

```
//all validators have 32 ETH one has 50 ETH (50k validators)
[] (Updater_STATE=0) -> 1/50000 : (stake0'=stake0+3100);
[voteBlock] (Updater_STATE=0) -> 4/5 : (stake0'=stake0+8);
[voteBlock] (Updater_STATE=0) -> 1/5 : (stake0'=stake0-6);

//all validators have 64 ETH (50k validators)
[] (Updater_STATE=0) -> 1/50000 : (stake0'=stake0+4384);
[voteBlock] (Updater_STATE=0) -> 4/5 : (stake0'=stake0+7);
[voteBlock] (Updater_STATE=0) -> 1/5 : (stake0'=stake0-5);
```

Listing 5.1: Stake analyses with different rewards and penalties.

5.5 Safety

Safety is an aspect strictly related to the consensus protocol used. As defined in [13], when considering the set of finalized blocks, denoted as $F(G)$, for any given view G, it is a fundamental condition that this set should never include two blocks that are in conflict with each other. An implication arising from the assurance of safety is that for any validator's view G, the set of finalized blocks ($F(G)$) can be seamlessly extended to form a distinctive subchain within the more extensive set of finalized blocks ($F(view(NW))$). This subchain commences from the genesis block and extends up to the last finalized block, effectively forming a continuous sequence known as the finalized chain.

As provided in [13] and [1], the best way to test the safety of the model is to verify the basic condition, i.e. that there are no two blocks or checkpoints aimed at the same height, increasing the network delay, thus also simulating a condition of maximum congestion.

As illustrated in Figure 5.12 starting from maximum congestion equal to $delay = 0.1$ and up to the ideal situation of no delay, i.e. $delay = 1$, it is shown that the network does not allow in any case to have two blocks at the same height. For each level of delay, a test was performed on 12 epochs. The label used in Prism to verify this feature is *sameHeigth* illustrated in

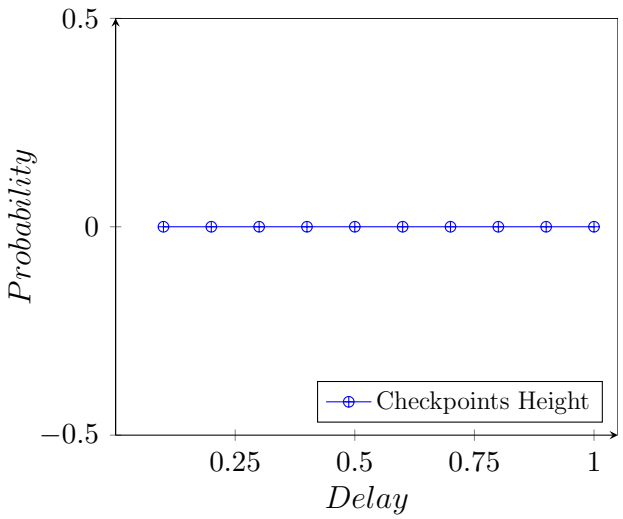

Figure 5.12: Safety condition.

Chapter 4. So, it is plausible to assume that the protocol is safe given that it respects the theorized and pre-established specifications.

Prism Code

Also in this case, the experiment is carried out considering the label *sameHeight*, which checks the presence of different checkpoints at the same height:

$$\mathbf{P} = [\mathbf{F} \leq \mathrm{T}\ \text{"sameHeight"}]$$

5.6 Security Analysis

As indicated in [13], the ultimate ledger achieved through Gasper's mechanisms is deemed secure. Nevertheless, it has been contended that its operational viability is confined to a distinct stochastic network delay model. Aligning with the principles espoused by [22] for the formulation and scrutiny of blockchain protocols, the analysis conducted in [21] examines Gasper through a conventional security framework, ultimately revealing vulnerabilities.

Specifically, the scrutiny exposed a potential liveness attack targeting Gasper within the confines of the standard synchronous model. This threat arises when adversaries manipulate message delays within the bounds of a known network delay threshold, potentially causing considerable disruption. Notably, this liveness attack, which capitalizes on a balancing strategy that leads to a division of votes across parallel chains, engenders a dire outcome: it compromises not only the system's operational fluidity but also endangers the integrity of the existing ledger. This jeopardy persists even in the absence of a network partition.

These comprehensive analyses were undertaken before the conclusion of 2021. Of note is the subsequent improvement in the resilience landscape of the Gasper Protocol. In addition to the LMD-Ghost algorithm, the integration of a limit of 4 epochs has been introduced, thus circumventing the security vulnerabilities outlined above because using 4 epochs as a time limit, i.e. if in 4 epochs not a single block is finalized, the *inactivity leak* described in the previous chapter begins to exclude some validators from the committee.

5.6.1 Liveness

Liveness pertains to the dynamic growth potential of the set of finalized blocks. Various definitions of liveness exist, in [13] were outlined two distinct aspects:

- Plausible liveness: This concept ensures that the protocol maintains a consistent capability for new blocks to achieve finality irrespective of prior occurrences like attacks or latency issues. This condition aims to prevent scenarios where the progression of honest validators could be hindered, except by someone voluntarily relinquishing their stake. Essentially, plausible liveness safeguards against the possibility of the protocol becoming *deadlocked*.

- Probabilistic liveness: In this scenario, the likelihood of new blocks at-

taining finality remains high, regardless of previous events. This likelihood is established based on certain probabilistic assumptions related to network latency, attacker capabilities, and other factors. Although this variant appears to encompass the concept of plausible liveness, a nuanced distinction arises. Plausible liveness purely addresses the protocol's logical operation, while probabilistic liveness relies on potentially robust assumptions concerning the broader implementation context. It necessitates these assumptions to ensure that the protocol predominantly functions as intended.

At a surface level, one might perceive probabilistic liveness to encompass plausible liveness. However, the distinction is more intricate: plausible liveness stands as a deterministic property intrinsic to the protocol's logic, while probabilistic liveness calls for potentially stringent contextual prerequisites to ensure the protocol's reliable operation under varying conditions.

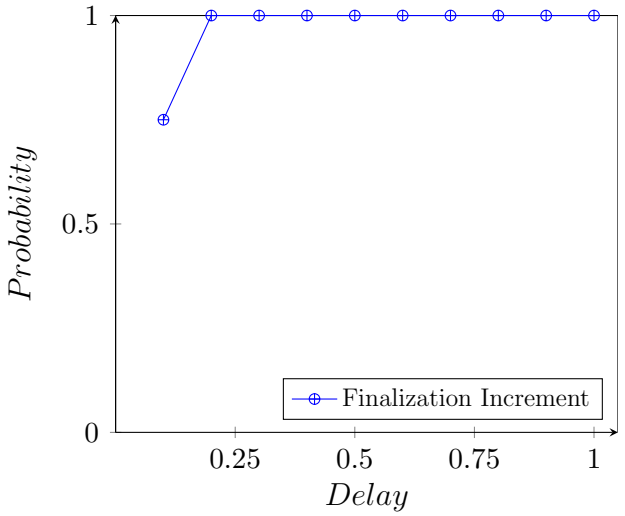

Figure 5.13: Probabilistic Liveness.

In Figure 5.13, the probability of observing an augmentation in the count of finalizations is depicted, taking into consideration a tenfold increment in delay. It is imperative to underscore that when the optimal delay is extended by a factor of ten, the likelihood of encountering an elevation in finaliza-

tion occurrences substantially diminishes relative to scenarios characterized by shorter delays. This observed behavior can be attributed to the inherent correlation between a significant augmentation in network latency and a proportionate extension in the temporal interval required for finalization to transpire.

Upon meticulous examination of the outcomes derived from epochs featuring delay values conducive to the finalization of one or more blocks, a consistent observation emerges: a positive probability persists in the detection of an escalation in the count of finalizations. This outcome unequivocally establishes that, even under conditions where network delay is augmented by an order of magnitude, thereby simulating an environment marked by high levels of congestion and severity, the protocol's commitment to ensuring liveness remains unswerving and impervious.

Prism Code

Once again the experiment is carried out considering the label, in this case *finalizationIncrease*, which precisely checks the presence of increases in the number of finalizations:

```
P = [F ≤ T "finalizationIncrease"]
```

5.7 Robustness to Attacks

In this section, there are described some of the most famous attacks designed for the Ethereum PoS network. Some of these have already been mitigated and resolved with updates scheduled for the Eth2.0 network after The Merge.

5.7.1 Bouncing Attack

The bouncing attack [29], within the context of Ethereum's Proof-of-Stake (PoS) consensus protocol, represents a sophisticated form of liveness attack capable of significantly disrupting the network's normal operation. In essence, it embodies a variant of the denial-of-service attack paradigm, wielding the potential to obstruct the conclusive finalization of blocks within the Ethereum ecosystem. In Figure 5.14, it is described a simple situation of bouncing attack. The numerical value contained within each hexagonal shape corresponds to the count of validators who have cast a checkpoint vote designating that specific checkpoint as their target. In the initial phase, we find ourselves in a scenario characterized by a blockchain fork. At this juncture, one of the chains boasts a justified checkpoint, while the other can rightfully accommodate a checkpoint from a higher epoch. Both checkpoints hold valid justifications. This situation transpires at the conclusion of the third epoch, during which honest validators have effectively distributed their votes across both blockchain branches. Subsequently, as we transition into the fourth epoch, often referred to as the Global State Transition (GST) phase, four honest validators have already submitted their votes, in alignment with the protocol's requirements. The pivotal moment occurs in the third step when dishonest validators initiate their actions and release their checkpoint votes for the opposing blockchain branch. This act effectively justifies the previously neglected checkpoint and subsequently alters the identity of the highest justifying checkpoint. This sequence of events can be perpetuated, enabling the bouncing attack to persist indefinitely through repeated iterations.

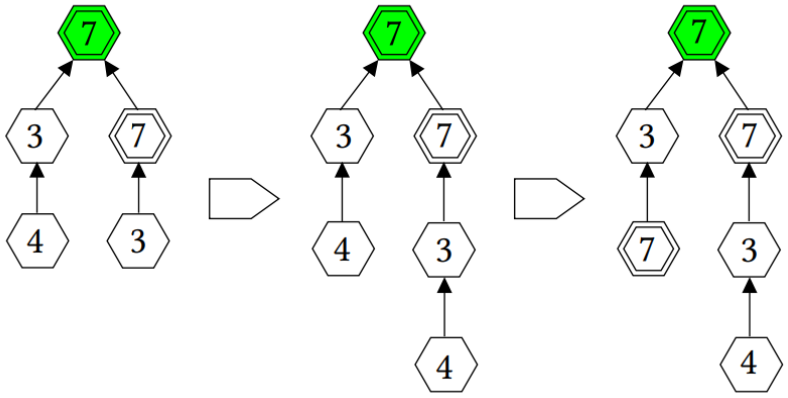

Figure 5.14: Simple Bouncing attack. Figure from [14].

It is essential to recognize that the bouncing attack poses a significant threat to the security of the Ethereum PoS consensus protocol.

The crux of this attack hinges on an inherent feature of Ethereum's fork choice rule, permitting validators to transition to an alternative blockchain if they perceive it as having a higher likelihood of becoming the canonical chain. This transition is enacted by means of casting votes in favor of a checkpoint located on the alternative chain.

The execution of a bouncing attack necessitates the malevolent actor's control over a substantial portion of validators. Initially, these attackers withhold their votes for a designated number of epochs, enabling legitimate validators to advance the chain unimpeded. Subsequently, after the honest validators have successfully justified a checkpoint, the attackers unleash their votes and facilitate a switch to a different chain. Consequently, honest validators realign their allegiance, commencing a cyclic recurrence of this process.

A countermeasure to mitigate the bouncing attack was introduced through a *patch* [33]. This patch imposes a restriction on validators, disallowing them from altering their preferences regarding justified checkpoints once a portion of the epoch has transpired. In practical terms, once a validator has cast its vote for a checkpoint, it is precluded from voting for a different checkpoint

within the same epoch. It is explained in the 5.7.1 subsection.

However, it is imperative to acknowledge that this patch does not constitute a definitive resolution to the bouncing attack quandary. [14] and [34] have revealed that the attack can still be orchestrated, albeit with a diminishing probability of success as time elapses. Ergo, the possibility of the attack endures, albeit with decreasing likelihood as temporal intervals progress.

Moreover, the authors have contributed a novel high-level formalization, elucidating the properties of liveness and availability within the Ethereum blockchain. This formalization holds promise for augmenting the analytical framework surrounding not only the bouncing attack but also other potential security vulnerabilities within Ethereum.

While the implemented patch is a noteworthy stride towards mitigating the bouncing attack, it does not offer an infallible deterrent. The attack remains plausible and retains the potential to disrupt network operations. Nevertheless, the patch does ameliorate the probability of success of the attack and provides a foundation for identifying and addressing additional security weaknesses in Ethereum. It is imperative to underscore that the bouncing attack represents but one among several conceivable security vulnerabilities within the Ethereum ecosystem.

Implemented Patch

The patch used comes from the observations, theories and experiments carried out in [35]. They present a simple modification of the Casper FFG (Friendly Finality Gadget) making it difficult for a malicious user to continue an attack as long as he does not have strong control over the network.

In the context of Casper FFG, it is noteworthy that the finalization rule does not commence its operation from the genesis block. Rather, it initiates from the latest justified checkpoint, referred to herein as the *start point*. This start point dynamically changes whenever a validator encounters a new latest justified checkpoint.

To enhance this mechanism, they have introduced a modification ensuring

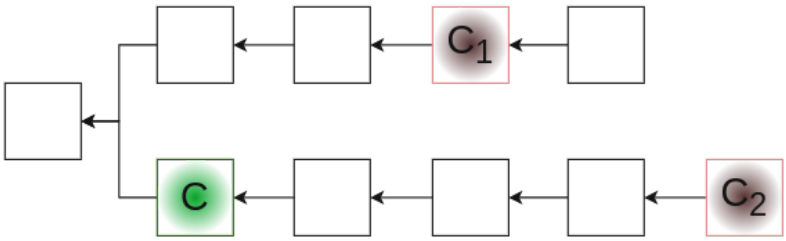

Figure 5.15: Example of Checkpoints in Bouncing attack.

that the transition of the start point to an alternate chain only occurs during the initial **k** slots of each epoch. More precisely:

1. When a validator identifies a new latest justified checkpoint that conflicts with the current start point:
   - If the validator's local clock indicates that it is within the first **k** slots, they promptly replace their start point with this new checkpoint.
   - If, however, the validator's local clock places it beyond the first **k** slots, they designate the new checkpoint as *pending*.
2. Upon the commencement of a new epoch, the validator recalculates the latest justified checkpoint for the start point. This calculation takes into account all justified checkpoints observed, including those marked as *pending*.

It is imperative to note that the parameter **k** must satisfy the condition of being less than $SLOTS\_PER\_EPOCH/3$ slots.

The effectiveness of this fix has been proven in [35]. As depicted in Figure 5.15, consider a scenario in which, at a particular juncture, the network achieves full synchronicity (i.e., with a delay of less than one slot), and the requisite conditions for bouncing come into play. These conditions entail the start point of honest validators aligning with a checkpoint denoted as

C, alongside the emergence of a subsequently justifiable checkpoint C_1 that directly conflicts with C.

For the sake of analysis, let C_2 be the checkpoint initially embraced by the honest validators. The premise here posits that an attacker lacks the capacity to render C_2 justifiable.

In instances where no new latest justified checkpoint emerges within the initial **k** slots, the honest validators effectively validate and justify C_2 in the same epoch. Conversely, if an attacker opts to justify C_1 and disseminates the associated votes within the initial **k** slots, the honest validators concede by adopting it as their new starting point. However, it is essential to underscore that C_2 remains immune to justification owing to the prescribed constraint, i.e., $\mathbf{k} < SLOTS\_PER\_EPOCH/3$. Consequently, the cumulative votes for C_2, denoted as $FFGVotes(C_2)$, consistently fall short of the threshold of $n/3$.

In consideration of the aforementioned analysis, the attacker's capacity to consecutively establish checkpoints that are justifiable (albeit not justified) and simultaneously conflicting is categorically impeded. Consequently, the phenomenon of bouncing ultimately ceases once the attacker exhausts all premeditated justifiable checkpoints that precede the network's full synchronization.

In the Capella Upgrade, this patch was eliminated from the Gasper protocol, since it added complexity to the LMD-Ghost fork-choice rule and since the conditions to reproduce the bouncing attack are too strict and complicated.

Bouncing Attack - Testing

The attack involves bouncing between two chains A and B at each checkpoint, so continuity is expected in the proposal of blocks in both chains. Assuming that the epoch $e = 0$ is valid to start the attack, therefore having two checkpoints at the same height and the network expresses a slight preference towards one of the two, but not an absolute preference. In this case, after at the end of that epoch all the honest validators have voted for one of the two checkpoints, there will be a situation in which one of the two checkpoints has more votes than the other, but not the 2/3 necessary to justify it. At the end of the $e = 1$ epoch, the dishonest validators will release their

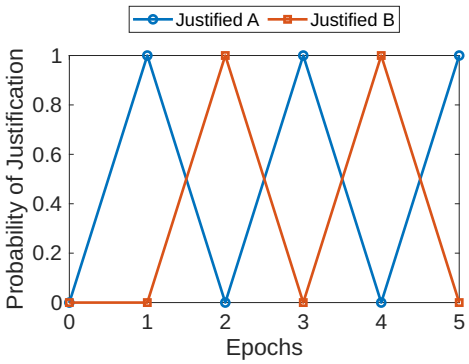

Figure 5.16: Bouncing from Chain A to B.

votes in order to justify one of checkpoint A or checkpoint B. Let's assume A. During the $e = 1$ epoch the validators continued to propose blocks on both chains, given that no checkpoint had been justified before, once again arriving at a situation similar to the previous one. At this point, the task of the dishonest people is to release their votes with a delay in order to justify the checkpoint on chain B. In this way, bouncing from A to B, it will not be possible to finalize any checkpoint. The Figure 5.16 shows this phenomenon of bouncing between two checkpoints. In Prism this situation is modeled using 13 validators, where 4 of them are dishonest and the differences in behavior are shown in Listing 5.2.

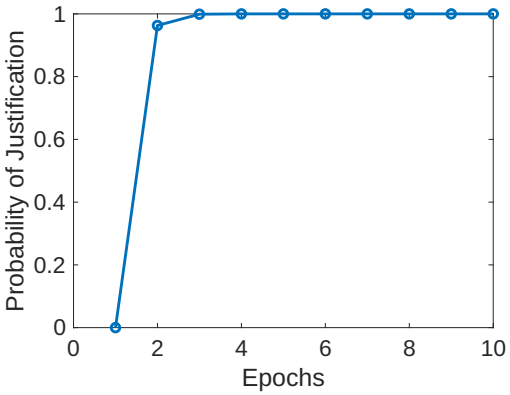

Figure 5.17: Justification rate during Bouncing Attack.

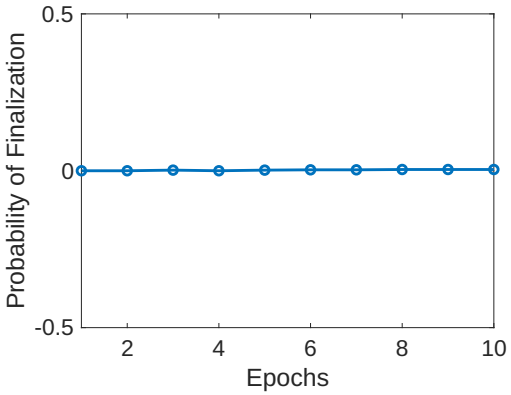

Figure 5.18: Finalization rate during Bouncing Attack.

Prism Code

Modeling the bouncing attack using Prism is only possible if at the beginning of the code execution, two parallel chains A and B are created such that half of the network receives checkpoints of A and the other half receives checkpoints of B. In this attack it is not necessary for validators to receive blocks of both chains, since the goal is to jump from one checkpoint to another, so it is enough to keep track of which chain belongs to them. If the validators received the blocks of both chains, the LMD-ghost algorithm, responsible for resolving the forks, would be able to make one chain preferable over the other, thus avoiding the formation of checkpoints at the same height. In this attack therefore, the first half of validators will always receive blocks of chain A while the other half will receive blocks of chain B. The dishonest validators, however, will receive the blocks of both chains saving them in two separate subsets, in order to preserve separately the checkpoints to be voted with delay to allow the attack.

```
module Validator_a
    M_a_STATE : [Start, Create, ReceiveA, ReceiveB, MoveA, MoveB
        , VoteA, VoteB, Check, Fin, Exit] init Start;
    b_a : block {m_a,0;genesis,0};
    block_a : block {m_a,0;genesis,0};
```

```
checkpointA: bool init false;
checkpointB: bool init false;
justifiedA: bool init false;
justifiedB: bool init false;

[] ((M_a_STATE=ReceiveA) & (checkpointA=false) -> 1 :
    (b_a=extractBlock(set_a)) &
    (M_a_STATE=MoveA);

 [] ((M_a_STATE=ReceiveB) & (checkpointB=false) -> 1 :
    (block_a=extractBlock(setNew_a)) &
    (M_a_STATE=MoveB);

 [extractBlock_a_A] (M_a_STATE=MoveA) & (isCheckpoint(b_a)) & (
    checkpointA=false) -> 1:
    (M_a_STATE'=Start) &
    (checkpointA'=true);

 [extractBlock_a_A] (M_a_STATE=MoveA) & (isHeadBlock(b_a)) ->
    1:
    (M_a_STATE'=Start);

 [extractBlock_a_B] (M_a_STATE=MoveB) & (isCheckpoint(block_a))
    & (checkpointB=false) -> 1:
    (M_a_STATE'=Start) &
    (checkpointB'=true);

 [extractBlock_a_B] (M_a_STATE=MoveB) & (isHeadBlock(block_a))
    -> 1:
    (M_a_STATE'=Start);

[] (checkpointA=true) & (checkpointB=true) -> 1:
        if justifiedA=true :
            (M_a_STATE'=VoteB);
        if justifiedB=true:
            (M_a_STATE'=VoteA);

 [voteBlock_bounceA] (M_a_STATE=VoteA) -> 1:
```

```
            (M_a_STATE'=Start) &
            (checkpointA=false) &
            (checkpointB=false) &
            (justifiedA=true) &
            (justifiedB=false);

    [voteBlock_bounceB] (M_a_STATE=VoteB) -> 1:
            (M_a_STATE'=Start) &
            (checkpointA'=false) &
            (checkpointB'=false) &
            (justifiedB'=true) &
            (justifiedA'=false);
endmodule
```

Listing 5.2: Dishonset validator in Bouncing Attack.

The network module takes care of the diffusion of the blocks created by the validators and during this attack it is also responsible for dividing the network into two subgroups and the consequent formation of two parallel chains.

```
module Network

    set_i : list [];
    setNew_a : list [];
    done: bool init false;

    [addBlock_i] (M_i_STATE=Create) & (done=true) & (isChainA(b_i))
       -> 1:
            for i from 0 to N/2:
                addBlock(set_i, b_i)
            for j from N/2 to N and isDishonest(j):
                addBlock(set_j, b_i)

     [addBlock_i] (M_i_STATE=Create) & (done=true) & (isChainB(b_i))
       -> 1:
            for i from N/2 to N:
                addBlock(setNew_i, b_i)
            for j from 0 to N/2 and isDishonest(j):
```

```
                addBlock(setNew_j, b_i)

[addBlock_i] (M_i_STATE=Create) & (done=false) -> 1:
        (done=true) &
        for i from 0 to N/2:
            addBlock(set_i, b_i)
            if isDishonest(i):
                addBlock(setNew_i, block_i)
        for j from N/2 to N:
            addBlock(setNew_j, block_i)
            if isDishonest(j):
                addBlock(set_j, b_i)
```

endmodule

Listing 5.3: Network module differences during Bouncing Attack.

In Figure 5.17 and Figure 5.18, the rate of justification and finalization within the network is shown. In Prism these experiments are modeled as described in previous sections.

5.7.2 Balancing Attack

The *Balancing Attack*, described in [21] represents a sophisticated threat to the Ethereum Proof of Stake (PoS) consensus protocol's fork choice rule. Its primary objective is to disrupt the ability of honest validators to reach a consensus regarding the canonical blockchain.

The orchestrated attack strategy unfolds in the following sequence:

1. **Initiation**: The attacker commences the assault by deliberately proposing two blocks, fully cognizant of the ensuing penalties they will incur as a consequence of this action. Subsequently, these two blocks are selectively disseminated to two distinct halves of the network. Consequently, both segments of the network proceed to cast their votes in favor of the block they have encountered, perpetuating a state of division within the network.

2. **Resolution**: Once all honest validators within the network have registered their votes, the impasse is effectively resolved through the intervention of validators under the control of the attacker, henceforth referred to as *swayers*. These swayers judiciously channel their attestations exclusively to one side of the network. This orchestrated maneuver ensures the perpetuation of the network's divided state.

To execute the balancing attack precisely, the attacker must fulfill several prerequisites. These prerequisites encompass the necessity to control a validator capable of initiating block proposals at the outset of an epoch, in addition to possessing a minimum of two validators entitled to deliver attestations during each epoch slot, hereinafter referred to as *swayers*. In the event that the total number of validators is an odd number, an additional validator, termed a *filler*, becomes imperative. Subsequently, after the chain undergoes division into two distinct sides, this filler assumes the role of an honest validator on the side inhabited by fewer validators.

Furthermore, the attacker must possess the competence to manipulate the timing of attestation transmissions to other network participants and they must possess a comprehensive understanding of the precise instants when the fork choice rule is used, following the protocol's specifications.

Proposer boost as Synchronization Bottleneck

The Balancing Attack was mitigated in [42] by introducing the *proposer boost*. The solution involves the incorporation of an explicit *synchronization bottleneck* mechanism [42] into the fork choice protocol, by introducing the following rules:

1. In cases where all attesters assigned to slot N collectively hold a total weight denoted as W, any participant occupying slot $N+1$ is mandated to deem attestations as valid only if the attestations were received before the conclusion of slot N from their temporal perspective.

2. The proposer in slot $N+1$ is anticipated to promptly initiate a proposal

at the commencement of slot $N+1$. This proposal implicitly designates a specific blockchain. From the vantage point of attesters in slot $N+1$, if they witness the proposal arriving prior to the one-third mark within the slot's duration, they treat this proposal as equivalent to an attestation carrying a weight of $W/4$. It is essential to underscore that this weight adjustment solely applies to slot $N+1$; subsequent to slot $N+1$, this weight adjustment is rescinded.

This boost was initially designed with a value equal to 25% of the weight of the committees with voting rights in that slot. It was subsequently increased to 30

Synchronization Bottleneck Analysis

The analysis of the proposed patch was done in [42], assuming that all clocks are in perfect synchronization throughout the network.

In each time slot, referred to as N, all validators receive a specific set of attestations. If there's an ongoing attack with at least k (where k is greater than or equal to 1) malicious attesters revealing their attestations during slot N, it's expected that validators may have different opinions about the scores of various blocks. However, it's crucial to note that this difference in opinions is limited, with a maximum limit of k.

When comparing two competing blocks, A and B, block A is considered to prevail if the difference in their scores ($score(A) - score(B)$) is greater than or equal to 0. Block B prevails otherwise. The possible range for disagreement in $score(A) - score(B)$ is restricted to $2k$. In mathematical terms, this means that each validator's interpretation of $score(A) - score(B)$ falls within the range $[z, z+2k]$, where the value of z remains constant.

The proposer, whose weight is denoted as W_p , in the event that he adheres to honest conduct, is bound by two key behaviors:

1. If the proposer observes that $score(A) - score(B) \geq 0$, they are obliged to propose a block for A. Conversely, if $score(B) > score(A)$, they will propose a block for B.

2. The proposer is also responsible for ensuring that their block is promptly presented, thereby guaranteeing that all attesters receive it well before the impending deadline.

With regard to the potential scenarios arising from $score(A) - score(B)$:

1. In cases where $z < -2k$, the proposer votes in favor of B. Consequently, attesters will perceive adjusted scores lying within the range $[z-W_p, z+2k-W_p]$. Notably, this entire range exhibits a negative value, signifying unanimous support for block B.

2. Conversely, if $-2k \leq z < 0$, the proposer casts their vote for A, leading attesters to witness adjusted scores distributed within the range $[z + W_p, z + 2k + W_p]$. In this instance, the entire range assumes positive values, indicating unanimous backing for block A.

3. In the case of scenario (2), the proposer's decision becomes pivotal. Depending on the proposer's personal assessment, either A or B gains their favor. Consequently, the range of disagreement can assume either the form (i) $[z - W_p, z + 2k - W_p]$ or (ii) $[z + W_p, z + 2k + W_p]$.

Reverting back to the scenario where $W_p = W/4$, disrupting the proposer synchronization bottleneck calls for a challenge to the foundational premise that $W_p \geq 2k$. In order to undermine this premise, there must exist a situation where more than $W/4$ attesters reveal themselves during each slot. Should the proposer synchronization bottleneck prove effective during any single slot, all honest attesters will uniformly cast their votes in that direction. This will further accentuate the disparity $score(A) - score(B)$ away from the equilibrium point. To prevent either side from achieving a decisive victory at this stage, the attacker is compelled to disclose a substantial number of attestations. This counteraction is required to offset the combined influence of all honest validators during that slot, accounting for the diminished impact of the proposer's vote as the slot concludes. Achieving this counterbalance would necessitate a substantially higher number of attestations than $W/4$.

Balancing Attack - Testing

The first experiment conducted had the main objective of evaluating the actual need for the implementation of the patch designed for the Balancing Attack, as previously described in the dedicated section.

The attack, as described in [21] requires a number of attackers of at least 6 in each slot, not specifying the total number of validators in the slot. In our case, having singleton committees and 3 committees for each slot, this consideration was not explored in-depth, and a number of dishonest validators $n_d < 1/3N$ were considered where N represents the total number of validators. It is necessary to clarify that some simplifications are used in this experiment given that the goal is to represent the balancing of votes for the blocks and the use of an increment of the duration of a slot regarding the weight of the attestations in that block.

The implementation of this experiment is carried out by setting a priori the behavior of the validators, in the first slot of the first epoch, in order to perform the first split of the network into two different chains. This happens given that half of the validators receive block A and therefore vote for it, while the other half receive chain B and vote for it.

In the simulation, this operation is modeled by modifying the behavior of validator 6 for the first slot of the first epoch. It is he who has the task of *equivocating*, that is, sending the two blocks to the two halves in the e epoch. This behavior is not punished in the simulation, since validator 6 will behave honestly throughout the experiment. Validator 5 will have the task of maintaining the balance of the two chains, keeping the blocks of chain A and the blocks of chain B in two separate subsets, in order to extract and vote, starting from the epoch $e + 1$, the chain which appears to have a lower number of votes than the other. This voting process by Validator 5 simulates the behavior of all the attacking validators, given that the stake of that validator, and consequently the weight of his vote, is 32ETH $\times$ n_d. In the end, it is necessary for the remaining dishonest validators to maintain the balance by releasing their votes as expected in [21]. Attacking validators

can only vote when it is their turn to vote, and to model the late sending of their votes, attackers are allowed to add only the blocks proposed during their voting turn to the two subsets A and B. Therefore preventing them from voting for each proposed block, as required by the protocol.

As illustrated in Figure 5.20, the attack allows dishonest members to hit the vitality of the protocol, affecting the possibility of finalizing new checkpoints. In the graph, it is possible to notice that the rate of finalize a block remains constant and equal to 0 in a period limited to 10 epochs. Figure 5.19 shows how the vote balancing process works, thus maintaining a constant balance between the two chains and consequently not making it possible to reach the supermajority to justify or finalize blocks.

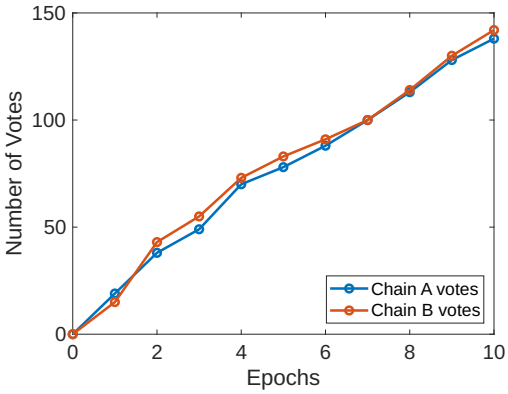

Figure 5.19: Votes during balacing attack.

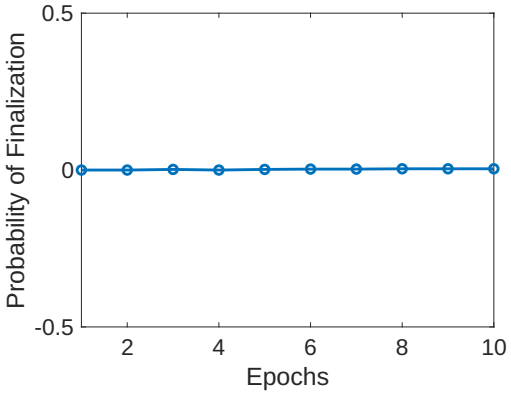

Figure 5.20: Finalization rate during balancing attack.

The results obtained are of great relevance, even if they are obtained from an approximation of the attack. It is clearly demonstrated that the execution of the Balancing attack, in the absence of the patch, has devastating effects on the possibility of reaching the finalization phase. In particular, it is clear that after the implementation of the attack, the blockchain is no longer able to complete the finalization process. To obtain a more in-depth understanding of this situation, it is important to underline that during the course of the experiment, it is observed that the possibility of completing the finalization is drastically eliminated, not reaching the aforementioned state,

even considering more than double the necessary epochs.

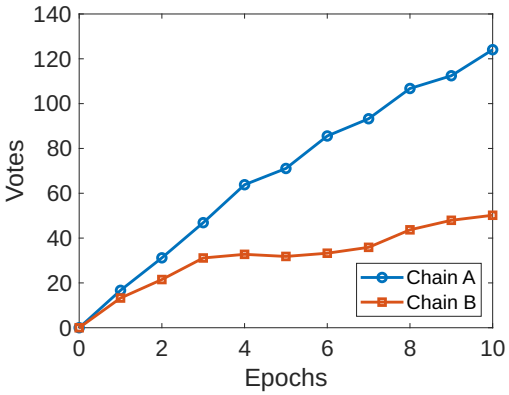

Figure 5.21: Votes with proposer boost.

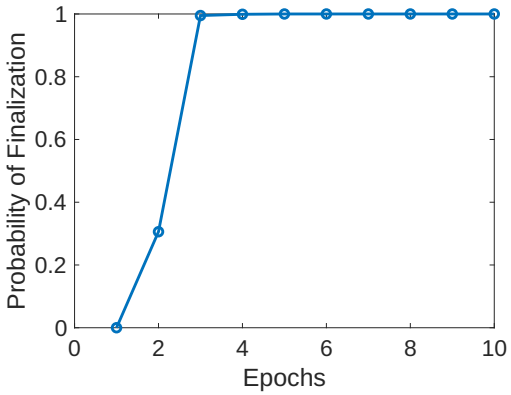

Figure 5.22: Finalization rate with proposer boost.

The results shown in Figure 5.21 and Figure 5.22 represent a significant confirmation of the importance of the patch implemented for the Balancing Attack. The use of proposal boosts is modeled in the simulation by simply sending an additional fictitious attestation for the proposed block to slot s of a value equal to 40% of the total attestations in the block, which will then be removed at slot $s + 1$. This assumption can be considered comparable, given that this attack provides for the release of votes by dishonest validators at the end of each epoch in order to maintain balance. As shown in Figure 5.21, the mechanism implemented to mitigate the attack is essential to guarantee the security and integrity of the finalization process in the blockchain, effectively preventing the implementation of the *balancing attack* and the resulting effects on the blockchain itself.

Prism Code

Modeling the balancing attack using Prism, like the bouncing attack described above, is only possible if, at the beginning of the code execution, two parallel chains A and B are created such that half of the network receives the blocks of A and the other half you receive B blocks.

In this attack, only in the starting epoch at slot 0 is there a separation between block transmission. Therefore only in $e = start$ and $s_e = 0$ is a block of chain A received by the first half of validators, while the second half of validators receives a block of chain B. As for bouncing attack, it is necessary for dishonest validators to receive both chains, in order to release their votes on the blocks belonging to the lighter chain in order to balance it with the heavier one.

Starting from the $s_e + 1$ slot, all validators will receive the blocks created regardless of whether they are from the A or B chain, given that the objective of this attack is precisely to not make the validators understand which is the canonical chain via the balancing of block votes. The votes received by both chains are updated continuously during code execution, via the global module using a Boolean guard that is always true.

In this subsection are illustrated the main code differences implemented for dishonest validators during the balancing attack and how the proposer boost is modeled.

```
module Validator_a
    M_a_STATE : [Start, Create, ReceiveA, ReceiveB, MoveA, MoveB
        , VoteA, VoteB, Check, Fin, Exit] init Start;
    b_a : block {m_a,0;genesis,0};
    block_a : block {m_a,0;genesis,0};
    votesA_a : int init 0;
    votesB_a : int init 0;
    B_a : blockchain [{genesis,0;genesis,0}];
    lastFinalized_a : block {genesis,0;genesis,0};
    lastJustified_a : block {genesis,0;genesis,0};
    lastCheck_a : block {genesis,0;genesis,0};

    [] ((M_a_STATE=ReceiveA) & (!isEmpty(set5)) & (attackStart=
        true) (votesA<votesB) -> 1 :
        (b_a=extractBlock(set_a)) &
        (M_a_STATE=MoveA);

     [] ((M_a_STATE=ReceiveB) & (!isEmpty(setNew5)) & (attackStart
```

```
        =true) & (votesA>votesB) -> 1 :
        (block_a=extractBlock(setNew_a)) &
        (M_a_STATE=MoveB);

    [extractBlock_a_A] (M_a_STATE=MoveA) & (voteID=ID_a) -> 1:
        (M_a_STATE'=VoteA);

    [extractBlock_a_B] (M_a_STATE=MoveB) & (voteID=ID_a) -> 1:
        (M_a_STATE'=VoteB);

    [voteBlock_A] (M_a_STATE=VoteA) -> 1:
            (M_a_STATE'=Start) &
            (votesA=votesA+(voteWeight));

    [voteBlock_B] (M_a_STATE=VoteB) -> 1:
            (M_a_STATE'=Start) &
            (votesB=votesB+(voteWeight));
endmodule
```

Listing 5.4: Dishonset validator in Balacing Attack.

The Listing 5.5 instead re-proposes the differences necessary to start the attack, i.e. dividing the network with two different views. During the first slot of the starting epoch, the validators are split into two half, the first half receive a block that will create chain A, while the second half receive a block that will create chain B. Even in this slot, the dishonest ones use save the blocks into two different subsets, in order to perform the attack.

After the first slot of the starting epoch, all the validators will receive all the proposed, making no distinction between chains A and B. The only distinction made concerns the dishonest ones, who will have to continue saving the blocks of chain B in a different subset compared to that of chain A. Therefore, depending on the proposer of that slot, they will choose which set to save the newly created block.

The ability to propose new blocks for dishonest validators after the starting condition has been removed, as there is no penalty for not proposing a block, and in order to simplify the network.

```
module Network

    set_i : list [];
    setNew_a : list [];
    done: bool init false;

    [addBlock_i] (M_i_STATE=Create) & (done=true) & (isChainA(b_i)
        -> 1:
            for i from 0 to N:
                addBlock(set_i, b_i);

    [addBlock_i] (M_i_STATE=Create) & (done=true) & (isChainB(b_i)
        -> 1:
            for i from 0 to N:
                if dishonest(i):
                    addBlock(setNew_a, block_i);
                else:
                    addBlock(set_i, b_i);

    [addBlock_i] (M_i_STATE=Create) & (done=false) -> 1:
            (done=true) &
            if i in (0, N/2):
                for j from 0 to N:
                     addBlock(set_i, b_i)
            if i in (N/2, N):
                for j from 0 to N:
                    if dishonest(j):
                        addBlock(setNew_j, b_i)
                    else:
                        addBlock(set_j, b_i);
endmodule
```

Listing 5.5: Network module differences during Balancing Attack.

In Listing 5.6 it is possible to see the changes made to the global module in order to always maintain an updated value of the weights of the two chains.

```
module Global
```

```
[] (true) -> 1:
      (votesA'=
          for i from 0 to N/2:
              +votesA_i) &
       (votesB'=
          for j from N/2 to N:
              +votesB_j));
endmodule
```

Listing 5.6: Global module differences during Balancing Attack.

The Listing 5.7 shows the differences in the code of an honest validator that uses the proposer boost when proposing a block. The weight is equal to 40% of the weight of the voters' attestations of that slot. This boost is removed from the overall weight of the chain after 1 temporal slot (12s).

```
module validator_i

    boostWeight: int init 0;
    boost_i: bool init false;
    votesA_i: int init 0;

    [] (M_i_STATE=Start)&(validatorID=ID_i) -> 1 :
            (b_i'=createBlock()) &
            (M_i_STATE'=Create) &
            boostWeight=(
                (for j in SlotVoters:
                    +weight(j)
                )
                × 40/100);

    [addBlock_i] (M_i_STATE=Create) -> delay :
          (B_i=addBlock(B_i,b_i)) &
          (M_i_STATE'=Start) &
          (boost_i=true) &
          (votesA_i'=votesA_i + weight(i) + boostWeight);

    [] (boost_i=true) -> slot :
            (boost_i=false) &
```

(votesA$_i$=votesA$_i$−boostWeight);

endmodule

Listing 5.7: Proposer Boost in honest validators. We assume that honest validators follow chain A.

5.7.3 Balancing Attack over LMD-Ghost

This attack was presented in [39], and is considered an improved balancing attack, which takes into account proposer boosting and exploits the ability to send equivocating votes.

The Longest Message-Driven (LMD) rule endows the adversary with a formidable capability to execute a balancing attack. Once the adversary establishes two competing chains, it gains the ability to equivocate on them. This involves strategically timing the release of equivocating votes so that the vote for the Left chain is received by half of the honest validators first, while the vote for the Right chain is received by the other half of the honest validators first. This intentional sequencing creates a divergence in the views of honest validators regarding the 'latest messages' from adversarial validators.

Despite all validators eventually receiving both sets of votes, the LMD rule ensures that the split view endures for a substantial duration. This persistence arises due to the absence of subsequent votes from adversarial validators for later slots. Consequently, half of the honest validators perceive the Left chain as leading and cast their votes accordingly, while the other half perceives the Right chain as leading and votes in favor of it. The inherent split among honest validators, approximately evenly divided, results in a balanced distribution of votes, and each faction continues to regard its respective chain as leading.

It is noteworthy that this effect is exceptionally pronounced and challenging to overcome through proposer boosting alone. Proposer boosting could potentially counteract the adversarial influence only if the proposal weight

significantly exceeds the adversary's equivocating votesârepresenting a fraction of the committee sizeâby a substantial constant factor. Failure to achieve such a substantial margin means that, in instances where the adversary leads by the constant factor number of slots, it can surpass the proposer boost.

An illustrated example is in Appendix A.

5.7.4 Time-Based Attacks

Time plays a pivotal role in orchestrating human activities and facilitating the development of distributed systems. Clock-driven algorithms, as elucidated in [36], offer substantial simplification in the realm of fault-tolerant distributed systems. In practice, numerous blockchains explicitly leverage the concept of time for the synchronization and coordination of participants, as exemplified in [37]. This temporal dimension serves as a fundamental cornerstone in the architecture of such systems, enhancing their efficiency and reliability.

Currently, Ethereum has no mechanisms for synchronizing the validators' clocks, although to function correctly it requires almost total synchrony, given that each operation can be carried out in specific time windows. The task of synchronization is left entirely to the individual validators. The most used and widespread technique regarding time synchronization concerns the use of Network Time Protocol (NTP). Over the years, numerous attacks have been carried out on this protocol, such as [51] and [52], which have demonstrated the possibility of slowing down or speeding up users' clocks. Assuming that a good part of the validators use the same server to sync, it is possible with a single attack to affect a large number of users.

The time-based attack [30] on Ethereum's Proof of Stake (PoS) system is a clever exploit that takes advantage of the requirement for validators to keep their clocks synchronized with the network's time.

This time-based attack can be used for various malicious purposes, such as double-spending funds or disrupting applications that rely on the blockchain's unchangeable history. Here's how:

1. **Double-Spending**: To double-spend, the attacker proposes a block containing a transaction that spends the same funds as one on the main blockchain. Then, they set their clock forward and propose another block with the same transaction. The network accepts this second block because it appears valid, even though it's in the past.

2. **Application Attacks**: In the case of applications relying on the blockchain's unchanging history, the attacker proposes a block that alters the application's state. The network accepts this block because it appears valid, despite being in the past.

The primary objective of this thesis is to analyze the Gasper protocol, therefore an accurate analysis of the problem will be carried out according to the specifications of the current Ethereum protocol. However, it is essential to underscore that the findings presented here and in [38] possess broader applicability, extending to other protocols grounded in Casper FFG with inactivity leakage, provided that a time attack can yield similar ramifications.

The core characteristic of the protocol under examination pertains to its handling of validator clocks. Specifically, when a validator's clock exhibits a noticeable delay compared to its peers, surpassing a predefined threshold, the protocol mandates the disregard of its attestations by others. Consequently, the potential exists for an attacker to exploit this mechanism with the aim of decelerating a specific validator's clock, resulting in its isolation from nodes with synchronized clocks. Importantly, this isolation remains unidirectional, with the sluggish validator retaining the ability to observe messages emanating from faster counterparts.

Regarding Gasper, some additional information is considered:

1. The protocol is grounded in the principles of Casper FFG and incorporates an inactivity leakage mechanism, progressively penalizing inactive participants.

2. The protocol enforces an upper limit on message delay, with any violation triggering the categorization of the sender as inactive, thereby

invoking the inactivity leakage mechanism.

The time attack strategy lies in its capability to partially segregate specific participants within the broader network. This partial isolation implies that fast participants find themselves unable to access messages originating from slower counterparts, while the slower participants maintain access to messages from their faster counterparts, albeit receiving them prematurely.

Time Attack exploiting Inactivity Leakage

This attack, also known as *Clock Attack*, is based on observations made in [38] and is hypothesized that it is possible for attackers to gain control of a specific number of validators. An attacker can target any validator that uses a specific and common synchronization protocol and then perform one of these operations:

- Control of correct synchronized validator clock.
- Control of a validator clock when it's set up incorrectly.

It is assumed that the clock control option over already incorrect clocks incurs significantly lower costs since it would be necessary to slow down that validator for less time before making it inactive. Essentially, each erroneously configured clock contributes to reducing the overall cost of a successful attack on the protocol.

Designations are employed using letters A and H to delineate two distinct sets of validators within the system: the attackers (Attacckers), and the validators with honest conduct (Honest). The encompassing set of all nodes is denoted as N, and it naturally follows that $A \cup H = N$.

The prevailing assumption is that the attackers lack the capability to exert complete control over a majority of validators. However, this analysis serves to elucidate how the absence of a synchronization rule or protocol in Ethereum can enable the attackers to compromise the protocol's safety or liveness while minimizing budgetary outlays.

Two possible cases are considered:

- Adversarial Majority case: $|N|/2 < |A| < 2|N|/3$. In this case, the attackers control the majority of validators, but they can not justify or finalize due they do not have 2/3 of the total stake.

  In this scenario, the adversary can slow down or speed up the clocks of honest validators. Thus, messages from H will be ignored by A validators, and correct validators will lose their balances due to the inactivity leak. As A constitutes the majority, but not the supermajority, they can break liveness but cannot justify/finalize epochs. However, as correct validators lose their balances, at some point in time, the adversary will be able to justify and finalize epochs.

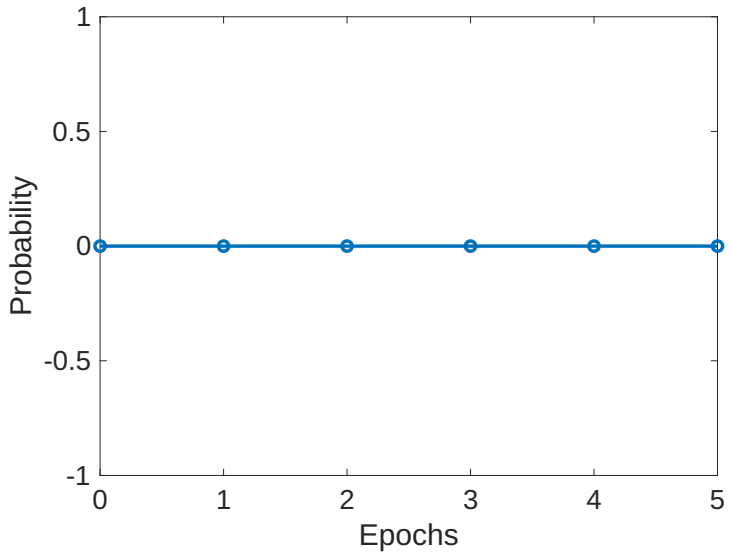

Figure 5.23: Honest Validators' Justification Rate during Clock Attack.

In Figure 5.23, one can observe the probability of justification among honest validators in the context of a clock attack. It is noteworthy that network operations occur within fixed time intervals, rendering any form of network delay a substantial concern. Such delays precipitate an inactivity leakage phenomenon [18], leading to the loss of a significant part of the total stake of a slow validator starting from the fourth epoch without justified blocks.

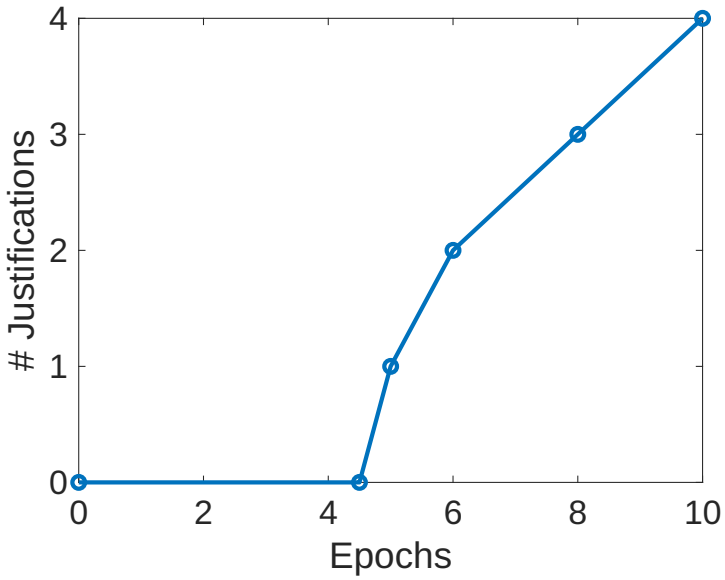

Figure 5.24: Attackers Justification Rate during Clock Attack.

After eliminating H, the adversary gains full control over the network. In Figure 5.24, an observable trend emerges wherein the rate of justification experiences an incremental rise as the inactivity leak mechanism unfolds its impact. Initially, this mechanism imposes penalties on validators with notably slower operational clocks within the network, categorizing them as *inactive*. Consequently, this categorization ultimately paves the way for malicious validators to secure the checkpoint. It is imperative to note that the depicted attack simulation has been accelerated for illustrative purposes, compressing the effect within a reduced number of epochs. This acceleration involves the application of a penalty equal to 5 ETH for the inactivity.

- Attackers Minority Case: If the condition holds that $|A| < |N|/2$, and the adversary does not have control over most of the validators' clocks, it is possible to perform a time attack to break the liveness, without taking control of the network. Whereas $|A| + |H| = |N|$, it logically follows that $|H| > |N|/2$. The vitality violation condition implies that $|A| > |H|/2$, which can be further expressed as $|A| > |N|/4$. To

violate viability without initially eliminating the opponent would have to attack more than $|N|/3$ validators.

Exploiting vulnerabilities related to inactivity leakage is not deemed a highly practical endeavor [38], primarily due to the extended timeframe required for an inactive balance to reach critically low levels, a process characterized by logarithmic progression. Consequently, such an attack is susceptible to detection by vigilant administrators overseeing the system's operations.

Nonetheless, the primary objective of this analytical examination is to underscore the considerable efficiency gained through the manipulation of validators' clock settings as a pivotal component within a sophisticated attack strategy. This efficiency is particularly noteworthy given the potential to isolate a substantial number of validators from the broader network, assuming these validators have erroneously configured their clocks, rendering them susceptible to attacks leveraging Network Time Protocol (NTP) vulnerabilities.

Prism Code

This section explains how this attack is implemented and tested in the Prism simulation. This attack requires attackers to have basic knowledge about the network protocols used by victims and additional knowledge to break them, which cannot be modeled in Prism. It is therefore assumed that the operations have already been carried out and the attackers are already able to attack the victims' clocks.

The 5.8 listing shows the code differences regarding the implementation of an honest validator as well as a time attack victim.

```
module validator_i

  M_i_STATE : [Start, Create, Receive, Move, Vote, Check, Fin]
      init Start;
   b_i : block{m_i ,0;genesis ,0} ;
   B_i : blockchain [{genesis ,0;genesis ,0}];
```

```
[] (M_i_STATE=Start)&(validatorID=ID_i) -> 1 :
    (b_i=createBlock()) &
    (M_i_STATE=Create);

[] (M_i_STATE=Start)&(voter_i=ID_i) -> 1 :
    (M_i_STATE=Vote);

[] (M_i_STATE=Start)&!ListIsEmpty()) -> rC :
    (M_i_STATE=Check);

 [] (M_i_STATE=Start) -> delayTimeAttack :
    (M0_STATE=Receive);

[addBlock] (M_i_STATE=Create) -> delayTimeAttack :
    (B_i'=addBlock());

endmodule
```

Listing 5.8: Implementation of delayed validator.

In the following Listing, the expected delay for this time attack is set a priori, in any case higher than the inclusion delay which is equal to 1 epoch. Considering that this value represents a probability, it could happen that sometimes the attacked validator manages to send the block in time. This aspect has not been considered within the code assuming that at each epoch the attacked validators are unable to send their attestations, receiving a penalty each epoch as shown in the Listing 5.10 of 1 ETH. This penalty is significantly higher than that initially predicted by the inactivity leak, but it is necessary to be able to simulate this attack in less time.

```
module Global

  const double delayTimeAttack = 1/(epoch + inclusionDelay);

  for i in victimValidators:
        exited_i : bool init false;
```

```
endmodule
```

Listing 5.9: Delay and delay and control variables for the activity of victim validators implemented in Global module.

Considering 13 validators in the simulation and that each slot gives the right to vote to 3 different validators, the number of times in which a validator is chosen as a voter of the slot in the model is greater than 1 for each epoch as required by the protocol, given the reduced number of validators present. In this case, a validator can be elected as a voter of the slot with probability $\frac{3}{12}$, given that the voters are 3 and the validators, excluding the proposer, are 12. By adding the probability of $\frac{1}{4}$ for 32 slots of the epoch, we obtain 8 times per epoch for a single validator.

```
module Updater

    [] (TimeAttack=true) & (stake_i>=16) -> 8/384 :
        (stake_i=stake_i-1) &
        (totalStake=totaltake-1);

    [] (TimeAttack=true) & (stake_i<16) & (exited_i=false) -> 1 :
        (stake_i=0) &
        (totalStake=totalStake-5) &
        (exited_i=true);

endmodule
```

Listing 5.10: Penalty implemented for votes with delay.

5.8 Idea of a Hybrid Attack

This new type of attack called the *Hybrid Attack*, is designed to exploit the protocol's vulnerabilities to time-based attacks, as described in Section 5.7.4, with the aim of reducing the capital required for execution, by exploiting a second type of attack, known as a *Balancing* attack and described in 5.7.2.

The motivations behind this new type of attack mainly derive from the limited ability of the network to resist possible time-based attacks, even if such attacks could be impractical in implementation, requiring considerable investments in terms of time and financial resources.

The idea for this attack was conceived by analyzing the average rewards received by a validator who locks up 32 ETH in their stake, which is the minimum deposit amount needed to become a validator and is also the limit value to become a block proposer.

In this analysis, the average daily income for a validator was determined to be 0.0035 ETH, equating to a daily return of 0.01% [50], as shown in Table 5.1.

Duration	ETH Stake	ETH Reward	Return %
Day	32 ETH	0.0035 ETH	0.01 %
Week	32 ETH	0.0242 ETH	0.08 %
Month	32 ETH	0.1074 ETH	0.34 %
Year	33.2 ETH	1.2641 ETH	3.95 %

Table 5.1: Average validator reward.

Subsequently, a more detailed analysis of the functioning of inactivity leakage was conducted, considering the formulas declared in 3.1.5. In the previous experiment, although its actual operation was accelerated for experimental purposes, this phenomenon was studied until the targeted validators were expelled from the network, which took approximately three weeks [49]. The new approach aims not to fully exploit inactivity but to significantly reduce a validator's ability to influence block voting and achieve supermajority.

This is achievable considering the daily gain of 0.0035 ETH and the fact that utilizing inactivity within a single day could result in the same validator losing 0.76 ETH, as shown in Table 5.2. Therefore, a fluctuating time-based attack scheme was devised, targeting only $\frac{1}{3}$ of the network's validators, instead of $\frac{1}{3} < n < \frac{N}{2}$ previously adopted in Clock Attack 5.7.4. This approach

allows for substantial ETH savings and makes it harder to detect, as it does not involve continuous attacks over time.

Days	Epochs	Lost Stake	Percentage %
<1	10	0.0001	<0.01%
<1	20	0.0004	<0.01%
<1	30	0.0009	<0.01%
21	4686	16	50%

Table 5.2: Percentage of stake lost due to the Inactivity Leak based on 32ETH staked. This data is an approximation and it may not entirely match the actual penalties.

By combining this non-continuous attack with a balancing attack on one-third of the validators, it is still possible to target 50% of the validators, as intended by the balancing attack. However, by exploiting the time-based attack, the aim is to exclude a part of the honest validators in order to balance the other part with the dishonest validators responsible for the balancing. This is in line with the patch, given that in this attack no equivocating votes are sent, but only delayed attestations, which in any case are not subject to penalties within the limits of the inclusion delay (1 epoch).

In this experiment the attack is started by postponing the start to the instant in which the first block is finalized, thus simulating the operation of an already active network. This simulation involves both the implementation of the patch required to balance the attack and the decision not to completely eliminate the portion of nodes targeted by the time attack. This is because the goal is not to remove them, but rather to equate their contribution to that of the dishonest validators involved in the balancing attack. As highlighted in Figure 5.25, once the initial condition for the attack is satisfied, i.e. the first finalization, the number of finalized blocks does not increase further. This is because, even if the number of dishonest nodes does not respect the 51% expected for a balancing attack, using it in combination with a *partial* time attack, it is possible to reach a deadlock situation, where the supermajority

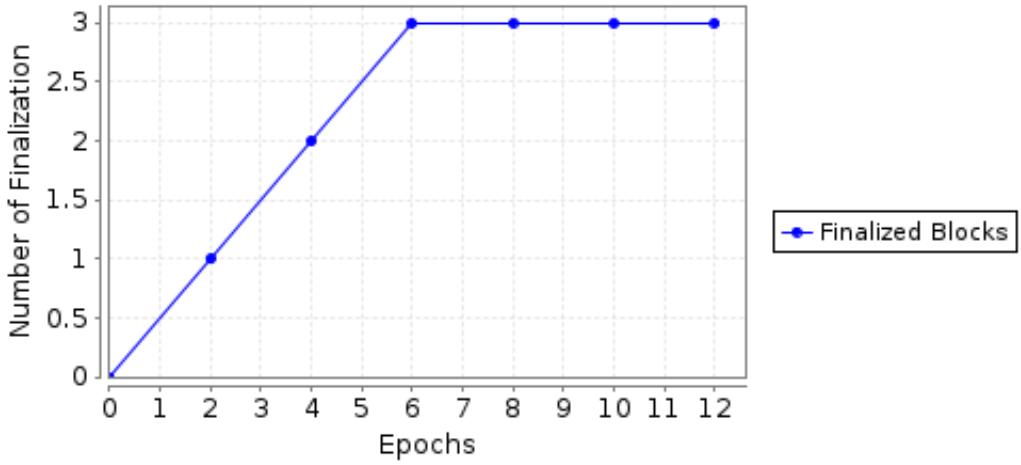

Figure 5.25: Number of Finalized Blocks - Hybrid Attack.

is not reached.

By partial time-based attack, we can mean an attack aimed at excluding some honest validators, randomly chosen in rotation, only for a limited period of time, in order to equate the honest validators to the dishonest ones. The choice to carry out a time attack only for short periods is due to the analysis of the rewards carried out previously, given that to get out of this deadlock, the victim validator must wait for the rewards to be recognized for his honest work, such as the timely creation of blocks and the timely provision of attestations, before seeing its original value or weight re-established.

A visual example of this attack is shown in Appendix B.

Chapter 6

Related Works

Several studies have put forth simulation-driven methods to estimate various metrics of blockchain-centric systems, assess the effects of altering parameters, and gauge the repercussions of potential attacks.

The experiments and analyses presented in this thesis build upon the foundation laid in the initial work conducted by *Veschetti et al.* [1]. In their preliminary research, they embarked on the development of a model capable of simulating the hybrid Ethereum consensus mechanism, encompassing both PoS and PoW elements. Notably, their model still incorporated mining activities to select validators responsible for executing blocks added to the blockchain. The significance of their previous experiment was underscored by its ability to demonstrate the model's consistency, as evidenced by comparing the expected outcomes of fork, finalization, and justification rates.

The next phase of the analysis involved a comprehensive examination of the distinctions between the Proof of Work (PoW) and Proof of Stake (PoS) consensus mechanisms. This scrutiny aimed to discern the primary rationale behind the selection of one over the other, taking into account the inherent advantages and disadvantages of both approaches. As elucidated in the study presented in [46], PoW boasts several notable advantages. Firstly, it facilitates swift consensus attainment due to the complexity of solving cryptographic puzzles, which, despite their simplicity to verify, necessitate signifi-

cant computational effort. This results in rapid consensus through block validation. Secondly, PoW holds historical significance as the oldest consensus mechanism, serving as the foundational choice for the first cryptocurrency, despite facing stability and security challenges. Lastly, PoW acts as a deterrent to spam, as the computational resource demands associated with sending an email dissuade spammers. Insufficient computational resources and high computation costs act as formidable barriers, discouraging the mass transmission of unsolicited emails, even when the spammer possesses the requisite resources. However, PoW is not without its disadvantages. Smaller networks experience diminished security, as the risk of hackers producing fraudulent blocks increases with the relative ease of acquiring network resources. Additionally, PoW necessitates substantial electricity consumption and resource usage. Despite only one miner being required to mine a block, extensive electricity and network resources are squandered, and miners with access to cost-effective electricity often monopolize the mining process, exacerbating energy waste. Lastly, PoW's reliance on electricity and mining technology has driven it toward centralization, with miners tending to cluster in regions with affordable electricity and abundant mining resources, posing data security risks. In contrast, PoS offers compelling advantages. Firstly, it mitigates centralization concerns by not demanding extensive resources and electricity, making it more eco-friendly and fostering decentralization. Secondly, its uncomplicated architecture minimizes resource requirements, resulting in a smaller environmental footprint. Lastly, PoS drastically reduces electricity consumption, obviating the need for energy-intensive algorithms, thereby benefiting the environment. Nonetheless, PoS is not without its drawbacks. Users with a substantial coin stake possess the potential to exert influence over the network through the PoS protocol. Additionally, PoS necessitates miners to invest their stake in the network to mine a new block. It is noteworthy that the threat of network manipulation by entities with significant stakes is a more pronounced concern in PoS, a consideration that has been extensively addressed within Ethereum's PoS protocol, Gasper.

The research continued to delve deeper, commencing with an in-depth examination of the official GASPER protocol [13]. This phase aimed to provide further theoretical insights into the protocol, primarily to showcase the network's enhanced efficiency with the incorporation of GASPER. Additionally, an experimental perspective is pursued to simulate the network's functioning under the new protocol. To acquire empirical data, the necessity arose to construct a network simulation adhering to the protocol's stipulated specifications. The simulation, tailored for deployment within the Prism model checker framework, represents an extension of the simulation previously established in [1]. This extension adheres to the foundational principles outlined in [45].

It is worth noting the significance of delving into stake analysis, as meticulously conducted in the study presented by [1]. This analytical endeavor holds a pivotal role in deciphering the intricate reward and penalty system delineated by the protocol. The stake analysis serves as a fundamental tool in unraveling the intricacies of how rewards and penalties are structured and applied within the framework of the protocol, offering invaluable insights into the underlying mechanics of the blockchain ecosystem. Through this comprehensive examination, a deeper understanding of the protocol's mechanisms and their impact on network participants can be gleaned, facilitating informed decision-making and policy formulation.

Additional experiments are conducted to assess the security and resilience of the network within this new framework. To inform the research, some of the analyses conducted by *Pavloff et al.*, as documented in [14], are considered and replicated. Notably, this work included the reproduction of their *Bouncing Attack*, which, back in 2021, posed a significant threat to the network, even though initial patch proposals had been put forth.

In the same year, the publication by *Hunseler et al.* [20] featured one of the initial simulations conducted on the Ethereum Beacon Chain. This endeavor aimed to validate the heightened security measures implemented by the new protocol, ensuring optimal efficiency and scalability. Within this con-

text, two novel types of attacks were introduced: the *Time-based Attack* [38] and the *Balancing Attack*[39]. The first type of attack is formally described and subsequently subjected to experimental testing. The second type of attack is subjected first to a theoretical examination and next to experimental testing, by considering some approximations, due to the formidable challenges associated with reproducing such conditions using a model-checking tool. This type of attack has catalyzed extensive research endeavors aimed at enhancing the LMD-GHOST fork algorithm, culminating in the conceptualization of a potential remedy known as RLMD-GHOST, or Recent Last Message Driven GHOST, as documented in [44], in the year 2023. This novel algorithm not only ensures dynamic availability but also upholds safety levels during periods characterized by bounded asynchrony. It is pertinent to note that these findings are yet to undergo empirical validation; however, they lay the groundwork for innovative solutions and considerations pertaining to network security.

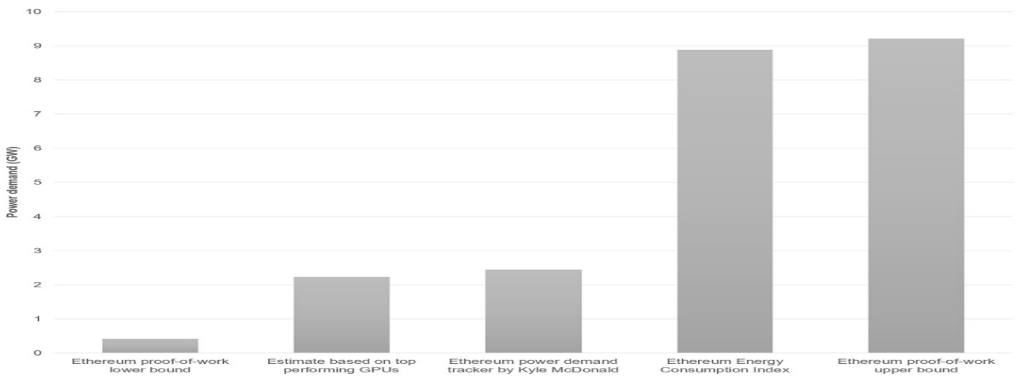

Figure 6.1: Power consumption of Ethereum PoW in different analyses. Figure from [24].

Furthermore, it's essential to emphasize that the research extended its focus beyond the technical aspects of protocol transitions. We also took into account the broader implications of transitioning from PoW to PoS, including the significant environmental benefits associated with such a shift, introduced in [46], also providing reasons that push towards the use of this

new consensus mechanism. Notably, a pivotal study highlighted in [23] and the analyses conducted in [47] underscore the environmental advantages of this transition.

It has been reported in Figure 6.1 that the Ethereum network has achieved a remarkable 99.95% reduction in energy consumption, showcasing a remarkable leap towards sustainability [24]. It is approximately 50,000 times lower than the energy consumption of Bitcoin [41]. The energy consumption values were also compared with additional technologies, in order to shed new light on Ethereum 6.2.

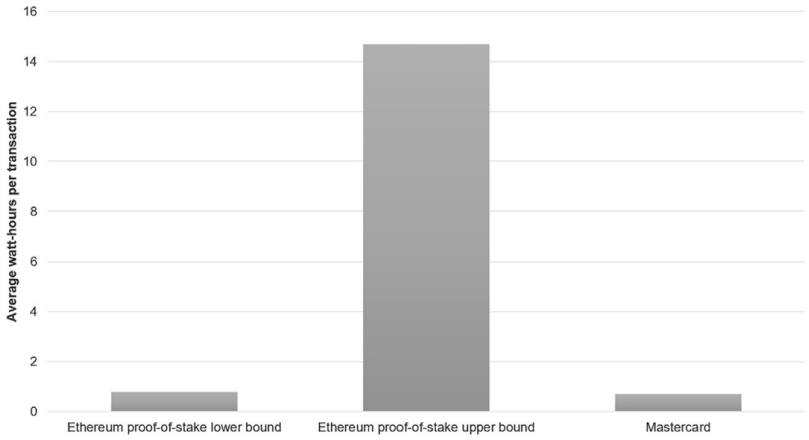

Figure 6.2: Power consumption of Ethereum PoS, lower and upper bounds, compared with the MasterCard payment circuit. Figure from [24].

This substantial reduction in energy usage not only aligns with the global drive towards eco-friendliness but also positions the Ethereum network as a responsible and sustainable blockchain platform. Our exploration and analysis of these environmental aspects further enrich the comprehensive perspective presented in this thesis, shedding light on the holistic impact of protocol transitions on the blockchain ecosystem.

Conclusions

This master's thesis represents a comprehensive exploration of Ethereum blockchain technology and the significant transition from Proof of Work (PoW) to Proof of Stake (PoS) through the Gasper protocol. Additionally, it delves into the event known as *The Merge*, which marked a significant milestone in Ethereum's evolution, starting in September 2022.

The analysis commenced with a thorough examination of Ethereum's foundational principles, an exploration of blockchain fundamentals, and a deliberate assessment of the motivations behind the transition to PoS, taking into consideration environmental and operational factors. To provide a new perspective, it is also conducted a comparative analysis, contrasting the Hybrid Casper and Gasper protocols. The introduction of the Prism Model Checker shed light on its intricate functionality, elucidating illustrative use cases and its real-world applicability, thereby setting the stage for our investigation. A crucial aspect of the research was an extensive survey of related works, offering the necessary context to appreciate the significance of our study within the broader landscape of blockchain research.

Transitioning from theory to practice, the architecture of the Gasper protocol is scrutinized paying particular attention to its implementation using Continuous-Time Markov Chains (CTMC) within a modified version of the Prism Model Checker software, called *Prism+*. The practical implementation within Prism+ yielded invaluable insights, offering pseudocode and comprehensive module descriptions that covered key components via ad-hoc modules, such as Validator, Updater, Network, Global, and RanDAO. These

descriptions significantly enhanced the understanding of the experiments and the protocol in general.

The findings derived from experimental simulations affirmed the faithfulness of this simulation to Gasper's specifications. Finalization, justification, and block creation rates are computed providing empirical evidence of the protocol's robustness. Furthermore, our exploration of fork probability, guided by the LMD Ghost algorithm, underscored Gasper's resilience in the face of prolonged forks.

One noteworthy observation that emerged from our investigation is the Gasper protocol's ability to incentivize honest validators, encouraging gradual stake growth over time. However, it is also uncovered that this phenomenon is most prominent in a network environment free from significant latency. Delays incurred by validators are met with equivalent penalties, effectively tempering potential gains. Consequently, the growth of stakes among honest validators followed a more incremental trajectory rather than experiencing an exponential ascent, regardless of their ETH holdings.

Figure 6.3: ETH Capitalization.

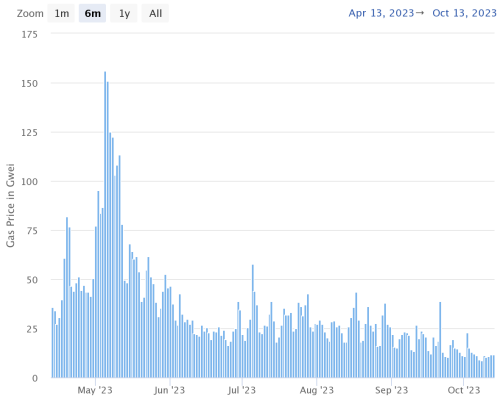

Figure 6.4: Average Gas price.

In recent months, approximately one year after *The Merge*, which marked the transition from a Proof of Work (PoW) to a Proof of Stake (PoS) consensus mechanism for the Ethereum network, there has been a growing consideration regarding the hypothesis of currency devaluation. This phenomenon is evident by analyzing the price trend of ETH, which decreased from around

$2,000 in April 2023 to approximately $1,500 in October 2023, as depicted in Figure 6.3. Another element supporting this theory is the behavior of the average Gas price, which decreased from an average value of 35 Gwei in April 2023 to approximately 11 Gwei in October 2023 (see Figure 6.4). It is worth noting that due to changes introduced in the validation process, which now takes into account various factors beyond computing power (PoW), such as the punctuality of certificate submission and block proposal, an expectation of increased earnings for individual validators has arisen.

This increase in rewards for validators may inevitably lead to an overall increase in ETH in circulation, resulting in a decrease in the currency's value. This decrease could have several effects, including reduced interest and participation by individuals, who represent the blockchain's final line of defense, as declared on the official Ethereum documentation [48].

Additionally, the decrease in value may increase the ease of carrying out various types of attacks, including those described in this thesis. While these attacks require substantial economic resources, they may become more accessible due to the reduced value of the ETH required to execute them. In summary, the transition from PoW to PoS has significantly reduced ETH emissions but has also introduced potential challenges related to the currency's value and blockchain security.

Future research endeavors will be dedicated to the exploration and evaluation of novel attack methodologies. These investigations aim to comprehensively assess the evolving threat landscape and identify potential vulnerabilities emerging in blockchain systems. Upcoming studies will center on developing and analyzing innovative reward and penalty structures. The goal is to avoid flooding the market with an excessive supply of cryptocurrency, thus preserving the currency's value and overall stability. This pursuit entails the crafting of mechanisms that strike an intricate balance between incentivizing participation and validating transactions while discouraging excessive accumulation or dilution of digital assets.

Bibliography

[1] Galletta, Laneve, Mercanti, and Veschetti, *Resilience of Hybrid Casper Under Varying Values of Parameters*. ACM, 2023.

[2] Buterin, *Ethereum white paper*. GitHub repository 1, 2013.

[3] Hildenbrandt, Manasvi, Zhu, Rodrigues, Philip, Guth, Rosu, *KEVM: A Complete Semantics of the Ethereum Virtual Machine*. Illinois Library, 2017.

[4] Ethereum Documentation, *Proof of Stakes*. `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/`, 2022.

[5] Ethereum Documentation, *Blocks*. `https://ethereum.org/en/developers/docs/blocks/`, 2023.

[6] Yaga, Dylan, et al., *Blockchain technology overview*. arXiv preprint, 2019.

[7] Ethereum Documentation, *Intro to Ethereum*. `https://ethereum.org/en/developers/docs/intro-to-ethereum/`, 2023.

[8] Ethereum Documentation, *Intro to Ether*. `https://ethereum.org/en/developers/docs/intro-to-ether/`, 2023.

[9] Ethereum Documentation, *Transactions*. `https://ethereum.org/en/developers/docs/transactions/`, 2023.

[10] Ethereum Documentation, *Smart Contracts*. https://ethereum.org/en/developers/docs/smart-contracts/, 2022.

[11] Solidity Documentation, *Solidity*, https://docs.soliditylang.org/en/v0.8.21/. 2023.

[12] Ethereum Whitepaper, *Intro to Smart Contracts*. https://ethereum.org/en/whitepaper/#introduction-to-smart-contracts, 2023.

[13] Buterin, et al. , *Combining GHOST and casper*. arXiv preprint, 2020.

[14] Pavloff, Ulysse, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni, *Ethereum Proof-of-Stake under Scrutiny*. Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. 2023.

[15] Ethereum Documentation, *Blocks*. https://ethereum.org/en/developers/docs/blocks/, 2023.

[16] Buterin et al., *Incentives in Ethereum's hybrid Casper protocol*. International Journal of Network Management, 2020.

[17] Ethereum Documentation, *Attestations*. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attestations/, 2023.

[18] Buterin, Griffith, *Casper the friendly finality gadget*. arXiv preprint, 2017.

[19] Ethereum Documentation, *Gasper*. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/gasper/, 2023.

[20] Hunseler and Lemke-Rust, *Simulating an Ethereum 2.0 Beacon Chain Network*. IEEE, 2021.

[21] Neu, Nusret Tas, Tse, *Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma*. CoRR, 2020.

[22] Cachin, Vukolic, *Blockchain Consensus Protocols in the Wild*. CoRR, 2017.

[23] Lal, Apoorv, and Fengqi You, *Climate concerns and the future of nonfungible tokens: Leveraging environmental benefits of the Ethereum Merge*. Proceedings of the National Academy of Sciences, 2023.

[24] De Vries, *Cryptocurrencies on the road to sustainability: Ethereum paving the way for Bitcoin*. Patterns 4.1, 2023.

[25] Kapengut, Elie, and Bruce Mizrach, *An event study of the ethereum transition to proof-of-stake*. Commodities 2.2, 2023.

[26] Aziz, Adnan, et al, *Verifying continuous time Markov chains*. Computer Aided Verification: 8th International Conference Springer Berlin Heidelberg, 1996.

[27] Aziz, Adnan, et al, *Model-checking continuous-time Markov chains*. ACM Transactions on Computational Logic, 2000.

[28] Christian Decker and Roger Wattenhofer, *Information propagation in the Bitcoin network*. 13th IEEE International Conference on Peer-to-Peer Computing, 2013.

[29] Nakamura, *Analysis of bouncing attack on FFG*. `https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113`, 2019.

[30] dankrad, *Eth2 attack via time servers*, `https://ethresear.ch/t/eth2-attack-via-timeservers/8049`. 2021.

[31] Zhang, Ashu, *RANDAO: A DAO working as RNG of Ethereum*. `https://github.com/randao/randao`, 2022.

[32] STU [n.d.], *RANDAO: Under the Hood*. `https://blockdoc.substack.com/p/randao-under-the-hood`, 2022.

[33] Ethereum Group (Pull Request), *Fix bouncing attack tests*. https://github.com/ethereum/consensus-specs/pull/2301, 2021.

[34] D'Amato, Tse, et al. , *No More Attacks on Proof-of-Stake Ethereum?*. arXiv preprint, 2022.

[35] Nakamura, Buterin, *Prevention of Bouncing Attack on FFG*. https://ethresear.ch/t/prevention-of-bouncing-attack-on-ffg/6114, 2020.

[36] Lamport, *Using time instead of timeout for fault-tolerant distributed systems*. ACM Transactions on Programming Languages and Systems, 1984.

[37] Buterin, *Network-adjusted timestamps*. https://ethresear.ch/t/network-adjusted-timestamps/4187, 2018.

[38] Vlasov, *Time attacks and security models*. https://ethresear.ch/t/time-attacks-and-security-models/6936, 2020.

[39] Neu, Nusret Tas, and Tse, *Two More Attacks on Proof-of-Stake GHOST/Ethereum*. ACM, 2022.

[40] Schwarz-Schilling, Neu, Monnot, Asgaonkar, Nusret Tas, Tse, *Three Attacks on Proof-of-Stake Ethereum*. CoRR, 2021.

[41] Cambridge University, *Cambridge Bitcoin energy consumption index*. https://ccaf.io/cbnsi/cbeci/comparisons, 2019.

[42] Buterin, *Proposal for mitigation against balancing attacks to LMD GHOST*. https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation, 2021.

[43] Ethereum Documentation, *Proof of Stake Rewards and Penalties*. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/, 2023.

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

[45] Bistarelli, De Nicola, Galletta, Laneve, Mercanti and Veschetti, *Stochastic modeling and analysis of the bitcoin protocol in the presence of block communication delays*. Concurrency Computat Pract Exper, 2023.

[46] Anupama, Sunitha, *Analysis of the Consensus Protocols used in Blockchain Networks â An overview*. 2022 IEEE International Conference on Data Science and Information Systems (ICDSIS), 2022.

[47] Wendl, Doan, Sassen, *The environmental impact of cryptocurrencies using proof of work and proof of stake consensus algorithms: A systematic review*. Journal of Environmental Management, 2023.

[48] Ethereum Documentation, *People: The last line of defense*. `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attack-and-defense/#people-the-last-line-of-defense`, 2023.

[49] Patrick Mccorry, *The Inactivity Leak*. `https://www.cryptofrens.info/p/the-inactivity-leak`, 2023.

[50] Block Native Team, *Ethereum Staking Calculator*. `https://www.blocknative.com/ethereum-staking-calculator`, 2023.

[51] Selvi, *Bypassing HTTP strict transport security*. Black Hat Europe, 2014.

[52] Malhotra, *Attacking the Network Time Protocol*. Research Gate, 2016.

Appendix A

Balancing Attack exploiting LMD-Ghost

In this appendix, an illustrative example is described inspired by the scenario detailed in [39], which pertains to the execution of the Balancing Attack. The objective of this example is to enhance comprehension of the attack mechanism by elucidating its operational intricacies. Let $X = 50$ denote the number of validators per slot. Suppose the proposal weight is $X_p = 0.6X = 30$, and the fraction of adversarial validators is $\gamma = 0.2$. Additionally, assume that the attack begins when there are five consecutive slots with adversarial proposers. During the first four slots, the adversary creates two parallel chains A and B, each consisting of 3 blocks initially kept private from the honest validators. In each slot, the 10 adversarial validators from that slot vote on the blocks. Consequently, there are conflicting votes for the blocks proposed in the same slot. For the fifth slot, the adversary combines all the conflicting votes for chain A into a single block and appends it to chain A. Similarly, they merge all the conflicting votes for chain B into an equivocating block attached to chain B. This arrangement allows votes to be *batched* as follows: The adversary releases the two equivocating blocks from the fourth slot in such a way that approximately half of the honest validators see the block from chain A first and all the equivocating votes for

chain A, while the other half of the honest validators see the block from chain B first and all the equivocating votes for chain B.

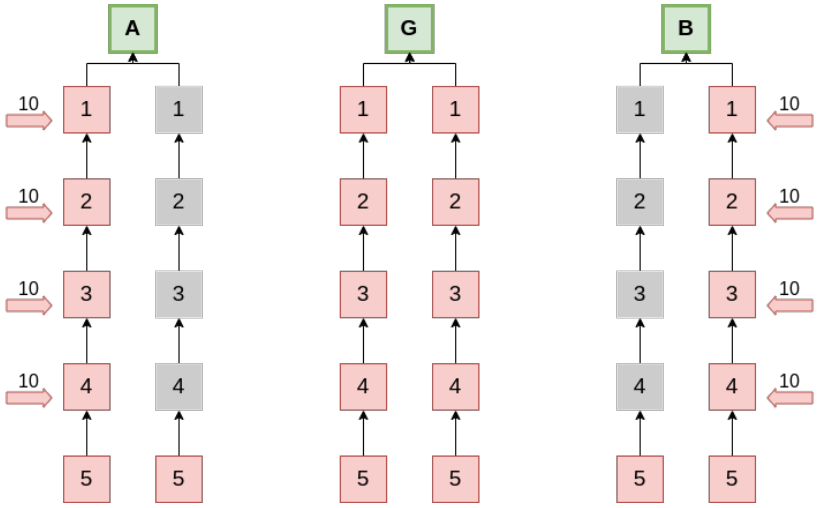

Figure A.1: Initial condition. Adversarial validators are tasked with initializing the two chains and keeping them hidden until the appropriate moment.

Now, let's suppose that the proposer of slot 6 is honest and from set HA. This validator proposes a block extending chain A. As a result, chain A gains a proposal boost equivalent to 30 votes. Thus, validators in HA perceive chain A as leading with 70 votes and cast their votes for it. Validators in HB, on the other hand, believe that chain A has 30 votes while chain B has 40 votes, so they vote for chain B. Consequently, their votes are evenly split, with both chain A and chain B receiving roughly half of the honest votes.

This pattern continues in subsequent slots, with the honest validators in HA and HB consistently voting for chains A and B, respectively. This ensures a balance of weights in the global view and perpetuates the adversarially induced divided perspective. In the LMD view of each validator, they continue voting for the chain they see as leading and find it perplexing why other honest validators persistently vote for the opposing chain.

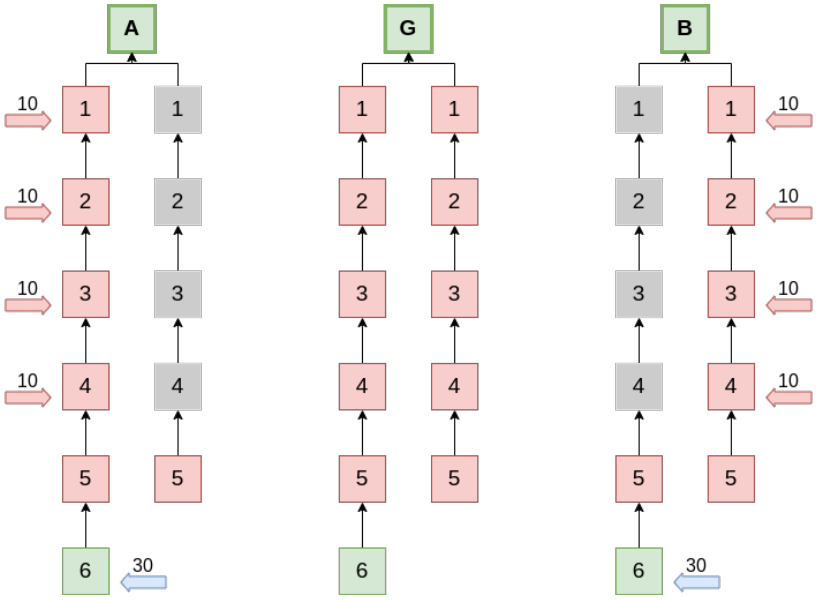

Figure A.2: Proposer boost balancing. An honest validator proposes his block, receiving the weight boost.

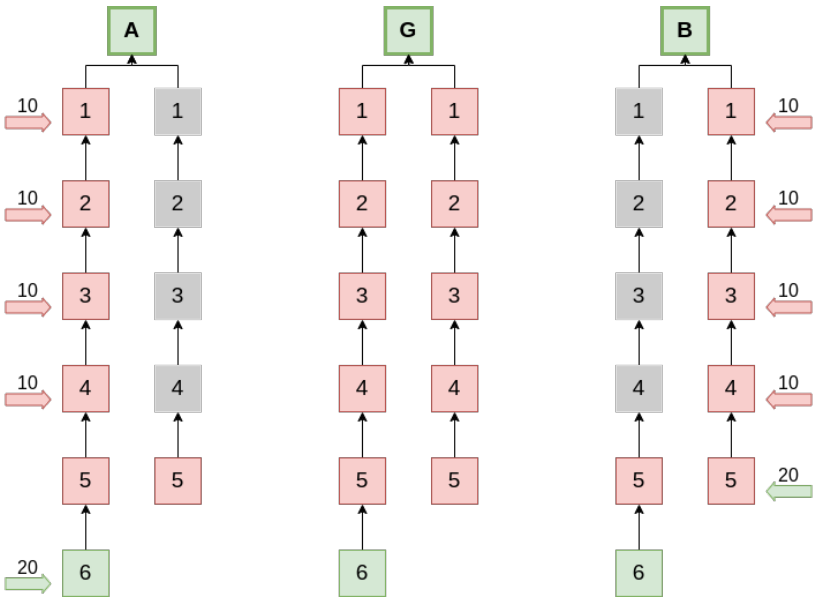

Figure A.3: Continued balancing. Even after the proposer boost, the balance between the two sets of validators continues to last over time.

This sequence of Figures A.1 - A.3 repeats in subsequent slots, with honest validators in HA and HB consistently casting their votes for Chains A and B, respectively. This ensures a balance of weights in the global view. In the LMD view of each validator, they continue to vote for the chain they perceive as leading and may find it puzzling why other honest validators persistently vote for the opposing chain, perpetuating the adversarially induced divided perspective.

Appendix B

Simple Hybrid Attack

This example serves to clarify some concepts expressed in Section 5.8. This example assumes $N = 100$, where N is the number of validators of each committee, therefore who have the right to vote in each slot of the epoch. Let's assume that $\frac{1}{3}$ of the validators is made up of attackers or dishonest ones, while the remaining part is made up of honest validators. In this simplified case, even if the number of dishonest validators is equal to $\frac{1}{3}$ of the total and therefore equal to the limit of Ethereum's BFT property, it is still considered that honest validators can justify a checkpoint, given that they are capable of increasing their stakes through rewards and consequently after an unspecified number of epochs they will have reached the ability to justify. In such a situation, given the majority of honest validators, it is not possible to carry out any type of attack on the liveness of the network. Half of the honest validators, who do not use any type of security mechanism regarding the synchronization of internal clocks given that Ethereum does not provide any additional control to that provided by the NTP, are therefore vulnerable. In this attack, in combination with a time attack, an early balancing attack is performed. In theory, this type of attack is no longer possible in the Ethereum network given the introduction of the boost proposer and in theory, LMD balancing could be used. Since the focus of this type of attack is on the vulnerabilities of validators to time synchronization in Ethereum, it is

considered a simpler and more basic balancing attack.

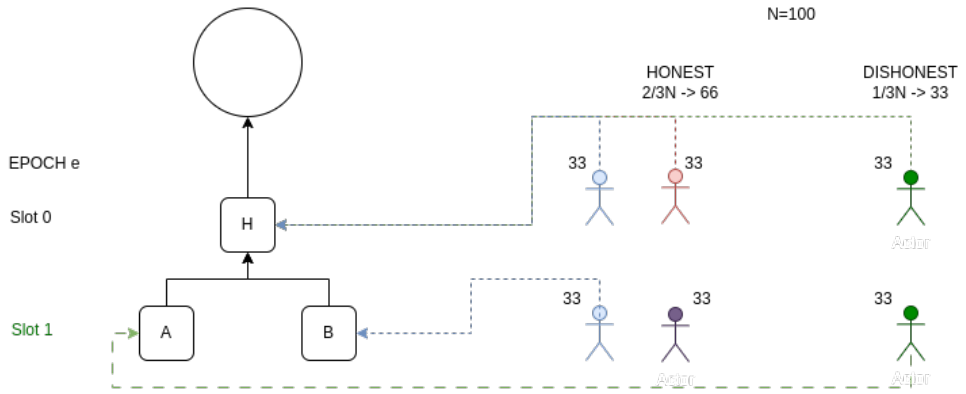

Figure B.1: Hybrid Attack: Start.

The attack begins, as can be seen in Figure B.1 when a dishonest validator is elected as proposer (slot 1 highlighted in green). In that slot, the validator proposes two blocks, a condition that will lead to its exclusion from the network due to *slashing*. At the same time, half of the honest validators who have been identified as vulnerable to a hypothetical time attack are attacked and slowed down.

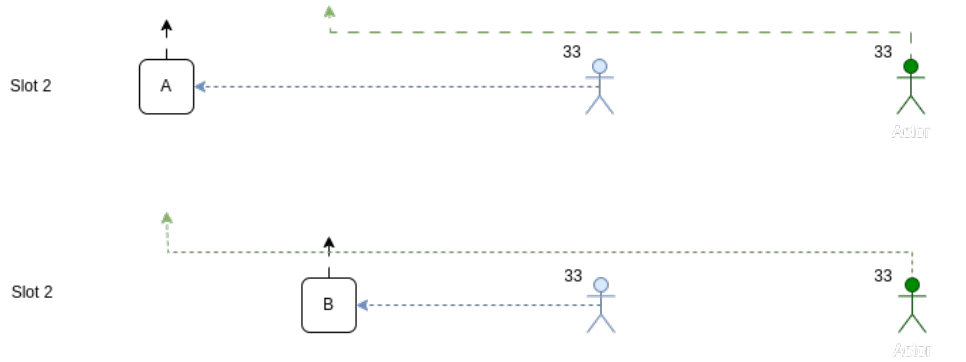

Figure B.2: Hybrid Attack: Balancing.

Figure B.2 shows the balancing phase that occurs between the validators. This process is simplified for demonstration purposes, as the slot 2 proposer

equally sends block A to half of the validators and block B to the other half. The validators who receive block A will continue to see chain A as the main one, not understanding why the other validators vote for chain B. The same reasoning applies to the validators who receive block B. In this phase, the inclusion delay is exploited, i.e. a time value (1 epoch) which is valid as the maximum delay for sending the votes by the validators. This period is essential in order to maintain a fair balance of votes, given that dishonest validators will send their attestations in order to keep the chains balanced. Figure B.3 describes the beginning of a new epoch, given that

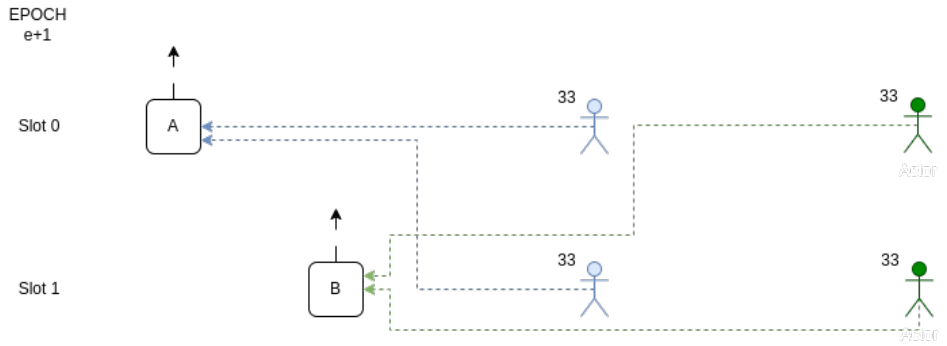

Figure B.3: Hybrid Attack: the beginning of an epoch.

to send attestations it is necessary to insert the first block of that epoch (checkpoint). Assuming a constant balance between A and B, it is assumed that there is a 50% chance that the first block of the epoch will be added to chain A and a 50% chance that it will be added to chain B. Similarly, it is also assumed for slot 1 of the same era. In the example, it is shown that if the block for slot 0 is proposed in chain A, then the dishonest validators enter a waiting situation, in order to release their votes in slot 1, i.e. when a block will be proposed as a child of the chain B. After this initial phase, operations will continue as shown in Figure B.2. In this way, even if in slot 0 attestations will be sent by some validators for chain A, given that the other half of the total validators still see B as the main chain, from slot 1 the balance will be re-established counting on the release with delay of votes by dishonest validators.