A Game-Theoretic Approach to the Study of Blockchain's Robustness

PhD Thesis from Paris-Saclay University

PhD thesis prepared at the research unit **LIST** (CEA, Paris-Saclay University), under the supervision of **Sara TUCCI-PIERGIOVANNI**, laboratory head, and co-supervised by **Yackolley AMOUSSOU-GUENOU**, associate professor.

Thesis defended at Paris-Saclay, on November 5, 2024, by

Ulysse PAVLOFF

Jury Composition

Jury members with deliberative voting rights

Sonia BEN MOKTHAR Director of Research, LIRIS CNRS	Reviewer & Examiner
Sébastien TIXEUIL University Professor, Sorbonne University	Reviewer & Examiner
Sylvain CONCHON University Professor, Paris-Saclay University	Jury President & Examiner
Daniel AUGOT Director of Research, INRIA	Examiner
Sophie CHABRIDON Professor, Telecom SudParis	Examiner

PhD THESIS
NNT : 2024UPASG071

Title: A game-theoretic approach to the study of Blockchain's Robustness

Keywords: Blockchain, Ethereum, Distributed Systems, Game Theory

Abstract

Blockchains have sparked global interest in recent years, gaining importance as they increasingly influence technology and finance. This thesis investigates the robustness of blockchain protocols, specifically focusing on Ethereum Proof-of-Stake. We define robustness in terms of two critical properties: Safety, which ensures that the blockchain will not have permanent conflicting blocks, and Liveness, which guarantees the continuous addition of new reliable blocks.

Our research addresses the gap between traditional distributed systems approaches, which classify agents as either honest or Byzantine (i.e., malicious or faulty), and game-theoretic models that consider rational agents driven by incentives. We explore how incentives impact the robustness with both approaches.

The thesis comprises three distinct analyses. First, we formalize the Ethereum PoS protocol, defining its properties and examining potential vulnerabilities through a distributed systems perspective. We identify that certain attacks can undermine the system's robustness. Second, we analyze the inactivity leak mechanism, a critical feature of Ethereum PoS, highlighting its role in maintaining system liveness during network disruptions but at the cost of safety. Finally, we employ game-theoretic models to study the strategies of rational validators within Ethereum PoS, identifying conditions under which these agents might deviate from the prescribed protocol to maximize their rewards.

Our findings contribute to a deeper understanding of the importance of incentive mechanisms for blockchain robustness and provide insights into designing more resilient blockchain protocols.

“Our time is such that those who feel certainty are stupid, while those with any imagination and understanding are filled with doubt and indecision."

- Bertrand Russell

Acknowledgement

I am deeply grateful to Sara Tucci-Piergiovanni and Yackolley Amoussou-Guenou for their invaluable support and guidance. Spending these three years alongside you has been incredibly enriching. You have skillfully guided my interests and our research to continuously focus on stimulating topics. Thank you for all these captivating discussions and for your trust throughout this journey.

Thank you to the reviewers, Sonia Ben Mokhtar and Sébastien Tixeuil, for their careful reading and detailed feedback on my work. Thank you to the jury members, Sophie Chabridon, Daniel Augot, and Sylvain Conchon, for agreeing to be part of my thesis defense committee.

I would also like to thank Jérôme Lang and Tristan Cazenave for giving me a glimpse of what a PhD journey entails and for reinforcing my desire to undertake one through the excellent experience I had working with you.

Thank you to Ewa Pichon, who supported me throughout my thesis and made it possible to share scientific knowledge seamlessly.

A special thanks to my office colleagues during most of my thesis, Alexandre Rapetti, Hector Roussille, and Valentin Pasquale. Thank you for helping me through challenging times and, above all, for making sure I never worked too hard for too long.

I must also thank my roommate, Léo Satgé, always there to support me daily while reminding me that being a PhD student doesn't grant me any advantage when it comes to competing in board games or video games.

Finally, I would like to thank my family, who has always supported me throughout these three years, especially my father, Nicolas Pavloff. The explanations and questions we exchanged were of immense help. Thank you for inspiring me to undertake this PhD and for encouraging me at every step.

Reading tips

Throughout the manuscript, every text in purple can be clicked to navigate to the indicated reference. Footnotes function in this manner as well[1]. Moreover, citations are in blue and offer the same ability to take you back to the place of the citation in the text by clicking on the page number at the end of the citation [APM+23].

[1] ↑ Clicking on the footnote's number will return you to the place of the footnote in the text.

Contents

— Chapter I —

Introduction

Contents

The first block of the Bitcoin blockchain, the genesis block, contains a message taken from the title of the January 3, 2009 edition of The Times newspaper:*"Chancellor on brink of second bailout for banks"*. This statement serves as a powerful symbol of what Bitcoin set out to challenge: a flawed financial system that has proven to be unworthy of trust. Since then, blockchain technology has gained significant recognition and ignited widespread interest. To build a better understanding and start with a clear foundation of blockchains, we begin by outlining the core principles and essential knowledge for an informed discussion about blockchains.

Before explaining blockchains, let us be clear that blockchain and cryptocurrencies are different. Cryptocurrency is one application allowed by blockchains, but blockchains can be much more and this distinction is important.

I.1 . Blockchain

Blockchain technology has revolutionized how we think about data storage, ownership, and decentralized systems. To better understand this technology, we break it down into three key components: what it achieves, how it functions, and its potential applications.

I.1.1 . Functionalities

We begin with the functionalities brought by blockchains. What new capabilities does it offer that did not exist before? A simple way to describe blockchains is to say that **a blockchain is a decentralized computer**. Like a traditional computer, you can store data and run programs on it. The difference lies in its decentralized nature: it has no single owner or operator.

Imagine a global computer that anyone can access and use to store data and run programs without restriction, this is the promise of blockchain technology. One interesting feature of blockchain is that, paradoxically, it is possible to create a strong sense of ownership over digital data on this public computer. Nakamoto first demonstrated that it was possible to enforce property rights over digital data with Bitcoin. His solution focused on digital currency, but this concept can be applied to any type of digital data stored on this decentralized "computer in the sky."

I.1.2 . Implementation

The implementation refers to *how* the functionalities are accomplished. What are the technological building blocks necessary to create this solution? This part is the most fascinating one, as it is where most of the research is conducted, and this manuscript is no exception.

Similar to how advancements in hardware have improved global communication and enabled new internet applications, blockchain technology is evolving.

The essential components used to build blockchains are:

- **Consensus Mechanism**: These are algorithms that enable the distributed network to agree on the state of the blockchain. Bitcoin uses *Proof-of-Work*, also called *Nakamoto Consensus*. Other mechanisms exist, such as *Proof-of-Stake* and *Byzantine Fault Tolerance*.

- **Cryptography:** This branch of mathematics is essential for permitting ownership in a public environment through digital signatures. Cryptography is the backbone of blockchain's security, ensuring that transactions and data remain private, secure, and verifiable without a central authority. Numerous cryptographic tools are used in blockchain. One of these tools is the *hashing*, which provides a unique fingerprint for any data. Cryptography is the reason why *cryptocurrencies*—currencies that use blockchains—are named this way.

- **Distributed Ledger**: This is the distributed database that maintains a continuously growing list of records, called blocks, literally the chain of blocks. In practice, this data is stored by every node, which are the computers working to maintain the blockchain. The consensus mechanism ensures that nodes agree on the data stored.

Research on blockchains aims to improve and create new solutions for any of these components. Our work focuses on the consensus mechanism.

I.1.3 . Purpose

Now, the infamous question: "Okay, but what for?" Working in blockchain research, this might be the question I hear the most, even more than "What are blockchains?"

There are already compelling answers, ranging from traceability and decentralized money to digital ownership of goods and art. However, this is only the beginning. As of the writing of this manuscript, blockchain has existed since 2009, about 15 years. What would the answer to this question have been for the internet 15 years after its development? At that time, the internet was mostly used for emails, you could send files, albeit very slowly, and web pages were still in their infancy. Now, the internet's applications are almost limitless.

We are 15 years past the first use case of blockchain, it has only just begun. Like the internet, the reasons for using blockchain will become more numerous as time goes on. Just as the internet's potential became clearer over time, so too will blockchain applications expand and evolve.

The most well-known application of blockchain, cryptocurrency, which can be seen as decentralized money, is often deemed useless in the Western world. We in Western countries might not see the immediate need. However, millions of people in countries where the value of money is decreasing daily, where banks refuse to return what should be their money, have found value in blockchain. Blockchain provides a solution to actually own your money and use it without intermediaries.

While these issues may not resonate with everyone now, as more people begin to understand and experience the benefits of true digital ownership and decentralized finance, the value of blockchain will become increasingly evident.

Understanding blockchain's functionalities, implementation, and potential applications is crucial as we continue to explore and refine this transformative technology. In the following section, we delve into the history of the first blockchain: Bitcoin.

I.2 . Bitcoin

Bitcoin was described in a 2008 paper [Nak08] and implemented in 2009 by a mysterious author known as Satoshi Nakamoto. The technological components necessary to create Bitcoin had actually existed for about 15 years before its appearance. However, no solution had been found to make a system both decentralized and resistant to Sybil attacks and double spending which we detailed below. In this section, we will provide a historical overview of how Bitcoin came into existence and how it works.

Bitcoin answers the question: "How do you build a trustless system with an unknown number of participants?". The difficulty lies in the consensus mechanism required to make participants agree without knowing the number of participants. The problem of reaching consensus with a known number of participants has been addressed by the distributed systems field, notably through the Byzantine Generals Problem [LSP82].

Several challenges arise when trying to create decentralized digital money. The first challenge is preventing double spending.

Double spending. A major problem for digital money is that digital information can be copied effortlessly. Unlike a physical coin, which can only exist in one place at a time, digital assets can be copied, potentially allowing the same digital coin to be spent multiple times.

If you rely on a centralized authority such as a bank, this problem does not exist. The bank being the sole entity keeping track of your balance and authorizing transactions, it can prevent you from spending more than you own. This problem led the first digital money, *eCash*, to rely on banks for this very reason. David Chaum founded Digicash in 1989, building on the work in [CFN88], which enhanced his earlier work [Cha82].

To mitigate this issue in a decentralized system, one solution is to have the participants vote on the state of balances after each transaction. However, this leads to the second problem:

Sybil attack. A common issue when trying to reach consensus with an unknown set of participants is that some individuals could create multiple identities to increase their voting power, thereby gaining an unfair advantage in decision-making. This type of attack is called a Sybil attack, named after the book *Sybil* by Schreiber, which tells the story of a woman with dissociative identity disorder who experienced having 16 different personalities.

Voting is crucial in a decentralized system, and several solutions prior to Bitcoin struggled with this problem. One such solution was Wei Dai's *b-money*, proposed in 1998 [Dai98]. The ideas behind b-money are similar in many ways to Bitcoin, but to achieve consensus, Dai proposed using a fixed set of 'trusted' servers to keep track of balances. However, choosing and trusting these servers contradicts the principle of decentralization. We have an email from Satoshi Nakamoto to Wei Dai, sent in 2008, that reads:

> “I was very interested to read your b-money page. I’m getting ready to release a paper that expands on your ideas into a complete working system. Adam Back (hashcash.org) noticed the similarities and pointed me to your site.”

Another problem mentioned by Dai in the b-money text is the fairness in the distribution of newly created money. While Digicash and b-money laid the groundwork for digital currencies, it was Bitcoin that finally solved the key issues, enabling the creation of a fully decentralized and secure currency system. The solution found by Nakamoto actually uses the work of Adam Back: the *Proof-of-Work*.

I.2.1 . Bitcoin’s Solution

Bitcoin’s solution to the problems of double spending and Sybil attacks is *Proof-of-Work* (PoW). In 1992, a proposal to use a form of proof of work was presented by Dwork and Naor [DN92] as an anti-spam mechanism. They suggested requiring proof of computational work to send an email in order to prevent spam. Adam Back proposed a similar idea in 1997 [Bac97] and further developed it in 2002 [Bac02], this time using cryptographic hashes. Let us explain PoW and hashes in more detail.

Hash. A hash function is a cryptographic tool that creates a unique fingerprint for data. It takes any amount of data, scrambles it, and returns a short and unique result for that data.[1] Hash functions possess several desirable properties:

- **Deterministic:** For a given input, the function will always produce the same hash output.
- **Irreversible:** Also known as pre-image resistance, this property ensures that given a hash value, it should be computationally infeasible to reverse-engineer the original input. The hash function is a one-way function.
- **Avalanche effect:** Slightly different inputs should produce wildly different hash outputs.
- **Collision resistance:** This property ensures that two different inputs do not produce the same hash output.

Now that we understand what a hash function is, we can explain the Bitcoin blockchain and its PoW mechanism.

[1] ↑ To experience hash functions interactively, you can try this instructive website: online-SHA-256.

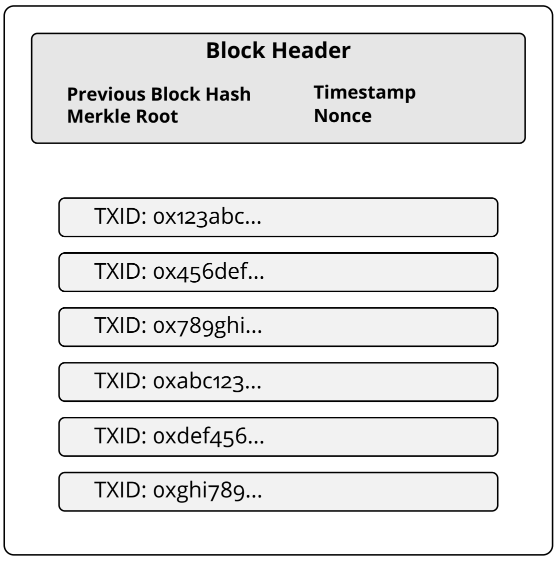

Figure I.1: Simplified representation of a Bitcoin block.

Bitcoin is a distributed computer focused on saving data about digital money and transactions. To achieve this, blocks containing transactions are regularly added and saved by every participant. The order of these blocks is crucial since it serves as a historical record of transactions. It is important to verify that someone has received 10 bitcoins before they can spend them. Each block has a structure similar to the simplified representation shown in Figure I.1.

Bitcoin Block. Each block contains a *block header* that summarizes its information. The block header includes:

- **Previous Block Hash:** This links the blocks in order by referring to the hash of the preceding block, its unique fingerprint.
- **Timestamp:** The time at which the block was created.
- **Merkle Root:** The hash of the combined transactions (tx) in the block. This provides a single fingerprint representing all the transactions included in the block.
- **Nonce:** A 'number used once' that is useful for the Proof-of-Work mechanism, explained below.

With this information, we can understand how Proof-of-Work operates.

Proof-of-Work (PoW). To add a new block to the blockchain, the hash of its block header must begin with x zeros. For instance, if $x = 3$, the hash must start with three zeros, e.g., 00019... where only the first three digits matter. Since hash functions produce unpredictable results, the block proposer must generate many different blocks to find one whose hash starts with the required number of zeros.

The probability of achieving a hash starting with three zeros is low, approximately 0.001. Therefore, successfully producing such a block serves as proof that a significant amount of computational work has been done.

The participants searching for valid blocks are called *miners*. They generate numerous blocks by slightly modifying the nonce field in the block header, hoping to find a hash that meets the requirements. The value of x, which determines the difficulty of mining a block, varies such that a valid block is found approximately every 10 minutes. As more people mine for blocks, the value of x, the difficulty, increases. It is as if, while mining for gold in your own cave, you would find gold more easily when fewer people are mining for gold elsewhere and vice-versa.

Miners search for blocks and build on top of previous ones, but what happens if two miners find valid blocks simultaneously and attempt to add them to the blockchain at the same time?

Fork choice rule. When several blocks share the same parent—meaning they reference the same previous block hash—we encounter a *fork*. When multiple branches exist, the rule prescribed by Nakamoto is to build on the longest branch[2], the one with the most blocks. If the branches are of the same length, you should add blocks to the one you saw first until, eventually, one becomes the longest.

> "Each node must be prepared to maintain potentially several 'candidate' block chains, each of which may eventually turn out to become the longest one, the one which wins. Once a given block chain becomes sufficiently longer than a competitor, the shorter one can be deleted." - Hal Finney [Fin08]

PoW solves all the problems of a decentralized computer. As stated in the Bitcoin white paper—the initial document explaining its functionality:

> "The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers."

Thus, double-spending for a transaction becomes less likely as more blocks are built on top of the block containing the transaction. The current rule of thumb is to wait for six blocks to be built after a block to trust the transactions in it. This way, it is very unlikely that a new branch will appear and become the rightful chain while possibly excluding the transaction. The Sybil attack problem is solved by making one vote equal to one CPU. A miner cannot mine (or vote) on different chains at

[2]↑ In reality, the rule is to build on the branch of blocks that has consumed the most energy to build. In practice, this almost always corresponds to the longest chain.

the same time. Every unit of energy used to search for a block on one chain is energy that cannot be used to search for a block on another chain. By making one CPU equal to one vote, as long as the majority of CPUs are owned by honest participants, malicious actors cannot control the network[3].

This clever mix of PoW and the longest chain rule is known as the *Nakamoto Consensus*. The chain with the most accumulated proof of work (i.e., the longest chain) is considered the valid one. As blocks are built upon and become older, they become increasingly less likely to be reverted. This concept is known as *probabilistic finality*, where a finalized block is one that will permanently remain in the chain.

I.3 . Ethereum

After Bitcoin showed the way, more and more blockchains began to emerge. The one we are interested in, and which is the focus of this manuscript, is Ethereum.

In 2011, a young boy named Vitalik Buterin stumbled upon Bitcoin and found the idea fascinating. He fell down the rabbit hole and started to learn more about it. Around that time, he co-founded Bitcoin Magazine. After realizing that blockchains could offer more than just digital money, he tried to convince the community to improve Bitcoin. However, Bitcoin is very conservative and slow when it comes to changes. He then embarked on a mission to convince people to help him create a new blockchain—one that wasn't centered around a single application, such as cryptocurrency, but rather for any purpose, leaving the freedom to its users to decide how to use it. After gathering like-minded individuals interested in his project, they created Ethereum [But14] in 2014.

Unlike Bitcoin, which is designed primarily for one application—digital money—Ethereum is a general-purpose blockchain. Ethereum is an open platform that allows people to build their own applications on top of it. Anything built on Ethereum is protected and secured, with every transaction checked by the entire network of millions of computers around the world that protect and verify every transaction on the blockchain.

Ethereum started as a PoW blockchain, with a fork choice rule similar to that of Bitcoin. From the start, the plan was to eventually transition to a different type of consensus called *Proof-of-Stake* (PoS).

Proof-of-Stake (PoS). The concept of PoS actually originated from a 2011 post on the BitcoinTalk forum [Qua11]. The post included the following thought:

[3] ↑ The case in which a majority of CPUs are owned by malicious actors is referred to as the 51% attack. Similar to democratic systems in general, if the majority agrees, the decision is theirs.

“I am wondering if as bitcoins become more widely distributed, whether a transition from a proof of work based system to a proof of stake one might happen. What I mean by proof of stake is that instead of your "vote" on the accepted transaction history being weighted by the share of computing resources you bring to the network, it’s weighted by the number of bitcoins you can prove you own, using your private keys.”

As explained, the idea behind PoS is that voting power is not determined by computing resources (as in PoW), but by the number of digital coins one owns. In PoW, one CPU equals one vote; in PoS, one digital coin equals one vote. This naturally raises many questions: How are the coins initially distributed? Does this system simply make the rich richer? How are blocks proposed? Different blockchains have answered these questions in various ways, and Ethereum has developed its own unique solution.

One thing to note is that, while Bitcoin is slow and conservative in its goals and development, Ethereum’s ethos is to ‘move fast and break things’. Although it took eight years to transition from PoW to PoS, many of these changes can be questioned. It seems that the Ethereum community prefers to implement changes quickly and refine them as needed, rather than engage in prolonged debates over potential risks. This dynamic approach renders Ethereum an interesting blockchain to study as it fosters the emergence of new ideas. Plus, the challenge of ideas is welcomed by the community, which can lead to direct impacts on the blockchain.

We dive into a thorough explanation of the protocol in Chapter III as this is part of our contribution.

I.4 . Terminology

Throughout this manuscript, we will utilize specific blockchain terminology. To ensure clarity and consistency, this section will serve as a glossary to explain recurring terms. Additionally, we will address instances where different terms have equivalent meanings.

Protocol. The protocol of a blockchain refers to the set of rules that users must follow to communicate and act within that blockchain. We may also refer to the *specifications* of a protocol, which holds the same meaning. In practice, the protocol is defined by the code that blockchain participants use to send messages, propose blocks and transactions, and perform other necessary actions.

Participants. "Participants" is a term with many synonyms. Participants can also be called agents, nodes or processes interchangeably. There are different types of participants: those who take part in the blockchain consensus, and those who do not. A participant who only sends transactions is considered a user of the blockchain; they use the blockchain but do not have a role in its consensus. In Ethereum, participants involved in the consensus process are known as **validators**, while in Bitcoin, they are called miners. The terminology varies depending on their role, which is defined by the protocol. In practice, consensus participants are machines running programs to follow and engage in the blockchain protocol.

Byzantine. A Byzantine participant can deviate arbitrarily from the prescribed protocol. This appellation stems from the Byzantine General Problem introduced by Lamport, Shostak, and Pease [LSP82]. This problem is an analogy to simulate how reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. The problem is stated as follow:

> “A group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement.”

This thought experiment outlines the challenges of reaching consensus if some members of the group are compromised. One of the solution proposed involves making the assumption that, among the n participants there are only f traitors[4], such that $f < n/3$.

This threshold stems from the two constraints we have in order to reach consensus: (a) the honest participants $n - f$ should be able to make a decision even if the traitors do not respond, and (b) the traitors should not be able to cause honest participants to make two different decisions. For (a), this means that $n - f$ must constitute a majority. For (b), this means that half of the honest participants $\frac{n-f}{2}$ plus the traitors f must not constitute a majority. This translates in:

$$\begin{aligned} n - f &> \frac{n-f}{2} + f \\ \Leftrightarrow \qquad 2n - 4f &> n - f \\ \Leftrightarrow \qquad \frac{n}{3} &> f. \end{aligned}$$

This solution is fundamental in proving that reaching consensus is possible even in adversarial settings. The requirement for two-thirds of participants to agree in order to reach consensus is often called a supermajority. A protocol that tolerates the presence of a Byzantine adversary while maintaining its guarantees is deemed *Byzantine Fault Tolerant* (BFT).

[4]↑ f stands for faulty, we use this term interchangeably to talk about Byzantine participants.

Finalization. In blockchains, we say that a block is finalized if it permanently belongs to the chain. Non-finalized blocks are not guaranteed to always belong to the chain.

Consensus. The consensus holds the same meaning as in everyday life, this is an agreement among a set of agents. The only difference is that in blockchain systems, agreements concern blocks, and consensus is repeatedly achieved to agree on increasingly larger sets of blocks. In this context, our formalization of consensus in blockchain, partly based on Dolev et al. [DDS87], is the following.

Definition I.1 (Consensus)**.** *A blockchain protocol achieves consensus if it satisfies the following three properties:*

- ***Safety**: No concurrent blocks can be finalized.*
- ***Liveness**: The set of finalized blocks continuously grows.*
- ***Validity**: The blocks agreed upon must have been proposed by one of the participants.*

The validity property is often taken for granted in blockchain and what we are really interested in are the Safety and Liveness properties. This was part of our work to define them adequately before beginning our analysis, the formal definitions are presented in Chapter III.

The consensus in blockchain can be summarized as we want to make sure we agree on the same thing and there should never be a point after which we cannot agree anymore. The protocol must not reach a point where finalization stops, preventing any new blocks from being finalized. In this case, the protocol is considered live.

These two properties are more thoroughly described in Chapter III and constitute part of our contribution.

I.5 . Contribution

Our research focuses on the analysis of blockchain robustness. By robustness, we mean the blockchain's ability to avoid unsolvable forks (ensuring *Safety*) while always maintaining the possibility to add new blocks (ensuring *Liveness*). Our work ranges from distributed systems to game theoretic analysis. Distributed systems consider two types of agents: honest and Byzantine. Honest participants follow the prescribed protocol while Byzantine participants deviate from it arbitrarily. This binary classification overlooks the nuances introduced by rational players of game theoretic models, who act based on incentives rather than strict adherence to the protocol or malicious intent. The lack of research in distributed systems regarding blockchains' incentives that are yet paramount for participants motivated our work.

In particular, we focused our analysis on one blockchain: Ethereum. Ethereum is the second biggest blockchain in terms of market capitalization and changed from a PoW consensus to a PoS one at the beginning of our work (2021). This transition called *The Merge* brought a lot of changes with it and motivated our work. We often refer to Ethereum protocol as Ethereum PoS protocol to emphasize this change of paradigm. This resulted in an involved protocol lacking study and explanation. By focusing on Ethereum, we saw an opportunity to contribute meaningfully to the field while closely observing the transition and its impacts on the properties of the blockchain.

The organization of this manuscript follows the chronological progression of our research. After establishing the necessary properties and definitions, we first analyze the protocol from a distributed systems perspective, initially without considering rewards and penalties. We then extend this analysis by incorporating penalties. Finally, the last technical chapter before the conclusion examines the protocol from a game-theoretic perspective. Here is a more detailed overview of each chapter:

- Chapter II presents the remaining terms useful throughout the manuscript and gives the essential properties and model we use for our analysis.

- Chapter III is the first part of our work, focusing on understanding the Ethereum PoS protocol. We started from scratch by reviewing the code to extract its properties. This led to the publication of our first paper [PAT23] that we extended for a ACM DLT journal recently accepted.

- Chapter IV pushes the protocol analysis further by taking into account penalties present in the protocol. This is uncommon in distributed system analysis. This work led to another publication [PAT24a]

- Chapter V is the last technical chapter and contains elements of our last paper [PAT24b]. We model the interactions between block proposers and attesters as a game. We investigate the most profitable behaviors for the players.

- Chapter VI summarizes our results and opens on possible future research linked to our work.

I.6 . Related Work on Blockchain Analysis

As mentioned, our work during this thesis revolves around the case study of Ethereum Proof-of-Stake. We contribute to the field by formalizing the protocol, defining properties, presenting attacks against these properties, and then analyzing the strategies of rational validators. In this chapter, we present related works corresponding to each of these efforts.

I.6.1 . Blockchain Formalization

The category of papers that aim to formalize blockchains includes all the *white papers*. These are articles explaining the main features of a protocol, often written by the team behind the blockchain. This trend started in 2008 with the first blockchain, Bitcoin [Nak08], and has been followed almost religiously by subsequent protocols. Ethereum is no exception, having released its first white paper in 2014 [But14] when the Ethereum blockchain operated with a PoW consensus.

Following the release of a white paper by a protocol's team, other papers have emerged to challenge or complement the blockchain's description. For Bitcoin, this is exemplified by the work of Garay et al. [GKL15], who analyzed the protocol's pseudo-code and deduced some of its properties. Similarly, Amoussou-Guenou et al. [APPT19] proved the correctness of the Tendermint protocol, thereby complementing the initial white paper [BKM18]. Alturki et al. [ACL+19] used a proof assistant to prove the safety of the Algorand's blockchain [CM19]. Similarly, García-Pérez and Schett [GS19] provided a formal correctness proof of the Stellar Consensus Protocol (SCP). Amores-Sesar et al. [ACM20] study revealed that the Ripple protocol might violate both safety and liveness, challenging the initial claims of Ripple's Byzantine fault tolerance.

The first section of our work has a similar aim. We extract the pseudo-code from the Ethereum specifications [Fou24], which is the description of how to implement the protocol, to formalize its properties. The most recent Ethereum white paper was released to explain the new protocol following the transition from PoW to PoS [BHK+20].

Outside of this line of work focusing on specific protocols, other efforts have aimed to provide formal foundations for blockchains. Anceaume et al. [ADL+19] proposed a formalization of blockchains and their evolutions as Block Trees. The work of Anceaume et al. [ADRT21] described the different ways blockchains can ensure that blocks permanently belong to the chain, finalizing them. We rely on the definitions of Block Tree and finality to express the properties of the Ethereum protocol.

I.6.2 . Attacks and Vulnerabilities

A considerable amount of work has been focused on identifying protocol vulnerabilities. The most famous example is probably the seminal work of Eyal and Sirer [ES18], which presents the selfish mining attack on Bitcoin. Eyal and Sirer show that in Bitcoin (and proof-of-work in general), miners can benefit from deviating from the prescribed protocol by withholding blocks for a while, to the detriment of honest miners. Many other examples exist: for instance, Amoussou-Guenou et al. [ADPT18] pointed out a liveness vulnerability in the Tendermint protocol, and Neuder et al. [NMRP20] presented an attack where nodes can reorganize Tezos' Emmy+ chain and then perform a double-spend attack. The protocol has been updated since then for a more robust solution proposed by Astefanoaei et al. [ACP+21].

Our work focuses on Ethereum, which is no stranger to protocol attacks. Neu et al. [NTT21] exhibited a balancing attack, highlighting the shortcomings of a consensus mechanism divided into two layers (finality gadget and fork choice rule). Mitigation against this attack was proposed, but Neu et al. [NTT22] overcame this mitigation with a new balancing attack. Schwarz-Schilling et al. [SNM+22] presented *reorg attacks*, attacks where the chain is reorganized leaving previous blocks orphaned, revealing that proposers could gain from disturbing the protocol by releasing their blocks late.

Nakamura [Nak19b] presented an attack called *splitting attack*, in which the adversary sends messages to split the set of validators. However, Nakamura assumes that the adversary needs to control and manipulate network delays, which is a strong and potentially unrealistic assumption. More recently, Schwarz-Schilling et al. [SNM+22] demonstrated through experiments that attackers can predict the proportion of validators receiving a given message within a specific timeframe with sufficient accuracy. This contradicts Nakamura's claim that the attack necessitates the adversary to control network delay.

We contribute to this line of work by identifying flaws on the Ethereum PoS protocol. First, we outline a flaw regarding the liveness of the current Ethereum Proof-of-Stake protocol, emphasizing the importance of reconciling availability and finality. Our approach differs from Galletta et al. [GLMV23], who aim to formally verify the *Hybrid Casper* protocol [BG17], focusing on an outdated version. We formalize the current implementation of the protocol through pseudo-code and expose a liveness attack on the protocol. Our work presents a form of the splitting attack where a message received by honest validators at different times is differently perceived to be on time or too late, splitting the validators into two 'views'. This different perception greatly influences which chain they consider canonical. This attack is based on the assumption that the adversary knows the network delay (in line with Schwarz-Schilling et al. [SNM+22]) but does not control it.

I.6.2A Incentives

Very few efforts in the literature have taken the incentive mechanism of protocols into account to evaluate how Byzantine validators could exploit it. Initial efforts were made to intertwine the study of incentives with considerations of liveness and safety properties of the Ethereum protocol [BRLP20]. However, this early exploration discussed a preliminary version of the protocol [BG17] and did not include an analysis of the inactivity leak. The inactivity leak is the mechanism that penalizes inactive validators by reducing their stake. The most recent version of the protocol by its founder [BHK+20] does not mention this mechanism. The inactivity leak still lacks a detailed examination, and our work aims to fill this gap.

While mechanisms similar to Ethereum's inactivity leak exist elsewhere (e.g., [Woo16, Goo14]), to the best of our knowledge, there has very few analysis of the risk associated with potentially draining honest stake in a Byzantine-prone environment. An investigation linking penalties with the actions of Byzantine validators is presented by Zhang et al. [ZLD23]. This work demonstrates how Byzantine validators can maliciously cause attestation penalties for honest validators.

Our work is similar to Zhang et al. in scope, however we focus on more substantial penalties, i.e., the inactivity penalties and slashing. During the inactivity period, attestation penalties tend to be less significant. We found that the penalties could be detrimental for the protocol's safety if exploited by Byzantine participants.

I.6.3 . Rational Agents in Blockchain

Incentives are not often considered in the distributed systems field. Game theory precisely addresses this gap.

Following the influential work of Eyal and Sirer [ES14, ES18], subsequent works (e.g., [GP20, NKMS16, SSZ16, ZET20]) use game-theoretic tools to analyze the maximum gain a rational miner can achieve by selfish mining, i.e., deviating from the proof-of-work protocols by withholding found blocks to gain an advantage in mining subsequent blocks. Also considering only rational participants, the work of Biais et al. [BBBC19] proves that while playing the proof-of-work game and following the longest chain's rule is an equilibrium -where no participant can improve their outcome by changing their strategy-, multiple other equilibria exist where forks may persist.

At the intersection of distributed systems, which model agents as either honest or Byzantine, and game theory, which models agents as rational, a mixed model was proposed: BAR (Byzantine, Altruistic, and Rational) [AAH11]. However, the complexity of the analysis rises quickly with so many types of agents. The work of Halpern and Vilaça [HV16] considers rational participants who can fail by crashing. They prove that in such a setting, there is no ex-post Nash equilibrium solving the fair consensus problem. Amoussou et al. [ABPT20] consider agents being either rational or Byzantine, exhibiting different equilibrium depending on the proportion of each.

These works do not apply to Ethereum since its PoS mechanism is too different from classic PoW or classic BFT. Regarding PoS in general, Saleh [Sal20] showed that the *nothing-at-stake*, problem in which PoS participant can extend simultaneously different fork without cost was prevented due to the value of the blockchain and thus their stake being decreased by such actions. For the Algorand blockchain Fooladgar et al. [FMJR20] showed that the cost and rewards of the protocol did not create an equilibrium in which participants followed the protocol. They proposed an adjustment of the rewards to entice selfish participants to cooperate. Comparing our work to the game theory literature on the Ethereum PoS *consensus*, Roughgarden [Rou20] conducted a game-theoretic analysis of an Ethereum Improvement Proposal (EIP) to evaluate its impact on transaction and proposer rewards considering rational agents. Many works focus on MEV (Maximal Extractable Value), which involves taking advantage of the transaction ordering in a block. In contrast, we focus on game-theoretic analyses affecting the consensus directly. We differ from works like [BCC$^+$23], which consider the possibility of making ransom demands without being detected. Schwarz-Schilling [SNM$^+$22] studies the optimal timing for proposers to propose their blocks.

Our endeavor is closest to the work of Carlsten et al. [CKWN16] and Tsabary and Eyal [TE18] that study selfish behavior in Bitcoin using game theory when the *only* source of rewards is transaction fees (no more coinbase transactions). Tsabary and Eyal show that the Bitcoin blockchain becomes unstable since block miners fork the Bitcoin blockchain to obtain the most lucrative transactions. Ethereum PoS does not reward block proposers with coinbase transactions; however, the presence of attesters and a different fork choice rule than Bitcoin's makes the analysis more complex. We focus on analyzing the behavior of block proposers and attesters in Ethereum PoS using game theory and demonstrate that the protocol tends to stabilize, even though a proposer might gain more by deviating from the prescribed strategy due to the initial asynchronous setting. Moreover, transaction fees do not play a crucial role in our analysis due to rewards stemming from attestations.

— Chapter II —

Background

Contents

Conducting science primarily involves two approaches: empirical and theoretical. Empirical science observes systems and studies the data and results from experiments, inferring general laws and their functioning. Theoretical science uses models to represent problems and employs these models to derive theorems and rules about the subject of study. When creating a model, you must make assumptions and hypotheses that define the constrained reality you aim to understand. In this chapter, we outline models relevant to blockchain protocols and consensus.

II.1 . Distributed Computing Model

Analyzing the Ethereum PoS, we consider participants to be *validators*, forming a finite set Π. There are a total of n validators. Being in PoS system, each validator owns a *stake*, which refers to the amount of cryptocurrency (ETH) they possess. This stake serves as a metric of their investment and influence in the consensus protocol. Throughout this manuscript, the term "proportion" is used concerning a validator set to denote the ratio of their combined stake to the total staked. A validator is interested in owning a stake as it comes with responsibilities that are rewarded. Initially capped at 32 ETH, a validator's stake has the potential to decrease.

II.1.1 . Fault Model

In distributed systems, the goal of a protocol is to guarantee certain properties despite the presence of *faulty* participants. In our analysis, faulty participants will always be Byzantine ones [LSP82].

Following the well-known work of Castro and Liskov [CL99], we consider the worst-case scenario in which all faulty nodes are controlled by a single adversary. This assumption of a strong adversary is crucial for ensuring the reliability of critical distributed systems, where certain guarantees are expected even in the event of unexpected failures. However, during our analyses the adversary does not manipulate message delays between honest validators.

Unlike the dynamic adversary model considered by Chen and Micali [CM19], where the adversary can change the set of faulty nodes during execution, the works presented in this manuscript consider a *static* adversary. This means that the set of faulty participants is determined at the start of the analysis and does not change throughout. We denote the proportion of Byzantine validators, which is the ratio of the sum of the stake of all Byzantine validators over the stake of all validators, by β with $\beta < 1/3$. When analyzing changes in the Byzantine proportion over time, we define the *initial* proportion as $\beta_0 < 1/3$.

The Ethereum PoS protocol aims to achieve Byzantine Fault Tolerance (BFT), ensuring the preservation of both Safety and Liveness properties as long as the initial proportion of Byzantine validators (β_0) remains strictly below $1/3$.

II.2 . Synchronization and Communication

II.2.1 . Time

Each participant maintains its own clock to keep track of time. We assume that all clocks are synchronized and run at the same pace. Any discrepancies in clock synchronization are considered as part of the message delay.

In the Ethereum PoS protocol, time is measured in periods of 12 seconds, called *slots*, with a period of 32 slots making up an *epoch*, which serves as the largest time unit in the protocol. These timeframes are used to assign specific roles to validators at particular moments.

II.2.2 . Network

We assume a *partially synchronous model* [DLS88], which consists of an asynchronous period of unknown length followed by a synchronous period:

- During the asynchronous period, there is no bound on message delay. A message sent during this period has no guarantee of reaching its recipient before the asynchronous period ends.
- Conversely, in the synchronous period, there is a known bound Δ, ensuring that any message sent at time t is received by time $t + \Delta$ at the latest.

The partially synchronous model begins with an asynchronous period that lasts until a global stabilization time (GST), after which the synchronous period begins.

Studying protocols under this model is common to ensure resilience under both good and bad network conditions. It is important to note that even with synchronized clocks, the presence of an asynchronous network before GST still qualifies the system as partially synchronous.

Validators communicate through message passing. We assume the existence of an underlying broadcast primitive, which operates as a best-effort broadcast. This means that when an honest validator broadcasts a value, all honest validators eventually receive it. Messages are signed with a digital signature, providing a mechanism for cryptographic identification and validation within the protocol.

II.3 . Game Model

This manuscript presents three distinct analyses: the first two from a distributed systems perspective, and the last one from a game-theoretic perspective. The shift in focus necessitates a corresponding change in the modeling approach.

In Chapter V, we model the Ethereum PoS consensus protocol as a game where each player[1] acts either as a *proposer* or an *attester*. In the ideal scenario, proposers propose blocks, and attesters broadcast attestations. The game unfolds over s sequential *slots*. There is one proposer and $a \in \mathbb{N}$ attesters per slot, leading to a total of s proposers and as attesters. The total number of slots s is unknown to the players.

Similar to the approach in [CKWN16], our game is based on the following assumptions: (i) The game occurs during a synchronous period where the network is fully synchronous, meaning there is no latency. This implies that once information (such as a block, attestation, or transaction) is broadcast, all players immediately become aware of it. (ii) The synchronous period follows an asynchronous period, during which there may have been delays in information transmission. This assumption aligns with the Ethereum protocol's network behavior hypothesis [BHK+20].

We model the interactions between proposers and attesters during n slots in Ethereum PoS as a dynamic game in which actions occur sequentially. In each slot, the sequence of events is as follows: (i) a block is proposed at the beginning of the slot, (ii) new transactions are proposed, and (iii) all attesters for the slot simultaneously send their attestations. The actions, rewards, and strategies of validators will be thoroughly described and analyzed in Chapter V.

[1] ↑ The players are the participants of the Ethereum PoS that we also consider as a game for our analysis.

— Chapter III —

Ethereum PoS Analysis under the Distributed Computing Model

Contents

All of our work revolves around the Ethereum protocol. We intend to study this blockchain, which was once a PoW blockchain and is now a PoS blockchain, to develop a general understanding using a famous case study. To do so, we start with the roots of any protocol: its code. The Ethereum Foundation and the inventor of Ethereum, Vitalik Buterin, have produced a paper [BHK+20] to explain the protocol and prove its properties. The issue is threefold: the paper discusses an outdated version of the protocol, not the entirety of the protocol is taken into account, and their results seem to contradict the crucial CAP theorem [Bre00]. For all these reasons combined, we begin our analysis from scratch, starting from the code. We begin our contribution by defining crucial properties for blockchains. We then use pseudo-code that reflects the protocol and the properties newly defined to evaluate the guarantees provided by the protocol. This first analysis does not consider the incentives part of the protocol.

III.1 . Safety and Liveness Properties

Once the protocol has been laid out, we can investigate its properties. Despite their names, blockchains are closer to *block trees*. Forks can occur and cause the blockchain to have several branches rather than a unique chain. We adopt the formalization of Anceaume et al. [ADL+19] of blockchain data structure as a block tree. Indeed, the blockchain takes the form of a tree in which every node is a block pointing to its unique parent, and the tree's root is the *genesis block*. Among the different branches of the block tree, the protocol indicates a unique branch, or chain, to build upon with a so-called fork choice rule (e.g., the longest chain rule in Bitcoin). The selected chain is called the *canonical chain*.

Definition III.1 (**Canonical chain**). *We call **canonical chain** the chain designated as the one to build upon by the fork choice rule. Considering the view of the chain of an honest validator i, i's associated canonical chain is noted $\mathcal{C}_i$.*

The blocks in the canonical chain can be finalized or not.

Definition III.2 (**Finalized block**). *A block is finalized for a validator i if and only if the block cannot be revoked, i.e., it permanently belongs to the canonical chain $\mathcal{C}_i$.*

Note: It stems from the definition that all the predecessors of a finalized block are finalized.

Definition III.3 (**Finalized chain**). *The finalized chain is the chain constituted of all the finalized blocks.*

Note: The finalized chain $\mathcal{C}_{fi}$ is always a prefix of any canonical chain $\mathcal{C}_i$.

To analyze the protocol, one needs to examine the capability of the Ethereum Proof-of-Stake protocol to construct a consistent blockchain (safety), to allow validators to add blocks despite network partitions and failures (availability), and to make progress on the finalization of new blocks (liveness). These are paramount properties characterizing blockchains. Safety, availability, and liveness are expressed as follows:

Property III.1 (**Safety**). *A blockchain is consistent or **safe** if, for any two honest validators with a finalized chain, one chain is necessarily the prefix of the other. More formally, for two validators i and j with respective finalized chains $\mathcal{C}_{fi}$ and $\mathcal{C}_{fj}$, $\mathcal{C}_{fi}$ is the prefix of $\mathcal{C}_{fj}$ or vice versa.*

Property III.2 (**Liveness**). *A blockchain is **live** if the finalized chain is ever growing.*

Property III.3 (**Availability**). *A blockchain is **available** if the following two conditions hold: (1) any honest validator is able to append a block to its canonical chain in bounded time, regardless of the failures of other validators and the network partitions; (2) the canonical chains of all honest validators are eventually growing, i.e., given a block b_k added to a canonical chain at a distance d from the genesis block b_0, where the distance is the number of blocks separating b_k from b_0, eventually a block b_l will be added to the canonical chain at a distance $d' > d$.*

The key difference between the finalized chain and the canonical chain is that blocks in the finalized chain are permanent and cannot be revoked. In contrast, the canonical chain can switch branches, meaning blocks in the previously chosen branch can potentially be revoked. Availability, on the other hand, guarantees that adding blocks to the canonical chain is a wait-free operation whose time to complete does not depend on network failures or Byzantine behaviors. Availability also implies that blocks are constantly added in such a way that the height of the canonical chain eventually grows. This property avoids the pathological scenario in which all the blocks are added to the genesis block to form a star.

As in any distributed system, blockchains are faced with the dilemma brought by the CAP Theorem. This theorem states that no distributed system can satisfy these three properties at the same time: *consistency*, *availability*, and *partition tolerance*. Indeed, if network partitions occur, either the system remains available at the expense of consistency, or it stops making progress until the network partition is resolved to guarantee consistency. This means that no blockchain can simultaneously be available and consistent. However, by maintaining the canonical and the finalized chain simultaneously, Ethereum Proof-of-Stake aims to offer both safety and availability. The canonical chain aims to be available but without guaranteeing consistency all the time, while the finalized chain falls on the other side of the spectrum, guaranteeing consistency without availability. Therefore, the finalized chain will finalize blocks only when it is safe to do so, whereas the canonical chain will still be available during network partitions (caused by network failures or attacks). The only caveat here is that the finalized chain grows by finalizing blocks of the canonical chain, which means that the properties of the two chains are interdependent. In particular, to assure liveness, it is necessary that the canonical chain steadily grows. This interdependence is a source of vulnerability as we show in the remainder of our analysis.

III.2 . Ethereum PoS protocol

III.2.1 . Overview

The Ethereum Proof-of-Stake (PoS) protocol design is quite involved. We identify, similarly to [NTT21], the objectives underlying its design as follows: (i) finalizing blocks and (ii) having an *available* canonical chain that does not rely on block finality to grow. To this end, the Ethereum PoS protocol combines two blockchain designs: a Nakamoto-style protocol to build the tree of blocks containing the transactions and a BFT finalization protocol to progressively finalize blocks in the tree. The objective is to keep the blockchain creation process always available while guaranteeing the finalization of blocks through Byzantine-tolerant voting mechanisms. The finalization mechanism is a *Finality Gadget* called *Casper FFG*, and the fork choice rule to select canonical chains is *LMD GHOST*.

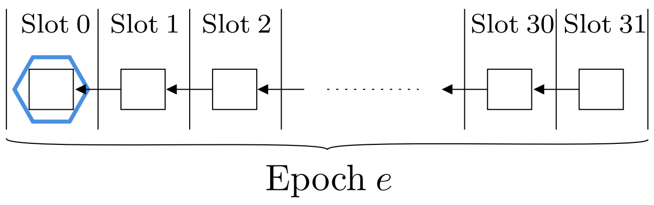

Figure III.1: Ethereum protocol Structure

Before introducing how the fork choice rule and the finality gadget work together, we will introduce the following basic concepts: (i) slots, epochs, and checkpoints, which set the pace of the protocol allowing validators to synchronize together on the different steps, (ii) committees formation and assignment of roles to validators as proposers and voters for each slot, and (iii) the different types of votes the validators must send in order to grow and maintain the canonical chain as well as the finalized chain.

In this section, we focus on providing a formalized version of the protocol through pseudo-code, following the specification given by the Ethereum Foundation [Fou24]. Note that a description of an initial plan of the protocol was proposed by Buterin et al. in [BHK+20]. We describe and formalize the current implementation of the protocol [Fou24].

III.2.1A Slots, Epochs & Checkpoints

In proof-of-work protocols, such as originally described in [Nak08], the average frequency of block creation is predetermined in the protocol, and the mining difficulty changes to maintain that pace. In contrast, in Ethereum PoS, it is assumed that validators have synchronized clocks to propose blocks at regular intervals. More specifically, the protocol uses time frames called *slots* (12 seconds) and *epochs* (6 minutes and 24 seconds). Each epoch is comprised of 32 slots. Epoch 0 contains slot 0 to slot 31, then epoch 1 slot 32 to slot 63, and so on. These slots and epochs allow associating the validators' roles to the corresponding time frame.

An essential feature of epochs is the *checkpoint*. A checkpoint is a pair (block, epoch) (b, e) where b is the block of the first slot[1] of epoch e. Figure III.1 represents the structure of an epoch in Ethereum PoS, with the checkpoint being represented by the hexagonal blue shape.

[1]↑ In the event of an epoch without a block for the first slot, the block used for the checkpoint is the last block in the canonical chain, belonging to a previous epoch. On the contrary, if the proposer of the first slot proposes multiple blocks, this will create multiple checkpoints for the other validators to choose from using the fork choice rule.

III.2.1B Validators & Committees

Validators have two main roles: *proposer* and *attester*. The proposer's role consists of proposing a block during a specific slot[2]. This role is pseudo-randomly[3] assigned to 32 validators per epoch (one for each slot). The attester's role consists of producing an attestation sharing the validator's view of the chain. This role is assigned once per epoch to each validator.

In each epoch, a validator is assigned to exactly one committee (of attesters). A committee $\mathtt{C}_j$ is a subset of the whole set of validators. Each validator belongs to exactly one committee, i.e., $\forall j \neq k, \mathtt{C}_j \bigcap \mathtt{C}_k = \emptyset$ and for each epoch $\bigcup_i \mathtt{C}_i = \Pi$. Each committee is associated with a slot. During this slot, each member of the committee will have to cast an *attestation* to indicate its view of the chain.

In short, during an epoch, validators are all attesters once and have a small probability of being proposers ($32/n$). The roles of proposer and attester are entirely distinct, i.e., the proposer of a slot is not necessarily an attester of that slot.

III.2.1C Vote & Attestation

There are two types of votes in Ethereum PoS: the *block vote*[4] and the *checkpoint vote*[5]. The message containing these two votes is called an *attestation*. During an epoch, each validator must make one attestation. The attestation should be sent during a specific time slot, which is determined by the validator's committee. The two types of votes, checkpoint vote and block vote, have very distinct purposes. The checkpoint vote is used to finalize blocks and grow the finalized chain, while the block vote is used to determine the canonical chain. Although validators cast their two types of votes in one attestation, an important distinction must be made between the two. Indeed, the two types of votes do not require the same conditions to be taken into account. The checkpoint vote of an attestation is only considered when the attestation is included in a block. In contrast, the block vote is considered one slot after its emission, whether it is included in a block or not.

The code associated with the production of attestations is described in Algorithm 3 at subsection III.2.2. We then describe in Algorithm 6 how the reception of attestations is handled.

III.2.1D Finality Gadget

The finality gadget is the mechanism that aims at finalizing blocks. The finality gadget grows the finalized chain independently of block production. This decoupling of the finality mechanism from block production permits block availability even

[2] ↑ The current protocol specifications [Fou24] indicate that honest validators should send their block proposition during the first third of their designated slot.

[3] ↑ Detailed explanation in subsubsection III.2.1F.

[4] ↑ Also called GHOST vote in [BHK+20] and in the specifications [Fou24].

[5] ↑ Also called FFG vote in [BHK+20] and in the specifications [Fou24].

when the finalizing process is slowed down. This differs from protocols like Tendermint [BKM18], where a new block can be added to the chain only after being finalized.

The finality gadget works at the level of epochs. Instead of finalizing blocks one by one, the protocol uses checkpoint votes to finalize entire epochs. We now present in more detail how the finality gadget of Ethereum PoS grows the finalized chain.

Recall that to be taken into account, a checkpoint vote needs to be included in a block. The vote will then influence the behavior of validators regarding this particular branch. Thus, in Algorithm 9 of subsection III.2.2, the function `countMatchingCheckpointVote` only counts the matching checkpoint votes of attestations included in a block.

Justification The justification process is a step towards achieving finalization[6]. It operates on checkpoints at the level of epochs. Justification occurs thanks to checkpoint votes. The checkpoint vote contains a pair of checkpoints: the checkpoint *source* and the checkpoint *target*. We can count with `countMatchingCheckpointVote` the sum of balances of the validators' checkpoint votes with the same source and target. If validators controlling more than two-thirds of the stake make the same checkpoint vote, then we say there is a *supermajority link* from the checkpoint source to the checkpoint target. The checkpoint target of a supermajority link is said to be *justified*.

More formally, a checkpoint vote is in the form of a pair of checkpoints: $\big((a, e_a), (b, e_b)\big)$, also noted $(a, e_a) \to (b, e_b)$. For the checkpoint vote $(a, e_a) \to (b, e_b)$, we call (a, e_a) the checkpoint source and (b, e_b) the checkpoint target. The checkpoint source is necessarily from an earlier epoch than the checkpoint target, i.e., $e_a < e_b$. In line with [BHK+20], we say there is a *supermajority link* from checkpoint (a, e_a) to checkpoint (b, e_b) if validators controlling more than two-thirds of the stake cast an attestation with the checkpoint vote $(a, e_a) \to (b, e_b)$. In this case, we write $(a, e_a) \xrightarrow{\mathsf{J}} (b, e_b)$, and the checkpoint (b, e_b) is justified.

Finalization The finalization process aims at finalizing checkpoints, thus growing the finalized chain. Checkpoints need to be justified before being finalized. Let us illustrate the finalization process with the two scenarios that can lead to finalization. The first case presents the main scenario in the synchronous setting. It shows how a checkpoint can be finalized in two epochs, the minimum number of epochs needed for finalization.

Case 1: The scenario is depicted in Figure III.2.

[6] ↑ The genesis checkpoint (i.e., the checkpoint of the first epoch) is the exception to this rule: it is justified and finalized by definition.

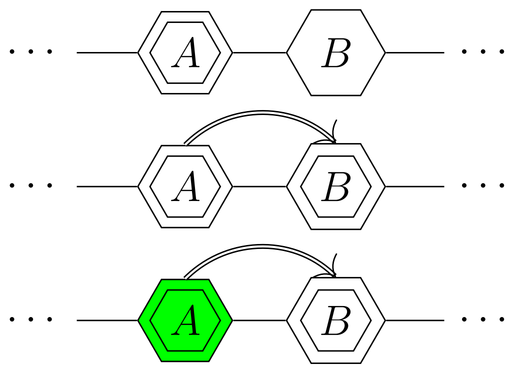

Figure III.2: The figure depicts the finalization scenario of **Case 1** with the 3 steps from top to bottom. We represent a checkpoint with a hexagon, a justified checkpoint with a double hexagon, and a finalized checkpoint with a colored double hexagon. The arrow between two checkpoints indicates a supermajority link.

1. Let $A = (a, e)$ and $B = (b, e+1)$ be checkpoints of two consecutive epochs such that $A = (a, e)$ is justified.
2. A supermajority link occurs between checkpoints A and B where A is the source and B the target. This justifies checkpoint B. Hence, we can write: $(a, e) \xrightarrow{J} (b, e+1)$ or equivalently $A \xrightarrow{J} B$.
3. This leads to A being finalized.

The second case illustrates the scenario in which two consecutive checkpoints are justified but not finalized. This means that the current highest justified checkpoint (e.g., B in Figure III.3) was not justified with a supermajority link having the previous checkpoint A as its source. Then, a new justification occurs with the source and target being at the maximum distance (2 epochs) for the source to become finalized. It is important to note that there is no limit on the distance between two checkpoints for justification to be possible. This limit only exists for finalization.

Case 2: The scenario is depicted in Figure III.3.

1. Let $A = (a, e)$, $B = (b, e+1)$, and $C = (c, e+2)$ be checkpoints of consecutive epochs such that A and B are justified. Since there is no supermajority link between A and B, A cannot be finalized as in Case 1.
2. Now, a supermajority link occurs between checkpoints A and C where A is the source and C the target. This justifies checkpoint C, i.e., $A \xrightarrow{J} C$.
3. This leads to A being finalized.

These two cases illustrate the fact that for a checkpoint to become finalized, it needs to be the source of a supermajority link between justified checkpoints. Once a checkpoint is finalized, all the blocks leading to it (including the block in

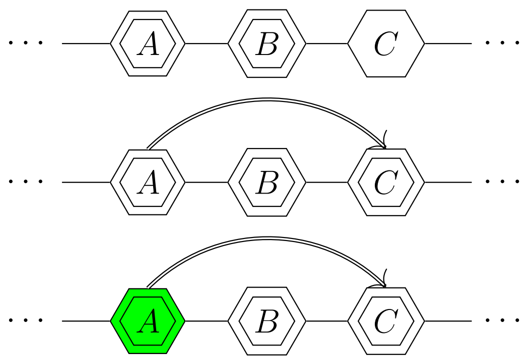

Figure III.3: The figure depicts the finalization scenario of **Case 2** with the 3 steps from top to bottom. We represent a checkpoint with a hexagon, a justified checkpoint with a double hexagon, and a finalized checkpoint with double hexagon coloured. The arrow between two checkpoints indicates a supermajority link.

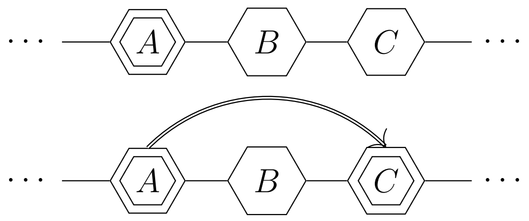

Figure III.4: This figure illustrates the case of two checkpoints A and C respecting all the conditions for finalization but the one that stipulates that a checkpoint B in-between must be justified for A to be finalized.

the pair constituting the checkpoint) become finalized. We now describe the conditions for a checkpoint to be finalized more formally. Let (a, e_a) and (b, e_b) be two checkpoints such that $e_a < e_b$. The checkpoint (a, e_a) is finalized if the following conditions are respected:

- **Source justified:** The checkpoint (a, e_a) is justified.
- **Supermajority link:** There exists a supermajority link $(a, e_a) \xrightarrow{\text{J}} (b, e_b)$.
- **Maximal gap:** $e_b - e_a \leq 2$.[7] Moreover, if $e_b - e_a = 2$, then the checkpoint in between at epoch $e_a + 1$ $(= e_b - 1)$ must be justified.

The importance of the last condition is illustrated by Figure III.4. In practice, these three conditions are only applied to the last four epochs. As mentioned in [BHK+20], at the implementation level, checkpoints more than 4 epochs old are not considered for finalization. All the conditions for finalization are illustrated by the last 4 conditions of Algorithm 9 in subsection III.2.2.

[7] ↑ This last condition necessitating the two checkpoints to be at most 2 epochs away from each other is also called *2-finality* [BHK+20].

III.2.1E Fork choice rule & Block proposition

The fork choice rule is the mechanism that allows each validator to determine the canonical chain depending on their view of the BlockTree and the state of checkpoints. The Ethereum PoS fork choice rule is LMD GHOST. The LMD GHOST fork choice rule stems from the Greedy Heaviest-Observed Sub-Tree (GHOST) rule [SZ15], which considers only each participant's most recent vote (Latest Message Driven). During an epoch, each validator must make one *block vote* on the block considered as the head of the canonical chain according to its view.

To determine the head of the canonical chain, the fork choice rule does the following:

1. Go through the list of validators and check the last block vote of each.

2. For each block vote, add a weight to each block of the chain that has the block voted as a descendant. The weight added is proportional to the stake of the corresponding validator.

3. Start from the block of the justified checkpoint with the highest epoch and continue the chain by following the block with the highest weight at each connection. Return the block without any child block. This block is the head of the canonical chain.

The actual implementation is presented in Algorithm 7 in subsection III.2.2. This algorithm is similar to the one already presented in [BHK$^+$20]. Albeit each *block vote* being for a specific block, the fork choice rule considers all the chains leading to that block. This reflects the fact that a vote for a block is a vote for the chain leading to that block. Figure III.5 offers an explanation with a visualization of how attestations influence the fork choice rule. At each chain intersection, the fork choice rule favors the chain with the most attestations.

III.2.1F Pseudo-Randomness

Ethereum's solution to incorporate randomness in the consensus is called *RANDAO*. RANDAO is a mechanism that creates pseudo-random numbers in a decentralized fashion. It works by aggregating different pseudo-random sources and mixing them.

Seed creation. Each epoch produces a seed. This seed is created with the help of the block proposers of the said epoch. Each valid block contains a field called `randao_reveal`[8]. The seed is the hash of an `XOR` of all the `randao_reveal` values of an epoch plus the epoch number.

[8] ↑ See subsection III.2.2 for a detailed explanation of its use.

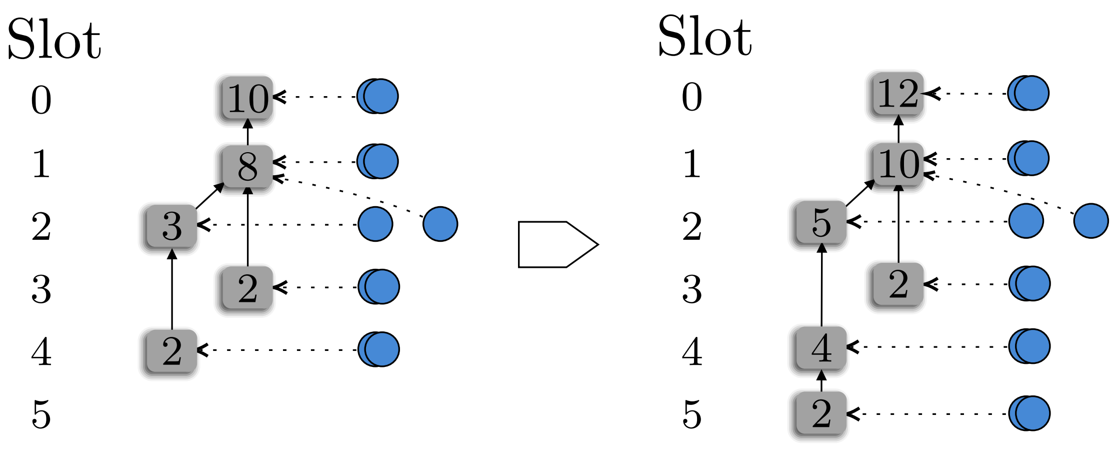

Figure III.5: Fork choice rule example observed from a validator i's point of view. We represent block votes with blue circles. Block votes point to specific blocks indicating the block considered as the *headblock* of the candidate chain at the moment of the vote. Each block has a number representing the value attributed by the fork choice rule algorithm (cf. Algorithm 7) to determine the candidate chain - we assume for this example that each validator has the same stake of 1. On the left we represent the chain at the end of slot 4, and on the right at the end of slot 5. On the left, i's fork choice rule gives the block of slot 4 as C_i's head. On the right, the fork choice rule designates the block of slot 5 as the head of the candidate chain.

Each block's `randao_reveal` must be the signature of specific data to prevent manipulation in the seed creation. The data to sign is the current epoch number. Anyone can then check that this signature is from the block proposer and for the correct data.

Seed utilization. The algorithm using the seed is called `compute_shuffled_index` (cf. Algorithm 12). This algorithm stems from the *swap-or-not* algorithm introduced by [HMR12]. `compute_shuffled_index` shuffles the validators list and assigns new roles depending on their shuffled index. This pseudo-random shuffling function is used two times in the Ethereum PoS consensus algorithm: for the proposer selection and the committee selection. The proposer selection is described in Algorithm 13 and the committee selection in Algorithm 14.

III.2.2 . Pseudo Code

This subsection can be skipped if the reader does not value the understanding on how we dissected the protocol. This step is as crucial as tedious, we outline the protocol based on the code that we formalized to be readable in the form of a pseudo-code.

In this section, we dive into a practical understanding of the mechanism behind the Ethereum PoS protocol. According to the specifications [Fou24] and various implementations (such as Prysm [Pry22] and Teku [Con22]), we formalize the main functions of the protocol through pseudo-code for better understanding and analysis purposes.

Each validator p runs an instance of this particular pseudo-code. For instance, when a validator p proposes a block, they broadcast the following message: $\langle PROPOSE, (slot,$ $\texttt{hash}(headBlock_p), content)\rangle$, where $slot$ is the slot at which the proposer proposes the block, the hash of the $headBlock_p$ is the hash of the block considered to be the head of the canonical chain according to the fork choice rule (see Algorithm 7), and $content$ contains data used for pseudo-randomness, among other things that we will not detail here. We instead focus on the consensus protocol.

We describe in the following paragraphs the variables and functions used in the pseudo-code and the goal of these functions.

Variables. During the computation, each variable takes a value that is subjective and may depend on the validator. We indicate with p that the value of variables depends on each process. The variable $tree_p$ is considered to be a graph of blocks with each block linked to its predecessor, representing the view of the blockchain (more precisely, $tree_p$ represents the view of all blocks received by the validator since the genesis of the system). Each $tree_p$ starts with the genesis block. $role_p$ corresponds to the different roles a validator can have, which can be none (i.e., for each slot, the validator can be proposer, attester, or have no role). $role_p$ is a list containing the role(s) of the validator for the current slot. The $slot_p$ is a measure of time. In particular, a slot corresponds to 12 seconds. $slot_p \in \mathbb{N}$. Slot 0 begins at the time of the genesis block and is incremented every 12 seconds. $headBlock_p$ is the head of the canonical chain according to p's local view and the fork choice rule.

A checkpoint C is a pair block-epoch that is used for finalization. C has two attributes, $justified$ and $finalized$, which can be true or false (e.g., if C is only justified, then $C.justified = \texttt{true}$ and $C.finalized = \texttt{false}$). $lastJustifiedCheckpoint_p$ is the *justified* checkpoint with the highest epoch. $currentCheckpoint_p$ is the checkpoint of the current epoch. The list $attestation_p$ is a list of size n (i.e., the total number of validators). This list is updated only to contain the latest messages of validators (of at most one epoch old). $CheckpointVote_p$ is a pair of checkpoints, so a pair of pairs, used to make a checkpoint vote. Let us stress the fact that all these variables are local, and at any time, two different validators may have different valuations of those variables.

Functions. We describe the main functions of the protocol succinctly before providing the associated pseudo-code and a more detailed explanation:

- `validatorMain` is the primary function of the validator, which launches the execution of all subsidiary functions.

- `sync` is a function that runs in parallel with the `validatorMain` function and ensures the synchronization of the validator. It updates the slot, the role(s), and processes justification and finalization at the end of the epoch and when a new validator joins the system.

- `getHeadBlock` applies the fork choice rule. This function indicates the head block of the canonical chain.

- `justificationFinalization` is the function that handles the justification and finalization of checkpoints.

We depict in Algorithm 1 the main procedure of the validator. This procedure initializes all the values necessary to run a validator. We consider the selection of validators already made to focus on the description of the consensus algorithm itself. The main procedure starts a routine called `sync` to run in parallel. Then there is an infinite loop that handles the call to an appropriate function when a validator needs to take action for its role(s).

Algorithm 1 Main code for a validator p

```
procedure validatorMain( )
    $tree_p \leftarrow nil$                                   ▷ The tree represents the linked received blocks
    $role_p \leftarrow [\,]$            ▷ $role_p$ can be ROLE_PROPOSER and/or ROLE_ATTESTER when it is not empty
    $slot_p \leftarrow 0$                                     ▷ $slot_p \in \mathbb{N}$
    $lastJustifiedCheckpoint_p \leftarrow (0, genesisBlock)$  ▷ A checkpoint is a tuple (epoch, block)
    $attestation_p \leftarrow [\,]$                           ▷ List of latest attestations received for each validator
    $validatorIndex_p \leftarrow$ index of the validator      ▷ Each validator has a unique index
    $listValidator \leftarrow [p_0, p_1, \dots, p_{N-1}]$     ▷ A list of the validators index
    $balances \leftarrow [\,]$                                ▷ A list of the balances of the validators, their stake

    start sync($tree_p, slot_p, attestation_p, lastJustifiedCheckpoint_p, role_p, balances$)

    while true do
        if $role_p \neq \emptyset$ then
            if ROLE_PROPOSER $\in role_p$ then
                prepareBlock()
            if ROLE_ATTESTER $\in role_p$ then
                prepareAttestation()
            $role_p \leftarrow [\,]$
        else
            no role assigned                                  ▷ No action required
```

The roles performed by the validator when acting as proposer or attester are defined in Algorithm 2 and Algorithm 3, respectively. The proposer of a block performs the following three tasks:

1. Get the head of its canonical chain to have a block to build upon.

2. Sign a predefined pair to participate in the process of pseudo-randomness.

3. Broadcast a new block built on top of the head of the canonical chain.

The attestation is composed of three parts: the slot, the block vote, and the checkpoint vote. The validator uses the fork choice rule presented in Algorithm 7 to obtain the block chosen for the block vote. Algorithm 7 and the one stemming from it, Algorithm 8, have already been defined in [BHK+20]. We restate them here for the sake of completeness. For the checkpoint vote, an honest validator should always vote for the current epoch as the target and take the justified checkpoint with the highest epoch (i.e., $lastJustifiedCheckpoint$) as the source.

In order to broadcast this attestation, the attester must wait for one of two things: either a block has been proposed for this slot, or $1/3$ of the slot (i.e., 4 seconds) has elapsed. This is ensured by the function `waitForBlockOrOneThird`.

Algorithm 2 broadcast block

```
1: procedure prepareBlock( )
2:     $headBlock_p \leftarrow$ getHeadBlock()
3:     $randaoReveal \leftarrow$ sign( epochOf($slot$))
4:     broadcast $\langle PROPOSE, (slot, \texttt{hash}(headBlock_p), randaoReveal_p, content)\rangle$
```

Algorithm 3 Broadcast Attestation

```
1: procedure prepareAttestation( )
2:     waitForBlockOrOneThird()        ▷ wait for a new block in this slot or $\frac{1}{3}$ of the slot
3:     $headBlock_p \leftarrow$ getHeadBlock()
4:     $currentCheckpoint_p \leftarrow$ (first block of the epoch, epochOf($slot$) )
5:     $CheckpointVote_p \leftarrow \left(lastJustifiedCheckpoint_p, currentCheckpoint_p\right)$
6:     broadcast $\langle ATTEST, (slot_p, \underbrace{\texttt{hash}(headBlock_p)}_{\text{block vote}}, \underbrace{CheckpointVote_p}_{\text{checkpoint vote}})\rangle$
```

The synchronization of the validator p is handled by the function `sync` described in Algorithm 4. This algorithm allows the validator to update its view of the blockchain, particularly the current slot, the list of attestations, the last justified checkpoint, the validator's role, and the balances of all validators. To determine its role(s), the validator verifies the index of the designated validator for the current slot and the set of indexes forming the committee of the current slot.

In more detail, two conditions assign a role to a validator for the current slot. The first condition calls Algorithm 13 and assigns the validator p the role of proposer if its index matches that of the current proposer. The second condition checks whether p belongs to the committee of the current slot (see Algorithm 14). The roles of proposer and attester are entirely distinct, i.e., the proposer of a slot is not necessarily an attester.

The synchronization function also starts two other routines, `syncBlock` and `syncAttestation`, corresponding to Algorithm 5 and Algorithm 6, respectively. These routines are used to handle the broadcasts from proposers and attesters.

In both functions, upon receiving a block or an attestation, the validator p verifies its validity using the `isValid` function. It is important to note that upon receiving a block, a validator can update the last justified checkpoint only if the current epoch has not started more than 8 slots ago. This particular condition is what the patch has introduced to prevent a liveness attack (see subsection III.3.2).

Algorithm 4 Sync

```
1: procedure sync($tree, slot, attestation, role, lastJustifiedCheckpoint,$)
2:     start syncBlock($slot, tree$)
3:     start syncAttestation($attestation$)
4:     repeat
5:         $previousSlot \leftarrow slot$
6:         $slot \leftarrow \lfloor$ time in seconds since genesis block / 12 $\rfloor$
7:         if $previousSlot \neq slot$ then                                   ▷ If we start a new slot
8:             roleSlotDone ← false
9:             if $validatorIndex_p$ = getProposerIndex(getSeed(current epoch), $slot$) then
10:                append ROLE_PROPOSER to $role_p$
11:            if $validatorIndex_p \in$ computeCommittee(getSeed(current epoch), $slot$) then
12:                append ROLE_ATTESTER to $role_p$
13:        if $slot \pmod{32} = 0$ then                                       ▷ First slot of an epoch
14:            jutificationFinalization($tree, lastJustifiedCheckpoint$)
15:    until validator exit
```

Algorithm 5 Sync Block

```
1: procedure syncBlock($slot, tree$)
2:     upon $\langle PROPOSE, (slot_i, \texttt{hash}(headBlock_i), randaoReveal_i, content_i)\rangle$ from validator $i$
   do
3:         $block \leftarrow \langle PROPOSE, (slot_i, \texttt{hash}(headBlock_i), randaoReveal_i, content_i)\rangle$
4:         if isValid($block$) then
5:             if $slot \pmod{32} \leq 8$ then
6:                 update justified checkpoint if necessary
```

Algorithm 6 Sync Attestation

```
1: procedure syncAttestation($attestation$)
2:     upon $\langle ATTEST, (slot_i, headBlock_i, checkpointEdge_i)\rangle$ from validator $i$ do
3:         $attestation_i \leftarrow \langle ATTEST, (slot_i, headBlock_i, checkpointEdge_i)\rangle$
4:         if isValid($attestation_i$) then
5:             $attestation[i] \leftarrow attestation_i$
```

Algorithm 9 can be considered the most intricate. This algorithm is responsible for justifying or finalizing the checkpoints at the end of each epoch. To do so, it counts the number of checkpoint votes with the same source and target. If this number corresponds to more than 2/3 of the stake of all validators, then the target is considered justified for the validator running this algorithm. The last four

Algorithm 7 Get Head Block

```
1: procedure getHeadBlock( )
2:     $block \leftarrow$ block of the justified checkpoint with the highest epoch
3:     while $block$ has at least one child do
4:         $block \leftarrow \underset{b' \text{ child of } block}{\arg\max}$ weight$(tree, Attestation, b')$
5:         (ties are broken by hash of the block header)
6:     return $block$
```

Algorithm 8 Weight

```
1: procedure weight($tree, Attestation, block$)
2:     $w \leftarrow 0$
3:     for every validator $v_i$ do
4:         if $\exists a \in Attestation$ an attestation of $v_i$ for $block$ or a descendant of $block$ then
5:             $w \leftarrow w+$ stake of $v_i$
6:     return $w$
```

conditions concern finalization. They verify among the last four checkpoints which one fulfills the conditions to become finalized. The conditions to become finalized are formally described in subsection III.2.1 and can be summarized as follows: the checkpoint must be the source of a supermajority link, and all the checkpoints between the source and target, inclusive, must be justified.

Algorithm 9 Justification and Finalization

```
1: procedure jutificationFinalization($tree, lastJustifiedCheckpoint$)
2:     $source \leftarrow lastJustifiedCheckpoint$
3:     $target \leftarrow$ the current checkpoint
4:     $nbCheckpointVote \leftarrow$ countMatchingCheckpointVote($source, target$)
5:     ▷ justification process:
6:     if $nbCheckpointVote \geq \frac{2}{3} *$ total balance of validators then
7:         $target.justified \leftarrow$ true
8:         $lastJustifiedCheckpoint \leftarrow target$
9:     ▷ finalization process:
10:    $A, B, C, D \leftarrow$ the last 4 checkpoints          ▷ With $D$ being the current checkpoint.
11:    if $A.justified \wedge B.justified \wedge (A \xrightarrow{J} C)$ then
12:        $A.finalized \leftarrow$ true                       ▷ Finalization of $A$
13:    if $B.justified \wedge (B \xrightarrow{J} C)$ then
14:        $B.finalized \leftarrow$ true                       ▷ Finalization of $B$
15:    if $B.justified \wedge C.justified \wedge (B \xrightarrow{J} D)$ then
16:        $B.finalized \leftarrow$ true                       ▷ Finalization of $B$
17:    if $C.justified \wedge (C \xrightarrow{J} D)$ then
18:        $C.finalized \leftarrow$ true                       ▷ Finalization of $C$
```

The pseudo-randomness requires a different seed for each epoch to yield different results. This is ensured by hashing the RANDAO mix and the epoch number, as shown in Algorithm 11. Adding the epoch number is helpful if no block is proposed during an entire epoch. This corner case would always result in the same

Algorithm 10 Get randao mix

```
1: procedure getRandaoMix($epoch$)
2:     $mix \leftarrow 0$
3:     $headBlock \leftarrow$ getHeadBlock()
4:     for each $block$ parent of $headBlock$ and belonging to $epoch$ do
5:         $mix \leftarrow mix \oplus$ hash($block$.randaoReveal)        ▷ $\oplus$ is a bit-wise XOR operator
6:     return $mix$
```

Algorithm 11 Get seed

```
1: procedure getSeed($epoch$)
2:     $mix \leftarrow$ getRandaoMix($epoch - 2$)        ▷ The seed of an epoch $i$ is based on the randao
   mix of epoch $i - 2$
3:     return hash($epoch + mix$)
```

seed if it were not for the epoch number.

The RANDAO mix is computed in Algorithm 10. The computation of the RANDAO mix for a given epoch consists of *XORing* all the *randaoReveal* values of the blocks in that particular epoch. We consider only the blocks of that particular epoch that belong to the canonical chain.

The RANDAO mix of epoch $e - 2$ determines the role of validators in epoch e. Hence, with Algorithm 14, as soon as epoch $e - 2$ is over, validators can know to which committee they belong at epoch e. `computeCommittee` (Algorithm 14) is the function that, given a seed and an epoch, returns the list of validator indices corresponding to the committee for the specified slot. The number of validators in each committee[9] is computed to be less than $N/32$ (with n being the total number of validators). Then, using the shuffled index computed with Algorithm 12, a committee of the given size is drawn according to the slot in question. All committee validators will have to perform the role of attester during this slot.

Since the balance can change until the previous epoch, block proposers are known at the end of epoch $e - 1$ for epoch e. Algorithm 13 handles the selection of a proposer for a designated slot. It starts by creating a seed specifically for the slot in question. Then, a loop starts with a pseudo-random selection of the validator's index. The loop stops only when a validator meets the condition criteria. This condition is equivalent to being selected with a probability depending on the balance. Thus, the validator with index *proposerIndex* is selected with probability $\frac{\textit{effectiveBalance}}{32}$, with *effectiveBalance* being the stake of *proposerIndex* capped at 32, i.e., $\min(\textit{balance}, 32)$.

Both Algorithm 13 and 14 use Algorithm 12 to imbue randomness in the proposer and committee selection. As mentioned in section III.2, Algorithm 12 stems from the algorithm *swap-or-not* [HMR12]. Its name helps us understand the principle behind the algorithm: select a validator and its opposite (based on a pivot)

[9] ↑ In the actual implementation, committees have a maximum size of 2048 [Fou24].

and swap them or not. The selection of the validator and the swap depend on the value of a hash. An essential aspect of this algorithm is that it can get the index of validators in the shuffled list without having to compute the shuffling of the whole list of validators. This reduces unnecessary computation.

Algorithm 12 Compute shuffled index

```
1: procedure computeShuffledIndex($index, seed, nbValidators$)
2:     for $i = 0$ to $90$ do
3:         $pivot \leftarrow$ hash($seed + i$) $(mod\ \ $nbValidators$)$
4:         $flip \leftarrow pivot + nbValidators - index\ (mod\ \ $nbValidators$)$
5:         $position \leftarrow \max(index,\ flip)$
6:         $bit \leftarrow$ hash($seed + i + position$) $\pmod 2$
7:         if $bit = 0\ (mod\ \ $nbValidators$)$ then
8:             $index \leftarrow flip$
9:     return $index$
```

Algorithm 13 Get proposer index

```
1: procedure getProposerIndex($seed$, $slot$)
2:     MAX_RANDOM_BYTE $\leftarrow 2^8 - 1$
3:     $i \leftarrow 0$
4:     $proposerSeed \leftarrow$ hash($seed$+$slot$)
5:     $nbValidators \leftarrow$ length($listValidator$)
6:     while true do
7:         $proposerIndex \leftarrow listValidator$[computeShuffledIndex($i, seed, nbValidators$)]
8:         $randomByte \leftarrow$ first byte of hash($proposerSeed + i\ \pmod{nbValidators}$)
9:         effectiveBalance $\leftarrow listValidators[proposerIndex]$.effectiveBalance
10:        if effectiveBalance $*$ MAX_RANDOM_BYTE $\geq$ MAX_EFFECTIVE_BALANCE $*randomByte$ then
11:            return $proposerIndex$
12:        $i \leftarrow i + 1$
```

Algorithm 14 Compute Committee

```
1: procedure computeCommittee($seed$, $slot$)
2:     $committee \leftarrow [\,]$
3:     nbValidatorByCommittee $\leftarrow \lceil$lenght$(listValidator)/32\rceil$
4:     for $i$ = $(slot\ \pmod{32})*$nbValidatorByCommittee to $(slot + 1\ \pmod{32})*$nbValidatorByCommittee $-1$ do
5:         $committee$.append($listValidator$[computeShuffledIndex($i, seed, nbValidators$)])
6:     return $committee$
```

III.3 . Robustness Analysis

We now have formalized the protocol and the blockchain properties necessary for our analysis. We will start by analyzing if the protocol is safe, and then if is live.

III.3.1 . Safety

In order to prove the safety of the protocol, we begin by presenting lemmas concerning the justification of checkpoints. The first lemma rules out the possibility of two different justified checkpoints having the same epoch. New validators that want to join the set of validators must send the amount they wish to stake to a specific smart contract[10]. This transaction triggers the process for a validator to join the set of validators. The last step required for the activation of a validator (allowing it to send attestations and propose blocks) requires that the block adding the validator to the validator set gets finalized[11]. This means that the set of validators is fixed between two finalized checkpoints.

Lemma III.1. *If checkpoints C and C' of the same epoch e are justified, it must necessarily be that $C = C'$.*

Proof. By hypothesis, we know that Byzantine validators are at most $f < n/3$. For the sake of contradiction, let us assume that C and C' are different checkpoints. Let V be the set of at least $2n/3-f$ honest validators that cast a checkpoint vote for checkpoint C in epoch e, and V' be the set of at least $2n/3-f$ honest validators that cast a checkpoint vote for checkpoint C' in epoch e. The intersection of the two sets of honest validators is $|V \cap V'| \geq (2n/3-f)+(2n/3-f)-(n-f) = (n/3-f) > 0$. $|V \cap V'| > 0$ implies that at least one honest validator voted for both checkpoint C and checkpoint C' in epoch e. This is a contradiction since, according to the protocol specification[12], an honest process signs at most one unique block per epoch. Therefore, $C = C'$. This proves there cannot be more than one justified checkpoint per epoch. □

The following lemma explains why the finalization of a checkpoint necessarily means that a checkpoint cannot be justified on a different chain afterward.

Lemma III.2. *If a checkpoint C of epoch e is finalized on chain c, and a checkpoint C' of epoch e' is justified on chain c' with $e' > e$, it necessarily means that c and c' have a common prefix until epoch e.*

Proof. C' being justified on chain c' means that at least $2n/3-f$ honest validators must have cast a checkpoint vote with C' as the checkpoint target for epoch e'.

[10] ↑ Currently, 32 ETH is needed to become a validator.

[11] ↑ The exact process involves placing the validator in the activation queue to be finalized. See more at https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md.

[12] ↑ This is specified in the specs `https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#attester-slashing`, and implemented in the actual client Prysm `https://github.com/prysmaticlabs/prysm/blob/0fd52539153e32cfbd0a27ee51f253f8f6bb71c4/validator/client/attest.go#L140`. This corresponds to the only attestation done by an honest validator during an epoch, see Algorithm 4.

For the sake of contradiction, let us say that c and c' have a common prefix until epoch $e-1$ at most. For a checkpoint to be justified on chain c' at an epoch strictly superior to e, it implies that a set V' of at least $2n/3 - f$ honest validators must have cast a checkpoint vote with a checkpoint target on chain c' and a checkpoint source with an epoch less than $e - 1$.

Checkpoint C of epoch e being finalized on chain c, we have two possibilities. Either the checkpoint at epoch $e+1$ on chain c has been justified with checkpoint C as the source, or the checkpoint at epoch $e+2$ on chain c has been justified with checkpoint C as the source, and the checkpoint at epoch $e+1$ is justified. Either way, a justification occurred on chain c with checkpoint C as the source, and no justification occurred on a different chain before its finalization.

Hence, we know that a set V of at least $2n/3 - f$ honest validators have cast a checkpoint vote with C as the checkpoint source before a justification on any other chain.

Seeing that $|V \cap V'| > 0$, at least one honest validator has cast a checkpoint vote with C as the checkpoint source and then a checkpoint vote with a checkpoint source of at most epoch $e - 1$ and a target on chain c'.

Therefore, at least one honest validator has cast a checkpoint vote with a checkpoint source from an epoch less than $e - 1$ after seeing checkpoint C at epoch e justified. However, the fork choice rule of the protocol (cf. Algorithm 7) requires honest validators to vote on the chain with the highest justified checkpoint. This contradiction proves the lemma. □

We saw with Lemma III.1 that two checkpoints of the same epoch could not be justified, hence finalized. We then showed with Lemma III.2 that after a finalization on one chain, no checkpoints could become justified on any other chain. These are the conditions required to have safety, as we prove now.

Theorem III.1 (Safety)**.** *There cannot be two finalized checkpoints on different chains in Ethereum PoS.*

Proof. Thanks to Lemma III.1, we know that two different checkpoints C and C' of the same epoch cannot be justified, hence finalized.

For the sake of contradiction, let us assume that two checkpoints C and C' are finalized on different chains c and c' at epochs e and e', respectively. We assume without loss of generality that $e < e'$. C being finalized, we know thanks to Lemma III.2 that C' cannot be justified on a different chain c', let alone be finalized. □

The blockchain preserves the property of safety at all times. The Ethereum PoS is safe.

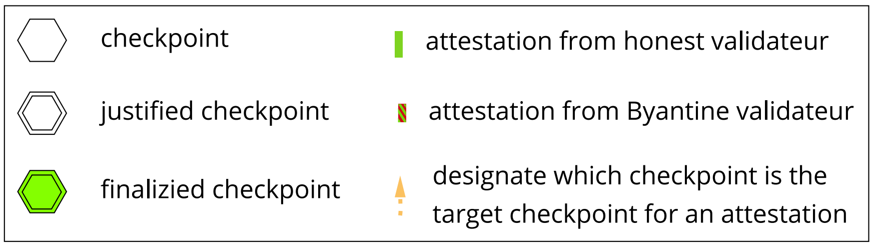
checkpoint
justified checkpoint
finalizied checkpoint
attestation from honest validateur
attestation from Byantine validateur
designate which checkpoint is the target checkpoint for an attestation

Figure III.6: This figure serves as a summary of the signification of the main diagrams of other figures.

III.3.2 . Probabilistic Liveness

Spoiler alert: the protocol is not guaranteed to be live; rather, it is probabilistically live. This means that as time goes on, the probability of it being live approaches 1. However, the probability that it is not live is not zero, although it remains very small. To prove this point, we will explain an attack that targets the protocol's liveness.

In order to explain this attack which is by no means simple we start by describing a simpler liveness attack called the *bouncing attack*. This attack delays finality in a partially synchronous network after GST. Previous works also exhibit liveness attacks against the protocol using the intertwining of the fork choice rule and the finality gadget [Nak19a, NTT21]. To prevent this attack, the protocol now contains a "patch" [Req19] suggested on the Ethereum research forum [Nak19b]. We show that the implemented patch is insufficient, and this attack is still possible if certain conditions are met. This is a probabilistic liveness attack against the Ethereum Proof-of-Stake protocol. Our attack can happen with less than 1/3 of Byzantine validators, as discussed in subsubsection III.3.3A. We also consider the adversary to be static because Byzantine validators are chosen before the computation.

III.3.2A Bouncing Attack

The *Bouncing Attack* [Nak19a] describes a liveness attack where the suffix of the chain changes repetitively between two canonical chains, thus preventing the chain from finalizing any checkpoint. The Bouncing Attack exploits the fact that the canonical chains should start from the justified checkpoint with the highest epoch. It is possible for Byzantine validators to divide honest validators' opinions by justifying a new checkpoint once some honest validators have already cast their vote (made an attestation) during the asynchronous period before GST.

The bouncing attack becomes possible once there is a *justifiable* checkpoint in a different branch from the one designated by the fork choice rule with a higher epoch than the current highest justified checkpoint. A *justifiable checkpoint* is a checkpoint that can become justified only by adding the checkpoint votes of Byzantine validators. If this setup occurs, the Byzantine validators could make honest validators start voting for a different checkpoint on a different chain, leav-

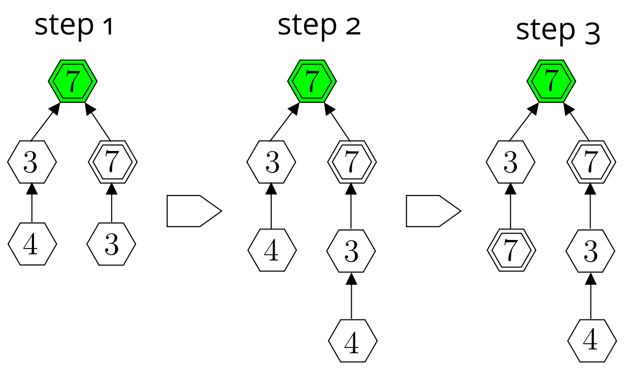

Figure III.7: A bouncing attack presented in 3 steps. We have 10 validators, of which 3 are Byzantines. The number inside each hexagon corresponds to the number of validators who made a checkpoint vote with this checkpoint as target. **1st step:** We start in a situation where there is a fork. A checkpoint is justified on one of the chains and a checkpoint of a higher epoch is *justifiable* on the other. We are at the end of the third epoch in which honest validators have divided their vote on each side. **2nd step:** We have reached `GST` at the beginning of the fourth epoch and 4 honest validators have already voted (rightfully so). **3rd step:** Here is the moment Byzantine validators take action and release their checkpoint vote for the concurrent chain, thus justifying the previously forsaken checkpoint and thereby changing the highest justifying checkpoint. By repeating this process, the bouncing attack can continue indefinitely.

ing a justifiable checkpoint again for them to repeat their attack and thus making validators *bounce* between two different chains and not finalizing any checkpoint. Hence the name Bouncing attack.

Let us illustrate the attack with a concrete case. In Figure III.7, we show an oversimplified case with only 10 validators, among which 3 are Byzantine. To occur, the attack needs to have a justifiable checkpoint with a higher epoch than the last justified checkpoint. We reach this situation before `GST`, which is presented in the left part of the figure. After reaching `GST`, Byzantine validators wait for honest validators to make a new checkpoint justifiable. When a new checkpoint is justifiable, the Byzantine validators cast their votes to justify another checkpoint, as shown in the right part of the figure. This will lead honest validators to vote for the left branch, thus reaching a situation similar to the first step, allowing the bouncing attack to continue. The repetition of this behavior is the bouncing attack. We emphasize this example in more detail in Figure III.8 by detailing the sequence of votes allowing a "bounce" to occur and leaving a justifiable checkpoint on the other branch.

III.3.3 . Implemented Patch

The explanation of the patch is described for the first time on the Ethereum research forum [Nak19b]. The solution found to mitigate the bouncing attack is to engrave in the protocol the fact that validators cannot change their minds regarding justified checkpoints after a part of the epoch has passed.

The goal of the proposed solution is to prevent the possibility of justifiable

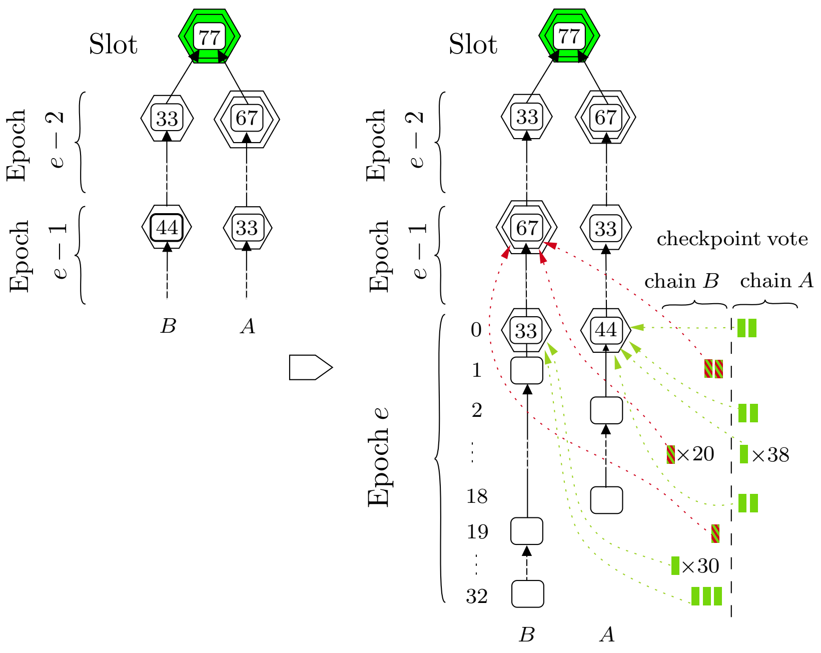

Figure III.8: This figure presents a detailed version of the bouncing attack. In this example, we have a total of 100 validators, of which 23 are Byzantines. A block in a checkpoint corresponds to the block associated with that checkpoint. The number inside each hexagon (hovering a block) corresponds to the number of validators who made a checkpoint vote with this checkpoint as target. We distinguish between two sorts of checkpoint votes, the Byzantine ones, which are bi-color rectangles, and the honest ones, which are uni-color rectangles. We compile the 3 steps of Figure III.7 in 2 with more information on how justification's turning point is accomplished because of the Byzantine agents. **First step:** We begin from a situation where epoch $e-1$ just ended and we now reach `GST`. Notice that the canonical chain is chain A because the checkpoint with the highest epoch is on chain A but not chain B. **Second step:** In this step, the checkpoint vote released during epoch e can change the last justified checkpoint to change the canonical chain for chain A to chain B. Byzantine validators released their checkpoint vote from the previous epoch during epoch e. They send their last checkpoint vote at slot 23 once the checkpoint of epoch e on chain A has reached 44, thus becoming justifiable (i.e., not yet justified but with enough votes so that Byzantine validators can justify it). This triggers the canonical chain to change from chain A to chain B starting the *bounce*.

checkpoints being left out by honest validators. To prevent honest validators from leaving a justifiable checkpoint, the patch must stop validators from changing their view of checkpoints before more than 1/3 of validators have cast their checkpoint vote. This condition stems from the fact that we reckon the proportion of Byzantine validators to be at most $1/3 - \epsilon$. To apply this condition, the patch designates a number of slots after which honest validators cannot change their view of checkpoints. Since validators are scattered equally among the different slots to cast their vote (in attestations) within a specific time frame, stopping validators from changing their view after a certain number of slots is equivalent to stopping them from changing their view after a certain proportion of validators have voted. This does appear to be a solution to prevent Byzantine validators from influencing honest validators into forsaking a checkpoint that is now *justifiable* for them.

To enforce this behavior, called the "fixation of view," the protocol has a constant j called `SAFE_SLOTS_TO_UPDATE_JUSTIFIED` in the code (cf. Algorithm 5 in subsection III.2.2). This constant is the number of slots[13] during which validators can change their view of the justified checkpoints. The patch introducing this constant j mentions a possible attack called the *splitting attack*. As they point out, the splitting attack relies on a "last minute delivery" strategy whereby releasing a message late enough causes some validators to consider it too late while others do not. This could split the validators into two different chains, unable to reconcile their views before the end of the epoch. After the beginning of the next epoch views can be reconciled during j slots however the split can occur once again by another last minute delivery. They consider the assumption that attackers can send a message at the right time to split honest validators too strong. In subsubsection III.3.3A, we present a new attack inspired by the splitting attack with more realistic assumptions.

III.3.3A Probabilistic Bouncing attack - why the patch is not enough

In this part, we present our novel attack against the protocol of Ethereum Proof-of-Stake. The attack is visually explained in Figure III.9.

Attack Condition. Our attack takes place during the synchronous period and uses the power of *equivocation* by Byzantine processes. Equivocation is caused by a Byzantine process that sends a message only to a subset of validators at a given point in time and potentially another message or none to another subset of validators. The effect is that only a part of the validators will receive the message on time. More in detail, the bounded network delay is used by a Byzantine validator to convey a message to be read on a specific slot by some validators and read

[13] ↑ At the time of writing this manuscript, $j = 8$ [Fou24].

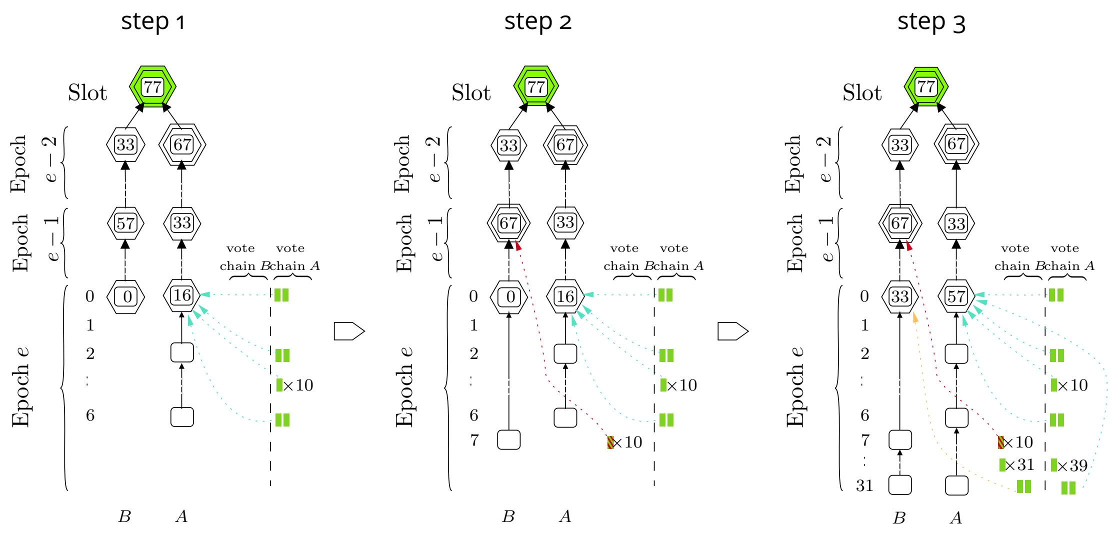

Figure III.9: This figure presents the probabilistic bouncing attack. In this example, we consider 100 validators, of which 10 are Byzantine. A block in a checkpoint corresponds to the block associated with that checkpoint. The number inside each hexagon (hovering a block) corresponds to the number of validators who made a checkpoint vote with this checkpoint as the target. The example starts at the slot before the attack in step 1. `GST` has been reached in epoch e and honest validators have started to vote on chain A. This is the correct action because the justified checkpoint with the highest epoch is on chain A (at epoch $e-2$). During the next slot in step 2, before reaching $limitSlot$, a Byzantine validator sends a block with withheld votes for the checkpoint at epoch $e-1$ on chain B. It is released just in time for a set of honest validators to consider it and too late for the remaining validators. The honest validators that see the block in time will update their view of the justified checkpoint with the highest epoch and consider chain B as the canonical chain. We now show how the epoch continues with step 3. The block produced by a Byzantine, having been released just in time, causes $(1/3)$ of honest validators to change their view. This results in a situation where Byzantine validators can perform the same attack during the next epoch provided that at least one Byzantine validator is selected to be block proposer on chain A for one of the first 8 slots.

on the next slot by the other validators. Note that if a protocol is not tolerant to equivocation, then it is not BFT (Byzantine Fault Tolerant), since equivocation is the typical action possible for Byzantine validators.

Attack Description and Analysis Let $\beta \leq f/n$ be the fraction of Byzantine validators in the system. The attack setup is the following. First, as in the traditional bouncing attack, we start in a situation where the network is still partially synchronous. A fork occurs and results in the highest justified checkpoint being on chain A at epoch e, and a justifiable checkpoint at epoch $e+1$ on chain B. Assume now that `GST` is reached. The attack can proceed[14] as follows:

1. Since `GST` is reached, the network is fully synchronous. Chain A is the canonical chain for all validators.

2. Just before validators must stop updating their view concerning the justified checkpoint (i.e., before reaching the limit of j slots[15] in the epoch corresponding to the condition in line 6 of Algorithm 5), a Byzantine proposer proposes a block (cf. Algorithm 2) on chain B. This block contains attestations with enough checkpoint votes to justify the justifiable checkpoint left by honest validators. The attestations included in the block are those of Byzantine validators that were not issued in the previous epoch when they were supposed to be. The block must be released just in time, that is, right before the end of slot j, so that $(1/3 - \beta)$ of the validators change their view of the canonical chain to be active on chain B while the rest of the honest validators continue on chain A. This is possible due to the patch preventing validators from changing their mind after j slots.

3. Repeat the process.

An important aspect to consider in the attack is the probability of Byzantine validators becoming proposers. This is crucial because, without the role of proposer, validators cannot propose blocks and add new attestations containing checkpoint votes on the concurrent chain.[16] The probability of being selected to be a proposer directly impacts how long the probabilistic bouncing attack can continue. In the following theorem, we establish the probability of a probabilistic bouncing attack lasting for a specific number of epochs.

[14] ↑ Note that before `GST`, no algorithm can ensure liveness since communication delays may not be bounded.

[15] ↑ At the time of writing, 8 slots.

[16] ↑ Note that Byzantine validators cannot use their role as proposer during the previous epoch to release a block with the right attestations because it might not be the last block of the epoch. Indeed, because some honest validators are on the concurrent chain, they add blocks. The checkpoint votes contained in the Byzantine attestations must be on the same chain as the attestation to justify the justifiable checkpoint, making the checkpoint justifiable in the first place.

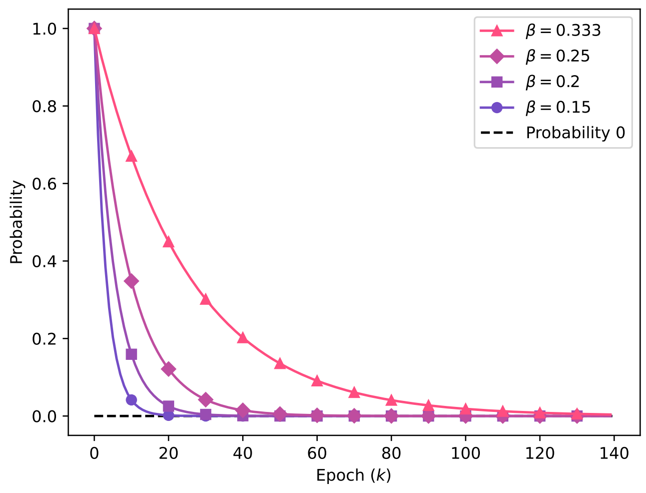

Figure III.10: The figure presents the probability of the bouncing attack depending on the proportion of Byzantine validators β and the number k of epochs during which the bouncing attack lasts. The probability is computed based on Equation III.1, knowing that $\alpha = 1-\beta$.

Theorem III.2. *The probabilistic bouncing attack occurs during k epochs after `GST` and a favorable setting with probability:*

$$P(bouncing\ k\ times) = (1-\alpha^j)^k, \tag{III.1}$$

with $\alpha \in [0,1]$ being the proportion of honest validators and j the number of slots before locking a choice for justification.

Proof. We denote by α the proportion of honest validators and j the number of slots before locking the choice for justification. We want to know the probability of delaying the finality for k epochs. Once we assume a setup condition sufficient to start a probabilistic bouncing attack, the attack continues until it becomes impossible for Byzantine validators to cast a vote to justify the justifiable checkpoint. To cast their vote, Byzantine validators need one of the j first slots of the concurrent chain to have a Byzantine validator as proposer. Considering the probability of choosing between each validator, the chance for a Byzantine validator to be a proposer for one of the first j slots is $(1-\alpha^j)$, with α being the proportion of honest validators. For k epochs, we take this result to the power of k. □

We depict the probability of the bouncing attack over time with several proportions of Byzantine validators β in Figure III.10. The closer the proportion of Byzantine validators is to $1/3$, the higher the probability of the attack lasting for k epochs (for any k).

The probability of the bouncing attack continuing for k epochs depends on two factors: α $(= 1-\beta)$, the proportion of honest validators that cannot be controlled, and j, the number of slots before which validators are allowed to switch branches.

Reducing j to 0 would prevent the bouncing attack from happening (the probability falls to 0), but it would mean that validators are never allowed to change their view of the canonical chain. This naive solution would create irreconcilable choices among the set of validators and prevent any new checkpoint from being justified, which is a more severe threat to the liveness of Ethereum PoS.

Reducing the number of slots during which validators can change their view of the blockchain implies that different views cannot reconcile quickly. At the very least, the window of opportunity for doing so gets smaller. Theoretically, the proportion of Byzantine validators necessary to perform this attack is $1/n$. This is because we assume a favorable setup and that Byzantine validators can send messages so that only a desired portion of honest validators receive them on time. Our analysis focuses on the course of action of the attackers during the attack rather than the conditions necessary for it to occur.

This analysis highlights the delicate balance between fixing the view of validators and letting them change their view too much. There is a non-zero probability for the attack to last k epochs for any k. However, the probability starts to plummet rapidly. With a proportion of $\beta = 0.3$, the probability of the attack lasting 100 epochs[17] is 0.02. Nonetheless, there is a non-negligible probability of delaying the finalization for several hours even with $\beta \approx 0.25$.

We conjecture that with the current design of Ethereum PoS, it is impossible to completely avoid such issues. Attempts to patch this probabilistic bouncing attack may not ensure a safe or live protocol; i.e., mitigating one attack might give rise to other vulnerabilities. For example, committee-based blockchains with single-shot finality employ complex systems to prevent conflicting and irreconcilable views [ADPT18, APPT19, YMR+19]. These mechanisms typically require the exchange of messages from a quorum of validators to update one's perspective. However, such mechanisms are not feasible in Ethereum PoS since halting the blockchain's availability is not an option. More broadly, this issue aligns with existing literature [NTT21, LPR20] and their connections to the CAP theorem [Bre00]. Consequently, a possible straightforward solution could be to transition to a classical BFT consensus protocol with single-slot finality. Yet, such a change would fundamentally alter the protocol and should not be considered a mere mitigation.

III.4 . Conclusion

In this study, we propose a novel distinction between the definitions of blockchain liveness and availability properties. This distinction is crucial for pinpointing differences between Nakamoto-style consensus and BFT (Byzantine Fault Tolerance) consensus, enabling a comparison between the two. We describe a framework for a high-level description of how the Ethereum PoS (Proof of Stake) protocol

[17] ↑ 100 epochs is about 10 hours.

functions—a committee-based BFT protocol strongly inspired by Nakamoto-style consensus. Through this formalization, we demonstrate that the Ethereum PoS protocol satisfies the safety property and that no conflicting blocks can be finalized on different branches. We also present patches implemented to mitigate some attacks demonstrated on previous and preliminary versions of the protocol. However, we exhibit an attack on finalization, showing that the Ethereum PoS protocol exhibits probabilistic liveness.

We supplement our analysis with an examination of the protocol's rewards and incentives, which were not considered here, in Chapter IV. We are interested in how the incentive mechanism can impact the safety of the protocol as well as the behavior of rational validators. We intend to implement the probabilistic attack and simulate its outcome while considering penalties.

Such analyses could be conducted empirically to closely monitor the actual behavior of the validators, especially in scenarios where an attack occurs.

— Chapter IV —

Ethereum PoS Analysis under the Distributed Computing Model with Penalties

Contents

Beginning with the consensus part of the protocol without the incentives, it now remains to consider them. We knew from the beginning that taking into account the incentives would yield different results. However the complexity of the protocol was such that a first analysis focusing solely on the consensus part was necessary. We start this chapter by precising the network model for this analysis. It is very similar to the previous one but note that we define an *initial* Byzantine proportion because it will vary during our analysis. We also redefine some properties and explain the protocol succinctly as a remainder. Then comes the analysis of the incentive mechanism called *inactivity leak*.

IV.1 . System Model

The model for our analysis is defined in Chapter II. For more clarity, there is a point of clarification regarding the network conditions.

During our analysis, we assume a network configuration where, during asynchronous periods, honest validators are split into two distinct partitions. Communication between these partitions is restricted, simulating a scenario where two regions are temporarily isolated from each other but maintain internal communication within each region. This setup emulates a situation where two regions of the world are temporarily unreachable from one another while maintaining unaffected communication within each region.

IV.2 . Protocol and Properties

In this section, we briefly remind the reader of the essential properties and definitions, as well as the necessary elements of the protocol previously described.

Ethereum PoS Properties Validators keep a local data structure in the form of a tree containing all the blocks perceived, and then a consensus protocol helps to choose a unique chain in the tree. Ethereum has a particular trait that consists of having a *finalized* chain as the prefix of a chain vulnerable to forks. A metaphor for this is that the finalized chain is the trunk that possibly supports various branches, and as time passes, the trunk grows and branches are trimmed[1].

Intuitively, the Safety property of Ethereum states that the finalized chain is not forkable, while the Liveness property states that the finalized chain always grows. The nuance with respect to classical consensus protocols is the existence of an Availability property on the entire chain that guarantees constant growth of the chain despite failures and network partitions. The complete definition can be found in section III.1.

Based on the definition of safety, we consider forks within the finalized chain as a loss of Safety. As explained in the subsequent section, forks occurring within the candidate chain suffix, which has not yet been finalized, are resolved by the fork choice rule of the protocol. This rule determines the chain upon which validators vote and build. However, this rule has not been explicitly designed to handle forks impacting the finalized chain.

The protocol is not intended to fork the finalized chain, as the finalization process depends on a super-majority vote, ensuring Safety when the Byzantine stake is less than one-third, i.e., $\beta_0 < 1/3$. We look at two types of Safety loss: (1) the finalization of two conflicting chains, and (2) the break of the Safety threshold, meaning the Byzantine stake proportion is more than one-third.

IV.3 . Safety Attack

[1] ↑ We use the terms "chain" and "branch" interchangeably.

The attestation contains two votes, a *block vote* and a *checkpoint vote*. The block vote is used in the *fork choice rule*, which determines the chain to vote and build upon for validators. As its name suggests, the checkpoint vote points to checkpoints constituting the chain. It is used to justify and finalize blocks to grow the finalized chain. Justification is the step prior to finalization. If validators controlling over two-thirds of the stake make the same checkpoint vote, then the checkpoint target is justified. Finalization occurs when there are two consecutive justified checkpoints (one in epoch e and the following one in epoch $e+1$).

Let us note that if justification occurs only every other epoch, finalization is not possible.

IV.3.1 . Incentives

The Ethereum PoS protocol provides validators with rewards and penalties to incentivize timely responses for reaching consensus. There are three different types of penalties: slashing, attestation penalties, and inactivity penalties.

(i) *Slashing penalties.* Validators face slashing if they provably violate specific protocol rules, resulting in a partial loss of their stake and expulsion from the validator set.

(ii) *Attestation penalties.* To incentivize timely and correct attestations (votes), the protocol rewards validators for adhering to the protocol and penalizes those who do not. If an attestation is missing or belatedly incorporated into the chain, its validator gets penalized.

(iii) ***Inactivity penalties.*** Each epoch a validator is deemed inactive, its *inactivity score* increments. However, if the protocol is not in an inactivity leak, all inactivity scores are reduced.

When finalization occurs regularly, a validator that is deemed inactive only receives attestation penalties. This changes when there is no finalization for four consecutive epochs: the inactivity leak begins. During the inactivity leak, which starts when there is no finalization for four consecutive epochs, all validators will receive inactivity penalties directly linked to their stake and *inactivity score*. The inactivity score varies with the validator's activity.

In addition to penalties, rewards are attributed for timely and correct attestations but not during the inactivity leak. Our analysis of the impact of the inactivity leak on the protocol takes into consideration the slashing and inactivity penalties across five different scenarios (cf. section IV.5).

Having provided a comprehensive overview of the Ethereum PoS consensus mechanism, we are now well-positioned to delve into the specifics of the *inactivity leak*.

IV.4 . Inactivity Leak

The Ethereum PoS blockchain strives for the continuous growth of the finalized chain. Consequently, the protocol incentivizes validators to actively finalize blocks. In the absence of finalization, validators incur penalties.

The inactivity leak, introduced in [BG17], serves as a mechanism to regain finality. Specifically, if a chain has not undergone finalization for four consecutive epochs, the inactivity leak is initiated. During the inactivity leak, the stakes of *inactive* validators are drained until active validators amount to two-thirds of the stake. A validator is labeled as inactive for a particular epoch if it fails to send an attestation or sends one with a wrong target checkpoint.

During the inactivity leak, there are no more rewards given to attesters[2], and additional penalties are imposed on inactive validators.

IV.4.1 . Inactivity Score

The *inactivity score* is a dynamic variable that adjusts based on a validator's activity. The inactivity score of a validator is determined based on the attestations contained in the chain. It is important to note that if there are multiple branches, a validator's inactivity score depends on the selected branch. Within an epoch, being active on one branch implies[3] inactivity on another (for honest validators).

More precisely, the inactivity score is updated every epoch: if validator i is active, then its inactivity score is reduced by 1; otherwise, 4 is added to it. When the inactivity leak is not in place, the inactivity scores are decreased by 16 every epoch, which often nullifies low inactivity scores.

During an inactivity leak, at epoch t, the inactivity score, $I_i(t)$, of validator i is:

$$\begin{cases} I_i(t) = I_i(t-1) + 4, & \text{if } i \text{ is inactive at epoch } t \\ I_i(t) = \max(I_i(t-1) - 1, 0), & \text{otherwise.} \end{cases} \tag{IV.1}$$

Each attester thus has an inactivity score that fluctuates depending on its (in)activity. In the protocol, the inactivity score is always greater than zero. A validator's inactivity for epoch t is determined by whether it sent an attestation for this epoch and if the sent attestation contains a correct checkpoint vote. Here "correct" implies that the target of the checkpoint vote belongs to the considered chain.

IV.4.2 . Inactivity penalties

Validators that are deemed inactive incur penalties. Let $s_i(t)$ represent the stake of validator i at epoch t, and let $I_i(t)$ denote its inactivity score. The penalty at each epoch t is $I_i(t-1) \cdot s_i(t-1)/2^{26}$. Therefore, the evolution of the stake is expressed by:

$$s_i(t) = s_i(t-1) - \frac{I_i(t-1) \cdot s_i(t-1)}{2^{26}}. \tag{IV.2}$$

[2] ↑ Actually, the only rewards that remain are for the block producers and the sync committees.

[3] ↑ This is true as long as the chains differ for at least one epoch.

IV.4.3 . Stake's functions during an inactivity leak

In this work, we model the stake function s (see Equation IV.2) as a continuous and differentiable function, yielding the following differential equation:

$$s'(t) = -\frac{I(t) \cdot s(t)}{2^{26}}. \tag{IV.3}$$

We then explore three distinct validator behaviors during an inactivity leak, each influencing their inactivity score and, consequently, their stake.

(a) Active validators: they are always active.

(b) Semi-active validators: they are active every two epochs.

(c) Inactive validators: they are always inactive.

Note that, in the case of a fork, this categorization depends on the specific branch under consideration as different branches may yield different evaluations of each validator's behavior.

This categorization is orthogonal to the Byzantine-Honest categorization. For instance, an honest validator can appear inactive in one branch due to poor connectivity or an asynchronous period (due to network partition or congestion). On the other hand, a Byzantine validator intentionally chooses one of these behaviors (e.g., being semi-active) to execute the attacks.

We illustrate in Figure IV.1 the evolution of the validators' stake depending on their behaviors. We also account for the ejection of validators with a stake lower than or equal to 16.75.

Using these newly defined stake functions, we explore five scenarios in section IV.5.

The first scenario, with only honest validators, serves as a baseline to assess the impact of Byzantine validators. Even in this seemingly straightforward setting, Safety is compromised.

In the second scenario, Byzantine validators come into play and aim to expedite the finalization of conflicting branches. They do so by performing slashable actions. Thus, they will get ejected from the set of validators once communication is restored among honest validators and evidence of their slashable offense is included in a block. We outline their impact based on their initial stake proportion. With an initial stake proportion of $\beta_0 = 0.2$, the finalization on conflicting chains occurs after 3107 epochs. With $\beta_0 = 0.33$, the conflicting finalization occurs only after 503 epochs.

In the third and fourth scenarios, Byzantine validators exhibit non-slashable behaviors. Specifically, Byzantine validators are semi-active, meaning they are active on both chains but in a non-slashable manner. In the third scenario, they aim to finalize conflicting branches as soon as possible, achieving conflicting finalization in 556 epochs with an initial stake proportion of $\beta_0 = 0.33$. In the fourth

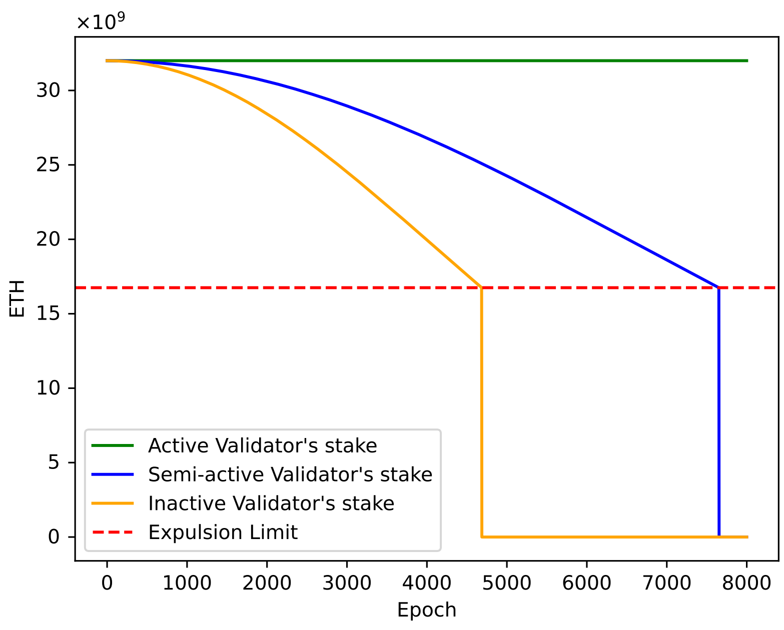

Figure IV.1: This figure shows three different stake trajectories in the event of an inactivity leak: the stake of a validator active every epoch, the stake of a validator active every other epoch (semi-active), and an inactive validator. The inactive validators get ejected at epoch $t = 4685$. The semi-active validators get ejected at epoch $t = 7652$. For reference, 5000 epochs is about 3 weeks.

scenario, their goal is to increase their stake proportion to exceed the $1/3$ threshold.

The last scenario delves into the effect of the probabilistic bouncing attack regarding the Byzantine stake proportion, considering the inactivity leak. In this attack, Byzantine validators initially aim to delay finality by being alternately active (bouncing) on both chains of a fork. This confuses honest validators, causing them to also bounce from one chain to the other. We detail how to find the distribution of honest validators' stakes in this setting, considering the inactivity penalties. We also cover how the Byzantine validators' stake proportion can exceed $1/3$ if their initial proportion is close to $1/3$.

The scenarios unfold within the context of a partially synchronous network while offering a meticulous examination of the property of Safety and the evolution of the proportion of Byzantine validators. Each scenario's initial conditions and outcomes are summarized in Table IV.1.

IV.5 . Analysis

In this section, we study the robustness of the Safety property within the context of the inactivity leak. By construction, in the case of a prolonged partition, two different chains can potentially be finalized, leading to conflicting finalized blocks.

Scenario		Outcomes
IV.5.1	All honest	2 finalized branches
IV.5.2A	Slashable Byzantine	2 finalized branches
IV.5.2B	Non slashable Byzantine	2 finalized branches
IV.5.2C	Non slashable Byzantine	$\beta > 1/3$
IV.5.3	Probabilistic Bouncing attack	$\beta > 1/3$ probably

Table IV.1: Analyzed scenarios associated with their outcomes. Initially the proportion of Byzantine's stake is smaller than $1/3$ and is zero for the first scenario.

We delineate scenarios that can produce such a predicament.

Considering the presence of Byzantine validators, we study how the proportion of Byzantine validators' stake evolves during an inactivity leak. Furthermore, we are interested in scenarios where the inactivity leak mechanism becomes the backbone of an attack strategy, potentially causing the proportion of Byzantine stakes to exceed the $1/3$ security threshold (cf. subsubsection IV.5.2C and subsection IV.5.3).

IV.5.1 . GST upper bound for Safety

In this first subsection, we look for an upper bound on GST before which no finalization on conflicting chains can happen in case of a partition. We study the case of an inactivity leak with these conditions: (i) only honest validators, no Byzantine validators, and (ii) the network is asynchronous (before GST).

In case of catastrophic events, during an instance of a particularly disrupted network, an arbitrarily large set of honest validators might be unreachable before GST. During this asynchronous period, the subset of validators still communicating with each other will continue to try to finalize new blocks. We assume that, within each partition, the message delay is bounded as in the synchronous period; however, communication between partitions is not restored before GST. As the system model mentions, Byzantine validators can communicate between partitions without restriction but cannot manipulate the message delay between honest validators. The active validators must represent more than two-thirds of the stake to be able to finalize. After 4 epochs without finalization, the inactivity leak starts.

All the validators deemed inactive will have their stake progressively reduced. This will continue until the active validators constitute at least two-thirds of the stake and can finalize anew.

Two finalized chains A noteworthy scenario arises during asynchronous periods that can lead to a network partition and the creation of two distinct finalized chains. If this partition persists for an extended period, both chains independently drain the stakes of validators they consider inactive until they finalize again. Although the protocol permits this behavior by design, it results in finalizing two conflicting chains, thereby compromising the Safety property.

This outcome aligns with Ethereum PoS prioritizing Liveness over Safety. However, to the best of our knowledge, this corner case has not been discussed in detail.

We can theoretically assess the time required to finalize both branches of the fork. Suppose honest validators remain in their respective branches due to the partition. In this case, by understanding the distribution of these validators across the partitions, we can compute the time it takes for the proportion of active validators' stake to return to $2/3$ of the stake on each branch, permitting new finalization.

Let n_{H} and n_{B} denote the initial number of honest validators and Byzantine validators at the beginning of the inactivity leak ($n_{\mathrm{H}} + n_{\mathrm{B}} = n$). Additionally, n_{H_1} and n_{H_2} represent the number of honest validators active on branch 1 and on branch 2, respectively ($n_{\mathrm{H}_1} + n_{\mathrm{H}_2} = n_{\mathrm{H}}$).

We denote by $p_0 = n_{\mathrm{H}_1}/n_{\mathrm{H}}$ the initial proportion of honest validators remaining active on branch 1, and $1 - p_0 = n_{\mathrm{H}_2}/n_{\mathrm{H}}$ represents the proportion of honest validators active on branch 2 (hence inactive on branch 1). In this first scenario, with only honest validators and no Byzantine validators, p_0 represents the proportion of all validators active on branch 1. Indeed, since $n_{\mathrm{H}}/n = 1$, we have that $n_{\mathrm{H}_1}/n_{\mathrm{H}} \times n_{\mathrm{H}}/n = p_0$.

We have assessed how validators' stakes vary based on their level of activity. Consequently, we can express the ratio of active validators on branch 1 at time t as:

$$\frac{n_{\mathrm{H}_1} s_{\mathrm{H}_1}(t)}{n_{\mathrm{H}_1} s_{\mathrm{H}_1}(t) + n_{\mathrm{H}_2} s_{\mathrm{H}_2}(t)}, \tag{IV.4}$$

with s_{H_1} and s_{H_2} being the stake of honest active and inactive validators, respectively. We know the function of their stake according to time, and by dividing the numerator and the denominator by the total number of validators ($n = n_{\mathrm{H}}$), we can rewrite Equation IV.4 as:

$$\frac{p_0}{p_0 + (1 - p_0)e^{-t^2/2^{25}}}. \tag{IV.5}$$

The initial stake value s_0 is factored out of the equation. This function is critical as the moment it reaches $2/3$ or more, finalization can occur[4] on the branch.

To establish the upper bound on GST under which two conflicting branches finalize, we must find when finalization occurs on each branch for each initial proportion of active validators p_0 and inactive validators $1 - p_0$. We simulate the evolution of the ratio of active validators (Equation IV.5) during an inactivity leak with

[4] ↑ With a proportion of two-thirds of validators' stake active, justification and then finalization can occur in 2 epochs.

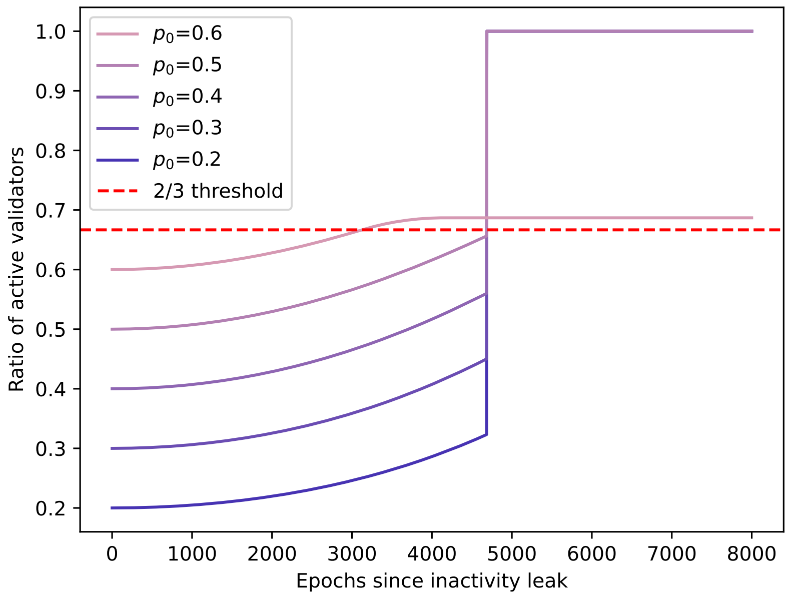

Figure IV.2: Evolution of the ratio of active validators depending on the proportion p_0 of active validators on the branch. This follows the ratio given in Equation IV.5 before regaining $2/3$ of active validators or the expulsion of inactive validators at epoch $t = 4685$.

different values of p_0 in Figure IV.2. The simulation starts with both active and inactive validators at 32 ETH. At epoch 0, the inactivity leak begins.

For $p_0 = 0.5$ or less, the ratio jumps to 1 at $t = 4685$; this is due to the fact that validators with a stake below 16.75 ETH are ejected from the set of validators. Conversely, for $p_0 = 0.6$, the proportion of active validators does not jump drastically as $2/3$ of active validators is regained before the ejection of inactive validators, permitting the active validators to finalize, hence ending the inactivity leak. Interestingly, with $p_0 = 0.6$ we can see that the ratio still increases several epochs after the proportion of $2/3$ of active validators' stake is reached. This is because the penalties for inactive validators take some time to return to zero.

As expected and shown by Figure IV.2, a chain with more active validators will regain finality faster. To ascertain how quickly, we seek when the ratio is equal to 2/3. Taking into account the expulsion[5] of inactive validators at $t = 4685$, we can find the value t at which the $2/3$ threshold is reached:

$$t = \min\left(\sqrt{2^{25}[\log(2(1-p_0)) - \log(p_0)]}, 4685\right). \tag{IV.6}$$

This calculation pertains to $0 < p_0 < 2/3$ (when there are less than $2/3$ of active validators), ensuring that the epoch t can be computed.

Conflicting finalization occurs once the slowest branch to finalize has regained finality. Our observation highlights that the lower the proportion of active validators, the slower the branch will regain finality. Hence, the fastest way to reach

[5] ↑ We drew inspiration for this initial work from the insights presented in [Edg3].

finality on both chains would be for honest validators to be evenly proportioned, with half of the validators active on one chain and the other half on the other chain ($p_0 = 1 - p_0 = 0.5$). In this case, the ratio of active validators amounts to $2/3$ on both chains at $t = 4685$ epochs (about 3 weeks). We can note here that even with the best configuration to finalize quickly on conflicting branches, it is impossible to be faster than 4685 epochs. Thus, with only honest validators, whatever their proportion on each branch, the last chain to finalize will always finalize at $t = 4685$.

Finality on both chains is achieved precisely at 4686 epochs after the beginning of the inactivity leak. Adding an epoch is necessary after gaining $2/3$ of active stake to finalize the preceding justified checkpoint. This finalization ends the inactivity leak, which has lasted approximately 3 weeks. *Any network partition lasting longer than 4686 epochs will result in a loss of Safety because of conflicting finalization. This is an upper bound for Safety on the duration of the inactivity leak with only honest validators.*

IV.5.2 . Upper bound decrease due to Byzantine validators

In a trivial setup with only honest validators, Safety does not hold if the inactivity leak is not resolved quickly. This prompts us to study the scenario in the presence of Byzantine validators to evaluate how much they will be able to hasten the conflicting finalization. We describe two possible outcomes: the first one violates Safety, but Byzantine validators get slashed; the second one violates Safety as well, but no validators get slashed. A slashing penalty entails an ejection from the validator set as well as a loss of part of the validator's stake. Both scenarios expedite the time t at which Safety is breached, with different velocities depending on the chosen method.

We study the inactivity leak under these conditions: (i) at the beginning, less than one-third of the stake is held by Byzantine validators ($\beta_0 = n_\mathrm{B}/n < 1/3$), with the rest held by honest validators ($1-\beta_0 = n_\mathrm{H}/n$); (ii) the network is asynchronous (before `GST`); and (iii) Byzantine validators are not affected by network partitions.[6]

The situation is as follows:

- Honest validators are divided into branches 1 and 2; a proportion $p_0 = n_{\mathrm{H}_1}/n_\mathrm{H}$ of the honest validators are active on branch 1, while a proportion $1 - p_0 = n_{\mathrm{H}_2}/n_\mathrm{H}$ are active on branch 2. This means that on branch 1, a proportion $n_{\mathrm{H}_1}/n_\mathrm{H} \times n_\mathrm{H}/n = p_0(1 - \beta_0)$ are honest and active, and a proportion $n_{\mathrm{H}_2}/n_\mathrm{H} \times n_\mathrm{H}/n = (1 - p_0)(1 - \beta_0)$ are honest and inactive.

- Byzantine validators are not restricted to either partition; they are connected

[6] ↑ In a model without partitions, one needs to give Byzantine validators more power to recreate our scenario. They must be able to control the network delay to allow them to be active on both branches while preventing honest validators from even observing the branch on which they are not active. They can manipulate message delays between groups of honest validators to simulate a partition between them.

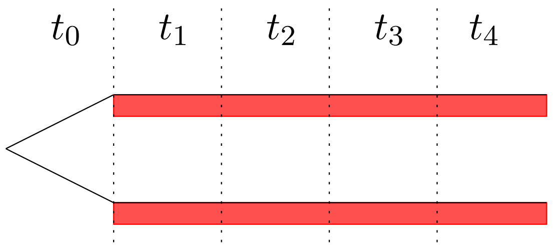

Figure IV.3: Byzantine validators are active on both chains of a fork simultaneously during asynchronous times.

to both.

IV.5.2A With slashing

In the event of a fork during asynchronous times, Byzantine validators can be active on both branches (Figure IV.3). Being active on two branches means sending correct attestations on both every epoch. Such behavior is considered a slashable offense, incurring penalties, but only if detected by honest validators. The slashable offense is punished once proof of conflicting attestations during the same epoch has been included in a block. Thus, before GST, Byzantine validators could operate on both branches without facing punishment as long as honest validators are unaware of the conflicting attestations. Byzantine validators have control over the message delay before GST, making this behavior possible. They can thereby expedite the finalization on different branches.

We study here the time needed for finalization to occur on conflicting branches depending on the proportion of Byzantine validators. The ratio of active validators at epoch t is:

$$\frac{n_{\mathrm{H}_1} s_{\mathrm{H}_1}(t) + n_{\mathrm{B}} s_{\mathrm{B}}(t)}{n_{\mathrm{H}_1} s_{\mathrm{H}_1}(t) + n_{\mathrm{B}} s_{\mathrm{B}}(t) + n_{\mathrm{H}_2} s_{\mathrm{H}_2}(t)}, \tag{IV.7}$$

with s_{H_1}, s_{B}, and s_{H_2} being the stake of honest active, Byzantine active, and honest inactive validators, respectively. This can be rewritten as:

$$\frac{p_0(1-\beta_0) + \beta_0}{p_0(1-\beta_0) + \beta_0 + (1-p_0)(1-\beta_0)e^{-t^2/2^{25}}}. \tag{IV.8}$$

where β_0 represents the initial proportion of Byzantine validators, and p_0 denotes the initial proportion of honest active validators. In contrast to the analysis with only honest validators (cf. Equation IV.5), here, Byzantine validators are present and active on both chains. Nonetheless, as before, we can obtain the ratio of active validators on the other branch just by interchanging p_0 and $1-p_0$. Finality on conflicting branches occurs when the last of the two branches finalizes. Similarly to the previous example, the branch with the fewer initial honest active validators (p_0) will finalize the latest. This happens t epochs after the beginning of the inactivity leak, with

$$t = \min\left(\sqrt{2^{25}\left[\log(2(1-p_0)) - \log(p_0 + \frac{\beta_0}{1-\beta_0})\right]}, 4685\right). \qquad \text{(IV.9)}$$

Finality on conflicting branches is achieved the quickest when honest validators are evenly split between the branches of the fork, at $p_0 = 0.5$.

β_0	t
0	**4685**
0.1	4066
0.15	3622
0.2	3107
0.33	502

Table IV.2: Time before finalization on conflicting branches depending on the initial proportion of Byzantine validators β_0 for $p_0 = 0.5$ with slashing behaviour based on Equation IV.9

Table IV.2 gives the epoch at which concurrent finalization occurs for $p_0 = 0.5$. This outlines the rapidity at which finality can be regained depending on the initial proportion β_0 of Byzantine validators' stake. The table shows that 503 epochs (approximately 2 days) could suffice to finalize blocks on two different chains, but hypothetically it could be quicker than that. In fact, as β_0 gets closer to $1/3$, the number of epochs required before concurrent finalization occurs (Equation IV.9) approaches 0.

The explanation is that if β_0 were to start at exactly 1/3, then with $p_0 = 0.5$, it would mean that on each branch we would start with $p_0(1-\beta_0)+\beta_0 = 2/3$ of active validators, hence finalizing immediately. This explains why if β_0 is very close to 1/3, the proportion of active validators reaches $2/3$ rapidly. *Hence, Byzantine validators can expedite the loss of Safety. If their initial proportion is 0.33, they can make conflicting finalizations occur approximately ten times faster than scenarios involving only honest participants.*

One can notice that if Byzantine validators act in a slashable manner, they will be penalized after the asynchronous period ends. However, the harm is already done. Once the finalization on two branches has occurred, the branches are irreconcilable with the current protocol. Next, we demonstrate that Byzantine validators can employ more subtle strategies to break Safety without slashable actions.

IV.5.2B Without Slashing

Byzantine validators can hasten the violation of the Safety property without incurring a slashable offense. While not as rapid as being active on both branches

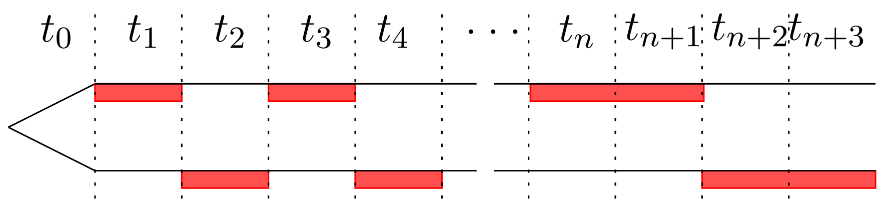

Figure IV.4: Byzantine validators active on both branches of a fork alternatively during asynchronous times.

simultaneously, they can be semi-active on both branches alternatively. Being semi-active on each branch means they are only active every other epoch. This approach diminishes their stake on each branch due to inactivity penalties. Nevertheless, at some point, they can finalize on two conflicting branches by being active two epochs in a row on one branch and then on the other (see Figure IV.4). Byzantine validators will be able to finalize when the proportion of their stake plus the proportion of the stake of honest active validators is above $2/3$ on the branch (cf. Equation IV.10).

At that point, Byzantine validators must remain active for two consecutive epochs on each branch to finalize them both. If they are only semi-active, they can alternate justifications for checkpoints on each branch but will not achieve finalization. However, by maintaining activity for two consecutive epochs, first on one branch and then on the other, they ensure two sequential justifications, leading to the finalization of a checkpoint.

We gave the different evolution of stakes depending on the activity of validators (subsection IV.4.3). Now that Byzantine validators are semi-active, their stake follows the curve $s_0 e^{-3t^2/2^{28}}$. We simplify the ratio as previously and we get that finalization occurs on the branch when the ratio

$$\frac{p_0(1-\beta_0)+\beta_0 e^{-3t^2/2^{28}}}{p_0(1-\beta_0)+\beta_0 e^{-3t^2/2^{28}}+(1-p_0)(1-\beta_0)e^{-t^2/2^{25}}} \tag{IV.10}$$

goes over 2/3, with β_0 and p_0 being the initial proportion of Byzantine validators and the proportion of honest active validators on the branch, respectively.

In contrast to the previous scenario, obtaining an analytic solution for t to determine the epoch when the ratio hits $2/3$ is not straightforward. Therefore, we apply numerical methods on Equation IV.10 with initial parameters $p_0 = 0.5$ and $\beta_0 = 0.33$, resulting in a calculated t value of 555.65. This means it will take 556 epochs to finalize, about 2 days and a half.

As previously, the proximity of β_0 to $1/3$ significantly influences the speed of finalization, as outlined in Table IV.3 and Figure IV.5. Figure IV.5 shows how the proportion of Byzantine validators affects the time of conflicting finalization. Notice that although the acceleration is not as pronounced as in the previous scenario, it remains noteworthy that Byzantine validators still exert a substantial impact on

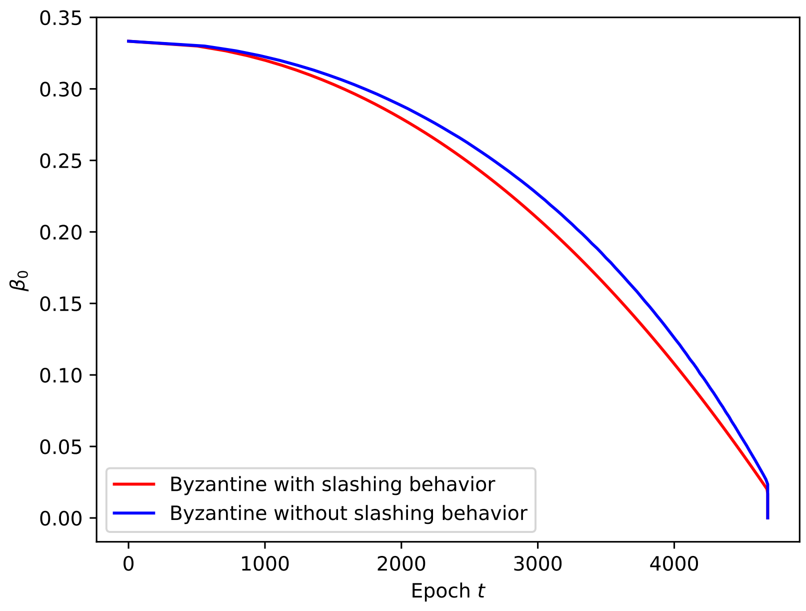

Figure IV.5: Time before finalization on conflicting branches, depending on the initial proportion of Byzantine validators β_0 and whether they engage in slashable actions.

breaching Safety, while not committing any slashable offense.

Hence, Byzantine validators can expedite the loss of Safety without committing any slashable action. If their initial proportion is 0.33*, they can make conflicting finalizations occur approximately eight times faster than scenarios involving only honest participants.*

β_0	t
0	**4685**
0.1	4221
0.15	3819
0.2	3328
0.33	556

Table IV.3: Time before finalization on conflicting branches depending on the initial proportion of Byzantine validators β_0 for $p_0 = 0.5$ without slashing behavior based on Equation IV.10.

Another consequence of being "semi-active" on both branches is that Byzantine validators can decide when to finalize on each branch. Indeed, even when the proportion of their stake plus the proportion of honest active validators' stake is above $2/3$, finalization only occurs when the Byzantine validators stay active for two consecutive epochs on the same chain. Being active for two epochs will justify the two consecutive epochs, thus finalizing an epoch.

There exists a scenario in which the Byzantine validators might delay finalization intentionally, aiming to increase their stake's proportion beyond the threshold of $1/3$ without incurring slashing afterward.

IV.5.2C More than one third of Byzantine validators

One may ask, why would Byzantine validators aim at going over the $1/3$ threshold? Indeed, we have just shown that Safety can be broken regardless of β_0; is it not the ultimate goal of Byzantine validators? It is not obvious to determine what behavior will harm the blockchain the most. We briefly discuss the impact Byzantine validators can have when they go over the $1/3$ threshold in subsection IV.5.3. We now examine the necessary conditions on β_0 and p_0 that permit the Byzantine validators' stake to go over the one-third threshold.

The key ratio that translates into what we are looking for is the proportion of Byzantine validators' stake $\beta(t, p_0, \beta_0)$ over time:

$$\frac{\beta_0 e^{-3t^2/2^{28}}}{p_0(1-\beta_0) + (1-p_0)(1-\beta_0)e^{-t^2/2^{25}} + \beta_0 e^{-3t^2/2^{28}}} \tag{IV.11}$$

As expected, at time $t = 0$, $\beta(0, p_0, \beta_0) = \beta_0$. Now, let us investigate when this ratio is above the threshold of 1/3, i.e.:

$$\beta(t, p_0, \beta_0) \geq 1/3 \tag{IV.12}$$

The main difference with the previous scenario is that Byzantine validators seek to go over the $1/3$ threshold, not to finalize quickly. This means that even after the proportion of honest active validators' stake and semi-active Byzantine validators' stake represents more than two-thirds of the stake on the branch, they do not finalize. Byzantine validators could finalize by staying active for two epochs in a row, yet they do not do so in order to reach a higher stake proportion.

We construct a set containing the pairs (p_0, β_0) that can lead β to go over $1/3$ (Equation IV.12). To do so, we take the point reached by the ratio when the validators deemed inactive are ejected. This point gives the highest value reachable[7] for a particular (p_0, β_0). For an intuition as to why this is the case, Figure IV.1 allows us to visualize that the biggest gap between semi-active Byzantine stake and honest inactive stake is at the moment of expulsion of the honest inactive validators. We have seen that inactive validators are ejected from the chain after 4685 epochs.

[7] ↑ There exist more values that can lead to going over one-third when considering a special corner case. If the Byzantine validators strategically finalize just before the expulsion of honest inactive validators, the decrease in inactivity penalties might not occur quickly enough to prevent the ejection of honest inactive validators. In this particular scenario, Byzantine validators could potentially eject honest inactive participants while incurring fewer penalties themselves. This subtlety underscores the intricate dynamics at play during the inactivity leak.

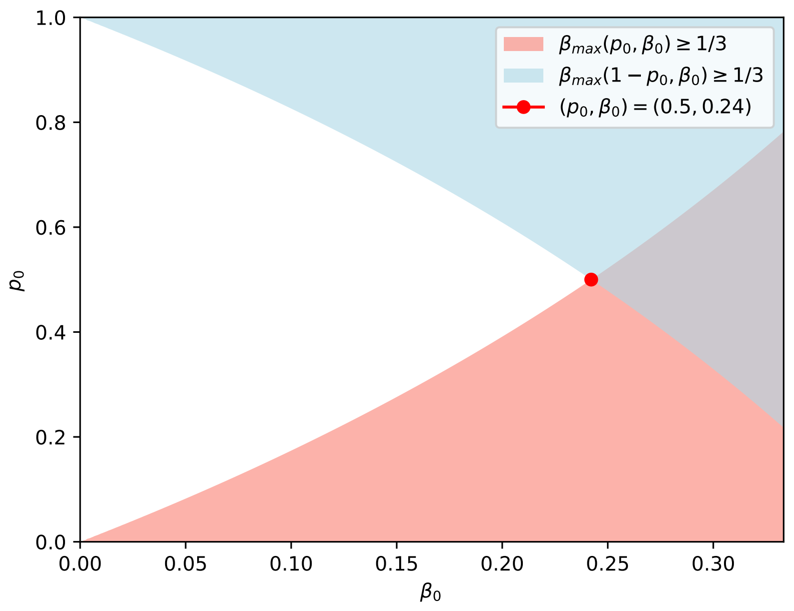

Figure IV.6: Pairs (p_0, β_0) such that $\beta_{\max}(p_0, \beta_0) \geq 1/3$. This figure gives a lower bound for which (p_0, β_0) can result in the proportion of Byzantine validators exceeding $1/3$ on both branches.

We can thus evaluate the maximum ratio reachable $\beta_{\max}$ at time $t = 4685$ when the inactive validators are ejected:

$$\beta_{\max}(p_0, \beta_0) = \frac{\beta_0 e^{-3\times(4685)^2/2^{28}}}{p_0(1-\beta_0) + \beta_0 e^{-3\times(4685)^2/2^{28}}}. \tag{IV.13}$$

When this ratio is greater than 1/3, Byzantine validators have reached their goal. We show with Figure IV.6 that Byzantine validators can actually go beyond the threshold of $1/3$ on both branches simultaneously. The lower bound β_0 before this becomes possible is for $p_0 = 0.5$ when $\beta_0 = 1/1 + 4e^{-3\times(4685)^2/2^{28}} = 0.2421$.

When the initial proportion of Byzantine validators is at least 0.2421, their proportion can eventually increase to more than $1/3$ of validators on both branches, exceeding the critical Safety threshold of voting power in each branch.

Having explored scenarios in which protocol vulnerabilities manifest exclusively before `GST`, we now focus on potential threats posed by Byzantine validators after `GST`. Given the acknowledged impact of the *Probabilistic Bouncing Attack* on Liveness (cf. subsubsection III.3.3A), our study extends to take the inactivity leak into account.

IV.5.3 . Revisiting the Probabilistic Bouncing Attack

This subsection revisits the Probabilistic Bouncing Attack subsubsection III.3.3A, showing that Byzantine validators could exceed the Safety threshold even during the synchronous period. Contrary to the previous scenarios, this one starts in the asynchronous period but unfolds in the synchronous period. This demonstrates that the inactivity leak poses significant challenges even within the synchronous period, revealing its broader implication for blockchain security.

As mentioned, while analyzing the probabilistic bouncing attack, we did not consider the penalties. Here, we fill this gap.

Let us note that there is no problem with conflicting finalization as the attack is progressing after `GST` in the synchronous period. In synchronous time, there is not enough delay for honest validators to miss a finalization on another branch. There would need to be more than two-thirds of the active stakes owned by Byzantine validators to break Safety in the synchronous period.

We briefly discussed the differences in gravity between conflicting finalization and having more than $1/3$ of the stake owned by Byzantine validators. We left the actual comparison and the in-depth analysis of the gravity of going beyond the infamous threshold as future work.

We primarily focus on identifying specific scenarios that would disrupt the network. Thus, we give a detailed explanation of a scenario that could lead to Byzantine validators breaking the $1/3$ threshold even during synchronous period (after `GST`).

Let us remind how the attack takes place for self-containment.

Probabilistic Bouncing Attack Summary The attack can be summarized as follows: (1) A favorable setup that partitions honest validators into two different views of the blockchain occurs. (2) At each epoch, Byzantine validators withhold their messages from honest validators, releasing them at the opportune time to make some honest validators change their view. (3) This attack continues as long as at least one Byzantine validator is proposer in the j^{th} first slots of the epoch, where j is a parameter of the protocol. The probability of the attack continuing for k epochs with a proportion of $(1-\beta_0)$ honest validators is $(1-(1-\beta_0)^j)^k$.

We start by analyzing the outcome of a fork where a proportion p_0 of the *honest* validators start on chain A and $1-p_0$ of the honest validators start on chain B.

We consider how a *Probabilistic Bouncing Attack* would unfold, taking the inactivity leak into account. A probabilistic bouncing attack lasting more than 4 epochs will necessarily cause an inactivity leak. Knowing this, we analyze the stakes of honest and Byzantine validators in this setting.

For this attack to continue, at each epoch, Byzantine validators cast their vote with a different chain as their candidate chain. They are active on each chain alternatively. Due to their inactivity every 2 epochs, they will get ejected from the chain after a total of 7653 epochs (4 weeks and 6 days). Byzantine validators are active on each chain to ensure that justification only happens every two epochs, preventing finalization from occurring.

For this attack to continue indefinitely, Byzantine validators must ensure honest validators are split into two branches according to two conditions: (a) the hon-

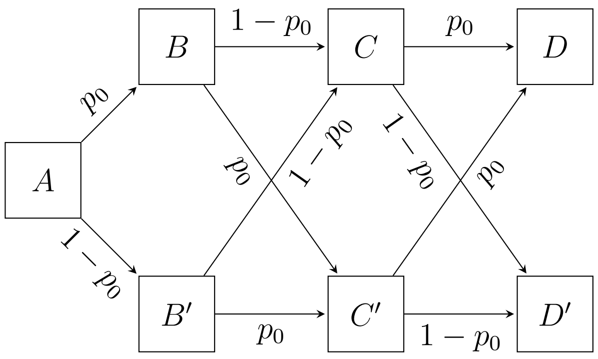

Figure IV.7: This figure represents, using a Markov chain, the probability of an honest validator changing branches or not every epoch. During the attack, the Byzantine validators ensure that a proportion p_0 of honest validators remains on one branch so they can justify this branch later with their withheld votes (Equation IV.14).

est validators are not enough to justify a chain on their own, and (b) the Byzantine validators can justify it afterwards with their withheld votes. This means that (a) p_0 must not represent more than $2/3$ of the stake, and (b) the proportions p_0 of honest validators and β_0 of Byzantine validators must represent more than two-thirds of the total stake. The two necessary conditions are that (a) $p_0(1-\beta_0) < 2/3$ and (b) $p_0(1-\beta_0) + \beta_0 > 2/3$. For the attack to function, we get that:

$$\frac{2-3\beta_0}{3(1-\beta_0)} < p_0 < \frac{2}{3(1-\beta_0)}. \tag{IV.14}$$

We can see that the closer β_0 is to 0, the closer p_0 has to be from 2/3. This is to be expected as otherwise the Byzantine validators would be unable to justify the checkpoint with withheld votes.

An illustration of an ongoing attack with the probability for honest validators to be on one chain or the other is depicted in Figure IV.7. At each epoch, a proportion p_0 of honest validators is on one branch, whereas a proportion $1-p_0$ is on the other.

Analytical Evaluation We are interested in the evolution of the proportion of Byzantine validators' stake β during the attack. To examine this, we analyze the evolution of the inactivity score over time for an honest validator randomly placed at each epoch. Referring to Figure IV.7, we observe that after two epochs, there is a probability $p_0(1-p_0)$ of having been on branch B for both epochs or on branch A for both epochs. The probability of being on both branches, regardless of the order, is $p_0^2 + (1-p_0)^2$. From the perspective of a chain, validators will be deemed inactive if they are active on the other chain. The probability of the inactivity score evolution after two epochs is the following:

$$\begin{cases} +8: & p_0(1-p_0) \\ +3: & p_0^2 + (1-p_0)^2 \\ -2: & p_0(1-p_0) \end{cases} \tag{IV.15}$$

We can notice that the time-dependent probability of the inactivity score is the convolution of two random walks. The first random walk moves +4 with probability p_0 and -1 with probability $(1-p_0)$. The second is the opposite, moving +4 with probability $(1-p_0)$ and -1 with probability p_0. We place ourselves in the continuous case to be able to continue our analysis and find the stake of validators with the inactivity score distribution over time (see section A.1 for details on the discrete and continuous case). To do so, we use the fact that a random walk follows a Gaussian distribution when time is large, using the central limit theorem. The expectation of the two random walks are $(5p_0-4)t$ and $(1-5p_0)t$, respectively, with both having a standard deviation of $25p_0(1-p_0)$. We disregard here the fact that the actual inactivity score is bounded by zero for analytical tractability. Allowing for negative values in the inactivity score can result in a reward instead of a penalty, which leads to a scenario conservatively estimating the loss of stake. The convolution of these two random walks is the probability of the inactivity score I:

$$\phi(I,t)=\frac{1}{\sqrt{4\pi Dt}}\exp\left(-\frac{(I-Vt)^2}{4Dt}\right), \tag{IV.16}$$

with $D=25p_0(1-p_0)$ and $V=3/2$. It now remains to find the distribution function of the stake s. We rewrite here the differential equation of the stake depending on I previously described in Equation IV.17:

$$\frac{ds}{dt}=-\frac{I(t)s}{2^{26}}. \tag{IV.17}$$

Using this (details in section A.2) we find the distribution function of the stake s to be:

$$P(s,t)=\frac{2^{26}}{s\sqrt{\frac{4}{3}\pi Dt^3}}\exp\left(-\frac{(2^{26}\ln(s/32)+Vt^2/2)^2}{\frac{4}{3}Dt^3}\right), \tag{IV.18}$$

with D and V, the diffusion and the velocity. In our case $V=3/2$ and $D=25p_0(1-p_0)$. The stake follows a log normal distribution for which the cumulative function is:

$$F(s,t)=\frac{1}{2}+\frac{1}{2}\operatorname{erf}\left[\frac{2^{26}\ln(s/32)+Vt^2/2}{\sqrt{\frac{4}{3}Dt^3}}\right]. \tag{IV.19}$$

Currently, the probability P does not reflect the actual stake according to time since validators get ejected at 16.75 ETH and are stuck at 32 ETH. To emulate this mechanism, since we know the cumulative distribution function, we can compute the new probability law $\mathcal{P}$:

$$\mathcal{P}(x,t)=\begin{cases}F(a,t) & \text{if } x=0,\\ P(x,t) & \text{if } a<x<b,\\ 1-F(b,t) & \text{if } x=b,\end{cases} \tag{IV.20}$$

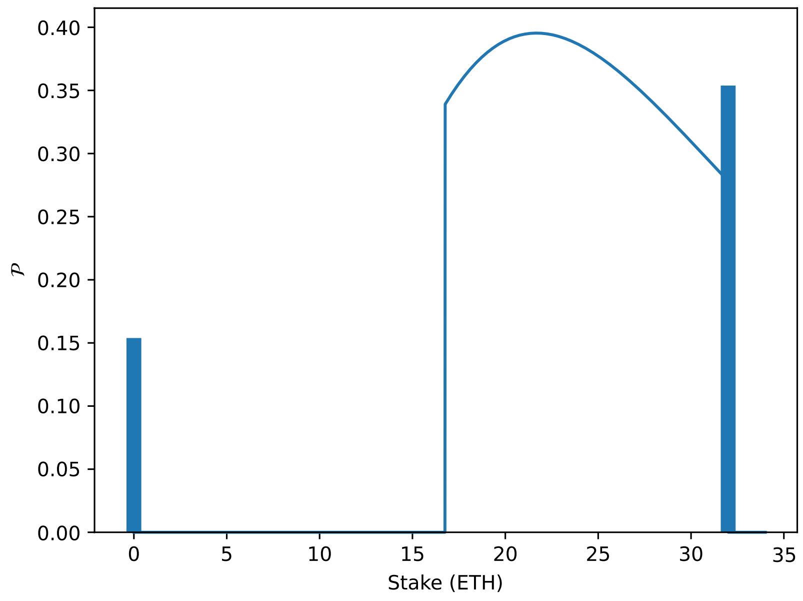

Figure IV.8: This is a representation of the distribution $\mathcal{P}$ at $t = 4024$ with an exaggerated standard deviation to provide a better intuition of the distribution behavior.

with $a = 16.75$ and $b = 32$. This new probability law takes into account the fact that if the stake is lower than 16.75 ETH, it becomes 0, and it is capped at 32 ETH. The explicit expression of $\mathcal{P}$ reads:

$$\begin{aligned}\mathcal{P}(x,t) =& \delta(x)\cdot F(a,t) + \delta(x-b)\cdot(1-F(b,t)) \\ &+ [H(x-a)\times H(b-x)]\cdot P(x,t),\end{aligned} \tag{IV.21}$$

where δ is the Dirac distribution, and H is the Heaviside function. Figure IV.8 shows a visual representation of the function $\mathcal{P}$.

The associated cumulative distribution function $\mathcal{F}$ of $\mathcal{P}$ is:

$$\begin{aligned}\mathcal{F}(x,t) &= \int_0^x \mathcal{P}(s,t)\,ds \\ &= F(a,t) + H(x-a)[F(x,t)-F(a,t)] \\ &\quad + H(x-b)[1-F(x,t)].\end{aligned} \tag{IV.22}$$

With this, we can evaluate the ratio of Byzantine validators and determine with what probability it will go beyond $1/3$. We denote by $s_{\mathrm{B}}(t)$ the stake of Byzantine validators and $s_{\mathrm{H}}(t)$ the stake of an honest validator. We are looking for the probability such that

$$\beta(t) = \frac{\beta_0 s_{\mathrm{B}}(t)}{\beta_0 s_{\mathrm{B}}(t) + (1-\beta_0)s_{\mathrm{H}}(t)} > \frac{1}{3}, \tag{IV.23}$$

depending on the probability of s_{H} that we now know. This translates into:

$$\mathcal{F}\left(\frac{2\beta_0}{1-\beta_0}s_{\mathrm{B}}(t), t\right), \tag{IV.24}$$

where $s_{\mathrm{B}}(t)$, the stake of a Byzantine validator, follows the stake of a semi-active validator.

We provide a representation of Equation IV.24 for several values of β_0 with $p = 0.5$ (note that p_0 has a minimal impact on the curve as it only slightly changes the variance) in Figure IV.9.

The figure illustrates how the proximity of β_0 to $1/3$ can be detrimental. This phenomenon occurs because the mean of the log-normal distribution approximates s_{B} when t is not too large. Referring to Equation IV.24, we observe that if $\beta_0 = 1/3$, we are examining $\mathcal{F}(s_{\mathrm{B}}(t), t)$, which explains why the probability is 0.5.

The probability increases sharply just before the expulsion of Byzantine validators; however, it is unlikely that the probabilistic bouncing attack would persist for that long. As an estimate, we can use the probability mentioned in the previous chapter (Equation III.1) to provide an upper bound on the probability of reaching epoch 7000: $(1-(1-\beta_0)^8)^{7000}$ is equal to 1.01×10^{-121} for $\beta_0 = 1/3$. This essentially negates any strategy by Byzantine validators that would require the probabilistic bouncing attack to last that long.

However, as Figure IV.9 shows, with β_0 nearing 1/3, Byzantine validators realistically have a high probability of quickly exceeding $1/3$ of the stake, especially considering the significant factor of the attack occurring on two branches. This means that if a validator is active during an epoch on one branch, it is inactive on the other. Hence, the probability can be doubled for each curve.

We can comprehend this by considering the case of $\beta_0 = 1/3$: after two epochs, the Byzantine validators have been active on each branch once. If one branch has more validators that have been active on it for two epochs, the other branch will have honest validators incurring, on average, more penalties than the Byzantine validators. On this latter branch, the Byzantine stake will represent more than one-third of the total stake.

These results imply that, theoretically, within the synchronous period and with a proportion of Byzantine stake sufficiently close to $1/3$ as well as a favorable initial setup, the probabilistic bouncing attack can pose a threat to the blockchain by allowing Byzantine validators to exceed the safety threshold of $1/3$.

IV.6 . Discussion & Conclusion

Our work presents the first theoretical analysis of the inactivity leak, designed to restore finalization during catastrophic network failure. We highlight situations where *Byzantine actions expedite the loss of Safety, either through conflicting finalization or by increasing the Byzantine proportion over the one-third Safety threshold*. No-

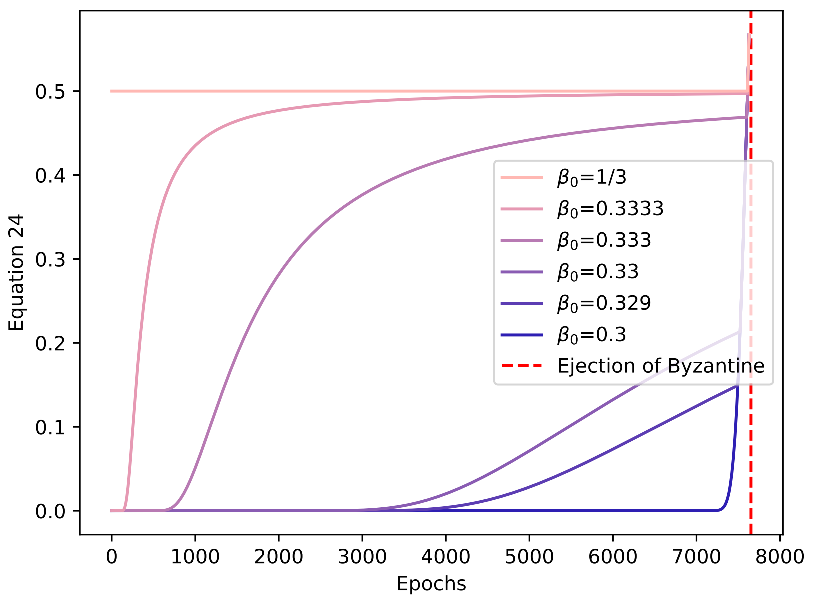

Figure IV.9: We represent Equation IV.24 according to time with various β_0.

tably, we demonstrate the possibility of Byzantine validators exceeding the one-third Safety threshold even during synchronous periods.

Our findings underscore the critical role of penalty mechanisms in BFT analysis. By illuminating potential issues in the protocol design, we offer insights for future improvement and provide tools to investigate them.

— Chapter V —

Ethereum PoS Protocol Analysis under the Game Theoretical Model

Contents

So far, we have analyzed the protocol through the lens of distributed systems, where agents were either honest or Byzantine. Although the second analysis did take into account an important part of the incentives, the model did not consider rational agents actually driven by incentives. In this final analysis, our goal is to study the strategies and equilibrium that can arise if all agents are rational. We aim to model agents with utility functions directly linked to the protocol rewards.

We start by presenting the necessary elements of the protocol required for our study. We then proceed with the definition of our *game* and its analysis.

V.1 . Ethereum protocol

We have simplified the complex functioning of the Ethereum Proof of Stake (PoS) protocol into the essential components for our analysis. At a high level, the set of participants, called validators, locally maintain a tree of blocks, denoted as $\mathcal{T}$. At any given moment, validators can evaluate the block tree to determine the branch that constitutes the current canonical chain, using a function known as the fork choice rule. The protocol requires validators to add new blocks to their local tree of blocks and broadcast these blocks to other validators.

Upon receiving a new block, validators assess whether it extends their canonical chain; if it does, a portion of them vote for it (attest to the block).

Thanks to a finalization protocol, a growing prefix of the canonical chain is maintained. This growing prefix cannot be forked, while the part of the chain beyond this prefix is forkable. In this study, we are interested in the protocol responsible for building the forkable part of the chain before finalization.

For our analysis, the elements of the protocol that we need are the following:

- **Slot.** Slots are the time frames dictated by the protocol for *proposers* and *attesters* to perform certain actions.
- **Proposer.** There is one proposer per slot. The proposer's role is to propose a block during a specific slot.
- **Attester.** There are a attesters per slot. The attester's role is to produce an attestation, which is a vote for a specific block. Attestations determine the weights of the blocks, which are used by the fork choice rule.
- **Fork choice rule.** The fork choice rule is the protocol's rule that determines, in case of a fork, which block is the head of the chain.
- **Canonical chain.** The canonical chain is the chain designated by the fork choice rule.

Proposers and attesters are assigned to slots in a deterministic and verifiable manner using a pseudo-random function included in the Ethereum protocol.

Fork Choice Rule As mentioned, despite their name, blockchains are closer to block trees. Forks can occur, causing the blockchain to have several branches rather than a single chain. To address this, the protocol defines a function called the fork choice rule, $\mathcal{F}$, which indicates, at each slot k, on which block to build or attest based on the tree of all blocks, $\mathcal{T}_k$, and the set of attestations, $\mathcal{A}_k$. This block is called the head of the canonical chain. To determine the head of the canonical chain, the fork choice rule follows these steps:

1. Traverse the set of all attestations $\mathcal{A}_k = \cup_{i=0}^{k} a_i$, where a_i is the set of attestations sent during slot i. Keep only the last attestation from each attester.
2. For each attestation, add a weight[1] to each block attested to, as well as to all of its parents. This process gives an *attestation weight* to each block using the tree of blocks $\mathcal{T}_k$ and the set of attestations $\mathcal{A}_k$ at slot k.

[1] ↑ In the protocol, the weight added is proportional to the stake of the corresponding validator. For simplicity, each validator is assumed to have the same stake in our analysis. Thus, without loss of generality, we choose the stake to be one, so that counting the attestation weight is equivalent to counting the number of attestations for this block or its descendants.

3. Start from the genesis block and continue along the chain by following the block with the highest attestation weight at each fork. Return the block that has no children. This block is the head of the candidate chain.[2]

During the execution of the protocol, it is prescribed that the block proposer of slot k executes the fork choice rule $\mathcal{F}(\mathcal{T}_{k-1}, \mathcal{A}_{k-1})$ to determine the parent of its block. During the same slot, the attester should use the fork choice rule $\mathcal{F}(\mathcal{T}_k, \mathcal{A}_{k-1} + \rho a)$ to determine which block to vote for. The notation $\mathcal{A}_{s-1} + \rho a$ indicates that an additional attestation weight of ρa is added on the block of the current round. The addition of the attestation with ρa is called the proposer boost and is explained below.

It is important to note that during periods of good network conditions, all validators are likely to be aware of every block and attestation within the corresponding slot. Therefore, all validators will have the same view of the attestation weights, implying that $\mathcal{F}$ will consistently return the same head when executed by different validators. In our model, we assume 'perfect' network conditions, meaning that any message sent is assumed to be received immediately.

Attestation Weight We previously introduced the concept of attestation weight, which is crucial throughout our analysis. Let us now briefly expand on it. The attestation weight of a block is the sum of all attestations sent for this block, as well as all attestations sent for the descendants of this block. This means that an attestation not only supports a single block but also the entire chain of blocks leading to it.

We refer to the *total* attestation weight of a branch of blocks as the sum of all attestations for that branch. It should be clear that the total attestation weight of a branch is equivalent to the attestation weight of the first block in the branch.

Proposer Boost To mitigate the balancing attack [NTT21], in which malicious validators withhold votes and release them at an opportune time to maintain a fork indefinitely, the *proposer boost* was created[3]. However, this modification of the protocol was found to be susceptible to new attacks involving equivocation [NTT22], leading to an update where the fork choice rule no longer considers conflicting attestations from the same attester.

The proposer boost, denoted as $\rho \in [0, 1)$, temporarily assigns ρa artificial attestations to a *timely* block, where a represents the total attestation weight per slot. This mechanism adds additional attestation weight to a block exclusively during the slot in which it is proposed. Specifically, if a block is received early in slot k

[2] ↑ We assume that the block with the highest attestation weight always starts from the genesis block. In practice, the fork choice rule begins from the justified checkpoint with the highest epoch, but we have simplified the protocol for our analysis.

[3] ↑ See ethereum/consensus-specs/pull/2730

(within the first four seconds), ρa artificial attestations are temporarily added to it. Currently, the proposer boost is set at 0.4, effectively adding $0.4 \times a$ attestations to the block's current weight. This adjustment influences the attestation weights so that during slot k, the timely block B_k carries additional weight, thereby affecting the fork choice rule.

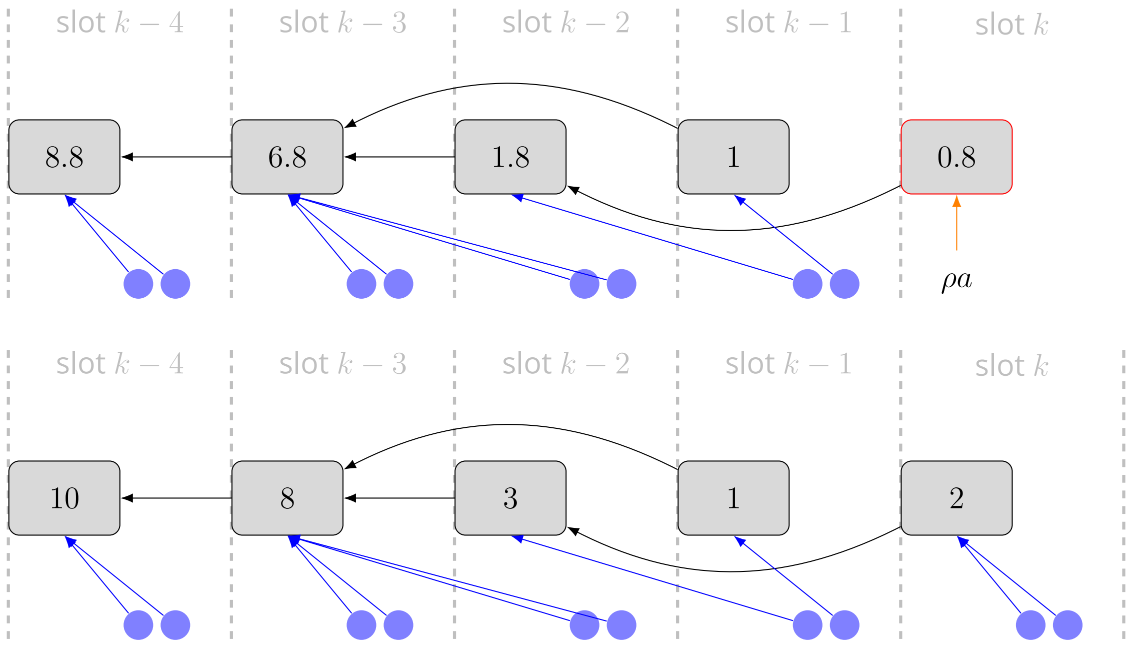

Figure V.1: This figure illustrates an evaluation of the fork choice rule executed after a timely block proposal in slot k. In this simplified representation, there are 2 attesters per slot, so two attestations are sent per slot, each represented by a circle pointing to the block being attested. Artificial attestations are created in the current slot k and add a weight of ρa for the fork choice rule. At the beginning of slot k, a timely block is proposed, and the proposer boost of $\rho a = 0.8 (= 0.4 \times 2)$ is applied. When slot k ends, the proposer boost is cleared, and we observe the two attestations sent during the slot that followed the fork choice rule.

An example of the fork choice rule with the proposer boost in action is illustrated in Figure V.1. The figure captures the chain at two different times during slot k: right after the timely block proposal and at the end of the slot. Each time, we show the weight of the blocks as computed by the fork choice rule. In our study, we remain agnostic about the value of ρ to examine its effects across different values.

Rewards The rewards for proposers and attesters are computed in a verifiable manner. Based on the content of a block in the canonical chain, which includes attestations and transactions, we can determine the rewards for the attesters responsible for these attestations and for the proposer who included them. The proposer also earns rewards from transaction fees.

Validators are incentivized to participate in the process of adding new blocks and attesting them through endogenous rewards inscribed in the protocol. A crucial factor in determining rewards is identifying the canonical chain. For instance, a block proposed but not included in the canonical chain will result in zero rewards for the proposer. All rewards for block proposers and attesters can be determined by examining the content of the blocks that form the canonical chain.

Because the proposer's rewards depend on the attesters' rewards, we first introduce the rewards for attesters.

Attester Rewards. An attester is rewarded for its attestation based on two factors: the *timeliness* and the *correctness* of its vote. Table V.1 indicates the reward for an attester depending on these two factors.

Timeliness	1 slot	≤ 5 slots	≤ 64 slots
Incorrect attestation vote	$20x/27$	$20x/27$	$6x/27$
Correct attestation vote	x	$20x/27$	$6x/27$

Table V.1: Attester's rewards based on the inclusion of the attestation in the chain and its blockvote.

Timeliness refers to the number of slots between the expected time for sending an attestation and its actual inclusion in a block. The fastest possible inclusion is 1 slot, meaning the attestation is included in the block of the subsequent slot. An attestation is considered correct if its vote points to the most recent block at the time of its slot that belongs to the canonical chain. Thus, both timeliness and correctness depend on the actions and votes of other validators. Timeliness depends on whether subsequent blocks include the attestation and eventually belong to the canonical chain. Correctness is affected by the possibility that an attester might vote for a block that is initially in the canonical chain but is later not.

As the finalized chain grows, it will eventually determine which blocks belong to the canonical chain. However, as shown in Table V.1, correctness only significantly impacts the attester's reward if the attestation is included in the following slot. This may incentivize attesters to align their votes with the proposer of the next slot, regardless of whether the block ultimately becomes part of the canonical chain.

Proposer Rewards. For the proposer, there are two types of rewards: rewards based on attestations and rewards based on transactions.[4] Importantly, unlike in Bitcoin, there is no coinbase transaction in each block guaranteeing a minimum reward for proposing a block. The proposer receives a proportion of the reward generated for each attestation it includes. Additionally, the proposer receives a reward for each transaction included in its block in the form of transaction fees.

[4] ↑ These rewards differ in that attestation rewards are received on the consensus layer, while transaction fees are received on the execution layer, but this distinction does not impact our analysis.

These rewards are formalized in subsection V.2.1, where the utility functions are defined.

V.2 . Model & Game

We model the Ethereum PoS consensus protocol as a game where each player[5] is either a *proposer* or an *attester*. Ideally, in the prescribed behavior, proposers propose blocks, and attesters broadcast attestations. The game evolves over s sequential *slots*. There is one proposer and $a \in \mathbb{N}$ attesters per slot, resulting in a total of s proposers and as attesters. The value of s is unknown to the players.

As described in section II.3 we have two main assumptions: (i) The game occurs during a synchronous period, where the network is considered fully synchronous with no latency, and (ii) The synchronous period follows an asynchronous period, during which there may have been delays in information transmission. Therefore, our game is set in the synchronous period, but the initial state is influenced by events that occurred during the preceding asynchronous period. In this initial asynchronous period, blocks may have an uneven distribution of attestations across slots, unlike in a permanently synchronous scenario.

Under these assumptions, the broadcast of the block proposal and attestations in our game are treated as atomic events. Thus, in each slot, there are three distinct events:

1. *Block proposal.* The designated proposer for the slot proposes a new block, selecting a previously existing block in the observed blockchain as its parent. When a proposer prepares a block, they add all available transactions and attestations to the block—i.e., all the transactions and attestations that are not yet part of the blockchain. Once the transactions and attestations are included, the block is proposed, meaning it is sent to the network.

2. *Generation of transactions.* Transactions are sent by users and observed by proposers and attesters.

3. *Attestations.* After the block proposal for the slot, all attesters of the slot choose which previously proposed block to attest and send their attestations simultaneously.

As detailed in section V.1, the protocol prescribes that proposers (and attesters) should select as the parent (or as the block to attest) the block identified by the fork choice rule, i.e., the head of the canonical chain.

These three events occur in sequence and are depicted in Figure V.2. Proposers and attesters are financially motivated to participate in the protocol. It

[5] ↑ We use the terms *players* and *validators* interchangeably.

remains to be seen whether the protocol is incentive compatible; it is the case if following the protocol maximize their gains.

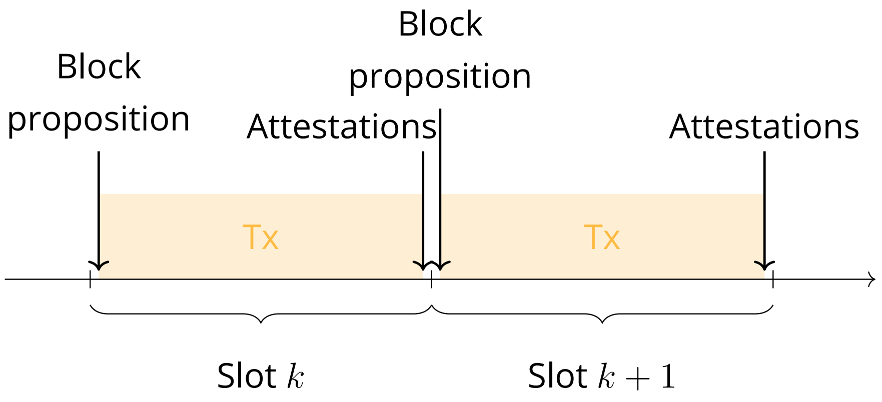

Figure V.2: This schema represents the atomicity of the block proposal and the attestation broadcast. In each slot, three phases occur in order: first, the block proposal; then, the generation of transactions (which will be available for the proposer of the next slot); and finally, all attestations are sent simultaneously.

The game

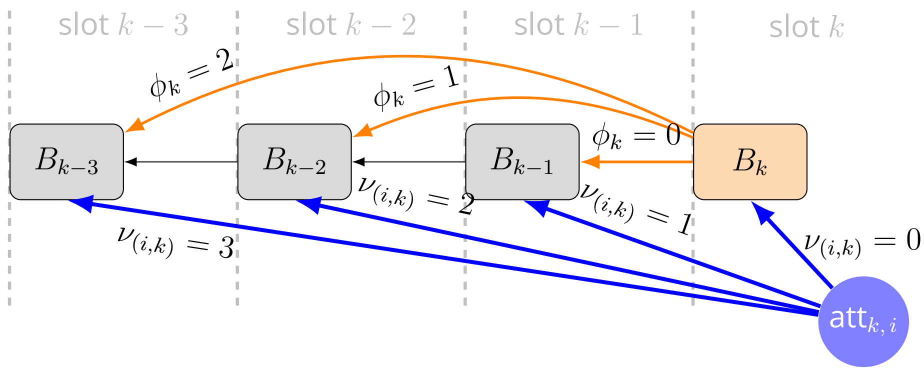

Figure V.3: The actions available to an attester (blue) and a proposer (orange). The attester i selects which block to attest with $\nu_{(i,k)}$. The proposer selects the parent of its block with ϕ_k. An example is shown where $\phi_k = \phi_{k-1} = \phi_{k-2} = 0$.

We denote the set of players (the validators) as $\mathcal{V} = \{P, A\}$, consisting of a set of proposers P and a set of attesters A. Per slot, following the Ethereum protocol, there is exactly one proposer and $a \in \mathbb{N}$ attesters. Hence, the number of proposers is $|P| = s$, and the number of attesters is $|A| = as$, with $a, s \in \mathbb{N}$.

We model the interactions between proposers and attesters during s slots in Ethereum PoS as a game. In each slot, the timeline of events is as follows: (i) a block is proposed at the beginning of the slot,[6] (ii) new transactions are proposed, and (iii) all the attesters of the slot send their attestations simultaneously. Therefore, the game is dynamic, with each stage corresponding to a slot. In each slot, the attesters play a simultaneous game following the proposal by the slot's

[6] ↑ Every block thus receives the proposer boost in our model.

proposer. Our interest lies in the actions that the proposers and attesters have, which we now describe.

Actions.

When it is their turn (recall that each proposer/attester is uniquely assigned to a slot, and this information is verifiable, allowing players to take action only in their corresponding slot), a proposer must choose which block to extend, and an attester must select which block to attest. The action will take the form of a variable that indicates how many slots prior a proposer attaches its block to, or an attester attests. More formally, the action of the proposer in slot k is to assign a value to its variable $\phi_k \in \mathbb{N}$, corresponding to the difference between the current slot and the slot of the block selected as the parent. Similarly, after the block proposal in slot k, each attester i of slot k must assign a value to its variable $\nu_{i,k} \in \mathbb{N}$ that represents the difference between the current slot and the slot of the block being attested. We depict a subset of the action space in Figure V.3. In more detail:

- At the beginning of slot k, a proposer p chooses the parent of its block B_k. We denote this action by $\phi_k \in \mathbb{N}$. $\phi_k = l$ means that B_k's parent is B_{k-1-l}. Thus, if B_k's parent is the block from the previous slot $k-1$, then $\phi_k = 0$.

  The blocks contain two types of data: attestations and transactions. There is no limitation on the number of transactions and attestations a block can contain.[7] A proposer always includes all available transactions and attestations. A transaction/attestation is considered available if it is not included in any of the predecessors of B_k.

- After the block proposal in slot k, all attesters of slot k simultaneously choose which block to attest. The attestation of attester i in slot k points to a specific block determined by $\nu_{(i,k)} \in \mathbb{N}$, i.e., the age of the block attested. $\nu_{(i,k)} = l$ means that the block B_{k-l} is the one attested by attester i in slot k. Thus, if validator i attests for the block in the current slot k, then $\nu_{(i,k)} = 0$.

These actions are repeated in each slot. Note that not proposing or attesting to a block is not an available action.

The last piece of data needed for our study is to determine whether a block B_k eventually belongs to the canonical chain. In our model, this information is represented by $\chi_k \in \{0,1\}$, where $\chi_k = 1$ if the block from slot k eventually belongs to the canonical chain, and $\chi_k = 0$ otherwise. This information becomes known at the end of slot s, which marks the conclusion of our game.

It is important to note that for any slot k, always assigning a value of 0 to ϕ_k as a proposer (or a value of 0 to $\nu_{(i,k)}$ as an attester) is not the prescribed action. The prescribed action is to follow the fork choice rule, as illustrated in Figure V.1.

[7] ↑ This simplification is similar to the one made in [CKWN16].

Strategies. A strategy of a player i is a function σ_i, which takes as input the entire tree of blocks in the blockchain, as well as the attestations sent so far, and produces as output a number, say $s \in \mathbb{N}$. Since the only information available are the tree of blocks and the attestations, the signature of a player's strategy is $\mathcal{T} \times \mathcal{A} \to \mathbb{N}$, where $\mathcal{T}$ is the set of blocks and $\mathcal{A}$ is the set of available attestations.

For the proposer of slot k, the prescribed strategy is $\sigma_{(0,k)}(\mathcal{T}_{k-1}, \mathcal{A}_{k-1}) = l$, where $\mathcal{F}(\mathcal{T}_{k-1}, \mathcal{A}_{k-1}) = B_{k-1-l}$. The prescribed strategy for an attester i at slot k is $\sigma_{(i,k)}(\mathcal{T}_k, \mathcal{A}_{k-1}) = l$, where $\mathcal{F}(\mathcal{T}_k, \mathcal{A}_{k-1} + \rho a) = B_{k-l}$. We say that a player deviates from the prescribed protocol when their strategy produces a number different from the slot of the block resulting from the fork choice rule.

A strategy profile $\sigma = (\sigma_{0,1}, \ldots, \sigma_{a,1}, \sigma_{0,2}, \ldots, \sigma_{a,2}, \ldots, \sigma_{0,s}, \ldots, \sigma_{a,s})$ is a vector where each component is a strategy of the corresponding player. We denote by $\mathcal{S}$ the set of all strategy profiles and by $\mathcal{S}_{(i,k)}$ the set of strategies for the player of component (i, k). In this notation, players with indices (i, k) where $i = 0$ are proposers, while players with indices (i, k) where $1 \leq i \leq a$ are attesters. For clarity, we denote by (σ_{-i}, σ'_i) the strategy profile σ where, instead of playing with strategy σ_i, player i deviates and uses strategy σ'_i instead. This applies to both attesters and proposers.

It remains to define the reward of the players at the end of the game. At the end of slot s, the payoff of all players is computed and given by the function $u : \mathcal{S} \to \mathbb{R}^{n+an}$ (defined in subsection V.2.1). The payoff of each player is given by its component in the reward vector, which depends on its type and is determined by a reward function. In the remainder of the chapter, for clarity, for any strategy profile σ, we write $u_{i,j}(\sigma)$ instead of $u(\sigma)_{(i,k)}$, where $u_{i,j}(\sigma)$ represents the payoff of player (i, j), and player $(0, j)$ is the proposer of slot j.

V.2.1 . Payoff

Attesters' rewards vary depending on when their attestations are included in a block and which block they attest to. This can incentivize them to align their attestations with the behavior of future block proposers. Block proposers have a clear incentive to accumulate the maximum transaction fees and lucrative attestations to maximize their rewards. One strategy to achieve this is to fork the chain and include in the new block all the attestations and transactions that do not belong to the new chain. However, if the block does not end up in the canonical chain, the block proposer will not receive any rewards. This incentivizes the proposer to consider other behaviors, as we will see.

Given a strategy profile σ, the reward of attester i in slot k, player (i, k), depends on a variable $x > 0$ set by the protocol and the slot in which the attestation is subsequently included in a block:

$$u_{(i,k)}(\sigma) = \begin{cases} x & \text{if } \sigma_{(i,k)} \text{ sets } \nu_{(i,k)} = \phi_{k+1} \text{ and } \chi_{k+1} = 1, \\ 20x/27 & \text{if } \chi_{k+2 \geq \cdots \geq k+5} = 1\text{, included in 2 to 5 slots following the attestation,} \\ 6x/27 = 2x/9 & \text{otherwise (if } \chi_{\geq k+6} = 1\text{).} \end{cases} \tag{V.1}$$

Here, $x > 0$ and χ_{n+1} denote the fact that the block of slot $s+1$ belongs to the canonical chain. The rewards for the attester can be understood as follows: they are maximized when the attestation votes for the parent of the block in the subsequent slot, and this block in the subsequent slot ends up in the canonical chain.

The actual rewards for an attester are detailed in Table V.1. Note that the reward is influenced by the correctness of the attestation only if it is included in the block of the next slot. Additionally, if the block of slot $k+1$ does not end up in the canonical chain, the attesters of slot k can never receive the maximum reward.

For the proposer of slot k, the reward function is given by:

$$u_{(0,k)}(\sigma) = \chi_k \sum_{j=n-\phi_k}^{n} \left(\frac{1}{7} \sum_{i=1}^{a} u_{(i,j)}(\sigma) + f_{j-1} \right), \tag{V.2}$$

The reward is the sum of attestation rewards and transaction fees over the slots separating the block from its parent, multiplied by the factor that indicates the block belongs to the canonical chain. Here, $f_{n-1} > 0$ represents the random value of transaction fees generated during slot $s-1$. The transaction fees are the incentives that motivate proposers to include transactions in their block. The proposer receives $1/7$ of what the attesters receive for their attestations being included in a block. The factor χ_s, indicating whether the block ends up in the canonical chain, applies to the entire reward since, if the block is not included in the finalized chain, it does not yield any rewards.

V.3 . Analysis

In this section, we explore a set of possible strategies for proposers and attesters. Each can either follow the obedient strategy or adopt a cunning strategy. The obedient strategy is the one prescribed by the protocol. In contrast, the cunning strategy may deviate from the protocol while exploiting the proposer boost as a means to remain part of the canonical chain. We begin by introducing the game-theoretic preliminaries necessary for our analysis.

V.3.1 . Preliminaries

To ensure clarity and self-containment, we redefine well-known game-theoretic concepts. These concepts are useful for categorizing strategy equilibria and exploring possible states of the game.

Definition V.1 (Best response)**.** *A strategy σ_i^* is a best response for player i to the strategy profile σ_{-i} of the other players if:*

$$u_i(\sigma_{-i}, \sigma_i^*) \geq u_i(\sigma_{-i}, \sigma_i), \quad \forall \sigma_i \in \mathcal{S}_i, \tag{V.3}$$

where u_i is the payoff function for player i, σ_{-i} is the strategy profile of all other players, and $\mathcal{S}_i$ is the set of all possible strategies for player i.

Definition V.2 (Nash equilibrium)**.** *A strategy profile $\sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_n^*)$ is a Nash equilibrium if each player's strategy σ_i^* is a best response to the strategies σ_{-i}^* of the other players. Formally,*

$$u_i(\sigma_{-i}^*, \sigma_i^*) \geq u_i(\sigma_{-i}^*, \sigma_i), \quad \forall \sigma_i \in \mathcal{S}_i \text{ and for all players } i, \tag{V.4}$$

where u_i is the payoff function for player i, σ_{-i}^* is the strategy profile of all other players in the equilibrium, and $\mathcal{S}_i$ is the set of all possible strategies for player i.

In summary, the concept of a best response helps identify the optimal strategy for a player given the strategies of the other players. Nash equilibrium defines a state where each player's strategy is a best response to the strategies of the other players, ensuring no player can benefit from unilaterally changing their strategy.

V.3.2 . Obedient

As Carlsten et al. [CKWN16], we first describe the strategy of proposers and attesters that act as prescribed by the protocol, we refer to them as obedient. However in the case of Ethereum, the actions prescribed by the fork choice rule are more complex compared to those described by Carlsten et al.

Obedient Proposer ($\sigma^O_{(0,k)}$):
Action: $\phi_k = l$, where $\mathcal{F}(\mathcal{T}_{k-1}, \mathcal{A}_{k-1}) \rightarrow B_{k-1-l}$.

The strategy of an obedient proposer at slot k is to propose a block B_k linked to the block designated by the fork choice rule $\mathcal{F}(\mathcal{T}_{k-1}, \mathcal{A}_{k-1}) \rightarrow B_{k-1-l}$.

Obedient Attester ($\sigma^O_{(i,k)}$):
Action: $\nu_{(i,k)} = l$ (Block attested is B_{k-l}.)

The obedient attester strategy of attester i is to attest to the block designated by the fork choice rule $\mathcal{F}(\mathcal{T}_k, \mathcal{A}_{k-1} + \rho a) \rightarrow B_{k-l}$.

We denote by $\sigma^O_{(i,j)}$ the obedient strategy of player (i, j) and by σ^O the strategy profile where all players act obediently.

When proposers and attesters follow the obedient strategy, we can evaluate the rewards each of them will receive. Since they will all follow the fork choice rule and there are no delays, no forks will occur, and attesters will attest to the block of their slot. Moreover, each attestation will be included in the following slot and will be correct. For proposers and attesters following the actions prescribed by the protocol, the rewards are as follows:

- For each attester i following the obedient strategy, the reward is: $u_{(i,k)}(\sigma^O) = x$, where σ^O is the strategy profile in which all proposers and attesters are obedient.
- For the proposer of slot k, the reward is: $u_{(0,k)}(\sigma^O) = \frac{ax}{7} + f_{k-1}$.

With this strategy profile, attesters obtain the maximum reward attainable (Equation V.1). However, there is no maximum reward for a block proposer, as their rewards increase the more ancient their block's parent is.

V.3.3 . Cunning Strategy

We now examine a strategy that could yield more rewards for validators than simply following the protocol. In some situations, deviating from the protocol can allow validators to accumulate more rewards without incurring penalties. We refer to this as the *cunning* behavior. For a proposer, the strategy involves choosing a block parent for its proposal that maximizes its rewards.

As the block parent's slot is further away from the new block, the proposer can include more transactions and attestations to increase its rewards. The ideal block parent, in theory, would be the genesis block. However, for the block to actually yield rewards, it must become part of the canonical chain. The cunning proposer will always propose a block that is considered the head of the canonical chain during its slot.

For instance, a cunning proposer will not strictly follow the fork choice rule to determine its block's parent. Instead, it will subtly test whether it can choose an older block as the parent while still having its block become the head of the canonical chain. The block that maximizes rewards—typically the oldest possible block—will be selected as the parent by the cunning proposer.

Cunning Proposer ($\sigma^C_{(0,k)}$):
Action: $\phi_k = \max\{x \in \mathbb{N} : \mathcal{F}(\mathcal{T}_x, \mathcal{A}_{x-1} + \rho a) = B_k\}$

The cunning proposer's block extends the block that leaves the most available transactions and attestations while still being selected as the head of the canonical chain by the fork choice rule (for attesters in the same slot) due to the proposer boost ρa.

We denote by $\sigma^C_{(0,k)}$ the cunning strategy of the proposer in slot k.

Remark V.1. *The obedient and the cunning strategy can result in the same action.*

It is important to note that while the cunning and obedient strategies are distinct, the actions resulting from them can sometimes be identical. Indeed, the action taken by a cunning proposer is to propose a block with the oldest possible parent while still ensuring the block is designated by the fork choice rule for the attesters of the slot. However, if the oldest possible parent is the same block initially designated by the fork choice rule, the action will align with the protocol, just as it would under the obedient strategy. In this sense, we say that a cunning player *acts* obediently if their action is the one prescribed by the protocol. Conversely, we say they act cunningly if the action taken differs from what is prescribed by the protocol, making the cunning strategy truly distinct from the obedient strategy.

Observation V.1 (Cunning condition)**.** *The divergence between cunning and obedient proposer behavior occurs when the branch containing the block designated by the fork choice rule, with a total attestation weight w_f, has a concurrent branch with a total attestation weight w_g such that:*

$$w_f - w_g \leq \rho a. \tag{V.5}$$

*We call this inequality the **cunning condition**.*

First, it is clear that w_f is always greater than w_g, as the block designated by the fork choice rule is on the branch with a total attestation weight of w_f. To understand the cunning condition, we consider two illustrations:

1. The first, and less intuitive, case is presented in Figure V.4. This showcases the scenario where $w_g = 0$. A branch can consist of many blocks, a single block, or, in this case, no block at all.

   The newly proposed block can become the head of the canonical chain by attaching itself to the first block with more than ρa attestation weight. If the block designated by the fork choice rule is on a branch with a total attestation weight less than ρa, the cunning behavior differs from the obedient behavior.

2. Another representation of the cunning condition is shown in Figure V.5. Here, there are two distinct branches, each with one block. The branch of the block designated by the fork choice rule has an attestation weight of $w_f = 3$, while the concurrent branch has $w_g = 2$. In this case, where $\rho a = 1.2$, the condition is met, allowing the proposer to act cunningly.

It should be noted that without the proposer boost, the cunning proposer strategy would never differ from the obedient proposer strategy. This strategy relies on the advantage provided by the proposer boost to ensure that its block becomes the head according to the fork choice rule.

Conversely, and intuitively, as the proposer boost increases, the opportunity for the cunning block proposer to act cunningly arises more frequently in the case all attesters are obedient.

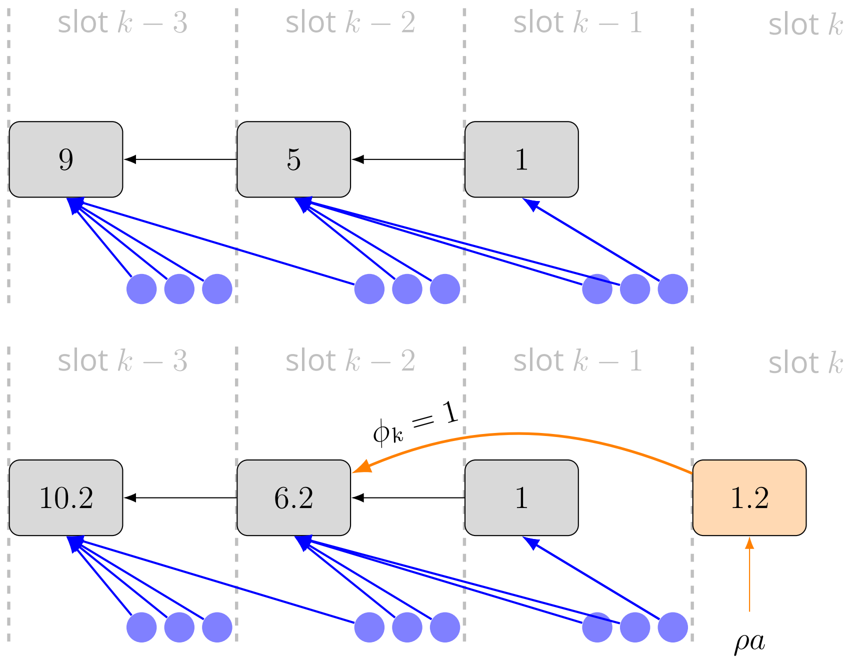

Figure V.4: Cunning proposer $(0, k)$ deviating from the prescribed protocol with $\rho a = 1.2 (= 0.4 \times 3)$ when the block designated by the fork choice rule has an attestation weight less than ρa.

Best response of a proposer among $s - 1$ obedient proposers and as obedient attesters. We first study the behavior of one proposer when all others are obedient with respect to their designated slot. In the case in which the proposer is associated to the first slot of the game, for this proposer the cunning strategy is the best response and the proposer deviates from the protocol if the cunning condition holds (Lemma V.1). In the case the proposer is associated to a subsequent slot of the game, then the two strategies obedient and cunning are equivalent; this means that the proposer will follow the protocol (Lemma V.2).

Lemma V.1. *When all other validators are obedient, the cunning proposer strategy is a best response.*

Proof. Let us denote by ϕ_k^C and ϕ_k^O the actions taken by proposer $(0, 0)$ under the cunning strategy and the obedient strategy, respectively. The cunning proposer strategy differs from the obedient strategy when $\phi_k^C > \phi_k^O$. Considering that the rest of the validators follow the obedient strategy, a proposed block that becomes the head of the chain at slot k will end up in the canonical chain ($\chi_k = 1$). By construction, $\phi_k^C \geq \phi_k^O$, and in both cases, the proposed block will be the head of the canonical chain and thus belong to the canonical chain ($\chi_k = 1$). Based on the definition of $u_{(0,k)}$ (cf. Equation V.2), the reward increases as the sum increases. This implies that $u_{(0,k)}(\sigma^O_{-(0,k)}, \sigma^C_{(0,k)}) \geq u_{(0,k)}(\sigma^O_{-(0,k)}, \sigma^O_{(0,k)})$. □

Let us note that when everyone else is obedient, the cunning strategy can only differ from the obedient strategy for the first proposer. All subsequent proposers

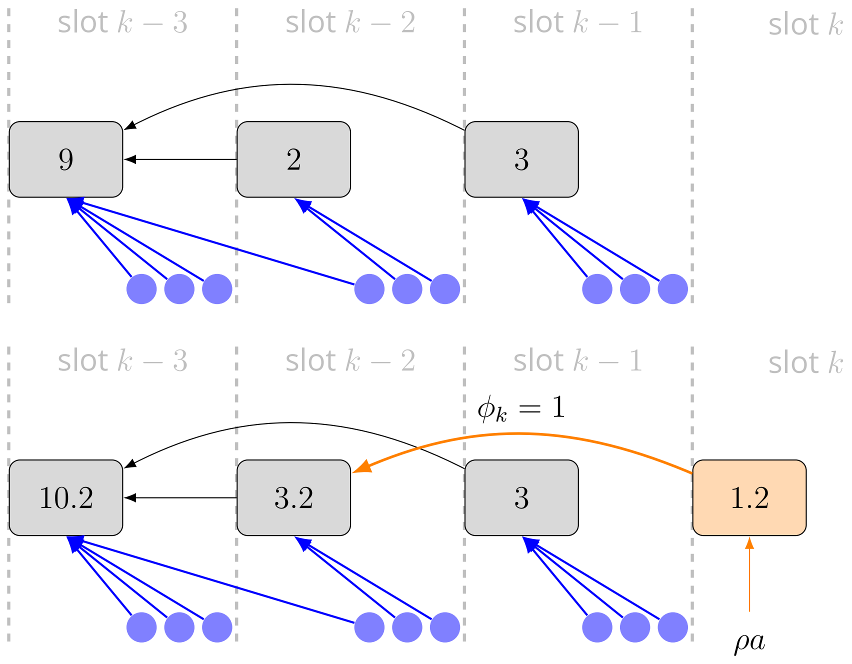

Figure V.5: Cunning proposer $(0, k)$ deviating from the prescribed protocol with $\rho a = 1.2 (= 0.4 \times 3)$ when two blocks have an attestation weight difference of less than ρa, and one of them is designated by the canonical chain.

will follow the protocol regardless of whether they follow the cunning or obedient strategy because the cunning condition is never satisfied. We formalize this in the following observation:

Lemma V.2. *For the strategy profile $\sigma^O_{-(0,i)}$ where all other validators follow the obedient strategy, $\phi^C_i \neq \phi^O_i$ only if $i = 0$, where ϕ_i is the action of proposer $(0, i)$.*

Proof. This can be explained because our model makes strong assumptions about synchronous network conditions. When considering that all attesters are obedient, this implies that in each slot, all attesters will send the same attestation. As a result, every new block will have an attestation weight that is a multiple of a. If the first proposer is obedient, it attaches its block to the branch with the highest attestation weight w_f. The obedient attesters will attest to it, adding a weight of a. This makes the cunning condition (Observation V.1) impossible for subsequent proposers, as $w_f + a - w_g \geq a$. Nevertheless, the first proposer can act cunningly since the network's state before the first slot is not predetermined, leaving any arrangement of blocks and attestation weights possible. ☐

Best response of a proposer among $s-1$ cunning proposers and as obedient attesters. We study the behavior of one proposer when all other proposers are cunning and attesters are obedient. We have two cases:

- $\rho < 1/2$. If the proposer is associated with the first slot of the game, the cunning strategy is the best response, and the proposer will deviate from the protocol if the cunning condition holds (Lemma V.3).

If the proposer is associated with any subsequent slot in the game, the cunning condition will never hold. In this case, the obedient and cunning strategies are equivalent, leading the proposer to act as prescribed by the protocol (Lemma V.11).

- $\rho \geq 1/2$. In this case, the cunning condition can apply to multiple consecutive proposers. If the cunning condition does not hold for the second proposer, then the cunning strategy is the best response for the first proposer (Lemma V.4), causing the first proposer to deviate from the protocol.

  If the cunning condition holds for more than just the first proposer, the cunning strategy becomes the best response if, and only if, the expected rewards gained from acting cunningly and hoarding the rewards over the two previous slots exceed the rewards from the most recent slot alone. (Lemma V.5). In this scenario, each proposer deviates from the prescribed protocol.

Lemma V.3. *The cunning proposer strategy is a best response for a proposer when all other proposers are cunning and attesters are obedient, provided $\rho < 1/2$. If the cunning condition holds, it will only do so for the first proposer, causing this proposer to deviate from the protocol.*

Proof. With $\rho < 1/2$, only the first proposer can act cunningly. Let's assume the first proposer acts cunningly, meaning that the cunning condition is satisfied. The maximum gap between w_f and w_g for the first proposer to act cunningly is ρa, such that $w_f = w_g + \rho a$. After the attestations sent by the obedient attesters in the first slot, the attestation weight of the branch designated by the fork choice rule becomes $w_g + a$. For the second proposer to act cunningly, it must hold that $w_g + a - w_f \leq \rho a$ (cunning condition for the second proposer). Substituting w_f with the maximum possible gap from w_g, this condition implies that the second proposer can act cunningly if and only if:

$$\begin{aligned} w_g + a - (w_g + \rho a) &\leq \rho a \\ \frac{1}{2} &\leq \rho. \end{aligned} \tag{V.6}$$

Thus, with cunning proposers and obedient attesters, only the first proposer can act cunningly, deviating from the obedient action. For the first proposer, acting cunningly will yield the maximum rewards. □

In fact, since only the first proposer can act cunningly when $\rho < 1/2$, the obedient strategy remains a best response for the rest of the proposers, even when all other proposers are cunning and attesters are obedient.

Lemma V.4. *The cunning proposer strategy is a best response for a proposer when all other proposers are cunning and attesters are obedient, if $\rho \geq 1/2$ and the cunning* *condition does not hold for the second proposer. If the cunning condition holds, it will only do so for the first proposer, causing this proposer to deviate from the protocol.*

Proof. If the attestation weight of the branch f designated by the fork choice rule, w_f, and the attestation weight w_g of a concurrent branch g are such that $w_g + a - w_f > \rho a$, the second proposer cannot act cunningly with obedient attesters (cunning condition false for the second proposer). We previously showed that if the first proposer is cunning and attaches its block to the concurrent chain g, the obedient attesters will follow, increasing the weight of g to w_g+a. By ensuring that $w_g + a - w_f > \rho a$, we prevent the second proposer from changing the canonical chain with the proposer boost. Thus, this condition ensures that the first proposer can be the only one to act cunningly, and in this case, the best response is the cunning strategy. □

Lemma V.5. *When all proposers are cunning, attesters are obedient, and $\rho \geq 1/2$, the cunning strategy is a best response if the cunning condition holds for the second proposer and:*

$$\frac{f_{k-2} - f_{k-1}}{2} \geq \frac{ax}{27}, \tag{V.7}$$

where f_k denotes the transaction fees emitted at slot k.

Proof. If the cunning condition is true for the second proposer ($w_g + a - w_f \leq \rho a$) it can attach its block to the branch with attestation weight w_f since the gap with the concurrent chain of weight $w_g + a$ is less than the proposer boost ρa. The obedient attesters of the second slot will add an attestation weight of a to w_f. Following this, the gap between the attestation weights of the two concurrent branches will always remain less than ρa, leading all cunning proposers to attach their blocks two slots prior.

We illustrate in Figure V.6 the "bouncing" that will unfold due to cunning proposers. Their resulting reward will be affected, as the repeated bouncing of the canonical chain between the two branches will cause the blocks from each chain to become canonical with a probability of $1/2$. No attesters will receive the maximum reward since they would never attest in accordance with the following proposer. The reward of the proposer $(0, k)$ following the cunning strategy will thus be:

$$\begin{aligned} u_{(0,k)}(\sigma_{-(0,k)}, \sigma^C_{(0,k)}) &= \frac{1}{2}\left(\frac{a}{7}\cdot\frac{20x}{27} + f_{k-2} + \frac{a}{7}\cdot\frac{20x}{27} + f_{k-1}\right) \\ &= \frac{a}{7}\cdot\frac{20x}{27} + \frac{f_{k-2}+f_{k-1}}{2}, \end{aligned} \tag{V.8}$$

with $\sigma_{-(0,k)}$ being the strategy profile in which every proposer is cunning and every attester is obedient.

Being cunning is a best response when $w_f + \rho a \geq a$, if and only if:

$$\begin{aligned} u_{(0,k)}(\sigma_{-(0,k)}, \sigma^C_{(0,k)}) &\geq u_{(0,k)}(\sigma_{-(0,k)}, \sigma^O_{(0,k)}) \\ \Leftrightarrow \quad \frac{a}{7}\cdot\frac{20x}{27} + \frac{f_{k-2}+f_{k-1}}{2} &\geq \frac{ax}{7} + f_{k-1} \\ \Leftrightarrow \quad \frac{f_{k-2}-f_{k-1}}{2} &\geq \frac{ax}{27}, \end{aligned} \tag{V.9}$$

where $\sigma_{-(0,k)}$ is the strategy profile in which every proposer is cunning and every attester is obedient, and $\sigma^O_{(0,k)}$ is the obedient strategy. Therefore, if f_{k-2} is not sufficiently greater than f_{k-1}, the best response is the obedient strategy; otherwise, the best response is the cunning strategy. □

Since transaction fees are positive, they cannot continue to decrease indefinitely with each slot. This implies that when the cunning condition holds for the second proposer, eventually one proposer will follow the obedient strategy, thereby stopping the fork.

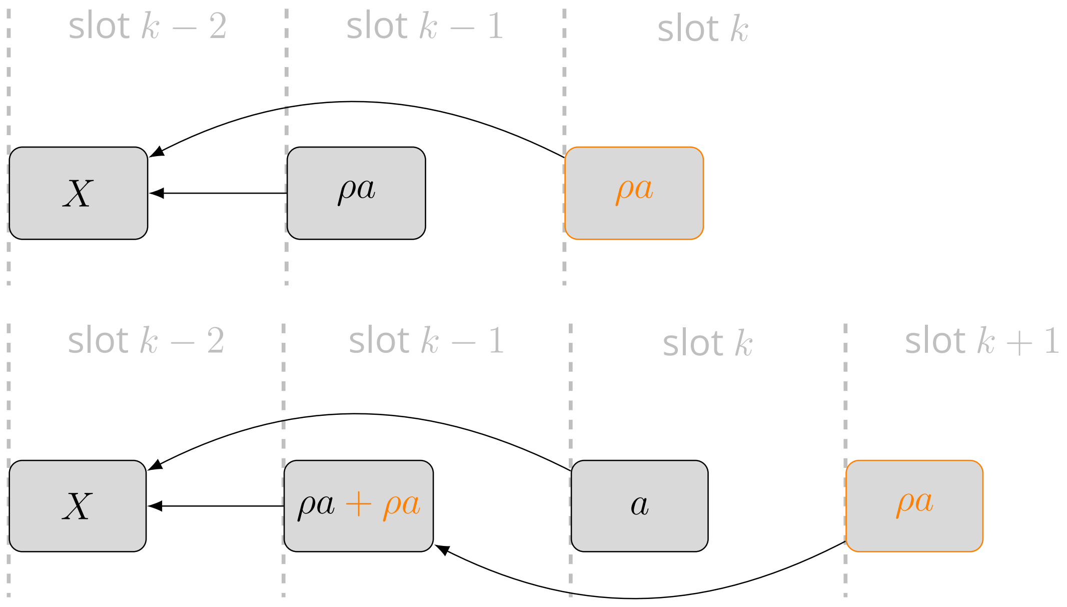

Figure V.6: X indicates that the value of the block is irrelevant. As a reminder, the proposer boost is equivalent to an attestation weight of ρa for a new block (in orange). In this scenario, the proposer of slot k is cunning and all the attesters are obedient. The proposer of slot k takes advantage of the proposer boost to become the head of the canonical chain. The proposer of slot $k+1$ can cunningly become the head of the canonical chain by attaching its block to the block from slot $k-1$ and can become the head of the chain only if $2\rho a \geq a$, which means $\rho \geq 1/2$. In conclusion, a proposer boost greater than $1/2$ can create a situation in which multiple forks occur in the presence of cunning proposers and obedient attesters.

Best response of an attester among s cunning proposers and $an-1$ obedient attesters. Until now, we have described a scenario where all proposers follow the cunning strategy, and attesters follow the obedient strategy, leading to a potentially long fork in which attesters pay the price for the cunning behavior of proposers and do not receive the maximum reward. Let us now introduce the cunning attester strategy, which takes advantage of knowing when the cunning proposer strategy is the best response to act accordingly and secure a higher reward.

Cunning Attester ($\sigma^C_{(i,k)}$):
Action: $\nu_{(i,k)} = \sigma^C_{0,k+1}(\mathcal{T}_k, \mathcal{A}_{k-1} \cup A^O_k)$.

The cunning attester (i,k) attests to the parent of the block in slot $k+1$, assuming that all other attesters in slot k will act obediently (A^O_k) and that the proposer of slot $k+1$ will act cunningly. The action is the same as the following cunning proposer's, i.e., $\nu_{(i,k)} = \phi^C_{k+1}$.

We found that when $\rho < 1/2$, the obedient attester strategy and the cunning attester strategy are equivalent. This implies that attesters will follow the protocol (Lemma V.6) when $\rho < 1/2$. Moreover, if $\rho \geq 1/2$, when all proposers are cunning and attesters are obedient, the best response is the cunning attester strategy. When the cunning condition holds for the second proposer the cunning attester strategy will deviate from the protocol. Otherwise, all attesters will follow the protocol (Lemma V.7).

Lemma V.6. *When $\rho < 1/2$, the obedient attester strategy and the cunning attester strategy are equivalent.*

Proof. This result stems from the fact that when $\rho < 1/2$, only the first proposer can act conspicuously cunningly, meaning they deviate from the protocol (cf. proof of Lemma V.3). Therefore, if all subsequent proposers act similarly to obedient proposers and follow the protocol, attesters will never have the opportunity to act cunningly and will follow the protocol as well. □

Lemma V.7. *When $\rho \geq 1/2$ and all proposers are cunning while all other attesters are obedient, the cunning attester strategy is a best response. If the cunning condition holds for the second proposer, the cunning attester strategy will lead the attester to deviate from the protocol. Otherwise, all attesters will follow the protocol.*

Proof. For attesters to exhibit cunning behavior, more than just the first proposer must act cunningly. This occurs if and only if the cunning condition holds for the second attester.

The reward for attester (i,k) following the cunning attester strategy $\sigma^C_{(i,k)}$, while all other attesters are obedient and proposers are cunning $\sigma_{-(i,k)}$, is:

$$u_{(i,k)}(\sigma_{-(i,k)}, \sigma^C_{(i,k)}) = \frac{47x}{54}. \tag{V.10}$$

Since the attester's reward depends on when their attestation is included in a block, it also depends on whether the blocks belong to the canonical chain. Each attestation is included in the next two blocks, which are on different chains, each having a 1/2 probability of being in the canonical chain. This gives the cunning attester a reward of $\frac{1}{2}(x + \frac{20x}{27})$. Following the obedient attester strategy leads to a reward of $\frac{20x}{27}$, as in both blocks, the attestation will either attest to the wrong block or be included too late. Thus, $u_{(i,k)}(\sigma_{-(i,k)}, \sigma^C_{(i,k)}) \geq u_{(i,k)}(\sigma_{-(i,k)}, \sigma^O_{(i,k)})$. □

Best response of an attester among s cunning proposers and $an-1$ cunning attesters. We study the behavior of an attester when all proposers are cunning and other attesters cunning. In this case the best response is the cunning attester strategy. This strategy only deviates from the protocol for the attesters of the first slot if the cunning condition holds for the second proposer. However this deviation will prevent the cunning condition to hold for the third proposer, effectively making all subsequent validators to follow the protocol.

Lemma V.8. *The cunning attester strategy is a best response for an attester when all validators are cunning. If the cunning condition holds for the second proposer, the cunning attester strategy will lead the attesters of the first slot to deviate from the protocol. Otherwise, all attesters will follow the protocol.*

Proof. For attesters to exhibit cunning behavior, more than just the first proposer $(0,0)$ must act cunningly. This occurs if the cunning condition holds for the second proposer $(0,1)$.

In this case, all attesters of the first slot will expect proposer $(0,1)$ to act cunningly and attach itself to the block designated by the fork choice rule at the beginning of the game ($\mathcal{F}(\mathcal{T}_0, \mathcal{A}_{-1})$), leading them to attest to the head of the branch with total attestation weight w_f. This scenario is represented in Figure V.7. As a result, the block proposed by the first proposer $(0,0)$ will not be attested by the attesters. The block proposed by $(0,1)$ will belong to the canonical chain since no other fork is possible for the subsequent proposers. The gap between $w_f + a$ and w_g is too large for the proposer boost to enable further cunning actions, the cunning condition cannot hold anymore. The attesters $(i,0)$ of the first slot, who act in accordance with proposer $(0,1)$, receive the maximum reward. Proposers $(0,2)$ and beyond will not have the opportunity to act cunningly, nor will the remaining attesters, resulting in all attesters receiving the maximum reward. □

Best response of a proposer among $s-1$ cunning proposers and as cunning attesters. Now that the attester can also act cunningly, let us evaluate the best response of proposers. We found that for $\rho < 1/2$, as usual, only the first proposers can deviate from the prescribed protocol. Doing so will yield more reward hence the cunning strategy is the best response for the first proposer.

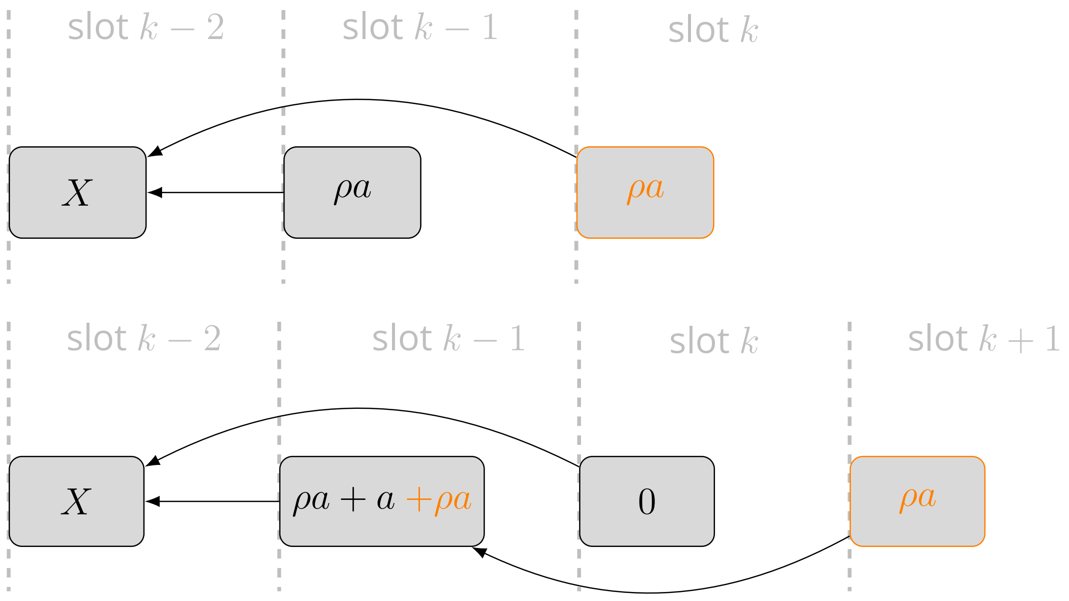

Figure V.7: X indicates that the value of the block is irrelevant. In this scenario, all validators act cunningly. The proposer of slot k attaches its block to the block from two slots prior. The cunning attesters in slot k attest to the block from slot $k-1$ to align with the following proposer's strategy. The proposer of slot $k+1$ is then compelled to attach its block to the block from slot $k-1$. As a reminder, the proposer boost is equivalent to an attestation weight of ρa for a new block (in orange). This results in the proposer of slot k forking alone and receiving no rewards.

When $\rho \geq 1/2$, there are two cases. If the cunning condition does not hold for the second proposer, the best response is the cunning proposer strategy and only the first proposer has the opportunity to deviate from the protocol. On the other hand, if the cunning condition holds for the second proposer, all attesters of the first slot will deviate from the protocol and not attest the block of the first proposer. This makes the obedient proposer strategy the best response for the first proposer as well as all other proposers that will follow the protocol regardless of the first proposer strategy when attesters are cunning.

Lemma V.9. *If $\rho < 1/2$ and all validators are cunning, the best response for the first proposer is the cunning strategy. If the cunning condition holds, it will only do so for the first proposer, causing this proposer to deviate from the protocol.*

Proof. This follows directly from Lemma V.6 and Lemma V.3. $\square$

Lemma V.10. *If $\rho \geq 1/2$ and all validators are cunning, the best response for a proposer is cunning, if the cunning condition does not hold for the second proposer. Otherwise, the obedient proposer strategy is the best response.*

Proof. If the cunning condition holds for the second proposer, Lemma V.8 describes how the scenario would unfold. A possible outcome is represented in Figure V.7. The result is that, for the first proposer, the best response is the cunning strategy only if the second proposer cannot act cunningly.

Otherwise, as described in the proof of Lemma V.8, if the first proposer remains cunning, they will receive zero reward. All other proposers follow the protocol regardless of the strategy of the first proposer. □

A counterintuitive finding is that with a higher proposer boost $\rho > 1/2$, the obedient proposer strategy can be favored. One reason is that the transaction fees gained may not be enough to compensate for the probability of not belonging to the canonical chain. Another reason is that being cunning can backfire if the attesters are also cunning, leading to the cunning block not being attested at all. The assurance of belonging to the canonical chain and the available rewards can be sufficient to make the obedient strategy the favored response.

Eventual Incentive compatibility We have seen that the strategy profile in which all validators are obedient and follow the prescribed protocol is not a Nash equilibria, participants can gain from changing strategy. In this sense the consensus protocol and more precisely the fork choice rule is not incentive compatible.

Nonetheless, we now show that in all equilibria, there exists a slot after which all validators follow the prescribed protocol (Theorem V.1).

Lemma V.11. *Once a proposer follows the protocol, all subsequent validators do so.*

Proof. A proposer $(0, j)$ following the protocol implies that if all attesters of its slot also follow the protocol, the next proposer cannot deviate. This is because all obedient attesters in that slot would give an attestation weight of a to the block proposed by $(0, j)$. Since it extends the branch designated by the fork choice rule, with an attestation weight $w_f \geq w_g$, where w_g is the attestation weight of any concurrent branch, adding a to w_f ensures that no proposer deviate from the protocol (cf. Observation V.1).

Knowing this, the attesters (i, j) of slot j will follow the protocol as well. No validators can deviate from the protocol after a block is attached to the head of the canonical chain. □

Theorem V.1. *In all Nash equilibria, there is a slot after which all validators follow the protocol.*

Proof. We know that once a proposer acts obediently, all subsequent validators do (Lemma V.11).

If there is an equilibrium in which one proposer follows the obedient strategy and extends the head of the canonical chain, the theorem is valid. We now look at proposers all following the cunning strategy. When all proposers follow the cunning strategy, to have more than the first to effectively act cunningly we need to have $w_g + a - w_f \leq \rho a$ otherwise the second proposer will extend the head of the canonical chain, validating the theorem. Then in the case of the fork continuing with each proposer thus attaching their block two slots prior, this makes each of their blocks have an expectation of 1/2 to belong to the canonical chain ($\chi = 1/2$). As computed in Lemma V.4, their reward will thus be in the form of: $f_{k-2} - f_{k-1} \geq \alpha$, where α is a positive number that depends on the attestation included and their reward associated. No matter the value of α, even taking $\alpha = 0$, a proposer will have as best response to be cunning only if the transaction fees gained with a probability 1/2 by being cunning are at least superior to the transaction fees obtained with certainty otherwise.

This condition cannot be true for all proposers as the transaction fees are positive and discrete. Eventually, a proposer $(0, k)$ will see previous transaction fees where $f_{k-2} < f_{k-1}$. The best response of proposer $(0, k)$ is to act obediently. $\square$

We can conclude that under perfect network conditions, regardless of the proposer boost, the obedient strategy will eventually prevail.

V.4 . Conclusion

In this chapter, we have analyzed the Ethereum PoS protocol through a game-theoretic lens, particularly focusing on the incentive mechanisms that influence the behavior of proposers and attesters. Our findings reveal that the current design leads rational validators to all eventually adhere to the protocol. Specifically the proposer boost mechanism does not permit prolonged forks while having good network conditions. Surprisingly, a high proposer boost (superior to $1/2$) can even prevent cunning behavior.

This initial analysis focused on two strategies: cunning and obedient. Future research will expand to include a broader range of strategies. Additionally, exploring different assumptions about network conditions, such as communication with more realistic delays, is expected to significantly impact the results.

— Chapter VI —

Conclusion and Perspectives

Driven by the need to evaluate the impact of incentive mechanisms on blockchain robustness, we embarked on this thesis. Traditional distributed systems approaches, which consider honest and Byzantine participants, often overlook incentives, while game theory focuses on incentives but typically disregards blockchain robustness. Our work aims to bridge this gap by examining the case study of Ethereum PoS.

Ethereum PoS is unique for two reasons: it has a very active research community, and its protocol is a hybrid of Nakamoto-style and BFT-like consensus. These features make it an ideal subject for studying complex ideas that apply broadly to other blockchain systems, as most blockchains incorporate at least one of these fundamental components.

Our analysis began with a detailed examination of the Ethereum PoS protocol. The complexities of this protocol necessitated a dedicated paper [PAT23] focused on its description. In this work, we redefined the crucial properties of safety and liveness, which are fundamental to blockchain robustness and serve as the foundation for our entire study. We formalized the protocol based on its code which we transcribed in pseudo-code. We then revealed that the protocol was safe and the liveness was probabilistic due to a possible attack we identified.

We then shifted our focus to the incentive mechanisms within the Ethereum PoS protocol. While our first analysis was in line with traditional distributed systems approaches and did not consider incentives, the second part of our work [PAT24a] explored the inactivity leak, an incentive mechanism that directly impacts the protocol's robustness. Our results show that the mechanism penalizing seemingly inactive validators to restore liveness in times of partitions could be subverted by Byzantine validators to break the safety.

The final part of our work addresses the behavior of rational participants within the protocol, bridging the gap between distributed system and game theoretic analysis. We investigated whether proposers and attesters could financially benefit from deviating from the protocol, specifically by exploiting the fork choice rule to their advantage. The strategy exploiting the fork choice rule is called cunning while the strategy that adheres to the protocol is called obedient. Our findings state that for any equilibrium, eventually all validators will behave obediently. There are two reasons for this. First we assume perfect network conditions during which messages sent are instantaneously received by all participants. Secondly, not following the obedient strategy can be less rewarding as it can imply that blocks do not eventually end up in the canonical chain, which would yield zero rewards in this case.

Our contributions highlight the critical role of incentive mechanisms in blockchains. Rewards and penalties can either prevent misbehavior, as shown in our game-theoretic analysis, or be detrimental if exploited by Byzantine participants, as illustrated in our analysis of the inactivity leak. This work contributes to the effort of making blockchain protocols more comprehensible. The complexity of the Ethereum protocol is undeniable, and we hope to serve as a resource for a detailed, formalized explanation of the protocol. While we have striven to provide an accurate description of the consensus mechanisms, the examination of how transactions are processed and executed remains. This constitutes an entire work on its own.

Although our work focuses on a single blockchain, the insights gained offer valuable guidance for the design of future protocols. Addressing the intersection of distributed systems and game theory is both recent and challenging. While Abraham et al. [AAH11] introduced the idea of combining these fields in 2011, the complexity involved has led to limited research in this area. One of our goals was to study the protocol with three types of agents: honest, Byzantine, and rational. However, the complexity of the protocol made adding this refinement to the participant model too daunting and, we believe, unfeasible.

Future work should aim to develop simpler protocols that remain robust in the presence of honest, Byzantine, and rational players. Our work provides a framework for evaluating protocols. However, a limitation of this thesis is the absence of a proposed solution to the problems identified.

Blockchain technology is still in its early stages, and we hope that our contribution will aid its continued development.

Bibliography

[AAH11] Ittai Abraham, Lorenzo Alvisi, and Joseph Y. Halpern. Distributed computing meets game theory: combining insights from two fields. *SIGACT News*, 42(2):69–76, 2011. (Cited on pages ↑ 15 and ↑ 96)

[ABPT20] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, and Sara Tucci Piergiovanni. Rational vs byzantine players in consensus-based blockchains. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 43–51. International Foundation for Autonomous Agents and Multiagent Systems, 2020. (Cited on page ↑ 15)

[ACL$^+$19] Musab A. Alturki, Jing Chen, Victor Luchangco, Brandon M. Moore, Karl Palmskog, Lucas Peña, and Grigore Rosu. Towards a verified model of the algorand consensus protocol in coq. In *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part I*, volume 12232 of *Lecture Notes in Computer Science*, pages 362–367. Springer, 2019. (Cited on page ↑ 13)

[ACM20] Ignacio Amores-Sesar, Christian Cachin, and Jovana Micic. Security analysis of ripple consensus. In *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, volume 184 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on page ↑ 13)

[ACP$^+$21] Lacramioara Astefanoaei, Pierre Chambart, Antonella Del Pozzo, Thibault Rieutord, Sara Tucci Piergiovanni, and Eugen Zalinescu. Tenderbake - A solution to dynamic repeated consensus for blockchains. In *4th International Symposium on Foundations and Applications of Blockchain 2021, FAB 2021, May 7, 2021, University of California, Davis, California, USA (Virtual Conference)*, volume 92 of *OASIcs*, pages 1:1–1:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on page ↑ 14)

[ADL$^+$19] Emmanuelle Anceaume, Antonella Del Pozzo, Romaric Ludinard, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Blockchain abstract data type. In *The 31st ACM on Symposium on Parallelism in Algorithms and*

Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019, pages 349–358. ACM, 2019. (Cited on pages ↑ 13 and ↑ 21)

[ADPT18] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Correctness of tendermint-core blockchains. In *22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China*, pages 16:1–16:16, 2018. (Cited on pages ↑ 14 and ↑ 47)

[ADRT21] Emmanuelle Anceaume, Antonella Del Pozzo, Thibault Rieutord, and Sara Tucci-Piergiovanni. On finality in blockchains. In *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, pages 6:1–6:19, 2021. (Cited on page ↑ 13)

[APM+23] Hagit Attiya, Antonella Del Pozzo, Alessia Milani, Ulysse Pavloff, and Alexandre Rapetti. The synchronization power of auditable registers. In *27th International Conference on Principles of Distributed Systems, OPODIS 2023, December 6-8, 2023, Tokyo, Japan*, volume 286 of *LIPIcs*, pages 4:1–4:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on page ↑ iii)

[APPT19] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Dissecting tendermint. In *Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers*, volume 11704 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2019. (Cited on pages ↑ 13 and ↑ 47)

[Bac97] Adam Back. A partial hash collision based postage scheme, 1997. (Cited on page ↑ 5)

[Bac02] Adam Back. Hashcash - a denial of service counter-measure, 2002. (Cited on page ↑ 5)

[BBBC19] Bruno Biais, Christophe Bisière, Matthieu Bouvard, and Catherine Casamatta. The blockchain folk theorem. *Review of Financial Studies*, 32:1662–1715, 05 2019. (Cited on page ↑ 15)

[BCC+23] Alpesh Bhudia, Anna Cartwright, Edward J. Cartwright, Darren Hurley-Smith, and Julio C. Hernandez-Castro. Game theoretic modelling of a ransom and extortion attack on ethereum validators. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*, pages 105:1–105:11. ACM, 2023. (Cited on page ↑ 16)

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017. (Cited on pages ↑ 14, ↑ 15, and ↑ 52)

[BHK+20] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. Combining GHOST and casper. *CoRR*, abs/2003.03052, 2020. (Cited on pages ↑ 13, ↑ 15, ↑ 19, ↑ 21, ↑ 24, ↑ 25, ↑ 25, ↑ 26, ↑ 28, ↑ 28, ↑ 29, and ↑ 33)

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. (Cited on pages ↑ 13 and ↑ 26)

[Bre00] Eric A. Brewer. Towards robust distributed systems (abstract). In Gil Neiger, editor, *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, page 7. ACM, 2000. (Cited on pages ↑ 21 and ↑ 47)

[BRLP20] Vitalik Buterin, Daniël Reijsbergen, Stefanos Leonardos, and Georgios Piliouras. Incentives in ethereum's hybrid casper protocol. *Int. J. Netw. Manag.*, 30(5), 2020. (Cited on page ↑ 15)

[But14] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. (Cited on pages ↑ 8 and ↑ 13)

[CFN88] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer, 1988. (Cited on page ↑ 4)

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 199–203. Plenum Press, New York, 1982. (Cited on page ↑ 4)

[CKWN16] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 154–167. ACM, 2016. (Cited on pages ↑ 16, ↑ 19, ↑ 78, and ↑ 81)

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. (Cited on page ↑ 18)

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. (Cited on pages ↑ 13 and ↑ 18)

[Con22] Consensys. Teku consensus client, 2022. (Cited on page ↑ 30)

[Dai98] Wei Dai. B-money, 1998. (Cited on page ↑ 4)

[DDS87] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987. (Cited on page ↑ 11)

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, pages 288–323, 1988. (Cited on page ↑ 18)

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992. (Cited on page ↑ 5)

[Edg3] Ben Edgington. *A technical handbook on Ethereum's move to proof of stake and beyond*. ETH2 Book, 2023-. (Cited on page ↑ 57)

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer, 2014. (Cited on page ↑ 15)

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, 2018. (Cited on pages ↑ 14 and ↑ 15)

[Fin08] Hal Finney. Mail from finney to, 2008. (Cited on page ↑ 7)

[FMJR20] Mehdi Fooladgar, Mohammad Hossein Manshaei, Murtuza Jadliwala, and Mohammad Ashiqur Rahman. On incentive compatible role-based reward distribution in algorand. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 452–463. IEEE, 2020. (Cited on page ↑ 16)

[Fou24] Ethereum Foundation. Consensus specifications github. https://github.com/ethereum/consensus-specs/tree/

a42d6706d89c414764eda7e2d0103e19f1e23761/specs, 2024. (Cited on pages ↑ 13, ↑ 24, ↑ 24, ↑ 25, ↑ 25, ↑ 25, ↑ 30, ↑ 36, and ↑ 43)

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. (Cited on page ↑ 13)

[GLMV23] Letterio Galletta, Cosimo Laneve, Ivan Mercanti, and Adele Veschetti. Resilience of hybrid casper under varying values of parameters. *Distributed Ledger Technol. Res. Pract.*, 2(1):5:1–5:25, 2023. (Cited on page ↑ 14)

[Goo14] L.M. Goodman. Tezos — a self-amending crypto-ledger, 2014. (Cited on page ↑ 15)

[GP20] Cyril Grunspan and Ricardo Pérez-Marco. Selfish mining in ethereum. In *2nd International Conference on Mathematical Research for Blockchain Economy, MARBLE 2020, online, August 24, 2020*, Springer Proceedings in Business and Economics, pages 65–90. Springer, 2020. (Cited on page ↑ 15)

[GS19] Álvaro García-Pérez and Maria Anna Schett. Deconstructing stellar consensus. In *23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland*, volume 153 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited on page ↑ 13)

[HMR12] Viet Tung Hoang, Ben Morris, and Phillip Rogaway. An enciphering scheme based on a card shuffle. In *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, 2012. (Cited on pages ↑ 30 and ↑ 36)

[HV16] Joseph Y. Halpern and Xavier Vilaça. Rational consensus: Extended abstract. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 137–146. ACM, 2016. (Cited on page ↑ 15)

[LPR20] Andrew Lewis-Pye and Tim Roughgarden. Resource pools and the cap theorem, 2020. (Cited on page ↑ 47)

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. (Cited on pages ↑ 4, ↑ 10, and ↑ 17)

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. (Cited on pages ↑ 4, ↑ 13, and ↑ 24)

[Nak19a] Ryuya Nakamura. Analysis of bouncing attack on ffg, 2019. (Cited on pages ↑ 40 and ↑ 40)

[Nak19b] Ryuya Nakamura. Prevention of bouncing attack on ffg, 2019. (Cited on pages ↑ 14, ↑ 40, and ↑ 41)

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 305–320. IEEE, 2016. (Cited on page ↑ 15)

[NMRP20] Michael Neuder, Daniel J. Moroz, Rithvik Rao, and David C. Parkes. Defending against malicious reorgs in tezos proof-of-stake. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 46–58, 2020. (Cited on page ↑ 14)

[NTT21] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 446–465. IEEE, 2021. (Cited on pages ↑ 14, ↑ 23, ↑ 40, ↑ 47, and ↑ 73)

[NTT22] Joachim Neu, Ertem Nusret Tas, and David Tse. Two more attacks on proof-of-stake ghost/ethereum. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus, ConsensusDay 2022, Los Angeles, CA, USA, 7 November 2022*, pages 43–52. ACM, 2022. (Cited on pages ↑ 14 and ↑ 73)

[PAT23] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake under scrutiny. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023, Tallinn, Estonia, March 27-31, 2023*, pages 212–221. ACM, 2023. (Cited on pages ↑ 12 and ↑ 95)

[PAT24a] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Byzantine attacks exploiting penalties in ethereum pos. In *54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2024, Brisbane, Australia, June 24-27, 2024*, pages 53–65. IEEE, 2024. (Cited on pages ↑ 12 and ↑ 95)

[PAT24b] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Incentive compatibility of ethereum's pos consensus protocol. In *28th International Conference on Principles of Distributed Systems, OPODIS 2024, December 11-13, 2024, Lucca, Italy*, volume 287 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. (Cited on page ↑ 12)

[Pry22] Prysm. Code consensus client, 2022. (Cited on page ↑ 30)

[Qua11] QuantumMechanic. Proof of stake instead of proof of work, 2011. (Cited on page ↑ 8)

[Req19] Specification Pull Request. Bouncing attack patch, 2019. (Cited on page ↑ 40)

[Rou20] Tim Roughgarden. Transaction fee mechanism design for the ethereum blockchain: An economic analysis of EIP-1559. *CoRR*, abs/2012.00854, 2020. (Cited on page ↑ 16)

[Sal20] Fahad Saleh. Blockchain without Waste: Proof-of-Stake. *The Review of Financial Studies*, 34(3):1156–1190, 2020. (Cited on page ↑ 16)

[SNM$^+$22] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 560–576. Springer, 2022. (Cited on pages ↑ 14, ↑ 14, ↑ 14, and ↑ 16)

[SSZ16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 515–532. Springer, 2016. (Cited on page ↑ 15)

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, 2015. (Cited on page ↑ 29)

[TE18] Itay Tsabary and Ittay Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 713–728. ACM, 2018. (Cited on page ↑ 16)

[Woo16] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White paper*, 21(2327):4662, 2016. (Cited on page ↑ 15)

[YMR$^+$19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019. (Cited on page ↑ 47)

[ZET20] Roi Bar Zur, Ittay Eyal, and Aviv Tamar. Efficient MDP analysis for selfish-mining in blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 113–131. ACM, 2020. (Cited on page ↑ 15)

[ZLD23] Mingfei Zhang, Rujia Li, and Sisi Duan. Max attestation matters: Making honest parties lose their incentives in ethereum pos. *IACR Cryptol. ePrint Arch.*, page 1622, 2023. (Cited on page ↑ 15)

— Appendix A —

Mathematical Elaborations

A.1 . Discrete case inacivity score during a probabilistic bouncinc attack.

This gives two Bernoulli laws where the probability to have k moves to the left at time n are respectively:

$$P_X(k,n) = \binom{n}{k} p_0^k (1-p_0)^{n-k} \tag{A.1}$$

$$P_Y(k,n) = \binom{n}{k} (1-p_0)^k p_0^{n-k} \tag{A.2}$$

We can then compute the convolution that will give us the probability law to be:

$$P_{X+Y}(s,2n) = [P_X + P_Y](s) = \sum_{k=0}^{n} P_X(k) * P_Y(s-k) \tag{A.3}$$

$$= \sum_{k=0}^{n} \binom{n}{k} p^k (1-p_0)^{n-k} \binom{n}{s-k} (1-p_0)^{k-s} p_0^{n-s+k} \tag{A.4}$$

$$= \sum_{k=0}^{n} \binom{n}{k} \binom{n}{s-k} p_0^{n+2k-s} (1-p_0)^{n+s-2k} \tag{A.5}$$

We are interested in the inactivity score in an attempt to study the evolution of stake of honest validators. To determine the stake we have to give a continuous function for the probability of the inactivity score.

Continuous case There are several ways to approach the continuous case. We use the same technique as before using a convolution. Starting by saying that a random walk follows a Gaussian's distribution when time is big using the Central limit theorem. Knowing that the expectation of the two laws P_X and P_Y are $(5p_0 - 4)t$ and $(1-5p_0)t$ and their standard deviation is both $25p_0(1-p_0)t$, we have:

$$P_X(x,t) = \frac{1}{\sqrt{\pi 50 p_0(1-p_0)t}} e^{-\frac{(x-(5p_0-4)t)^2}{50p_0(1-p_0)t}} \tag{A.6}$$

$$P_Y(x,t) = \frac{1}{\sqrt{\pi 50 p_0(1-p_0)t}} e^{-\frac{(x-(1-5p_0)t)^2}{50p_0(1-p)t}} \tag{A.7}$$

$$P_{X+Y}(s,t) = \int P_X(x,t) P_Y(s-x,t)dx \tag{A.8}$$

Which gives:

$$P_{X+Y}(x,t) = \frac{1}{\sqrt{\pi 100 p(1-p_0)t}} e^{-\frac{(x-3t/2)^2}{100p_0(1-p_0)t}} \tag{A.9}$$

A.2 . From Gaussian white noise to log-normal distribution

In order to be able to find the probability of s, we need to change referential to stop I from drifting with time. To do so we start by noticing that with the change of variables $u = -2^{26}\ln|s|$ this implies $\frac{du}{dt} = -\frac{2^{26}}{s}\frac{ds}{dt} = I$. We now simplify what we looking for by introducing $\tilde{u}$ and $\tilde{I}$ the functions resolving these equations:

$$\begin{cases} I = \tilde{I} + Vt \\ u = \tilde{u} + \frac{1}{2}Vt^2 - 2^{26}\ln(s_0) \end{cases} \tag{A.10}$$

Where $s_0 = 32$, for the initial stake. We can write the probability of $\tilde{I}$ as:

$$\phi(\tilde{I}, t) = \frac{1}{\sqrt{4\pi Dt}} e^{-\frac{\tilde{I}^2}{4Dt}} \tag{A.11}$$

Looking at the derivative of $\tilde{u}$ we get:

$$\frac{d\tilde{u}}{dt} = \frac{du}{dt} - Vt = I - Vt = \tilde{I}. \tag{A.12}$$

Hence we find $d\tilde{u}/dt = \tilde{I}$. $\tilde{I}$ being a Brownian motion, $\tilde{u}$ is called an integrated Brownian motion. It is a well-known problem and this leads to :

$$P(\tilde{u}, t) = \frac{1}{\sqrt{\frac{4}{3}\pi Dt^3}} \exp\left(-\frac{\tilde{u}^2}{\frac{4}{3}Dt^3}\right). \tag{A.13}$$

Where :

$$\tilde{u} = u - \frac{Vt^2}{2} + 2^{26}\ln(s_0) \tag{A.14}$$

We have that $d\tilde{u} = -2^{26}\frac{ds}{s}$, then the only remaining step is using the fact $P(s) = P(\tilde{u})|\frac{d\tilde{u}}{ds}|$, hence:

$$P(s) = \frac{2^{26}}{s} P(\tilde{u} = -2^{26} ln(s/s_0) - Vt^2). \tag{A.15}$$

Thus, the probability of finding a stake s at time t for an honest validator during the probabilistic bouncing attack is:

$$P(s, t) = \frac{2^{26}}{s\sqrt{\frac{4}{3}\pi Dt^3}} \exp\left(-\frac{(2^{26}\ln(s/32) + Vt^2/2)^2}{\frac{4}{3}Dt^3}\right) \tag{A.16}$$

With D and V, the diffusion and the velocity. In our case $V = 3/2$ and $D = 25p_0(1-p_0)$. The stake follows a log normal distribution.

The density of log normal distribution is:

$$f_X(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right). \tag{A.17}$$

The cumulative distribution function of the log-normal distribution is the following:

$$F_X(x; \mu, \sigma) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left[\frac{\ln(x) - \mu}{\sigma\sqrt{2}}\right]. \tag{A.18}$$