universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Simulation and Analysis of Fork Behaviour in the Ethereum Blockchain

verfasst von | submitted by

Lukas Wegscheider BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Ing. Mag. Dr.techn. Edgar Weippl

Acknowledgements

I would like to express my deepest gratitude to my parents for making everything in my life possible, opening all doors for me, and for their unwavering support. I am also immensely thankful to my grandfather and grandmother for their encouragement and care throughout this journey. I would like to extend my thanks to the University of Vienna and the University of Stockholm, where I had the opportunity to make lifelong friends enriching my academic and personal experiences. Finally, I am sincerely grateful to my supervisor Nicholas Stifter for his invaluable guidance and mentorship along this path.

Abstract

Forks are a phenomenon that occur in permissionless blockchains utilizing variants of Nakamoto Consensus, such as Bitcoin or Ethereum, leading to competing chains and conflicting states. Non-finality, a consequence of the Nakamoto Consensus and forks, means that transactions or blocks are not immediately final and can be revoked or not included in any block if a competing chain with greater accumulated Proof-of-Work or another dominant resource, based on the consensus technique, emerges. Research on non-finality and forks is an active area with many open questions, both theoretically and in terms of understanding the risk of these phenomena in practice. Empirical data on forks is scarce, and although blockchains may appear to be immutable public ledgers, data on forks is generally not recorded and is challenging to obtain, since actual fork data has to be actively monitored. This is surprising given the potential negative impact forks can have on the network, as demonstrated by events like the DAO hack, which led to the split between Ethereum and Ethereum Classic and the fork between Bitcoin and Bitcoin Cash. These events showcased limitations in blockchain security and consensus mechanisms, highlighting the need for a deeper exploration of fork behavior. During Ethereum's transition from Proof-of-Work to Proof-of-Stake, a high-risk consensus change known as The Merge, the potential for unintended forks was high due to protocol upgrades and possible client implementation bugs. This risk was evident and did come true: a fork occurred that, to our knowledge, is not known within the community or among researchers. By monitoring this transition, we could retrieve data from this fork and this observation forms the motivation and starting point of this thesis. The transition confirmed noteworthy security challenges and underlined the potential for unintended forks, further emphasizing the critical nature of this research. This thesis sets out three core objectives to address these challenges and gain a deeper understanding of forks. Firstly, it aims to better understand forks through empirical analysis of fork data, mainly focusing on the observed merge fork. Secondly, it provides a comprehensive overview of the literature on forks, presenting the current state-of-the-art research and identifying gaps and open questions. Thirdly, it develops and adapts methods to better understand forks and their impact, which do not require active monitoring. Specifically, this includes the development of a blockchain fork simulator to model and analyze fork scenarios. By achieving these objectives, this thesis contributes to the broader understanding of blockchain consensus protocols, particularly in the context of non-finality and forks. The proposed simulator offers a valuable tool for researchers and developers to study and imitate the effects of forks, ultimately aiming to enhance the reliability and robustness of blockchain systems.

Kurzfassung

Forks sind ein Phänomen, das in permissionless Blockchains wie Bitcoin oder Ethereum auftritt, die auf dem Nakamoto-Konsens basieren. Forks führen zu konkurrierenden Chains und daraus folgenden widersprüchlichen Zuständen einer Blockchain. Ein entscheidender Aspekt des Nakamoto-Konsenses ist die Nicht-Finalität, die bedeutet, dass Transaktionen oder Blöcke nicht sofort als abgeschlossen betrachtet werden können. Diese können widerrufen oder nicht in einen Block aufgenommen werden, falls eine konkurrierende Chain entsteht, die mehr akkumulierte Rechenleistung oder eine andere dominante Ressource aufweist, die im Konsensverfahren relevant ist. Die Forschung zu Nicht-Finalität und Forks ist ein Thema mit vielen offenen Fragen, sowohl auf theoretischer Ebene als auch in Bezug auf das praktische Verständnis der Risiken dieser Phänomene. Empirische Daten zu Forks sind jedoch selten. Obwohl Blockchains als unveränderliche, öffentliche Systeme gelten, werden Fork-Daten oft nicht aufgezeichnet und sind schwer zu beschaffen. Dies macht es notwendig, Fork-Daten aktiv zu überwachen, um ein vollständigeres Bild zu erhalten. Angesichts der potenziell negativen Auswirkungen, die Forks auf ein Netzwerk haben können – wie der DAO-Hack, der zur Spaltung zwischen Ethereum und Ethereum Classic führte – ist dies überraschend. Solche Ereignisse haben die Risiken der Blockchain-Sicherheit und der Konsensmechanismen aufgezeigt und unterstreichen die Notwendigkeit einer tiefergehenden Erforschung des Fork-Verhaltens. Während des risikoreichen Wechsels von Ethereum von Proof-of-Work zu Proof-of-Stake, bekannt als „The Merge“, bestand ein erhebliches Risiko unbeabsichtigter Forks. Diese Gefahr ergab sich aus Protokoll-Updates und möglichen Implementierungsfehlern der Software. Tatsächlich trat dieses Risiko ein, und es kam zu einem Fork, der, soweit bekannt, in der Community und unter Forschern weitgehend unbemerkt blieb. Durch die Überwachung dieses Übergangs konnten wir Daten zu diesem Fork sammeln, was den Ausgangspunkt und die Motivation für diese Arbeit darstellt. Diese Arbeit verfolgt drei Hauptziele, um diese Herausforderungen zu adressieren und ein tieferes Verständnis für Forks zu entwickeln. Erstens soll durch eine empirische Analyse von Fork-Daten, insbesondere des beobachteten Merge-Forks, ein besseres Verständnis für Forks erreicht werden. Zweitens wird ein umfassender Überblick über die Literatur zu Forks geboten, der den aktuellen Forschungsstand darstellt und Lücken sowie offene Fragen identifiziert. Drittens werden Methoden entwickelt, um Forks und ihre Auswirkungen besser zu verstehen, ohne aktive Überwachung. Dazu gehört die Entwicklung eines Blockchain-Fork-Simulators, um Fork-Szenarien zu modellieren und zu analysieren. Durch die Erreichung dieser Ziele trägt diese Arbeit zu einem umfassenderen Verständnis von Blockchain-Konsensprotokollen bei, insbesondere im Kontext von Nicht-Finalität und Forks. Der entwickelte Simulator stellt ein wertvolles Werkzeug für Forscher dar, um die Auswirkungen von Forks zu untersuchen und nachzubilden, mit dem Ziel, die Zuverlässigkeit und Robustheit von Blockchain-Systemen zu verbessern.

Contents

List of Tables

List of Figures

Listings

1. Introduction

Blockchain technology is increasingly recognized for its applicability across various industries, offering solutions to a range of use cases. Its decentralized nature, combined with the promise of transparency and security, has positioned itself as a potential game-changer for numerous industries. At its core, blockchain offers a way of conducting and recording transactions without intermediaries, thereby challenging traditional centralized systems. However, like any new technology, blockchain has its challenges. As it continues to evolve and find applications beyond cryptocurrencies, it faces a series of technical and operational issues. Among these challenges, problems related to forks and non-finality stand out. [1] [2] In permissionless blockchains, where Nakamoto consensus is widely used, non-finality and forks are challenges that arise due to the consensus mechanism. Unlike traditional consensus protocols or Byzantine Fault Tolerant (BFT) systems, which achieve finality, these blockchains experience temporary forks and non-finality, leading to potential instability and uncertainty in transaction confirmations. Despite the critical nature of these issues, there is surprisingly limited research and empirical data. This gap in research underscores the need for more in-depth exploration and empirical analysis, which this thesis aims to address through a fork simulator.

In blockchains, a fork is a phenomenon in which the blockchain splits into two separate chains. Forks can occur for various reasons, such as general software updates, known as hard- and soft forks or temporary forks, when there is a brief disagreement among nodes about the longest and valid chain. [3] A soft fork is a backward-compatible change to the blockchain protocol. This means that nodes running the new software version can still communicate with nodes running the old version, but blocks created by outdated nodes won't be valid blocks. Soft forks are generally less contentious and have a higher chance of being adopted by the network. On the contrary, a hard fork is a change to the blockchain protocol that is not backward-compatible, meaning that nodes running the new software version cannot communicate with nodes running the old version. Hard forks generally require most network nodes to adopt the latest software version to succeed. Both types of forks can be defined as controlled forks. [4] [5] On the other hand, temporary forks cannot be controlled and happen due to the way hash-based Proof of Work consensus algorithms work. They lead to situations where multiple nodes propagate a newly created valid block in parallel. [3] Temporary forks happen predominantly, but not only, in blockchains following longest-chain style or Nakamoto-style consensus like Bitcoin and Ethereum (Proof-of-Work). Generally speaking, the longest chain, illustrated in Figure 1.1, refers to the longest valid chain of blocks in a blockchain network. Those chains are secure under certain assumptions, such as an honest majority of computational power in the case of Proof-of-Work, and the newest blocks always have the uncertainty of being revoked. At which point a block can be considered part of the longest chain is a dominant

research topic. [6] Garay et al. [7] define desirable properties that a transaction ledger should satisfy. The common-prefix property, designed by the authors, states that even if nodes prune a certain amount of blocks, k, depending on a general security parameter κ, the remaining part of their local blockchains should match the majority longest chain, meaning that blocks are considered final after a certain amount of blocks have been appended on top of them. This property ensures that correct nodes have the same prefix of blocks, except with negligible probability. It is important to note that this common prefix only holds probabilistically, meaning there is no guaranteed number of blocks k that ensures the remaining chains are always the same. k is considered to be the number of conformation blocks necessary to deem a transaction final, and choosing the correct value, k is a challenging task in practice. [8] However, what exactly are the consequences of such a scenario? Blocks include transactions. Transactions may or may not have certain valuable goods or can depend on transactions invoked in prior blocks. During a fork, a transaction that is valid and approved in one chain may or may not be included in the opposing chain and can only be declared final after a certain amount of time or a particular block depth level. These challenges are not merely theoretical; real-world events, such as the infamous DAO hack that led to the split between Ethereum and Ethereum Classic, underscore the vulnerabilities inherent in the system. Similarly, Ethereum's transition from a Proof-of-Work (PoW) to a Proof-of-Stake (PoS) consensus algorithm has brought security and network stability concerns to the foreground. Such events highlight the limitations of blockchain technologies and emphasize the need for research and development to address these challenges. Blockchain simulators have emerged as essential tools for researchers and developers in this context. These simulators offer a controlled environment to study the behavior of blockchains under various conditions, allowing for a deeper understanding of their dynamics and potential vulnerabilities. In general, not much empirical data is available on forks, as gathering such information requires active monitoring. Simulators can also help facilitate this issue by generating synthetic forks that can be studied in detail to allow researchers to explore various scenarios and their effects on the blockchain. However, it is crucial to validate the simulator to ensure that the synthetic scenarios accurately represent real-world conditions. In this thesis, we extensively investigate the current state of blockchain technology, specifically exploring the challenges related to forks, non-finality, and consensus. While these issues have been extensively discussed independently, we aim to consider them together in the same context, understanding their interdependencies and impact on blockchain networks' overall behavior. In the subsequent chapters, we will present a practical solution for a blockchain fork simulator, setting the stage for a detailed discussion.

1.1. Research Questions

This thesis investigates the potential impact of forks on cryptocurrencies and blockchains. Given the wide range of cryptocurrency designs and protocols, the scope of this thesis will focus on Ethereum. Ethereum's active developer community and extensive resources make understanding and testing fork scenarios easier. At first, this work will investigate

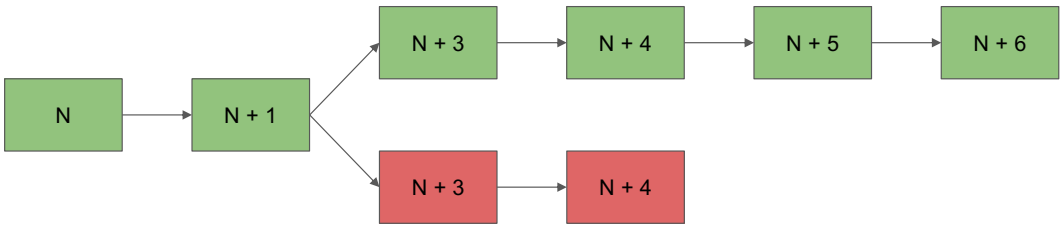


Figure 1.1.: Longest-Chain-Protocol: The longest chain refers to the valid chain with the most accumulated work or blocks in a blockchain network. In this visualization, N represents the current block number, the green blocks denote the current longest valid chain, while the red blocks represent a discontinued branch of blocks.

the risks of forks by analyzing a real-life deep fork that occurred during Ethereums' transition from Proof-of-Work to Proof-of-Stake. The fork we studied is, to the best of our knowledge, unknown in the community and was never mentioned in any grey or white literature. Generally, forks, more precisely deep forks, only occur occasionally, and there is little to no scientific analysis of such forking events and their impact available. We call them deep forks because a certain amount of blocks have already been appended to the forked chain. This leads to our first research question.

RQ1: What is the potential impact of (deep) forks in the Ethereum Blockchain?

One major point of this work is to assess the current state of the art of research in the field of blockchain forks, non-finality, and simulations, which leads to our second research question:

RQ2: What is the state-of-the-art research on fork behaviour of blockchain technologies?

We must explore the factors and techniques needed to create fork simulations that accurately reflect the complexities of real situations. This understanding is essential for developing a reliable and effective simulator.

RQ3: How can realistic fork simulations in Ethereum be performed?

Finally, the proposed simulator examines the risks and influence of blockchain forks. At first, though, it is essential to establish a proper validation and testing framework and

outline its limitations, our final research question:

RQ4: What are the appropriate techniques for validating fork simulation models?

1.2. Outline

The primary content of this thesis is separated into two distinctive parts: First, we will discuss the general problem regarding forks in both a theoretical and practical way and will then continue describing our process of developing our own practical solution for a blockchain fork simulator.

1. **Introduction**
   This chapter offers a short overview of this work and briefly introduces its key objectives.

Part I - Non-Finality and Forks in Cryptocurrencies and Distributed Ledgers

2. **Technical Background - Blockchain Glossar**
   This section covers the basic concepts of blockchain technologies and elaborates on commonly used terms in this thesis. Understanding these fundamentals is crucial for understanding more advanced topics and discussions in further chapters. We examine the general blockchain structure, decentralization, and consensus techniques.

3. **Motivation: Identification and Analysis of a Deep Fork during The Merge in Ethereum**
   This chapter provides an extensive examination of the transition from Proof-of-Work to Proof-of-Stake, referred to as "The Merge," within the Ethereum network. Hereby, the focus of attention is a largely unknown deep fork that occurred during the Merge, encompassing a thorough analysis of various aspects, including node statistics, block statistics, transaction statistics, and an investigation into state divergence.

4. **State of the Art**
   A systematic literature review is performed to understand existing research, identify gaps in knowledge, and establish a foundation for new research. Topics include non-finality, simulation models with a primary focus on blockchain forks.

Part II - Development of a Fork Simulator for Ethereum

5. **Execution & Consensus Layer**
   This chapter discusses the concept of Ethereum clients, their distributions in the network, and predominantly the interaction between consensus and execution clients.

6. **Proof of Concept - Implementation of a Fork Simulator**
   The stages and steps of creating a proof-of-concept Ethereum fork simulator implementation will be explained, with an exhaustive description of its architecture and functionality.

7. **Validation**
   The concept of how to validate the proposed fork simulation model is discussed in this chapter. Furthermore, validation results are presented, and any limitations and observations are outlined.

8. **Future Work**
   This chapter outlines potential areas for further investigation, addressing limitations, proposing extensions, and identifying unanswered questions.

9. **Conclusion**
   The results are summarized based on the research questions, limitations are discussed, and incentives for future research are given in this section.

Part I.

Non-Finality and Forks in Cryptocurrencies and Distributed Ledgers

Introduction

The first part of this thesis discusses the general problem of forks from both a research and practical perspective. Firstly, we will discuss blockchain-related terms and concepts, focusing on the Ethereum blockchain, although many concepts are also related to blockchains in general. To illustrate these concepts, we will then conduct an extensive examination of a deep fork during the transition from Proof-of-Work to Proof-of-Stake in Ethereum. As already mentioned, the fork we studied is unknown in the community to the best of our knowledge and was never mentioned in any grey or white literature. As far as we are aware, this is also the first extensive analysis of a deep fork, encompassing a thorough analysis of various aspects, including node statistics, block statistics, transaction statistics, and an investigation into state divergence. This analysis is crucial to understanding how a realistic fork simulation can be performed. Building on this real-world example, we will follow with a systematic literature review that analyzes the state-of-the-art research on forks and identifies possible research gaps. We compare and discuss the different types of forks, especially how and why they happen, including potential risks and implications that come with forks. Similarly to forks, we look closer at consensus in permissionless distributed ledgers and discuss the non-finality problem. This literature review aims to understand the connections between forks, finality, and consensus.

2. Technical Background - Blockchain Glossar

Within this thesis, we are using a lot of blockchain-specific terminology. Therefore, we are going to use this chapter as a glossary for certain technical terms. Generally speaking, there is a lot of ambiguity in blockchain-related terms, especially regarding consensus algorithms like Proof-of-Work and Proof-of-Stake, which are merely consensus techniques with different implementations and designs. Within the scope of this thesis, we will focus on Ethereum-related designs and leave other architectures to future work.

2.1. Proof of Work (PoW) - Ethereum

Before transitioning to Proof-of-Stake, Ethereum used a Nakamato-based Proof-of-Work consensus algorithm, similar to Bitcoin. [9] The concept of Proof-of-Work is to extend the longest chain by mining new blocks. [10] [11] Miners compete to create blocks, which can lead to multiple branches of the chain emerging at the same time until one of these chains becomes the accepted chain again. The Ethhash algorithm, Ethereums' Proof-of-Work protocol, required miners to brute-force search for a nonce that would lead to a valid proof-of-work over the Ethereum block header. Ethhash was explicitly designed to be ASIC-resistant, which aims to prevent using specialized hardware (ASICs) for mining, thus promoting a more decentralized mining ecosystem. [9]

2.2. Proof of Stake (PoS) - Ethereum

Next to Proof-of-Work, Proof-of-Stake counts as one of the most crucial consensus techniques, with the first functioning use of PoS for cryptocurrency being Peercoin [1] in 2012. [12] Ethereum switched its consensus algorithm to PoS in favor of Proof-of-Work. While Proof-of-Work follows a competitive mining concept in which computing power is the most valuable asset, stake supply is the backbone of the Proof-of-Stake consensus algorithm. Ethereums' Proof-of-Stake-based blockchain keeps track of a set of validators, and every user holding a sufficient amount (at the time of writing 32 Ether) of the blockchain's base cryptocurrency can be one of the validators. Validator rights on voting and validating the next block is a quasi-random process in which the selection accounts for the amounts of assets staked. For every accepted block, validators are rewarded, similar to the block reward used in Proof-of-Work. [12] [9]

[1] https://www.peercoin.net

2.3. Transactions

A transaction is a fundamental operation representing the transfer, modification, or creation of data or assets in a distributed ledger. It encapsulates a specific action or intent initiated by a participant within the blockchain network. A unique address identifies both the sender and the receiver. Depending on the consensus algorithm, transactions undergo verification by network nodes or validators, ensuring they adhere to the predefined rules and requirements. Once validated, they are included in a block and executed sequentially. A distinctive feature of Ethereum is the existence of smart contracts. Smart contracts find applications in decentralized crowdfunding, insurance, and entertainment programs. Their nearly Turing completeness allows for handling complex logic and automating diverse processes. [9] Smart contracts are only nearly Turing complete to hinder the halting problem. The halting problem, stating the impossibility of determining whether a Turing machine will halt or loop indefinitely for all possible inputs, is resolved using the concept of gas, an internal pricing system for executing transactions. Every computation in a transaction costs gas, and the transaction's sender sets the maximum amount of gas intended for the execution. [13]
In this thesis, we want to analyze the possible state divergence occurring whenever a fork happens, in which transactions play a decisive role. Because of a fork, a transaction may or may not be valid or could fail, and there are several reasons for that. If a transaction becomes invalid in a fork, it cannot be processed, which may lead to a diverging state. On the other hand, if a transaction fails in one branch but not in the other, it could also alter the state. We will focus on the reasons that could appear due to forks:

- Insufficient funds: The transaction is only valid if the sender has enough funds to cover the transaction amount or transaction fees. During a fork, there may be an imbalance in the account balance on the separate chains.

- The transaction is invalid if the nonce does not match the expected nonce for the sender's account. A nonce is a number that ensures the order of transactions from an account. [9] For example, a nonce mismatch can happen with the creator's equivocation, i.e., creating a second transaction with the same nonce or a situation where a previous transaction has become permanently invalid and, therefore, a transaction with the next nonce has no chance of eventually being included. We will go into a more detailed discussion in 7.3.1.

- Contract Execution failure: Due to errors in the code or code conditions not being met, the execution of the smart contract can fail. Smart contracts can act upon specific OPCODEs, that are able to access state variables, which could change the execution behavior and alter the state. [9] We refer the reader to 7.1 for a more in-depth discussion on the topic of OPCODES. Stifter et al. [14] also discuss the security concerns regarding the state-dependence of OPCODES during forks in greater detail.

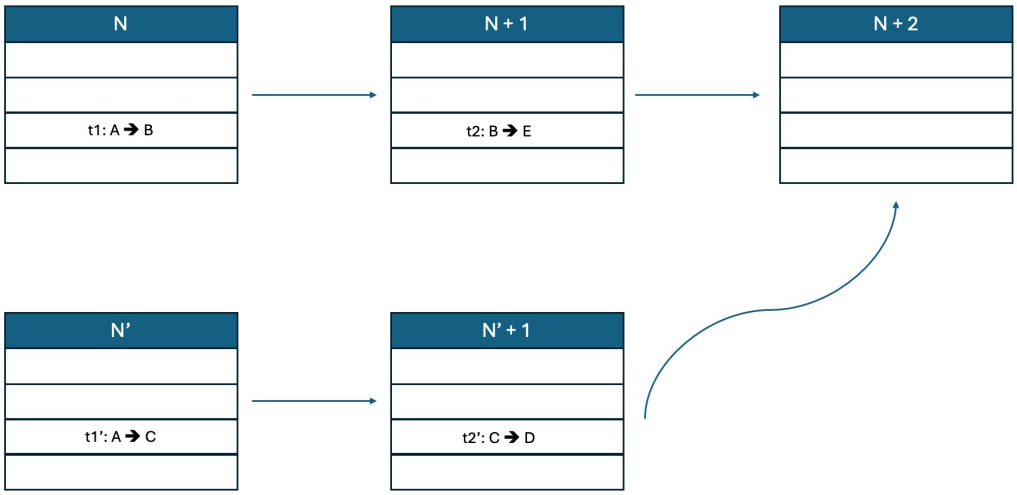


Figure 2.1.: This figure illustrates the possible state divergence influenced by a fork. A currently forked blockchain executes the same transaction t1 and t1' with the same initial state A, resulting in different final states after a transaction within the block N' was impacted by a fork. This results in further state discrepancies with transactions t2 and t2'.

2.4. State

The state in the Ethereum blockchain refers to a global snapshot of the entire network dataset, the entirety of all accounts, balances, and smart contracts. The information is stored in a specific data structure called a Merkle-Patricia Trie, a combination of a Merkle Tree and Patricia Trie. This data structure establishes a verifiable connection among all individual nodes in the tree, resulting in a root value, enabling the verification of properties related to the data, allowing efficiency for inserts, lookups, and deletes. [13] [9] The state is updated with every new block appended to the chain after every transaction is invoked and every smart contract is executed. Figure 2.1 visualizes a transaction with the same initial state A on two conflicting chains and a different final state after a transaction was impacted by a fork.

2.5. Difficulty

The difficulty of Ethereum was a quantifiable measure of the complexity of mining a new block. It acted as a dynamic modifier to ensure the block creation time stayed consistent. It was a scalar value calculated by the previous blocks' difficulty and timestamp. [15] [16] Because of the switch from Proof-of-Work to Proof-of-Stake, the difficulty parameter in

Ethereum has no use anymore, although it remains in the block structure. As part of the merge specifications, it is set to 0.

3. Motivation: Identification and Analysis of a Deep Fork during The Merge in Ethereum

The transition of Ethereum from a Proof-of-Work (PoW) to a Proof-of-Stake (PoS) consensus mechanism, referred to as "The Merge," has been a topic of anticipation and speculation for over half a decade. [17] After numerous delays and extensive testing, this monumental shift was officially implemented on September 15, 2022. The Merge signifies the unification of the original Ethereum Mainnet with a distinct PoS blockchain known as the Beacon Chain. [9] The Beacon Chain, launched in 2020, was a precursor to Ethereum's transition to a PoS consensus mechanism. Its primary function was to validate the security and sustainability of the PoS mechanism before its integration into the Ethereum Mainnet. Consequently, the original Ethereum chain discontinued its mining and consensus logic, shifting this tasks to the Beacon Chain. As reported by the Ethereum Foundation, this transition reduced Ethereum's energy consumption by 99.95%. [9] After analyzing the client code, we determined that a hard fork could occur if some miners continued using outdated client versions. Given that certain mining community members opposed Ethereum's transition to a new consensus mechanism, the likelihood of a hard fork became apparent. [18] [19] This presented a unique opportunity to gather real-world data on a hard fork, which has never been published in grey or white literature, providing valuable insights into fork behavior. Our monitoring efforts confirmed this hypothesis, as we observed a fork that emerged as expected. However, this fork was short-lived, continuing only for a few blocks up to block 15537420. We call this hard fork PoW Merge Fork.

3.1. Methodology

To gather all information effectively, we formulated separate research questions to guide our study process. These included: *Which client should we use? What is the optimal number of nodes to operate? Is the version of the client a factor that requires consideration?*
Upon thorough consideration, we established a node setup with two distinct configurations: a monitoring setup and an analyzing setup. This differentiation was essential in tailoring the setup to fulfill the specific requirements of each phase of our research. The monitoring setup was designed to observe and record real-time data, while the analyzing setup was configured to process and interpret the collected data.

3.1.1. Monitoring Setup

Our monitoring setup includes three virtual machines (VMs) running different Go-Ethereum (geth) client versions. With the Terminal Total Difficulty (TTD) being set at 58750000000000000000000[1], the implementation of a trigger for the impending Merge, we strategically selected versions without this updated TTD. Specifically, the Sentry Omega version (v1.10.23[2]) introduced the TTD, a threshold attained on September 15, 2022 at block height 15537393 at 6:42:42 UTC[3]. Consequently, two VMs were configured to operate nodes running versions v1.10.22[4] and v1.10.21[5]. Both versions did not implement the TTD and were, therefore, not merge-ready. We expected identical output from both nodes but chose to run both as a precaution in case one failed or a hidden TTD trigger was implemented in the previous version, v1.10.22. The Go-Ethereum client is programmed to reject blocks that are considered incorrect automatically. To capture all available data, including blocks potentially regarded as invalid by the protocol, we altered our clients' source code. This required a thorough comprehension of the client's operations, precisely the criteria triggering removing blocks deemed invalid by geth. We implemented a simple function that exports the desired block into a readable JSON file. These files were subsequently utilized to examine any anomalies during the merge process. One of the VMs ran an updated and current version of the Go-Ethereum client in archive mode. An archive node maintains a comprehensive record of all block states, encompassing every transaction from the genesis block to the most recent one, including every balance modification, contract creation, and contract function invocation on the network. Full nodes only maintain the blockchain state of the last 128 blocks. Such a node can compare the complete blockchain state at any block level with the corresponding block of potential forks, but it has some drawbacks. [9] An archive node requires over 12 TB of disk storage with fast read/write performance and immense synchronization times for bootstrapping a new archive node.

3.1.2. Analysis Setup

Besides the already mentioned geth nodes, we operated an additional archive node to analyze the scanned blocks. For this purpose, we chose an Erigon archive client, primarily due to resource efficiency considerations. The Erigon archive node only needs half the storage space as an archive node running Geth. [20] We used the two modified monitoring nodes to bootstrap the Erigon node to the chain head of the PoW Merge Fork chain. Bootstrapping refers to syncing a new node with the existing blockchain. In our case, the origin blockchain was the PoW Merge Fork chain. For data analysis, visualization, and interpretation, we utilized Jupyter Notebooks. Jupyter Notebook is an open-source web

[1] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-3675.md
[2] https://github.com/ethereum/go-ethereum/releases/tag/v1.10.23
[3] https://etherscan.io/block/15537393
[4] https://github.com/ethereum/go-ethereum/releases/tag/v1.10.22
[5] https://github.com/ethereum/go-ethereum/releases/tag/v1.10.21

application that facilitates creating and sharing documents containing live code, equations, visualizations, and narrative text. It is compatible with many programming languages and is extensively employed in data science, scientific computing, and education. [21] To interface directly with the blockchains, we used web3.py. Web3.py[6] is a Python library specifically designed for interaction with the Ethereum ecosystem.

3.2. Node Analysis

Before conducting an in-depth state analysis of the blocks and blockchain state on the PoW Merge Fork, we investigated the behavior of Proof-of-Work specific parameters and their impact on the entire ecosystem. Considering the obsolete nature of miners by the newly introduced Proof-of-Stake consensus algorithm, we focused our examination on the hash rate and the difficulty value. These parameters, integral to the PoW mechanism, underwent significant changes in the transition to PoS, and their analysis provides valuable insights into the impact of this transition on the Ethereum blockchain.

3.2.1. Hashrate Analysis

The primary question we addressed revolved around the reaction of miners to the shift in the consensus algorithm. This was investigated by analyzing the network's hash rate and drawing comparisons with other notable cryptocurrencies. The hash rate, which signifies the speed at which a miner's hardware can execute cryptographic calculations, is a crucial factor in a hash-based Proof-of-Work system. A higher hash rate enhances a miner's likelihood of being the first to solve the necessary proof-of-work for creating a valid block, thereby earning the cryptocurrency reward. [9] With Ethereum's decision to discontinue relying on Proof-of-Work and the associated mining process, miners were left with two options: shut down their mining equipment or transition to a similar cryptocurrency. We wanted to analyze whether the hashing rate dissolves or if miners switch to other cryptocurrencies. Miners can generate income with mining. To participate in mining, miners must invest in specific hardware and generally join mining pools to achieve more regular payouts. It's important to note that different PoW algorithms are better suited to particular types of hardware. [22]
The hash rate is calculated by measuring the number of hash operations performed per second by all the miners in the network. [23] Calculating the hash rate for all analyzed blockchains would require an extensive technical setup, including running at least one full node for each monitored blockchain, which was beyond the scope of this thesis. Therefore, we relied on publicly available data from third-party providers. However, as the data was not directly accessible, we used screenshots of the graphs provided by these sources.

We selected three cryptocurrencies to analyze the potential transition of miners to alternative blockchains after the Merge. The first two choices were apparent: Ethereum

[6] https://github.com/ethereum/web3.py

Classic, a hard fork of Ethereum, and Ethereum PoW, another hard fork proposed by miners in opposition to the Merge. These choices are logical since they run the same Proof-of-Work (PoW) algorithm, allowing miners to easily switch using the same hardware. The third cryptocurrency we analyzed is Ergo, which was chosen to compare another type of Proof-of-Work.

Ethereum Classic

As previously elaborated, Ethereum Classic is a blockchain successor to the original Ethereum blockchain. It was created in 2016 due to a hard fork of Ethereum, which resulted from a disagreement among Ethereum's developers and community members regarding the appropriate response to an attack on the Ethereum network, the DAO hack. [24] As illustrated in the subsequent Figure 3.1, the hash rate of Ethereum Classic exhibited a consistent growth trajectory, reaching up to 50 terahashes per second (TH/s) before the official merge day. Happening simultaneously with the Merge, the hash rate experienced a steep increase, peaking at 200 TH/s briefly before rapidly declining to 150 TH/s. At the time of writing, the hash rate stabilized at approximately 100 TH/s, marking a 100% increase relative to the pre-merge period.

Figure 3.1.: Ethereum Classic Hashrate before and after Ethereum switched to PoS on September 15, 2022 [25]

Ergo

Ergo is an open-source blockchain platform designed to deliver a more secure, private, and adaptable blockchain infrastructure. It employs a novel consensus algorithm known

as *Ergo Proof of Work*, which claims to be more energy-efficient than traditional Proof of Work algorithms. [26] As illustrated in the subsequent Figure 3.2, Ergo's hash rate experienced a notable increase on the merge day, reaching over 160 TH/s. However, this elevation was short-lived, and the hash rate rapidly reverted to its initial level of approximately 20 TH/s.

Figure 3.2.: Ergo Hashrate before and after Ethereum switched to PoS on September 15, 2022 [27]

Ethereum Pow

Ethereum PoW is a hard fork of the original Proof-of-Work Ethereum chain, creating a new blockchain with its first block mined some hours after the Merge. The primary objective of this hard fork of Ethereum is to stick to the proof-of-work mining process for ETH miners. [28] Ethereum PoW wanted to be an alternative to Ethereum PoS by implementing replay protection to prevent replay attacks in their own EIP-155[7]. The Ethereum Community implemented the original EIP-155[8] after replay attacks occurred with the Etherem Classic hard fork. A replay attack occurs when a transaction valid on one blockchain is maliciously or unintentionally repeated on another blockchain. While the nonce prevents the same transaction from being replayed on the same chain multiple times, in the case of a fork, the same transaction can be replayed on the fork, even if the user did not intend for it. [29] [30] As illustrated in the subsequent Figure 3.3, the blockchain started with a hash rate exceeding 60 TH/s and has stabilized at the time of writing, fluctuating between 15 and 20 TH/s.

3.2.2. Difficulty

With a substantial majority of connected node clients in the Ethereum network, exceeding 85%, being Merge ready shortly before the Merge, meaning they were running a client version that implemented the TTD, we expected a sharp decline in the difficulty parameter

[7]https://github.com/ethereum/go-ethereum/commit/c9be794d38f07ea926460077ff6282a2dd336a5e
[8]https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md

Figure 3.3.: Ethereum PoW Hashrate before and after Ethereum switched to PoS on September 15, 2022 [27]

and a minimal number of nodes participating in any potential fork. [18] The data we analyzed is gathered from an external provider and only verified with a string, being the client version, the node replies as part of its communication, which calls the legitimacy of the data into question.[18] As illustrated in Figure 3.4, the PoW Merge Fork blockchain manifested a decrease of nearly 50% in its difficulty within a brief span of 25 blocks. The specifics of the difficulty parameter are explained in Section 2.5.

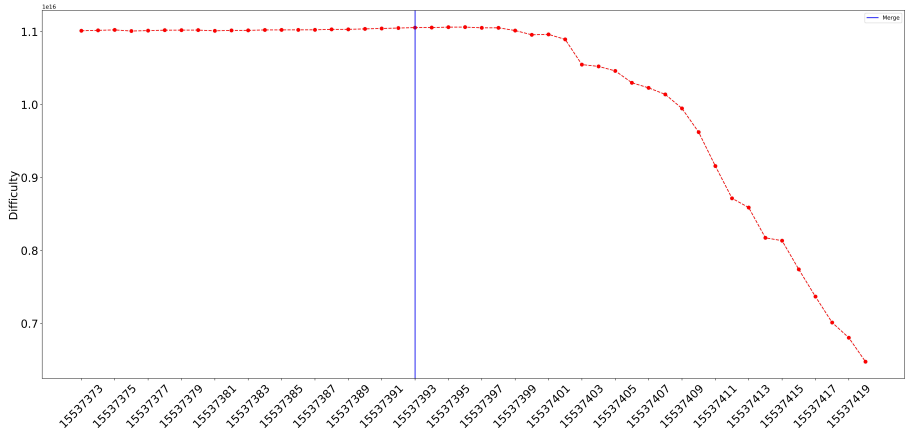


Figure 3.4.: Difficulity Decrease of the continued PoW Merge Chain after the Ethereum Merge to Proof of Stake. The x-axis represents the block height, and the blue line denotes the block height at which the Merge occurred.

3.3. Block Analysis

Using basic statistical methods, we began our analysis by thoroughly examining the blocks we gathered from the PoW Merge Fork chain and comparing them with those in the original blockchain. This investigation played a crucial role in identifying the

distinctions and similarities between the two chains, offering valuable insights into how the fork affected the Ethereum network. During our analysis, we observed 3368 transactions in the main network blocks, in contrast to a remarkably larger number of transactions, precisely 6523, in the PoW Merge Fork blockchain blocks, within the same number of blocks. The difference in transaction volume between the main network and the PoW Merge Fork blocks highlights the distinct dynamics within the two chains after the fork. Based on the transaction inclusion process described in Section 2.3 and visualized in the accompanying Figure 3.5, a substantial 92.6% of the transactions in the PoW Merge Fork blocks were eventually included in the canonical main blockchain until the time of writing. More interesting, though, is why certain transactions were not included in the main chain. The reasons include, also explained in Section 2.3, the dependency on a particular blockchain state or the senders' nonce. We performed random samples and saw different transactions with identical nonces occur on both the PoW Merge Fork and the Ethereum 2.0 chain. This may imply that the sender was actively aware of the fork and performing different actions or that other factors may have caused this, which warrants further investigation for future work. As shown in Table 3.1, the majority of the PoW

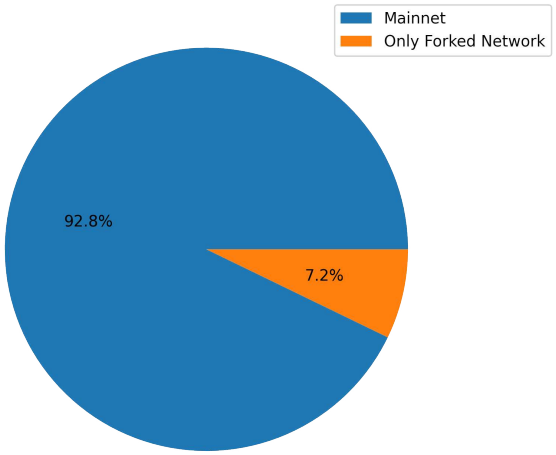


Figure 3.5.: Percentage of how many Transactions were included in the PoW Merge Chain only and not in the Main Chain

blocks received post-merge were generated by mining pools, including influential players like Ethermine [31] and Binance Pool [32]. Ethermine appeared as the most active, mining seven blocks, followed by the Luxor Mining pool[9]. The reasons behind the continued

[9]https://luxor.tech/mining

mining activities of large pools like Ethermine are subject to speculation. One plausible explanation could be an oversight in updating all nodes before the Merge. Given the magnitude of this software update, one of the biggest in the cryptocurrency domain, it is also possible that many mining pools chose to keep their mining equipment running as a precautionary measure in case any problems emerged. However, these remain speculative interpretations without further information and warrant further investigation. Generally speaking, an analysis of this behavior is very difficult since this was an unusual event.

Block Number	**Address**	**Etherscan.io Label**
15537393	0x5b310960a7922092fdcb9295ece336012f9cf87e	BTC.com Pool 2
15537393	0x829bd824b016326a401d083b33d092293333a830	F2Pool Old
15537394	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537395	0xc365c3315cf926351ccaf13fa7d19c8c4058c8e1	Binance Pool
15537396	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537397	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537398	0x646db8ffc21e7ddc2b6327448dd9fa560df41087	Anon1
15537399	0x6ebaf477f83e055589c1188bcc6ddccd8c9b131a	Anon2
15537400	0xc730b028da66ebb14f20e67c68dd809fbc49890d	Ezil Pool 4
15537401	0xc365c3315cf926351ccaf13fa7d19c8c4058c8e1	Binance Pool
15537402	0xc730b028da66ebb14f20e67c68dd809fbc49890d	Ezil Pool 4
15537403	0x6ebaf477f83e055589c1188bcc6ddccd8c9b131a	Anon2
15537404	0x5fc386e5312f24f9416644db87d215056ed4d55d	Unknown Address1
15537405	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537406	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537407	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537408	0xea674fdde714fd979de3edf0f56aa9716b898ec8	Ethermine
15537409	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537410	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537411	0x5fc386e5312f24f9416644db87d215056ed4d55d	Unknown Address1
15537412	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537413	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537414	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537415	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537416	0xf2e3718a3ea5e0af0fbf6de3cc479fd00d54b5dc	Unknown Address2
15537417	0x26b3eea1cd34a4aff7ce828a5f71daac042d38e0	Luxor Mining
15537418	0xf2e3718a3ea5e0af0fbf6de3cc479fd00d54b5dc	Unknown Address2
15537419	0xf2e3718a3ea5e0af0fbf6de3cc479fd00d54b5dc	Unknown Address2
15537420	0x09ab1303d3ccaf5f018cd511146b07a240c70294	Minerall Pool

Table 3.1.: List of Miners and Their Associated Addresses in the PoW Merge Chain Fork This table includes information on various PoW Merge Chain Fork miners. The table highlights the participation of different mining pools and individual miners, showing their contribution to block production.

After thoroughly examining the timestamps linked to the exported blocks, as shown in Table A.1, we noticed a difference between when a block was created and when it

arrived at our nodes. The block headers' embedded timestamps indicate that all blocks were produced on the same day as the Merge, within a window of approximately seven hours. However, block '15537420' was not propagated to our node until 25 days after its creation, when it was finally gossiped through the peer-to-peer (p2p) network. The observed increase in average block creation time seems to be directly correlated with the previously discussed reduction in Ethereum's hash rate, while the substantial delay in block propagation can likely be attributed to the decreased number of nodes actively participating in the fork. This relationship could be proven by the results of a Pearson correlation test with X being the hash rate values and Y being the block creation times. But as already discussed, obtaining the hash rate values is a time and resource consuming task. Therefore, a statistical evaluation of this relationship is unfeasible.

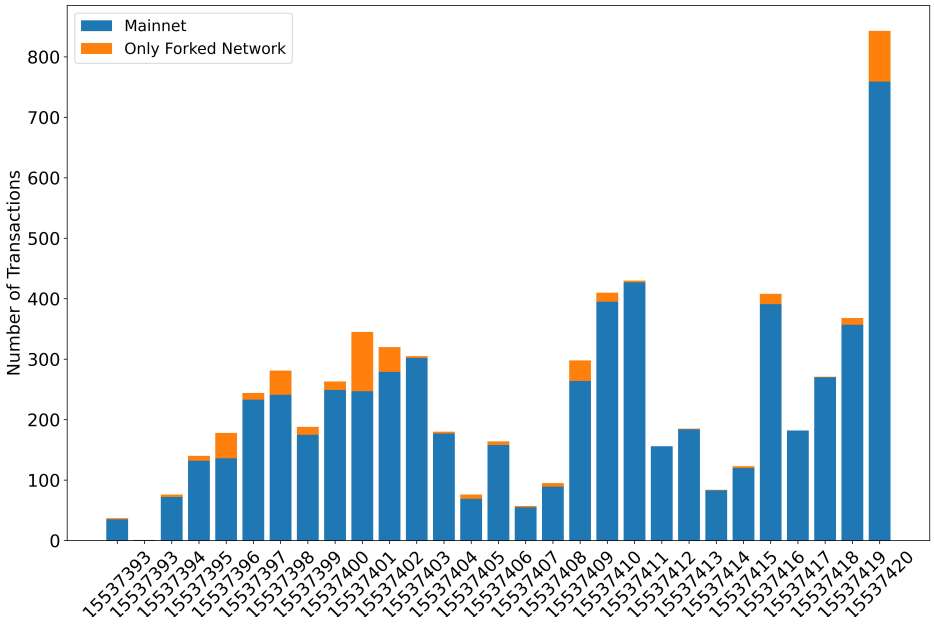


Figure 3.6.: Block Transaction Distribution of Transactions not included in the Ethereum 2.0 Chain per Block. The x-axis represents the block height.

Upon analyzing the transaction distribution shown in Figure 3.6, a notable observation from the extracted blocks is the tendency for many blocks to exceed the average transaction count per block compared to the main blockchain. We also examined the dataset in Table 3.2 and performed a Pearson correlation test, illustrated in Equation 3.7, to investigate the relationship between the transaction volume X and the block creation time[10] Y. The Pearson correlation test resulted in a coefficient of *0.54*, a moderate positive correlation between the block interval and the transaction volume.

[10]The smallest unit of time for block timestamps in Ethereum is seconds

$$c = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$

Figure 3.7.: The formula for Pearson's correlation coefficient (c), which measures the strength and direction of the linear relationship between two variables

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Figure 3.8.: The formula for Welch's T-test. Welch's T-test is used to compare the means of two groups, accounting for unequal variances and different sample sizes.

To examine the transaction volume of the PoW Merge Fork more closely, we compared it with random samples of blocks from the Ethereum 2.0 chain. The random selection process aimed to ensure a representative comparison between the two datasets. We conducted a Welch's T-test, illustrated in Equation 3.8, to determine if there was a statistically meaningful difference between the mean transaction volumes of the two samples. We opted for Welch's T-test because it accommodates the unequal sample sizes and variances, allowing us to use a larger sample from the Ethereum 2.0 chain to enhance the accuracy of our results.

The sample from the Ethereum 2.0 chain from 16367146 to 16477146 as *X1*, a total of 110000 blocks, was compared to the block interval from 15537393 to 15537420 as *X2* from the continued PoW Merge fork. Based on the results of Welch's t-test, which produced a t-statistic of *3.03* and a p-value of *0.005*, we can conclude that there is a statistically significant difference in the average number of transactions per block between the Ethereum chain and the continued PoW Merge fork. With a p-value well below the conventional threshold of 0.05, we can reject the null hypothesis of equal means.

Given that the comparison was conducted on a subset of the total blocks available in the blockchain, it's important to acknowledge the possibility that the sample might not fully represent the entire population of blocks. With only 110,000 blocks out of a total of 19,704,561 at the time of writing, the observed difference in the average number of transactions per block between the main chain and the continued PoW Merge chain may only partially be representative. While the statistical analysis indicates a significant difference between the two blockchain systems within the sampled data, it's essential to recognize the need for further exploration and validation.

If we examine the block gas limits for the PoW Merge chain, we observe that nearly all blocks, except for eight, use over 99% of their gas limit. In contrast, during the same block intervals on the Ethereum 2.0 chain, 18 blocks do not approach their gas limits.

When analyzing the blocks that have the most transactions not present in the main chain visualized in Figure 3.6, like block number 15537401, we see a gas usage of 99.9% in the PoW Merge chain and only 29.1% in the main chain. The same phenomenon can be seen with block 15537420, with 99.9% in the PoW Merge chain and 38% in the main chain. This analysis highlights the need for a statistical evaluation of hash rate distribution and the probability of block mining by various miners, as outlined in Table 3.1, within the PoW Merge Chain fork. Such an evaluation would involve comparing this distribution to an interval of blocks from the main chain prior to the Merge. However, since detailed information on the hash power of individual mining pools or miners is not publicly accessible, researchers like Ozisik et al. [33] have proposed algorithms to estimate a miner's hash rate based solely on block hash values. Due to the complexity and time demands of these calculations, conducting a comprehensive analysis in this area falls outside the scope of this thesis and is suggested for future research.

3.4. Transaction Analysis

In this section, we will analyze and discuss transaction-specific metrics such as gas prices, the amount of Ether transferred, and the identities of the most active senders and receivers. Ether is the main unit of currency in Ethereum, while Wei is its smallest subunit, with one Ether equal to 10^{18} Wei. [9] As previously detailed, the blocks in the continued PoW fork contain a noticeably higher number of transactions compared to those in the Ethereum 2.0 network. Our data review reveals that transactions unique to the PoW Merge Fork blockchain had a 62% higher mean gas price, quantified in Wei, and consumed 26% more gas than those also found on the main chain. This trend suggests that transactions on the PoW Merge Fork chain might be of increased computational complexity. Additionally, the average gas price for these fork-specific transactions was 596,527,151,027.67 Wei, which is considerably higher than the 357,083,417,699.84 Wei on the primary Ethereum blockchain, as described in Table 3.3. To underline the differences regarding gas, we again performed a Welsch T-test with the same block interval X from 16367146 to 16477146 in the main blockchain and Y, the blocks from the continued PoW merge chain. The test resulted in a p-value well below the conventional threshold of 0.05, statistically showing the difference in the average gas used per block of those two distributions. Regarding financial implications, the transactions in the forked blocks transferred a total of 127,488.80 Ether across 28 blocks, equating to an estimated value of $209 million. (15th September, 1 Ether = $1,639.67) 4,461.39 Ether more compared to the main network, as described in Table 3.4.

Upon analyzing the data, it became evident that a small number of addresses were responsible for the majority of transaction activity, as several addresses consistently appeared as either senders or receivers across multiple transactions. It was crucial to investigate these accounts to gain a deeper understanding of this behavior. However, due to the inherent pseudo-anonymity of blockchain technology, directly associating identities with specific blockchain addresses remains a challenge. De-anonymizing addresses can be achieved by clustering and heuristic analysis and has been a research subject for many

Block Number	Time Difference between Blocks (seconds)	Number of Transactions
15537393		37
15537393	0	1
15537394	9	76
15537395	2	140
15537396	14	178
15537397	27	244
15537398	12	281
15537399	83	188
15537400	115	263
15537401	1	345
15537402	132	320
15537403	594	305
15537404	58	180
15537405	123	76
15537406	298	164
15537407	141	57
15537408	177	95
15537409	363	298
15537410	617	410
15537411	2332	430
15537412	920	156
15537413	285	185
15537414	2946	84
15537415	106	123
15537416	2303	408
15537417	4068	182
15537418	1238	271
15537419	566	368
15537420	4146	843

Table 3.2.: Time Intervals Between Blocks and Corresponding Transaction Counts of the PoW Merge Fork
This table displays the time differences between successive blocks and the number of transactions recorded in each block. The varying time intervals and transaction volumes provide insights into the transaction throughput and block production dynamics during the deep fork event.

years. [34] [35] Because of its simplicity and accessibility, we decided to use the labeling feature of etherscan.io[11] for our exploratory analysis, which allowed us to identify a subset of the most active accounts. As presented in Table 3.5, FTX appears as the most active account on the PoW Merge Fork blockchain. FTX Trading Ltd., commonly known as FTX, was formerly a cryptocurrency exchange and hedge fund. However, by November 2022, it had declared bankruptcy. Remarkably, four addresses labeled by etherscan.io as FTX alone accounted for 1,594 transactions, constituting 24% of the total transactions in

[11] https://etherscan.io/labelcloud

Total Mean Gas	244437.93
Total Mean Gas Price	374150229144.19
Fork Only Mean Gas	302415.52
Fork Only Mean Gas Price	596527151027.66
Mainnet Mean Gas	239988.311
Mainnet Mean Gas Price	357083417699.84

Table 3.3.: Comparison of Mean Gas Consumption and Price across the PoW Merge Chain and the Ethereum 2.0 blockchain

Total Value	127488.80
Mainnet Value	123027.40
Extra Fork Value	4461.39

Table 3.4.: Total Value Comparison transferred in Ether between the PoW Merge Chain and the Ethereum 2.0 blockchain

the dataset. The identities of most other addresses remained unidentified.

Considering all the transactions not included in the Ethereum 2.0 network, we analyzed the top receiving addresses of all fork-only transactions. During our analysis, we found that 25% of the transactions only included in the forked blocks were directed to the address *0x57C1e0C2ADf6EECDb135BcF9ec5F23b319be2c94*[12], which belongs to the smart contract component of a MEV (Maximal Extractable Value) bot. MEV refers to the profit that can be extracted by reordering, including, or excluding transactions within a block during block validation. The address in question is not an externally owned account (EOA) but a smart contract specifically programmed to execute strategies for capturing MEV. [36] Table 3.6 illustrates that 9 of the 30 most active receiver addresses are smart contract components of MEV bots. MEV Bots are software programs designed to automate the exploitation of vulnerabilities in transactions to extract profits, with the bot's work happening both off-chain and on-chain via smart contracts. These bots can perform actions such as front-running, liquidity provisioning, and flash loan arbitrage to generate profit for the user. Their use can negatively impact the fairness and security of DeFi protocols and has been a topic of concern in the crypto community. [36] [14] [37] It is impossible to generalize why certain transactions were not executed on the Ethereum 2.0 mainnet. Each transaction would need thorough investigation, as multiple scenarios can lead to the non-inclusion of transactions. For instance, consider transaction *0xd1f13ad...f2a54bd7*[13]. This transaction was included in block 15537419 on the continued PoW chain. On the Ethereum 2.0 mainnet, it was only included in block 15630883, essentially two weeks later, or 93,464 blocks later, suggesting that we might observe more such inclusions in the future.

[12]https://etherscan.io/address/0x57C1e0C2ADf6EECDb135BcF9ec5F23b319be2c94
[13]https://etherscan.io/tx/0xd1f13adf11d130168715736372c010c115bae4ec3aed8f06b999a959f2a54bd7

Address	Number of Transactions	Etherscan.io Label
0x2FAF487A4414Fe77e2327F0bf4AE2a264a776AD2	780	FTX
0xC098B2a3Aa256D2140208C3de6543aAEf5cd3A94	673	FTX
0xBB50cE87be3443Ed137df1dfdbF2FB0ca8C0a9e0	320	Unknown1
0x7abE0cE388281d2aCF297Cb089caef3819b13448	84	FTX
0x25eAff5B179f209Cf186B1cdCbFa463A69Df4C45	57	FTX
0x19f494583c7C933be7B0eE58104DDaFac1E8aDfA	44	Unknown2
0xfaC3f677aaf29FB5c702bC51F6C28cBc48816B60	38	Unknown3
0x8EB8a3b98659Cce290402893d0123abb75E3ab28	35	Avalanche Bridge
0xe1e1AA9C7e93Bc0FB6259eb509be062dcEc96d43	23	Unknown4
0x6a09008c0041dBd389F64324b093c3936a4921fc	18	Unknown5
0xE8527f01BdF894AaAB08eE10D813e009a85eDFaB	18	Unknown6
0xec00eb6C067A004b277B8B336a560a90c97f6697	17	Unknown7
0x6ef49a9d00f024D73CD100dFab16047D2361429c	17	Unknown8
0x5f2Cc40000dFA88f9D34466a06e237A9492dBbB9	16	Unknown9
0xF33835D85c8ce17f41abE1f510fACB044806751a	16	Unknown10
0x87cbc48075d7aa1760Ac71C41e8Bc289b6A31F56	16	Unknown Exchange
0xeac2eBc38A14De6A78925c33c2739846A02c40f0	16	Unknown11
0x1aD42FB475192C8C0a2Fc7D0DF6faC4F71142c58	16	Unknown12
0x0EBd59317D3c6FF58104B5bEBf2F9D2F732b2694	16	Unknown13
0x3C2532CFba0310d5Bd0d0745F90000c3E48C2c27	16	Unknown14

Table 3.5.: Top Senders of Transactions on the PoW Merge Chain
This table presents the most active addresses responsible for sending transactions on the PoW Merge Chain. It includes the number of transactions initiated by each address and their corresponding labels from Etherscan.io. Notable entities, such as FTX and the Avalanche Bridge, are highlighted alongside a series of unidentified addresses. This distribution provides insights into the key players during the fork event.

3.5. State Divergence Analysis between Ethereum 2.0 Mainnet and PoW Merge Fork

The primary purpose of this fork investigation is to analyze the potential state divergence of a blockchain and its fork during a deep fork event. Specifically, we aim to gain insights into the relationship between fork depth and state divergence. Account balances in a cryptocurrency are a crucial component of the distributed ledger and serve as key indicators for identifying state divergence. With the retrieved data, particularly the list of the most active accounts, we can compare account balances of specific addresses to investigate discrepancies or anomalies in balances between the original blockchain and its fork, which are closely tied to transaction validation and execution.

3.5.1. Transaction Based Divergence Analysis

Before diving into a direct block-based comparison of the blockchain state, we first explored the differences on a transactional level. Given the extensive number of accounts initiating transactions in the PoW Merge Fork blockchain, it was essential to narrow our

Address	Number of Transactions	Etherscan.io Label
0x7Bc25283a29A3888CAb4555Ea86fF1a8C18Cc90a	779	SetInMergeERC721 Contract
0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48	318	Centre
0xdAC17F958D2ee523a2206206994597C13D831ec7	285	Bitfinex (Blocked)
0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45	184	Uniswap
0xaaAEBE6Fe48E54f431b0C390CfaF0b017d09D42d	159	Celsius Network
0x57C1e0C2ADf6EECDb135BcF9ec5F23b319be2c94	156	MEV
0xfc1726c4ead2393fA14407Ff54B336E2c8BB4aCA	144	Unknown2 (Genesis Contract)
0x8d8c3637293A2D84202438B2567dA5421238DBa7	122	Unknown3 (Token Contract)
0xe0c9BfC391cEE9e840825239aE9910C77064b356	118	Unknown4 (Subjects Contract)
0xcE0f25934dEAaddd174427F1978bCd487a85E9fA	101	Unknown5 (Truplay Contract)
0x00000000AE347930bD1E7B0F35588b92280f9e75	99	Unknown6
0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D	99	Uniswap
0xbaDc0dEfAfCF6d4239BDF0b66da4D7Bd36fCF05A	88	MEV
0xE42caD6fC883877A76A26A16ed92444ab177E306	80	Token (TheMergeNFT)
0x0e824d5f934Ad01A603101d6A41D723C1702822B	78	Unknown7 (Conclusion Contract)
0x5A98FcBEA516Cf06857215779Fd812CA3beF1B32	54	Lido Token
0x881D40237659C251811CEC9c364ef91dC08D300C	51	MetaMask
0x00000000006c3852cbEf3e08E8dF289169EdE581	50	Unknown8 (Seaport Contract)
0x346B5B8844D2548f6aD55089d8939CfFbE3aCBAF	45	Unknown9 (Kungfungus Contract)
0xDBd324B73f6f85bf9013B75C442021303b635Ff9	44	Sorare (Fantasy Football)

Table 3.6.: Top Recipients of Transactions on the Forked Chain
This table lists the most active addresses that received transactions on the forked blockchain, detailing the transaction count and their associated labels as identified on Etherscan.io. The list includes a mix of smart contracts associated with MEV bots, DeFi platforms, and other smart contracts, highlighting their activity during the fork event.

focus. We concentrated on the top 30 accounts primarily due to their increased activity since each account was responsible for at least ten transactions. To compare the balance of any account across both chains, we first identified the relevant data entries, especially for the Ethereum 2.0 mainnet. This led us to define the starting and ending blocks for our analysis. The starting block was set at the height where the fork occurred (i = 15537393). The ending block was determined by the block in which an account's last observed fork transaction was processed on the Ethereum 2.0 mainnet. For instance, consider the most active account in our study,
0x2FAF487A4414Fe77e2327F0bf4AE2a264a776AD2. Its starting block is i = *15537393*, while its ending block is j = *15537909*, 516 blocks. For this account, we extracted the balance for all blocks between i and j on the Ethereum 2.0 mainnet. In contrast, for the continued PoW chain, we set the starting block to *i'=15537393* and the ending block to *j'=15537420*, marking the last recorded block of the fork.

We could identify a pattern for some of the most active accounts, a notable balance discrepancy. For example, account
0xc098b2a3aa256d2140208c3de6543aaef5cd3a94 exhibits a deficit of 75,976 Ether in the PoW Merge Fork blockchain compared to the main network, as illustrated in Figure 3.9. The most notable differences in account balances were nearly always addresses associated with Coin Exchanges. Coin exchanges refer to platforms or services allowing individuals

to buy, sell, or trade cryptocurrencies with other users. Consequently, the top accounts with the highest difference were both FTX addresses, as labeled by etherscan.io.

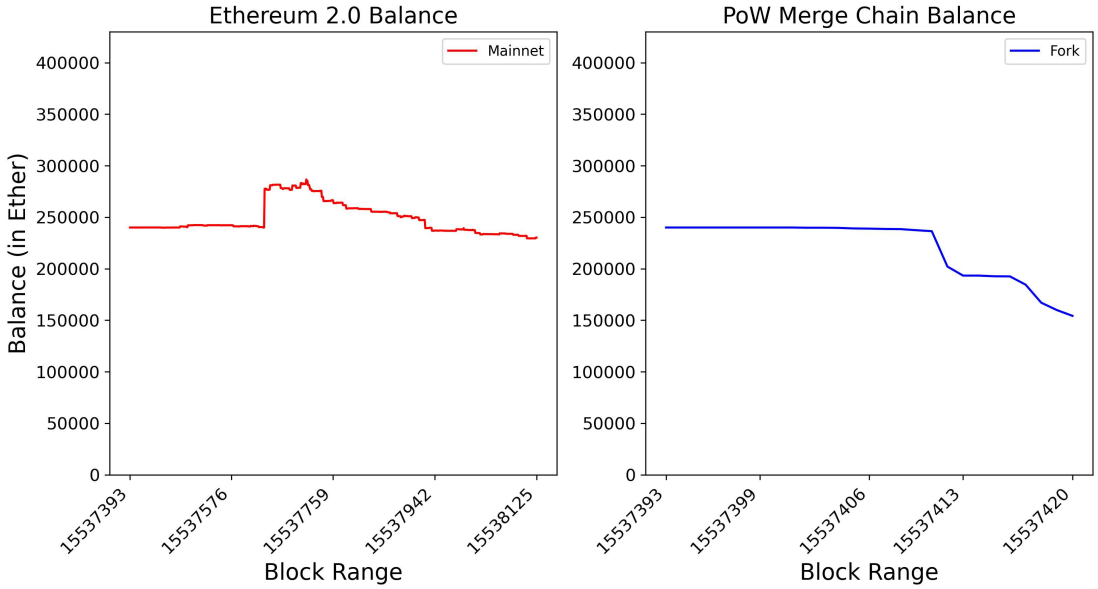


Figure 3.9.: Account Balance comparison of Account 0xc098b2a3aa... in the Ethereum 2.0 Chain and in the PoW Merge Chain (Transaction based). The block range for the Ethereum 2.0 chain spans from the block height where the fork occurred (i = 15537393) to the block where the last observed fork transaction from 0xc098b2a3aa... was processed. The block range for the PoW Merge Chain spans over the 28 recorded blocks during the Merge. (Transaction based)

While addresses associated with Coin Exchanges had substantial account balance discrepancies, they were not the exclusive entities with such divergences. The address *0xeB6c4bE4b92a52e969F4bF405025D997703D5383*, not labeled as a coin exchange by etherscan.io, showed a marked difference of approximately 810 Ether, as illustrated in Figure 3.10. Interestingly, this account maintained a consistent balance of around 2178 Ether in the PoW Merge Fork chain. In contrast, its balance in the mainnet increased to 3000 Ether and remained relatively stable at that level.
It is debatable how meaningful this measurement is. Suppose one compares the state of an account on two distinct blockchains with a tremendous difference in block heights. In that case, the account balance will likely diverge, especially for accounts engaged in active trading, such as accounts associated with exchange hot wallets. Hot wallets are always connected to the internet and are actively used for transactions.

3.5.2. Block Based Divergence Analysis

In our subsequent analysis, we set a fixed block interval to explore potential account balance disparities. Similar to the transaction-based approach, the starting block (i)

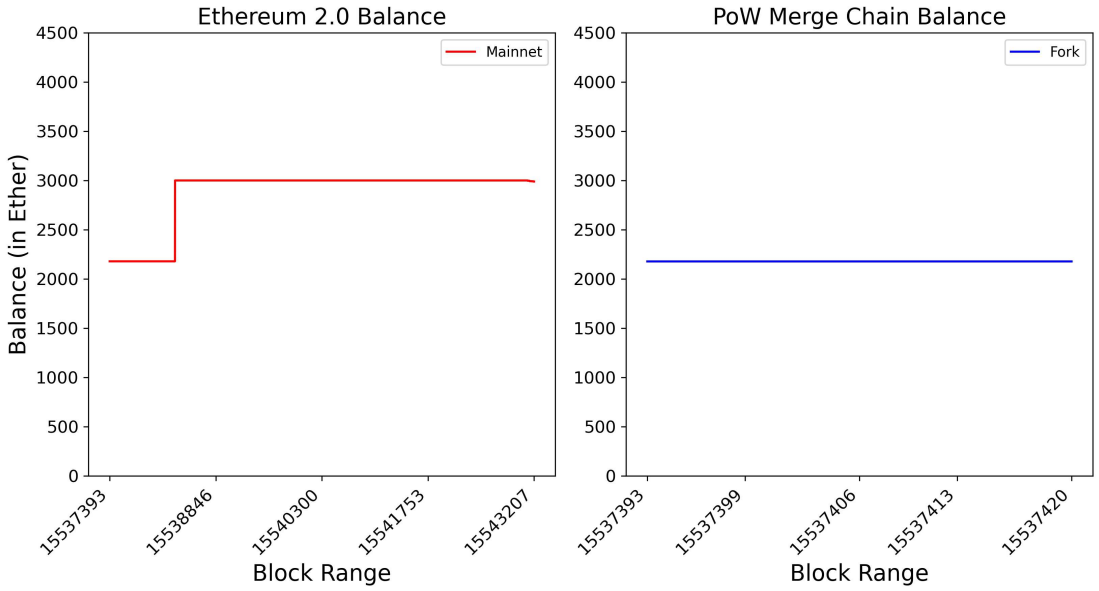


Figure 3.10.: Account Balance comparison of Account 0xeB6c4bE4... in the Ethereum 2.0 Chain and in the PoW Merge Chain (Transaction based). The block range for the Ethereum 2.0 chain spans from the block height where the fork occurred (i = 15537393) to the block where the last observed fork transaction from 0xeB6c4bE4... was processed. The block range for the PoW Merge Chain spans over the 28 recorded blocks during the Merge. (Transaction based)

is positioned at 15537393, and j represents the last recorded block in the PoW Merge Fork, 15537420. This range applies to both the main and PoW Merge Fork chain. With the outcomes of this analysis, we try to gain insights into the relationship between fork depth and state divergence. We used so-called tracer functions in Ethereum clients to gather all relevant account addresses, including accounts used in internal transactions. Blockchain transactions also include internal transactions, particularly during interactions with smart contracts. [9] Although these internal transactions involve both senders and receivers like regular transactions, they are not independently recorded. Instead, only the parent transaction appears in the block, as internal transactions are embedded within it. [9] In the context of Ethereum clients, tracer functions refer to functions that allow tracing transactions and account activities within the blockchain. Tracers replay the entire sequence of EVM operations, enabling the investigation of the stack, memory, etc. [38] With 85751 Ether difference, address *0xc098b2a3aa256d2140208c3de6543aaef5cd3a94* was again the account with the highest difference between the main and PoW Merge Fork chain. As already described in the transaction-based approach, account *0xc098b2a3aa...* is labeled by etherscan.io as an address of the former coin exchange FTX.

In the transaction-based experiment, this specific address had a difference of 75976 Ether. Notably, three of the top five addresses with the highest account balance difference were indirectly associated with FTX. Even accounts not belonging to FTX by etherscan.io had some association with FTX. For example, address *0x5c311563ea0bb8f9ca2471012eb2f495bd687d18*[14] to which some of the FTX addresses' final assets were transferred.
However, compared to the previously performed analysis with account *0xc098b2a3aa...*, we have a constant account balance line in the main net and a drastically falling curve in the PoW Merge Fork network, as illustrated in Figure 3.12. The phenomenon of a constant balance line in the mainnet was present for many accounts. We can also notice it with account *0xcde35b62c27d70b279cf7d0aa1212ffa9e938cef*, an address not associated with FTX based on the labeling feature of etherscan.io, as illustrated in Figure 3.11. Overall, we have an absolute difference between the mainnet and the PoW Merge Fork network of 225200 Ether – 358367516 USD.

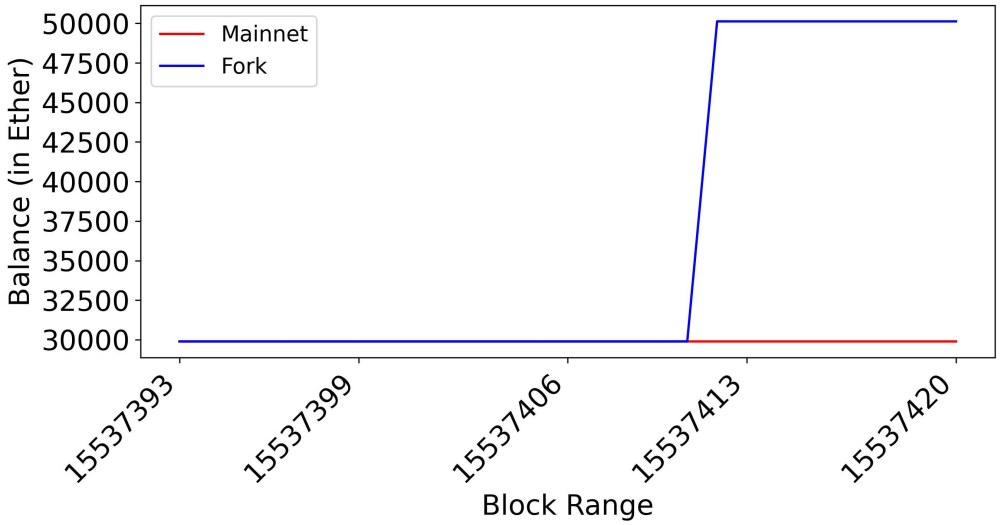


Figure 3.11.: Account Balance comparison of Account 0xcde35b62... in the Ethereum 2.0 Chain and in the PoW Merge Chain (Transaction based). The block range spans from the block height where the fork occurred to the last recorded block during the Merge. (Block based)

3.6. Discussion of Merge Analysis

With Ethereums' transition from Proof-of-Work to Proof-of-Stake, our analysis revealed that, in practice, the blockchain state can already show marked divergences from a

[14]https://etherscan.io/address/0x5c311563ea0bb8f9ca2471012eb2f495bd687d18

fork depth of only 28 blocks. By extracting the account balance of numerous accounts and analyzing general metrics like the number of transactions, we could see what an unexpected and uncontrolled fork could result in. Discrepancies in account balances come from differences in transaction order, execution, and validation. A massive shift in the account balance of several addresses associated with a coin exchange, like the analyzed accounts of FTX, diverges the overall blockchain state. Another exciting point we could explore was the behavior of the enormous mining community Ethereum used to have. When looking at similar EVM-based cryptocurrencies, we observed a shift in the hashing power from Ethereum to those other blockchains. Coins like Ethereum Classic or ERGO, in particular, appeared to experience a massive boost in their hashing power: ETC went up to 200 TH/s, an all-time high and growth of 300%, and ERGO had an 8x spike on the day of the merge. It was also interesting to notice how the PoW Merge fork chain reacted to the sudden drop in network participants. The difficulty parameter, responsible for adjusting the mean block time, could not compensate for the considerable drop in active nodes. Responsible for that might be the sudden decline of active miners in the network. Furthermore, this research offers some points that will be highly important in the future, like the role and behavior of the now former coin exchange FTX, which should be closely examined since even core developers of the Ethereum Foundation advised against carrying out important transactions or sending a large amount of assets during the merge. [19] We can only speculate about why FTX nevertheless sent a large number of transactions, some of which were highly valued. The role of the numerous MEV bots whose transactions were not available on the mainnet should also be closely observed. MEV and especially frontrunning bots are fascinating topics, and their behavior in forks should also be investigated.

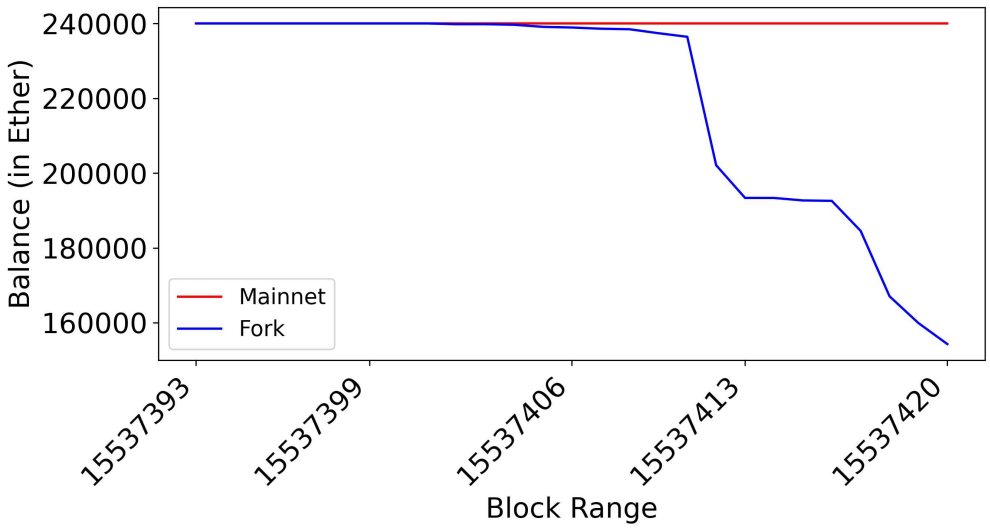


Figure 3.12.: Account Balance comparison of Account 0xc098b2a3aa... in the Ethereum 2.0 Chain and in the PoW Merge Chain (Transaction based). The block range spans from the block height where the fork occurred to the last recorded block during the Merge. (Block based)

4. State of the Art

Researchers have yet to discuss the consequences of blockchain forks regarding the non-finality of transactions. [14] Thus, we propose a discrete stochastic blockchain fork simulator to analyze how the blockchain state behaves with an ongoing fork. This chapter presents an analysis of the use of fork simulators in Ethereum, focusing on their potential to improve the network's security and functionality. A fork simulator allows researchers to measure the impact of forks, including the state-dependent nature of transactions, especially those involving smart contracts. Overall, we discuss the challenges related to forks, non-finality, and consensus in Ethereum and how fork simulators can be utilized to evaluate the impact of these challenges on the network's behavior.

4.1. Simulations

Before we analyze the specifics of simulation environments within blockchain technologies, it is essential to understand the basics of simulators. During the review process, two articles or books emerged as the most frequently referenced sources, serving as foundational resources for simulation and modeling within the field. Their common ground is discrete event simulation, a simulation approach often used for blockchain simulations. Maitz's [39] thesis inspired the literature and ideas for this chapter.

4.1.1. Discrete Event Simulation

Maria [40] elaborates and provides a simulation modeling and analysis tutorial by defining and answering critical questions to develop a working simulation model. The author discusses the difference between models and simulations and elaborates on their connection. The steps described by Maria [40] to define a simulation model serve as a fundament for any model. They are illustrated as shown in Table 4.1.1. The logical sequence of provided steps is furthermore subdivided into three distinct categories: "How to develop a simulation model?", "How to design a simulation experiment?" and "How to perform simulation analysis?". One should keep in mind, though, that this provided step-by-step model acts solely as a theoretical guideline, meaning that not all steps are possible nor necessarily required. Additional steps or iterations may be required. [40] To underline the presented theory, the author concludes with an intuitive case study in which each defined step is used and elaborated.
Banks et al.[41] provide a more broad and descriptive approach regarding simulations. The authors deliver a high-level definition of the essential concepts to understand how simulators are specified. They introduce the concept of systems and system environments, which components are necessary to form a system and deep-dive into discrete and

continuous systems. Similar to the research of Maria, Banks et al. [41] also elaborate on the difference between models and simulations and their connection. Furthermore, they also present a comparable logical step order to define a simulation study, which shows some similarities. Different from the study by Maria [40], the steps are divided into four distinct phases. The first phase is used to define the problem space precisely, the second phase deals with the model definition and data collection, phase three is related to running the model and experimenting with the simulation model, and the focus of the last phase, phase four, lies within implementation and documentation. [41]

Step	**Maria**	**Banks**
1	Identify the Problem	Problem foundation
2	Formulate the Problem	Settings of objectives and overall project plan
3	Collect and process real system data	Model conceptualization
4	Formulate and develop a model	Data collection
5	Validate the model	Model translation
6	Document model for future use	Verified?
7	Select appropriate experimental design	Validated?
8	Establish experimental conditions for runs	Experimental design
9	Perform simulation runs	Production runs and analysis
10	Interpret and present results	More runs?
11	Recommend further course of actions	Documentation and reporting

Table 4.1.: Simulation steps by Maria [40] and Banks [41]

4.1.2. Blockchain Simulators

Blockchain simulators are essential tools in distributed ledger technologies, they allow developers to test their configurations, contracts, and transactions in close to real-world scenarios to provide security and warranted functionality for their users. Most blockchain simulators are developed to serve only a single purpose, ranging from simulating miner behavior and fork behavior to attack simulators. [42] For this literature review, we looked closely at three blockchain simulators: EthAttackSim by Maitz [39], BlockSim by Faria and Correia [43] and BlockSim by Alharby and van Moorsel [44]. It is important to note that the two BlockSim projects are entirely separate initiatives that, unfortunately, share the same name, which could lead to confusion.

EthAttackSim

EthAttackSim is an Ethereum attack simulator that focuses on the simulation and analysis of different attack scenarios, precisely the verifiers' dilemma and the selfish mining attack, to understand their implications for the networks' stability. EthAttackSim follows the principle of a Discrete Event Simulator Engine, with its architecture consisting of two major parts: Firstly, the simulation world consisting, for example, of an event

queue, configuration abilities, timekeeping or randomness, and secondly, the node module, representing a single participation node providing network capabilities, a consensus engine, and the ledger storage. One of the critical objectives of EthAttackSim is extensive reconfigurability: It is possible to set essential properties like the overall hash power, CPU power of specific nodes, or even the average block time via config YAML files. Those configurations are adjusted according to the attack scenario planned to be simulated. Data collection for those properties is crucial. Maitz gathered these parameters from various sources like research papers, blockchain monitoring tools, or data extracted manually from execution clients. [39]

BlockSim (Faria and Correia)

Faria and Correia propose a blockchain simulator that provides a framework and blockchain-common base simulation models like blocks, miners, transactions, or nodes. BlockSim is a discrete-stochastic-event simulator that assists in designing, implementing, and evaluating existing or newly created blockchain technologies. With BlockSim, the authors experimented with Ethereums' configurations to analyze how the block propagation times behave with certain modifications. The project is publicly available on the code-sharing platform GitHub and is solely developed in Python. The complex architecture of BlockSim consists of six distinct components: A **Discrete-Event-Simulation-Engine** based on the discrete simulation framework SimPy, a **Simulation World** responsible for input and configuration parameters, a **Transaction and Node Factory** used for creating a batch of random transactions, a **Programmatic Interface and Simulation Example** allowing users to interact with the system and define their models, a **Monitor and Reports** component to capture specific blockchain-related metrics and lastly a **Blockchain Modelling Framework**. [43]

BlockSim (Alharby and van Moorsel)

BlockSim, developed and proposed by Alharby and van Morsel, is similar to the previously described implementation by Faria and Correia, a discrete event simulator for blockchain technologies. In direct comparison, BlockSim by Alharby and van Morsel is more lightweight but also implemented in Python. Of course, all required simulation models required for a valid blockchain are provided: Blocks, Nodes, and Transactions. As already mentioned, this project's architecture is way simplified compared to the proposal by Faria and Correia. It consists of three distinct layers: The **incentives layer**, the **connector layer**, and lastly, the **system layer**. The incentives layer is responsible for the decision-making process, whereas the connector layer includes the consensus layer, which is necessary to guarantee a valid chain. Furthermore, the system layer defines the basic configurations like the operating system or the network. BlockSim aims to be intuitive without unnecessary details and easily manipulated and adaptable. [44]

In conclusion, simulators have a broad range of applications, all applicable to different use

cases. Our simulation model should be capable of simulating blockchain forks incorporating real-life data. It should not create a separate simulation environment but instead use the existing blockchains as its basis. Blockchain forks should be simulated at any block height of Ethereum or its test networks. We do not want to rebuild Ethereums' protocol, we don't want to simulate nodes, and we do not want to simulate specific attacks. We simulate the view of the blockchain at different nodes. Precisely, in our simulation model, the goal is not to mimic the actions of individual nodes but rather the resulting state divergence of a blockchain fork as a whole, ideally using only minimal modifications to existing code to preserve realism.

4.2. Non-Finality and Consensus

The topic of reaching agreement, i.e. consensus, among distributed processes is a fundamental problem in distributed computing. The consensus problem has a rich history but has gained renewed attraction in the last decade due to the rise of cryptocurrencies like Bitcoin and Ethereum. [7] [45] Garay and Kiayias [7] define consensus as reaching an agreement distributedly in the presence of faults. Many different forms of consensus mechanisms are used in blockchains, all created for various systems with distinct properties. To better understand consensus, we first need to determine state-of-the-art research and practices, also regarding transaction finality, a property many consensus techniques do not offer or only offer partially, compared to traditional consensus algorithms. [7]

4.2.1. Consensus Algorithms

Bakmakan et al. [45] define consensus algorithms as a decision-making process where a group expresses their opinions to construct the decision that best estimates a process or system. Garay and Kiayias describe consensus as reaching an agreement distributedly in the presence of faults. [11] Bakmakan et al. [45] provide an analysis and comparison of different state-of-the-art consensus algorithms used in public, federated, and private blockchains with the aim of understanding existing challenges regarding consensus and non-finality in distributed ledger technologies. We are going to provide definitions for those different types of blockchains shortly. Not only are they analyzing the most used consensus algorithms like Bitcoins' Proof-of-Work or Ethereums' Proof of Stake, but also far more unfamiliar ones like Proof-of-Burn, Proof-of-Importance, or Proof-of-Capacity. The authors' primary focus is on providing a framework that evaluates the overall capabilities of those algorithms to review their possibilities and restraints. The framework identifies and weighs standard features of all consensus algorithms, like their goal or use-case, speed, and whether they are centralized or decentralized or commonly used in permission- or permissionless blockchains. Furthermore, general aspects regarding network participants, like miners' roles, energy efficiency, hardware dependency, and scalability, are taken into consideration. At last, the algorithms' reaction to specific blockchain attacks is examined and evaluated. [45]
A comparable work, published by Mingxiao et al. [12], also reviews and compares

commonly used state-of-the-art blockchain consensus algorithms, followed by a technical guideline for selecting the correct consensus algorithm for specific application use cases. The authors first define and explain the differences between the most commonly used consensus algorithms in blockchain technologies: Proof-of-Work, Proof-of-Stake, Delegated Proof-of-Stake, Practical Byzantine Fault Tolerance, and Raft. In order to analyze the characteristics of each algorithm, the authors investigate their performance, scalability, and shortcomings. Like in many other distributed networks, the danger of malicious actors is always present. Therefore Mingxiao et al. examine how the distinct consensus algorithms respond to performed double spending attacks. Similar to blockchain attacks, limitations like waste of resources or slow transaction throughput may also drastically impact a network's overall performance. The authors' research is concluded with different application scenarios. They describe which algorithms best suit the different forms of blockchain technologies: Private Blockchains, Permissioned Blockchains, and Public Blockchains. [12] Private blockchains are blockchain networks that restrict access and participation to a specific group. Permissioned blockchains are blockchain networks where access to network participation requires authorization. Public blockchains are decentralized networks where participation is open to anyone, and no central authority controls access or governance. [46] [47]
Bitcoin and many other cryptocurrencies based on the so-called Nakamoto consensus algorithm have the property that they do not offer consensus finality, which brings us to the next topic of our systematic literature review.

4.2.2. Non-Finality

Anceaume et al. [48] define blockchain finality as deterministic or probabilistic and immediate or eventual. Immediate finality guarantees that any written block cannot be revoked and is immediately finalized. Usually, permissioned blockchains, like Hyperledger Fabric, work with deterministic immediate finality, and blockchains like Algorand follow the principle of probabilistic finality. [48] Eventual finality increases when more blocks are appended to the blockchain. Major cryptocurrencies like Bitcoin guarantee eventual finality with some probability: The probability that a block can be revoked sinks with the number of further blocks appended to this specific block. [48]
Sankagiri et al. [49] illustrate an often ignored perspective, especially in the blockchain space, the CAP Theorem or Brewer's theorem. The CAP Theorem states that only two of the following three properties are guaranteed in distributed systems simultaneously: Consistency, Availability, and Partition Tolerance. [50] Cryptocurrencies, therefore, face the same restrictions. The CAP theorem claims that blockchain protocols can achieve adaptivity, also known as dynamic availability or finality, in the context of the CAP theorem known as consistency. Regarding CAP, the longest-chain protocol proposed by Nakamoto is an adaptive protocol that ensures availability and partition tolerance but cannot guarantee consistency. On the other hand, committee-based consensus algorithms favor strong finality, meaning consistency over availability. [49] [51]
Sankagiri et al. propose a new blockchain protocol, the checkpointed-longest-chain-protocol, a protocol used to analyze if the trade-off between adaptivity (dynamic availab-

ility) and finality (consistency) can be resolved at a user level instead of solely relying on a system/network solution. The blockchain still depends on honest nodes contributing their mining power to build upon the chain. However, it allows ordinary users to choose between two configuration rules: One rule guarantees finality, the other adaptivity. The block proposal mechanism is identical to Nakamotos Proof-of-Work algorithm, with the exception that certain blocks in predefined intervals are marked as checkpoints. Therefore any node accepts only a chain containing the latest checkpoint. A block is confirmed with the so-called k-deep rule. Any block can be confirmed as long as k blocks exist after it. [49]

4.3. Forks

In this thesis we focus on understanding forks thoroughly. We aim to study why forks happen, how they affect blockchain networks, how they are an artifact of consensus protocols, and how they relate to the non-finality problem of transactions. This research area has not received much attention, especially the potential state divergence between forks and possible implications. Since forks are the main concerns of this thesis, we formulated a specific research question to analyze the current research regarding forks:

RQ2: What is the state-of-the-art research on fork behaviour of blockchain technologies?

Therefore, we will go into a more in-depth literature analysis to understand the state of research and identify research gaps. To structure this process, we defined separate research questions.

- RQ 2.1: What are the different types of forks?
- RQ 2.2: Why and when do forks occur?
- RQ 2.3: What are the implications regarding forks?

4.3.1. Categorization and Definition of Different Types of Forks

When defining forks, we must differentiate between intended and accidental forks. [3] Since the various types of forks are loosely defined, we have established our own definitions based on the literature reviewed in the following sections, followed by defining them within the literature review:

Definition 4.3.1. ***Intended fork:*** *Intended forks are planned and deliberate changes in blockchain protocols, made to upgrade or modify the system, resulting in new* ***permanent*** *chains that reflect the updated rules.*

Definition 4.3.2. ***Accidental fork:*** *Accidental forks occur due to network delays or simultaneous block proposals, resulting in* ***temporary*** *divergent chains.*

It is essential to note that forks can also be distinguished between permanent and temporary forks:

Definition 4.3.3. ***Permanent fork:*** *Permanent forks are changes in blockchain protocols that result in a new, lasting chain.*

Definition 4.3.4. ***Temporary fork:*** *Temporary forks are short-lived divergences in the blockchain that resolve themselves as nodes reach consensus and re-align on a single chain.*

To help understand those loosely defined terms, we have visualized the different categories of forks in Figure 4.1.
Researchers generally talk about hard- and soft forks when talking about intended forks. Simplified are hard- and soft forks upgrades to the blockchain or simple to complex protocol changes. [5] [29] This could be, for example, the change in a blockchain's consensus algorithm, like Ethereums' switch from PoW to PoS. To be more precise, the so-called Merge is defined as a hard fork. A hard fork is a change to the blockchain protocol that is not backward-compatible, meaning newly created valid blocks might be considered invalid by non-upgraded nodes following the previous protocol rules. On the other hand, a soft fork represents a modification to the blockchain protocol that maintains backward compatibility. [3] [29] This implies that nodes operating on the updated software can interact with nodes using the older version. However, blocks generated by outdated nodes are invalid in the upgraded protocol. Hard- and soft forks generally have a specific condition embedded in the clients that trigger the update. [5] [52]
Accidental forks, on the other hand, are a phenomenon with many different names and various definitions. For example, Kiffer et al. define them as transient forks, which occur when two miners propose a new block simultaneously, both claiming to be the longest valid chain. [29] Chen et al. describe them as temporary forks: A fundamental phenomenon where miners discover the next PoW block simultaneously, triggering a chain split. [53] Decker et al. define blockchain forks as a disagreement over the current blockchain head. This disagreement also applies to the blockchain state, making the chain inconsistent and leading to security issues to the consensus itself. An attacker may attempt to rewrite the transaction history. [2]

4.3.2. Factors leading to Forks in Blockchains

There can be different reasons for forks to occur, but researchers primarily consider network delays, more precisely propagation delays, as the cause for temporary forks. Most research is focused on these temporary forks, reflecting the importance of understanding their impact on blockchain stability. [53] [2] [1] [54] Block propagation delay refers to the time it takes for a newly mined block to be transmitted and received by all nodes in a blockchain network. Da Silva et al. [54] describe what factors are typically responsible for forks in their work and go into greater detail regarding the propagation delay. They present an analytical model based on network delay and packet losses of TCP connections inside the blockchain network to calculate the probability of blockchain forks, proving that a more stable network without packet losses and faster transmissions leads to lower fork

probability. Chen et al. argue that the probability of a fork is also related to the hash rate of the miner who found the latest block. Generally, the faster the block generation speed, the higher the frequency of temporary forks. In fact, the frequency of temporary forks in Bitcoin is quite lower than that of Ethereum since Bitcoin has a higher block time. [53] In modern post-merge Ethereum, Ethereum 2.0, slots and attestations were introduced. Slots represent fixed time intervals during which validators propose and attest to blocks, lasting 12 seconds. Validators take turns proposing blocks during their assigned slots, with only one validator designated as the leader for each slot. Since slots can be missed, forks are still possible but less common with the help of attestations. Validators continue to attest to the validity of proposed blocks, even if they are not the designated leaders for a particular slot. These attestations serve as votes that contribute to the finality of blocks, helping to establish consensus on the main chain. [9] Neuder et al. [55] also describe malicious chain reorganizations as factors leading to forks and impacting the finality of transactions and blocks in Ethereum 2.0. Reorganizations, known as reorgs, happen when a fork-choice rule determines that a new branch replaces an existing one, effectively erasing blocks from the canonical chain, declaring the chain deemed by honest participants as the primary one a fork. Those reorgs also happen because of the already-mentioned network delays. [55] Decker et al. claim that large blocks propagate slowly in the network, leading to a higher propagation delay and a higher fork probability. [2]
Kwon et al. names selfish mining as an additional cause for temporary forks. Selfish mining is a strategy some miners employ in a blockchain network to gain an unfair advantage in the mining process. Instead of immediately broadcasting a successfully mined block to the network, a selfish miner intentionally withholds it, creating a temporary fork in the blockchain. [56] [57] For Ethereum PoS, the selfish mining problem can also be linked to malicious chain reorganization attacks, where attackers try to exploit those reorgs to attempt double-spending. [55]
Hard- and soft forks, as already described, happen because of software upgrades, such as a change in the network's consensus rules or a disagreement among network participants about the direction the blockchain should take. McCorry et al. present a list of significant hard forks up to 2017 that occurred in Bitcoin, Ethereum (ETH), and Ethereum Classic (ETC). Ethereum, for example, has experienced 19 hard forks at the time of writing. Compared to Ethereum, Bitcoin rather makes use of soft forks than hard forks. [5] In Ethereums' hard fork history, the two most influential ones are the fork that followed the DAO hack and the already mentioned switch from Proof of Work to Proof of Stake, which serves as the foundation and motivation for this thesis and will be discussed in greater detail in the later chapters. Although the Merge was mentioned multiple times in papers [19] [17], the outcomes have yet to be covered. The DAO (Decentralized Autonomous Organization) hack was a significant event in the history of blockchain technologies. [24] [29] It has been studied extensively by many academics and researchers since it highlighted the importance of secure coding practices, the need for rigorous testing, the auditing of smart contracts, and the associated challenges with decentralized governance. The DAO acted as a decentralized crowdfunding platform to fund Ethereum projects. Any account could transfer Ether (ETH), the cryptocurrency of Ethereum, in exchange for

voting power to select which projects should be supported. In June 2016, a vulnerability in the DAO's code was exploited by an attacker, allowing them to steal approximately one-third of the fund's assets (3.6 million ETH), valued at roughly $50 million at the time. In response to the DAO hack, the Ethereum community implemented a hard fork to revert the blockchain to a pre-attack state, effectively undoing the theft and returning the stolen funds to their rightful owners. [24] This decision was controversial, with some arguing that it violated the decentralization principles and immutability underpinning blockchain technology and resulted in a chain split, creating a new cryptocurrency, namely Ethereum Classic. Malicious actors can use those hard forks to launch several attacks on both networks, so-called Double-Spending or replay attacks. To analyze this phenomenon and assess the general behavior of forks, Kiffer et al. [29] ran nodes in both the Ethereum and Ethereum Classic blockchain and compared both networks. The authors' research provided multiple valuable insights on short-term and long-term fork dynamics: As expected, the Ethereum Classic blockchain suffered a massive drop in active node counts, severely decreasing blocks per hour. It took the network almost two days to stabilize the average block time again. Overall, Ethereum had a higher difficulty rate during the fork period, proving that most miners stayed with the main blockchain. [29]

4.3.3. Implications of Forks

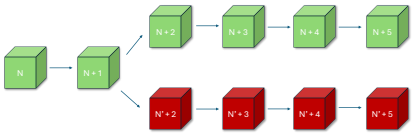
Nourmohammadi et al. [1] name temporary and permanent forks as one of the most critical challenges for blockchain technologies. Since forks need to be resolved, the chain needs to be reorganized, and transactions from blocks that do not belong to the new valid chain may or may not have occurred on the valid chain. Therefore, clients have to wait for more confirmations before finalizing transactions. [1] Decker et al. argue that blockchain forks are the reason transactions never reach finality and are only partially committed because of the tight coupling between the finality of transactions and forks. By adjusting their Bitcoin client to retrieve all incoming blocks without immediately rejecting invalid or forked blocks, which Bitcoin usually does, they observed 169 blockchain forks in 10000 block intervals. [2] Da Silva et al. [54] name forks as the reason for probable inconsistency in the network since they make transactions unreliable. Unreliable in the sense that the more forks happen, the more blocks have to be validated on top of a particular transaction to be declared final.
Other implications that need to be considered when thinking about blockchain forks are attacks like replay attacks. After the fork of Ethereum and the creation of Ethereum Classic, the lack of replay protection caused multiple accounts to lose ETC, Ethereum Classic's cryptocurrency. [30] To counter this, Ethereum introduced a special identifier in transactions with the Spurious Dragon [1] hard fork, the chain ID, which is supposed to prevent replayability. The chain ID only works in hard forks where the protocol is changed, e.g., temporary forks don't have replay protection as the chain ID remains the same. [5] [29]

[1] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-607.md

4.4. Discussion of the State of the Art Research

The research surveyed in this chapter has shown some research gaps regarding the connection between forks and non-finality. Some publications discuss the non-finality problem concerning forks, and some authors even mention state divergence and inconsistency [2] [58] but do not talk in detail about the consequences. Mighan et al. [58], for example, calculate the time of state inconsistency based on the probability of forks.
During our survey, we came across multiple implementations of blockchain simulators, with the limitation of only creating a complete simulated environment without incorporating real-life data. However, due to the complexity of the system state and its interactions, using real-world data appears to be the most promising approach for replicating the behavior and consequences of an actual blockchain fork. It is not yet clear if the effects can be easily abstracted to allow for more lightweight simulations.
We encountered numerous publications [53] [2] [1] [54] discussing the theoretical aspects of blockchain forks, like fork probability and its various factors. Still, we saw a need for more research on forks occurring in blockchains using the PoS consensus algorithm. Some researchers also highlight that forks do not exist in some PoS chains, depending on the type of PoS protocol. [58] The concept of non-finality of transactions regarding forks is a topic only a few researchers focus on. Analysis of fork behavior mainly discusses the reasons for the supposed events or their impact on the node behavior. Still, it doesn't account for a possible divergence of the blockchain state followed by missing transactions or overall different blocks in the split chains. Events similar to the DAO hack or Ethereums' transition from Proof-of-Work to Proof-of-Stake can cause discrepancies in transaction behavior and overall blockchain state. Those events are likely to continue to happen. A discrete stochastic blockchain fork simulator would allow researchers to analyze and quantify the effect of forks, the state-dependencies of transactions, in particular smart contracts, and the overall potential risks to the network.

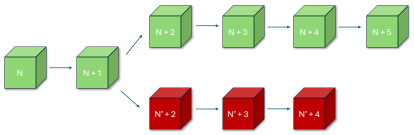
Intended (Permanent) Forks



A. Hard Fork resulting in 2 distinctive blockchains

B. Soft Fork introducing new Blocks with Protocol Changes

Accidental (Temporary) Forks



C. Accidental Fork after Network Delays or 2 simultaneous Block Proposals

Figure 4.1.: Visualization of different fork types: Forks A and B are applicable when considering permanent and intended forks. Fork C represents temporary and accidental forks.

Part II.

Development of a Fork Simulator for Ethereum

Introduction

The second part of this thesis focuses on developing a practical solution for a fork simulator to understand the phenomenon of forks better. Firstly, we discuss the concept of Ethereum clients, their distributions in the network, and predominantly the interaction between consensus and execution clients. After the transition from Proof-of-Work to Proof-of-Stake of Ethereum, we need to understand how the newly introduced consensus client initiates a block proposal and, respectively, a block import by analyzing the API used for communication between the two clients: The Engine API. Identifying and understanding the methods used by the clients is crucial to reproducing the block import process necessary to perform a valid fork simulation. We continue by implementing a practical solution of a fork simulator, following four stages of implementation: First, isolating the execution client's block creation, second, integrating the execution and consensus clients while masking the forked chain, third, forwarding transactions to replicate the original blocks accurately, and fourth, modifying transactions within a block. The Proof-of-Concept implementation is followed by the concept of how to validate the proposed fork simulation model.

5. Execution & Consensus Layer

Every Ethereum node is characterized by its underlying execution client, which plays a crucial role in defining and implementing the functionalities of the Ethereum protocol. These clients serve as the network's backbone, enabling nodes to participate in the networks' ecosystem and communicate with each other over a peer-to-peer network. The Ethereum community, such as the Ethereum Foundation, actively embraces the coexistence and interoperability of different computing clients as long as they adhere to the exact specifications and standardized communication protocols. All clients, including the network component, are open-source projects, meaning their source code is publicly available. These specifications provide a detailed and standardized description of the network's behavior and rules, ensuring a common understanding among different client implementations. At the time of writing, based on statistical analysis by 'clientdiversity.org,' [59] the Go-Ethereum (Geth) and Erigon clients are the most widely used in the Ethereum ecosystem. Geth, in particular, is responsible for approximately 49% of all nodes. It is debatable how accurate those statistics are since there is no way to know precisely what client a node is running. We chose to use the information from 'clientdiversity.org' since it is referenced by the Ethereum website and gathers data from multiple sources. [59] [9] However, it is essential to note that the Ethereum ecosystem is dynamic, and the popularity of client implementations may evolve as new projects emerge, developers innovate, and community preferences shift. [4] Especially with the transition from Proof-of-Work to Proof-of-Stake in September 2022, Ethereum 2.0 adopted a dual-client architecture where an Execution Layer (EL) client and a Consensus Layer (CL) client work together. Prior to the so-called Merge, a single execution client, mainly Geth or Erigon, was solely responsible for creating, proposing, validating, and propagating new blocks.

5.1. Execution Clients

In the post-merge Ethereum environment, execution clients are primarily responsible for the block creation process. Like the pre-merge client, the execution client is responsible for computing the global state of the blockchain, adhering to the protocol rules. Their role is focused on the construction and validation of blocks rather than being responsible for managing the Proof-of-Stake consensus-related functionality of the protocol. Transactions within blocks are gathered from the transaction pool. The transaction pool, often called the mempool is a temporary storage where pending transactions wait to be included in a block. Execution clients ensure the inclusion of only valid transactions within the proposed blocks and handle the computational tasks associated with processing those transactions, including executing smart contracts. Through their involvement in block

construction and validation, execution clients contribute to the overall maintenance and security of the blockchain network. [9] [60] As already mentioned above and illustrated in Figure 5.1, nodes running the Geth execution client are by far the most active network participants with over 49%, followed by Nethermind with approximately 25%, based on the statistics of 'clientdiversity.org'.

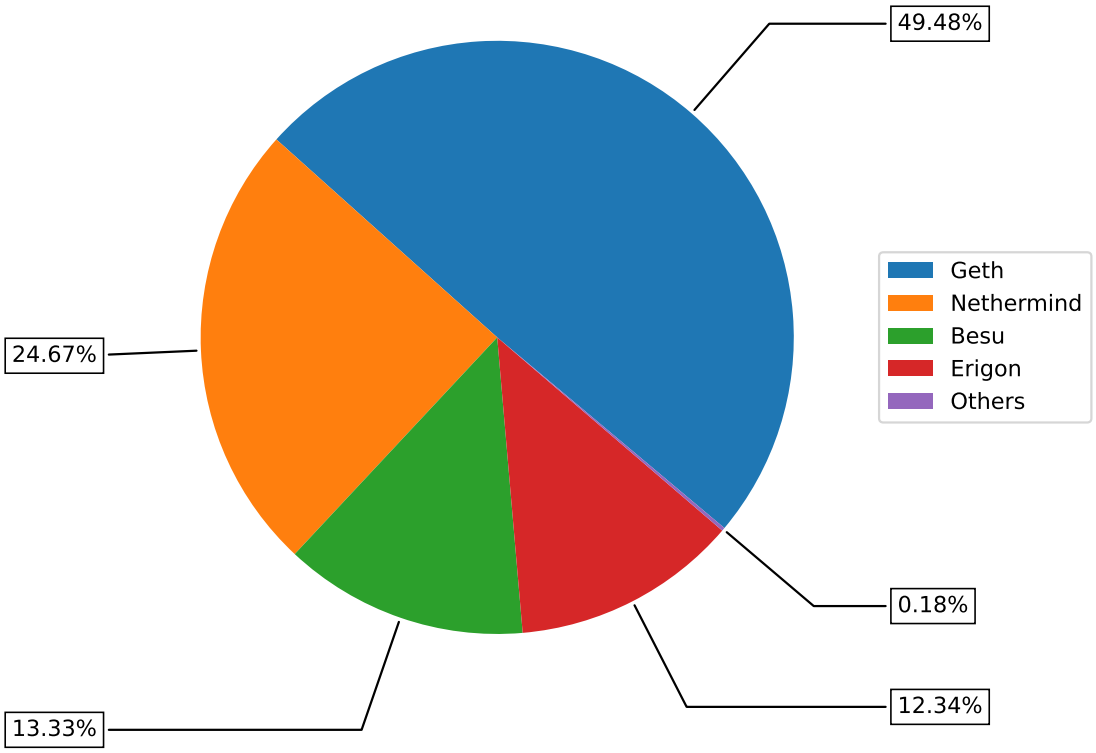


Figure 5.1.: Execution Client Distribution (October 2023) [59]

5.2. Consensus Clients

Consensus clients in the post-merge Ethereum ecosystem, operating under a PoS consensus mechanism, hold a key role in ensuring the integrity and security of the blockchain network. These clients actively participate in the consensus protocol and are responsible for block proposals, validation, and finalization. Validators, who function as consensus clients, are selected based on the amount of cryptocurrency they stake, serving as a measure of their economic commitment to the network. They instruct the execution client to create and broadcast blocks, including valid transactions. The execution client is still responsible for the validation of the block structure. In contrast, the consensus clients validate blocks by verifying the proposed block's correctness and adherence to the consensus protocol rules.

This involves confirming the validity of the transactions and ensuring that the block is constructed according to the required protocol specifications. However, the responsibility of proposing a block is not constant but is determined by a pseudo-random selection process within each slot, a unit of time in Ethereum's time structure. This selection process employs the RANDAO algorithm, which combines a hash from the block proposer with a seed that updates each block, producing a value that selects a specific validator from the total set. [9] The selection is fixed two epochs in advance to counter potential seed manipulation. The block proposer is tasked with broadcasting a signed beacon block that builds upon the most recent head of the chain. In a PoS-based consensus mechanism, consensus clients contribute to the finalization process, collectively agreeing on the canonical chain with so-called attestations. This process includes evaluating the level of agreement among validators and establishing a finality threshold to determine the accepted chain. [9] [61] To become a validator, node operators are required to deposit a minimum of 32 ETH into a dedicated deposit smart contract[1]. As seen in Figure 5.2 Prysm and Lighthouse are the most used consensus clients in Ethereum, both similarly distributed with over 33% network participants.

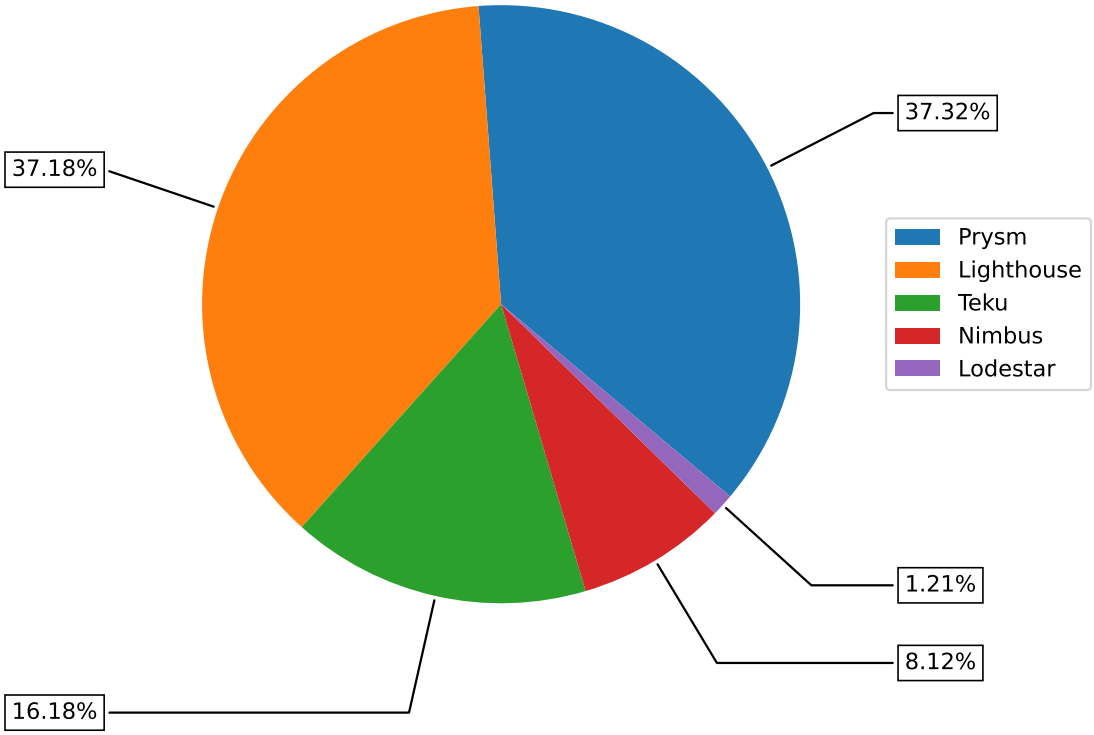


Figure 5.2.: Consensus Client Distribution (October 2023) [59]

[1] https://etherscan.io/address/0x00000000219ab540356cbb839cbe05303d7705fa

5.3. Communication Between Clients

A new network communication protocol called the Engine API was introduced during the transition from Proof-of-Work to Proof-of-Stake in Ethereum. It manages communication between the execution client and the new consensus client, which are usually operated on the same machine by the same entity. However, this API's available information and documentation were limited and fragmented within a single GitHub repository with several Markdown files. [2] Consequently, a substantial portion of our research effort was dedicated to identifying the essential methods, their purposes, and their proper usage, as the documentation was insufficient.

Key processes that required understanding included:

- How the Consensus Client communicates to the Execution Client about a newly proposed block that needs to be imported.
- How the Execution Client informs the Consensus Client about its current chain head.
- How the Consensus Client instructs the Execution Client to create and propose a new block.

Additionally, it was crucial to understand other methods defined by the Engine API that are vital for establishing functional and secure communication between the two clients.

5.3.1. Block Import

The RPC call *engine_newPayloadV2* in Listing 5.3 plays a crucial role in the interaction between the consensus client and the execution client. It instructs the execution client to add a new block to its local blockchain and provides all necessary block parameters like the block hash, block number, and timestamp as illustrated in the provided cURL (Client for URL) request in Listing 5.3. The execution client retains responsibility for validating and executing transactions within each block. Once the execution client has successfully processed the transactions, it updates its local blockchain by setting the block as the new head of the chain.

5.3.2. Block Creation

The block creation procedure is more complex than the previously described block import process since it includes multiple requests between the consensus and execution client. As illustrated in Figure 5.3, the consensus client has to send a total of four requests until a newly created block is added as the new blockchain head:

1. **engine_forkChoiceUpdatedV2**
   The *engine_forkchoiceUpdated* method, a crucial component of the Engine API, illustrated in 5.1, performs two essential functions within the Ethereum protocol.

[2] https://github.com/ethereum/execution-apis/blob/main/src/engine/common.md

First, it transmits updated consensus information from the consensus client to the execution client. This includes:

a) headBlockHash: The hash of the latest block, which is the most recent block constructed and validated by Ethereum validators. [62]

b) safeBlockHash: The hash of the safe block, which has received attestations from at least two-thirds of Ethereum's validator set. [62]

c) finalizedBlockHash: The hash of the finalized block is a justified block located one epoch behind the most recently justified block. A justified block is considered an actual final block with no chance of being revoked. [62]

Second, the *engine_forkchoiceUpdated*, illustrated in 5.2, method triggers the creation of a new block within the execution client. This process begins when the consensus client invokes the method with a non-null payload attributes parameter. This parameter includes critical data for block creation, such as:

a) timestamp: The proposed time for the new block.

b) prevRandao: The previous RANDAO value generates randomness in the network.

c) suggestedFeeRecipient: The address suggested for receiving the block's transaction fees.

d) withdrawals: Requests to withdraw funds, such as validator rewards or penalty withdrawals.

The method subsequently returns a payload ID, which is used in subsequent operations.

```
curl -X POST --data '{
  "jsonrpc": "2.0",
  "method": "engine_forkchoiceUpdatedV2",
  "params": [
    {
      "headBlockHash": "headBlockHash",
      "safeBlockHash": "safeBlockHash",
      "finalizedBlockHash": "finalizedBlockHash"
    },
    {
      "timestamp": "0x5",
      "prevRandao": "prevRandao",
      "suggestedFeeRecipient": "suggestedFeeRecipient"
    },
    "withdrawals": []
  ],
  "id": 67
}' http://127.0.0.1:8551
```

Listing 5.1: CURL request for engine_forkchoiceUpdatedV2

2. **engine_getPayloadV2**
The *engine_getPayloadV2* method, illustrated in 5.3, is a critical function within the

Ethereum Engine API that is used to fetch the execution payload for a specific payload build process. This method is invoked by the consensus client and is identified by a unique ID received as a response result by calling *engine_forkChoiceUpdated*. The execution payload fetched by this method contains crucial information about a block, including its transactions, state, and other attributes. This payload is used to propose a new block in the Ethereum network.

```
curl -X POST --data
{"jsonrpc":"2.0","method":"engine_getPayloadV2","params":
["0x00659f96de28ef20"],"id":1} http://127.0.0.1:8551
```

Listing 5.2: CURL request for engine_getPayloadV2

3. **engine_newPayloadV2**
   In the Ethereum network, the consensus client uniformly handles block imports, regardless of whether the block comes from the execution client or any other participant in the network. This is evident in the block creation process since the third call mirrors the procedure for importing a new block retrieved from the network. Specifically, the consensus client retrieves the necessary block information via the *engine_getPayload* method and subsequently imports the obtained execution payload using the *engine_newPayload* method. This standardized approach ensures consistency in block processing across the network. The parameters presented in the code example below represent the block structure of any Ethereum 2.0 block.

```
curl -X POST --data
{"jsonrpc":"2.0","method":"engine_newPayloadV2","params":[{
"parentHash" : parentHash,
"feeRecipient" : feeRecipient,
"stateRoot" : stateRoot,
"logsBloom" : logsBloom,
"prevRandao" : prevRandao,
"gasLimit" : "0x1c9c380",
"gasUsed" : "0x0",
"timestamp" : "0x6468c096",
"extraData" : "0x",
"baseFeePerGas" : "0x7d297cfb2",
"transactions" : [ ],
"withdrawals" : [ ],
"blockNumber" : "0x89c049",
"receiptsRoot" : receiptsRoot,
"blockHash" : blockHash}],"id":67} http://127.0.0.1:8551
```

Listing 5.3: CURL request for engine_newPayloadV2

4. **engine_forkChoiceUpdatedV2**
   To finalize the block creation process, it is necessary to call *engine_forkchoiceUpdated* again, but with the payload attributes being null. This is going to confirm the previously soft-imported block as the head of the chain. Now the execution client will officially regard the new block as part of the canonical chain.

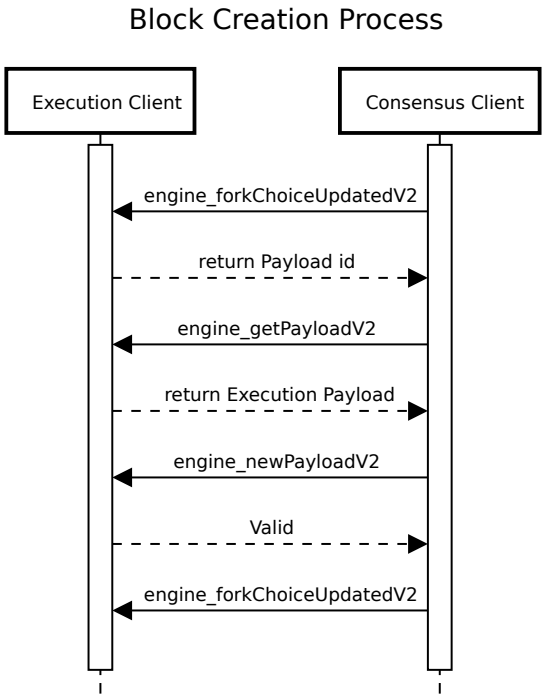


Figure 5.3.: Block Creation Process: This process involves four RPC calls between the EL and CL: 1. engine_forkChoiceUpdatedV2, 2. engine_getPayloadV2, 3. engine_newPayloadV2, and 4. engine_forkChoiceUpdatedV2.

Generally, the block is constructed by gathering data from the proposer's local chain view. Upon creation, the block is added to the proposer's local chain and distributed to peers via the consensus layer network. Validators receiving the block undertake a verification process, examining the data within, including the parent-child relationship of the proposed block, which ensures that the new block correctly references its immediate predecessor to maintain the continuity of the blockchain. They also verify slot correspondence, ensuring that the block was proposed within the correct time slot allocated to the validator, and confirm the proposer index to verify that the correct validator submitted the block. Additionally, validators evaluate the RANDAO reveal for its proper contribution to the network's randomness and review the proposer's slash status to determine if the proposer has previously breached protocol rules. Slashing may result from protocol violations such as double-signing or proposing conflicting blocks, leading to penalties, including the forfeiture of staked ETH, thereby upholding the security and integrity of the blockchain. [9] [61] [60] The execution payload is unbundled, and the validator's execution client re-executes the transactions in the list to verify the proposed state change. If the block passes all these checks, it is added to each validator's canonical chain. The block proposer is reimbursed for their work, receiving a base reward calculated as a function of the number of active validators and their adequate balances, similar to the previous Proof-of-Work chain mining reward. The proposer also receives a fraction of the base reward for every

valid attestation in the block. Additionally, a reward is given for reporting validators that should be slashed, equal to *1/512 * effective balance* for each slashed validator. [9] [61] [60]

5.3.3. Additional methods

Besides the stated specifications, including other non-described methods, the protocol requires several other RPC requests to be implemented to access the state and logs through the same connection. The required methods are as follows:

- eth_blockNumber
- eth_call
- eth_chainId
- eth_getCode
- eth_getBlockByHash
- eth_getBlockByNumber
- eth_getLogs
- eth_sendRawTransaction
- eth_syncing

5.3.4. Observations

In the process of implementing fork simulations and validating the outcomes, we identified several anomalies and instances of protocol specifications being disregarded with our execution client Besu and our consensus client Teku. It is important to emphasize that those anomalies were only experienced with Teku and Besu. We are going to discuss those specific client implementations in greater detail in Section 6.1. Other implementations may work differently but were not tested.
During the initial testing phase, we realized that specific parameters within the *engine_forkChoiceUpdated* function might not necessarily require proper configuration. Our observations indicated that the timestamp is the only field requiring accurate configuration. Provided that the *headBlockHash*, *safeBlockHash*, and *finalizedBlockHash* are legitimate former block hashes, the execution client does not require association with the correct and required epoch. If, by accident, *safeBlockHash* and *finalizedBlockHash* are block hashes of blocks appended many blocks prior to the current head of the chain (approx. 5000 blocks), the execution client Besu will not be able to pursue the block creation process since it uses a recursive function to verify the parent-child relationship between blocks. It starts with the current block and checks its parent hash against the provided *safeBlockHash* or *finalizedBlockHash*. If these hashes originate from multiple

blocks prior to the current chain head, the recursive function must traverse numerous blocks to validate the relationship. This could exceed the JVM stack limit or cause a significant delay, halting the block creation process.
In the process of implementing various methods of the Engine API and basic Remote Procedure Calls, it was discovered that the consensus mechanism does not appear to respond to all of these methods. This observation suggests a lack of integration between the implemented methods and the consensus mechanism. Specifically, methods such as *eth_ syncing* and *eth_ blockNumber* were found to be disregarded by the consensus mechanism. These methods play crucial roles in the Ethereum protocol; *eth_ syncing* returns an object with data about the sync status or false if not syncing, and *eth_ blockNumber* returns the number of the most recent block. Although intentionally sending outdated block information to the consensus client, notifying it of an unsynced blockchain state, the consensus client did not act upon those falsified requests to inform the consensus client of an unsynced execution client but kept sending newly validated blocks. This issue only occurred after the execution client had informed the consensus client that synchronization was complete and the local blockchain was fully synced at one point. After analyzing Tekus' source code, we observed that the additional methods described in Subsection 5.3.3 are exclusively utilized during the initial synchronization phase and are not included in the official Engine API specification within Teku's codebase.
Those discoveries suggest that the execution client's requirements for block validation may be less strict than initially assumed. This could have implications for the overall robustness and security of the protocol and warrants further investigation.

6. Proof of Concept - Implementation of a Fork Simulator

With the knowledge we have gained regarding the interaction between the CL and EL, we can now simulate a blockchain fork on a local node. Our goal is to bypass the standard protocol behavior by injecting artificial consensus client messages that mimic the presence of new blocks. Essentially, we avoid the actual consensus rules to locally define alternative blocks, thereby creating a simulated fork on our local node. Therefore, the primary task is to instruct the execution client to generate new blocks. Our practical solution is designed to be lightweight, open-source, and easily adaptable. To achieve this, we developed a systematic implementation strategy that advances through iterative steps.

- **Iteration 1**: This first implementation concerns commanding the execution client to produce new blocks by using a Python script to dispatch curl commands. To ensure a controlled environment focused solely on the execution client's ability to generate blocks, it is essential to disconnect the consensus client. This prevents the consensus client from receiving or syncing new blocks, allowing us to isolate and test the execution client's capacity to create blocks without interference from incoming data.

- **Iteration 2**: The second iteration integrates the execution and consensus clients with our designated fork client. When the consensus client attempts to transmit a new block to the execution client, the steps outlined in iteration 1 are used to create new blocks. Since the consensus client expects a response after the block import, we provide a valid response each time a block is incorporated into the forked chain, embodying the genuine block hash originating from the original chain. This maneuver effectively masquerades the forked chain, making it indistinguishable from the original chain from the consensus clients' perspective.

- **Iteration 3**: The third stage retains the functionalities of Iteration 2 but introduces an essential modification. Whenever the consensus client transmits a new block, the blocks' transactions are forwarded to the execution client when creating a new block. This procedure permits the creation of an accurate replica of the original block within the forked chain.

- **Iteration 4**: This iteration serves as a refinement of Iteration 3, involving modifications to the transactions of the original block. These adjustments could contain a spectrum of changes, from a simple reordering of transactions to more complex operations, such as eliminating or replacing transactions. However, it remains

crucial throughout this iterative process to uphold the validity of all blocks, ensuring the forked chain's operability and integrity.

Notably, Iterations 1 and 2 do not necessitate any modifications to the execution client, yet they are sufficient enough to enforce a chain fork. As previously detailed, the consensus client communicates to the execution client via a Remote Procedure Call to trigger the block creation process. Although the payload for these methods is pre-established, developing a new RPC method to encapsulate transaction data for Iterations 2 and 3 might be necessary. An alternative approach involves incorporating this functionality by appending the needed attributes to the existing method, thereby eliminating code duplication. This comprehensive plan, therefore, serves as a roadmap for creating a blockchain fork simulation. As this plan is implemented, further modifications and refinements may become apparent, underlining the importance of flexibility and adaptability.

6.1. Architecture

The architecture of the proposed fork simulator was intentionally designed with simplicity as a core principle. In addition to the execution client and consensus client, users only require one additional component. Therefore, instead of requiring just two clients, each node in the simulation setup requires a total of three clients. While Geth may be the most widely used execution client, it is essential to note that most simulator features of the herein-proposed design are exclusively compatible with Besu. Besu, an open-source Ethereum client developed by ConsenSys, is implemented explicitly for enterprise use cases. Implemented in Java, Besu has advanced capabilities for controlling permissions, includes features to enhance privacy, and is optimized for better performance through the ability to process transactions in parallel. Given its focus on enterprise usage, Besu distinguishes itself by providing comprehensive and well-structured code and detailed documentation. [60] This was instrumental for thoroughly understanding the block creation process and communication between the execution and consensus clients.
Users have the flexibility to choose any consensus client based on their preferences. In our implementation, we did not alter the consensus engine code. While we did not test this directly, our fork simulation framework is designed to accommodate any consensus client, suggesting that integration should be possible. All of our experiments used Teku for consensus purposes. Teku, similar to Besu, is an open-source consensus client developed by ConsenSys. Written in Java, Teku caters to validators participating in the PoS consensus mechanism within the Ethereum ecosystem with performance optimizations like multi-threading and efficient network communication, enhanced scalability, and responsiveness. Moreover, Teku offers monitoring and management features, simplifying the operational aspects of validator engagement. Teku is a comprehensive and reliable Java-based solution for Ethereum's PoS consensus validators. [61]
Our Fork Simulator itself is written in Python using the framework Flask. Flask is a minimalist web framework offering a straightforward and efficient approach to web

application development. It allows developers to quickly design and implement web applications while maintaining flexibility and scalability. Flask's modular architecture and intuitive design principles are ideal for academic projects and research where a lightweight and adaptable web framework is desired. [63] The essential concept of the defined architecture is the interception of the fork simulator component of the communication between the consensus client and the execution client. In the earlier implementations of the fork simulator, this altered communication process was kept primitive but evolved with the implementation of the simulator itself. Since the execution and consensus engine need to be connected at all times, the fork simulator acts as a **middleware** with the execution client and consensus client connected to it, as illustrated in Figure 6.1.

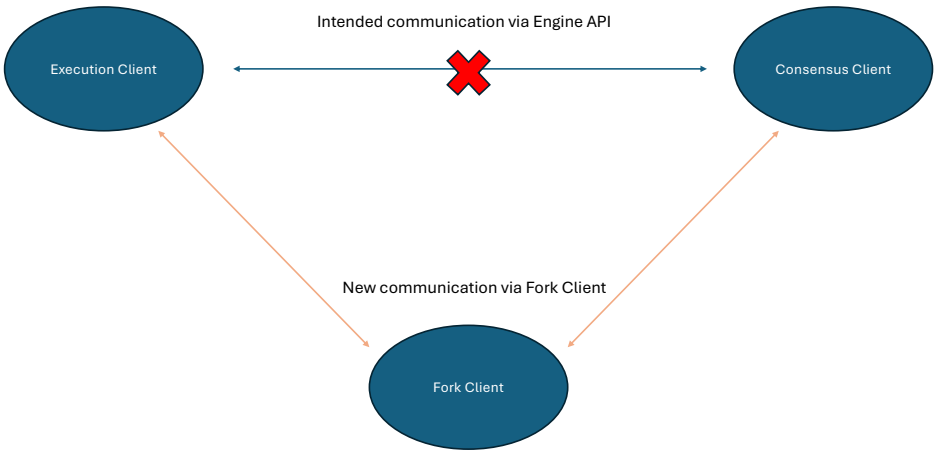


Figure 6.1.: The interaction of the three necessary components for the proposed fork simulation involves the additional Fork Client as the central element, which modifies the communication flow between the EL and CL

6.2. Implementation

In order to mimic the block creation process described in Subsection 5.3.2 it is necessary to first define all four essential block creation methods since every version of the fork simulator must undergo the same procedure.

The entry point for any block creation process is the method called *engine_forkchoiceUpdatedV2* with the following parameters: *HeadBlockHash*, *safeBlockHash*, *finalizedBlockHash*, *timestamp*, *withdrawals*, and *suggestedFeeReceipient* as illustrated in the Listing 6.1 and already discussed in Subsection 5.3.2. All parameters are encapsulated in a template JSON and passed as the body parameter for the HTTP request, including the necessary header parameters.

```
url = "http://127.0.0.1:8551"
headers = {'Content-Type': 'application/json'}

engine_forkchoiceUpdatedV2 = {
    "jsonrpc": "2.0",
    "method": "engine_forkchoiceUpdatedV2",
    "params": [
        {
            "headBlockHash": headBlockHash,
            "safeBlockHash": safeBlockHash,
            "finalizedBlockHash": finalizedBlockHash
        },
        {
            "timestamp": timestamp,
            "prevRandao": "0x60EFEF1D",
            "suggestedFeeRecipient": suggestedFeeRecipient,
            "withdrawals": []
        }
    ],
    "id": 67
}

response =
    requests.post(url, json=engine_forkchoiceUpdatedV2, headers=headers)
```

Listing 6.1: Python HTTP Request to initiate Block Creation Process. (Step 1)

The second Python method, illustrated in Listing 6.2, is responsible for retrieving the created payload of step 1. It is vital to note that the requests can only be sent if the previous *engine_forkchoiceUpdatedV2* call returns a valid response since the response parameter *payloadId* is crucial for the *engine_getPayloadV2* request.

```
if response.status_code == 200:
    response_data = response.json()
    payload_id = response_data['result']['payloadId']

    engine_getPayloadV2 = {
        "jsonrpc": "2.0",
        "method": "engine_getPayloadV2",
        "params": [payload_id],
        "id": 1
    }

    response = requests.post(url, json=engine_getPayloadV2, headers=
    headers)
```

Listing 6.2: Python HTTP Request to retrieve newly created Block. (Step 2)

The last block creation method, illustrated in Listing 6.3, is responsible for importing the created block in the blockchain. All essential block parameters are returned by *engine_ getPayloadV2*.

```
if response.status_code == 200:
    response_data = response.json()
    execution_payload = response_data['result']['executionPayload']

    engine_newPayloadV2 = {
        "jsonrpc": "2.0",
        "method": "engine_newPayloadV2",
        "params": [execution_payload],
        "id": 1
    }

    response =
        requests.post(url, json=engine_newPayloadV2, headers=headers)
```

Listing 6.3: Python HTTP Request to soft import Block. (Step 3)

To confirm the soft import of a block, we defined a similar method as in step 1, with the only difference being that the passed JSON object omits any payload parameters such as *timestamp*, *suggestedFeeReceipient*, and *withdrawals*, as illustrated in Listing 6.4.

```
if response.status_code == 200:
    response_data = response.json()
    finalized_hash = response_data['result']['latestValidHash']

    engine_forkchoiceUpdatedV2_finalize = {
    "jsonrpc": "2.0",
    "method": "engine_forkchoiceUpdatedV2",
    "params":
            [
                {
                    "headBlockHash": finalized_hash,
                    "safeBlockHash": safeBlockHash,
                    "finalizedBlockHash": finalizedBlockHash
                }, None
            ],
            "id": 67
    }
    response =
        requests.post(url, json=engine_forkchoiceUpdatedV2_finalize,
    headers=headers)
```

Listing 6.4: Python HTTP Request to hard import Block. (Step 4)

Every version or stage of the fork simulation implementation uses these defined methods, which replicate the block creation process exactly. We refer to these methods in the descriptions of each version as *ForkChoiceUpdate*, *engineNewPayload*, and *engineGetPayload*.

6.2.1. Version 1

Version 1 of our proposed fork simulator is kept simple, as it was primarily intended to understand the methods and parameters of the Engine API. Despite its simplicity, it can enforce a blockchain fork without relying on any consensus engine. The fork simulator acts as a consensus engine in this setup by signaling the execution client to create a new block. It is important to note that Version 1 is designed to process one block at a time. The simulator invokes a new *ForkChoiceUpdate* with a valid payload to initiate the block creation sequence in the execution engine. When the execution client receives this request, it builds a block by selecting optimal transactions from the transaction pool and then returns a payload ID to the fork simulator. With this payload ID, the simulator retrieves the necessary payload attributes and submits them as the execution payload in the *engineNewPayload* method. This instructs the execution engine to soft-import the newly created block as the current head of the chain. "Soft import" means that the block is considered the new head of the chain but still requires confirmation from the consensus client, in this case, the fork simulator, to become officially acknowledged. Finally, the fork simulator invokes a concluding *ForkChoiceUpdate* without any payload attributes to facilitate the final creation and import of the new block. This process ensures that the block is fully integrated into the blockchain after consensus validation. Figure 6.2 visualizes all necessary method calls described in Section 6.2 as a sequence diagram.

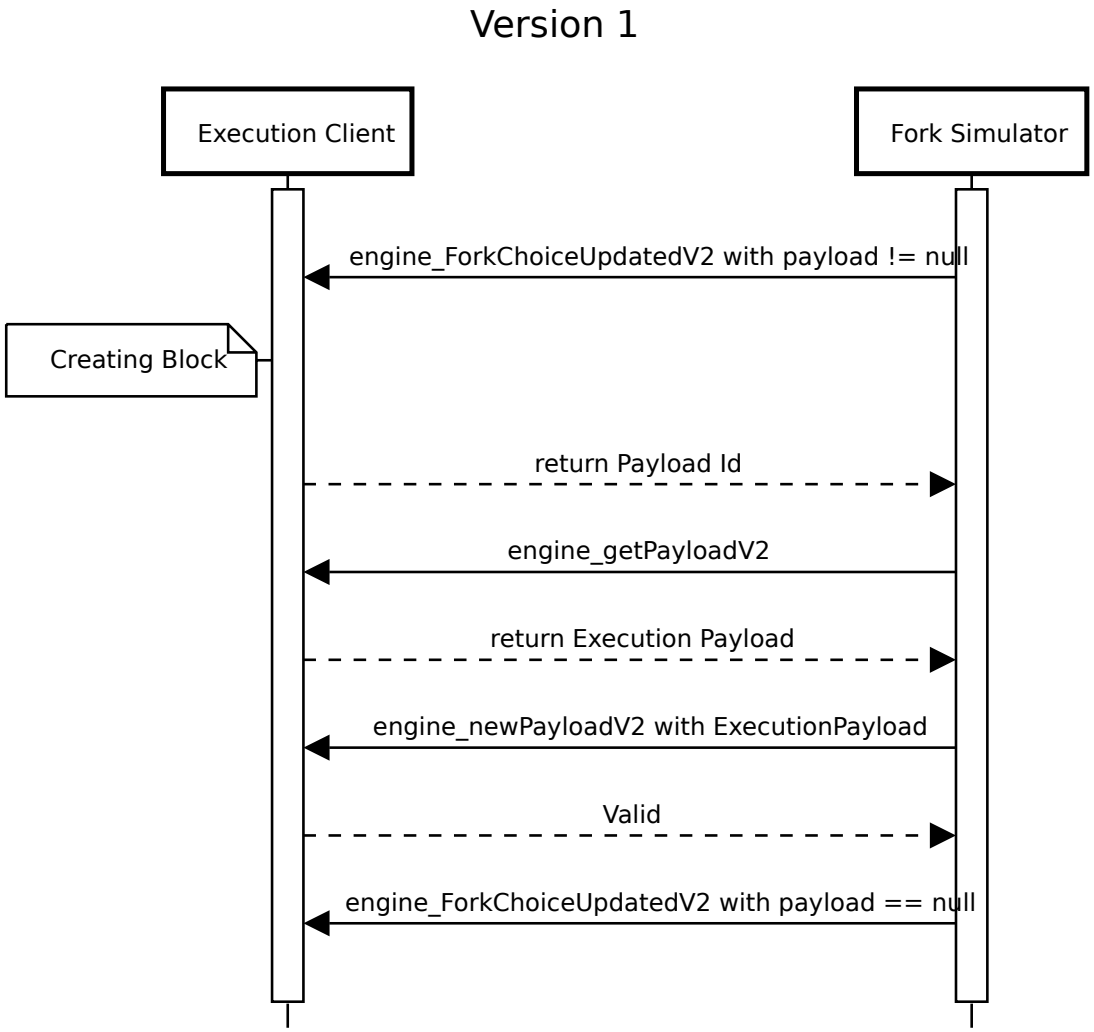


Figure 6.2.: Visualization of all necessary function calls between the components for the first version of the Fork Simulator

6.2.2. Version 2

Version 2 of our fork simulator introduces the integration of the consensus engine; however, it comes with some limitations. In this version, the integration is simplified to handle the basic forwarding of imports exclusively. Instead of direct API requests to the execution engine, the consensus client must now interact with the fork simulator, which acts as an intermediary redirecting proxy to the execution client. The information transmitted through this process is limited to the fee recipient and the list of withdrawals. When the consensus client notifies the fork simulator about a new block for importing, the fork simulator initiates the block creation procedure by executing a sequence of requests identical to version 1. In this updated implementation, the execution client still retrieves

transactions from the transaction pool to form a new block, meaning that the block is entirely different from the original main block, leading to an imminent state divergence.

```
if method == "engine_newPayloadV2":
    block_number = int(params[0]["blockNumber"], 16)
    block_hash = params[0]["blockHash"]
    fee_recipient = params[0]["feeRecipient"]
    withdrawals = params[0]["withdrawals"]
    print(block_number)

    response = {
        "jsonrpc": "2.0",
        "id": 1,
        "result": {
            "status": "VALID",
            "latestValidHash": block_hash,
            "validationError": None
        }
    }
    global last_hash
    last_hash = create_block(last_hash, fee_recipient, withdrawals)

    return jsonify(response)
```

Listing 6.5: Handles the engineNewPayload method to process block data, reflecting the fork client's intermediary role in Version 2.

However, there's a significant change: the fork simulation becomes dynamic and ensures a continuous flow of new block imports from the consensus client. The simulation achieves this by returning the actual block hash of the original mainnet block to the consensus client. This step is necessary because consensus clients perform sanity checks to verify the consistency and integrity of the blockchain state. If the consensus client receives an unexpected block hash, it might discontinue the block imports, recognizing the inconsistency as a potential issue. By returning the original block hash, the simulation maintains the appearance of a synchronized chain, allowing the consensus client to proceed with block imports without interruption. Again, Figure 6.3 visualizes all necessary methods as a sequence diagram, and Listing 6.5 shows the implementation of the code.

It is important to note that for Version 2 and the subsequent Version 3 (see Subsection 6.2.3), the main Ethereum network itself posed limitations for real-time processing. Due to the high computational demands on the mainnet, it was impossible to simulate forked blocks dynamically. Specifically, the execution client's efforts to create new blocks were frequently interrupted by the consensus client, which was continuously importing new blocks from the live P2P network. In contrast, these issues were not prominent during testing on Ethereum 2.0 test networks, where the computational load is significantly lower. Potential solutions for the Ethereum 2.0 mainnet could involve isolating the node from the P2P network and feeding it blocks at a slower rate or using a message queue to manage incoming blocks, only processing them after the previous block import was successful. Both approaches would require additional modifications. Consequently, our

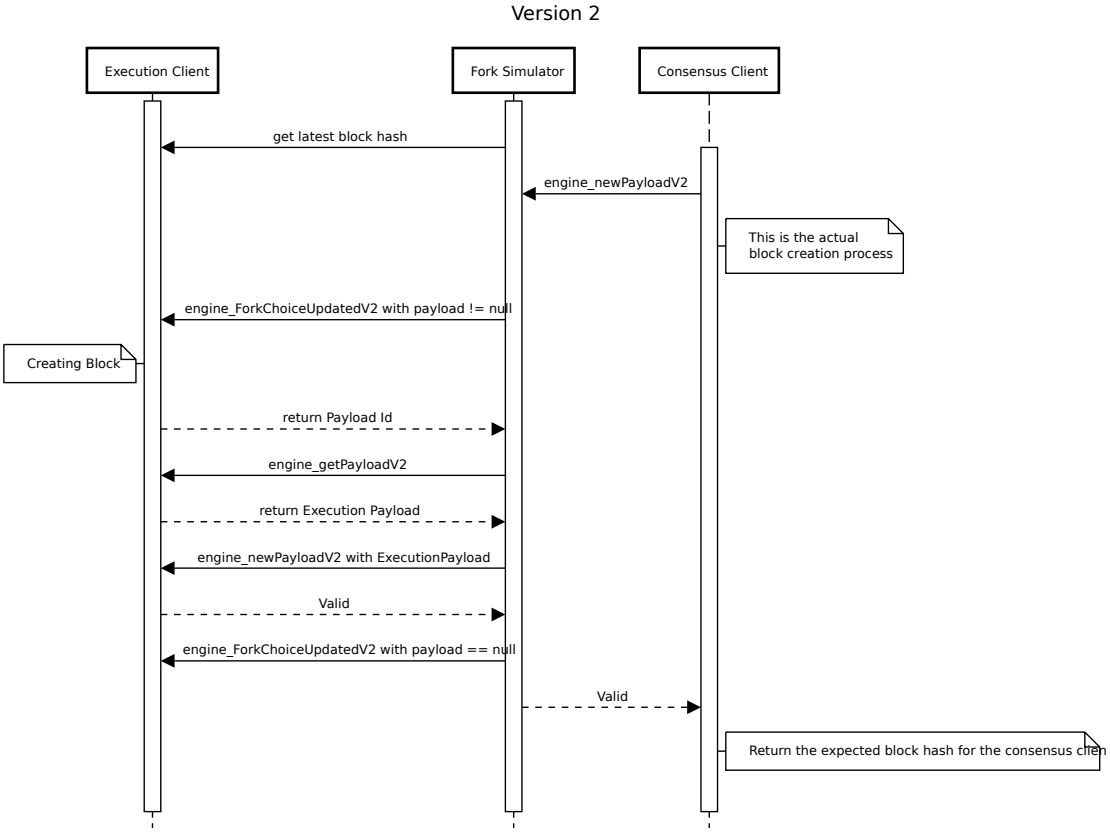


Figure 6.3.: Visualization of all necessary function calls between the components for the second version of the Fork Simulator

experiments were carried out exclusively on Ethereum 2.0 test networks to avoid these mainnet challenges.

6.2.3. Version 3

When comparing the sequence diagrams of version 2 6.3 and version 3 6.4, the primary distinction lies in including transactions in the block creation process. While the fork simulator continues to serve as a redirection component, it is no longer limited to only passing withdrawals and fee recipient parameters. Instead, it also extracts the RLP-encoded transactions from incoming blocks and transmits them as an additional payload attribute in the *engineForkChoiceUpdated*. As a result of these protocol adjustments, modifications are also required in the code of the execution engine. Several methods must be overloaded to include transactions in the block creation process, and certain functions must be altered.

```
final EngineForkchoiceUpdatedParameter forkChoice =
    requestContext.getRequiredParameter(0,
    EngineForkchoiceUpdatedParameter.class);
final Optional<EnginePayloadAttributesParameter> maybePayloadAttributes =
    requestContext.getOptionalParameter(1,
    EnginePayloadAttributesParameter.class);
final Optional<ForkTransactions> forkTransactions =
    requestContext.getOptionalParameter(2, ForkTransactions.class);

////////////////////////////////////////////////////////////////////////

Optional<PayloadIdentifier> payloadId = null;
if (transactions == null) {
    payloadId =
        maybePayloadAttributes.map(
                payloadAttributes ->
                        mergeCoordinator.preparePayload(
                            newHead,
                            payloadAttributes.getTimestamp(),
                            payloadAttributes.getPrevRandao(),
                            payloadAttributes.getSuggestedFeeRecipient(),
                            withdrawals));
} else {
    EnginePayloadAttributesParameter payloadAttributes =
    maybePayloadAttributes.get();
    payloadId =
        Optional.ofNullable(mergeCoordinator.preparePayload(newHead,
              payloadAttributes.getTimestamp(),
              payloadAttributes.getPrevRandao(),
              payloadAttributes.getSuggestedFeeRecipient(),
              withdrawals,
              transactions));
}
```

Listing 6.6: Adaptation of the Besu implementation of the ForkChoiceUpdated RPC endpoint to initiate the block creation process. The *EngineForkchoiceUpdatedParameter* was modified to include RLP-encoded transactions as an additional parameter.

In the process of implementing the fork simulation, we introduced a mechanism to handle the integration of transactions into the block creation process. This involved adding the capability to include transactions directly within the payload attributes of *engineForkChoiceUpdated*. A new component, displayed in Listing 6.6 was developed to manage these transactions, ensuring they are appropriately encoded and included in the block creation sequence. This process starts with the synchronization method triggering block creation, which then moves through a series of steps within the block creation framework. The process ensures that the transactions are correctly prepared and integrated into the new block, leading to its final construction and addition to the local blockchain. The overall concept of this workflow is illustrated at a higher level of abstraction in Appendix A.1, offering a clearer understanding of how the block creation

is managed within the simulation.

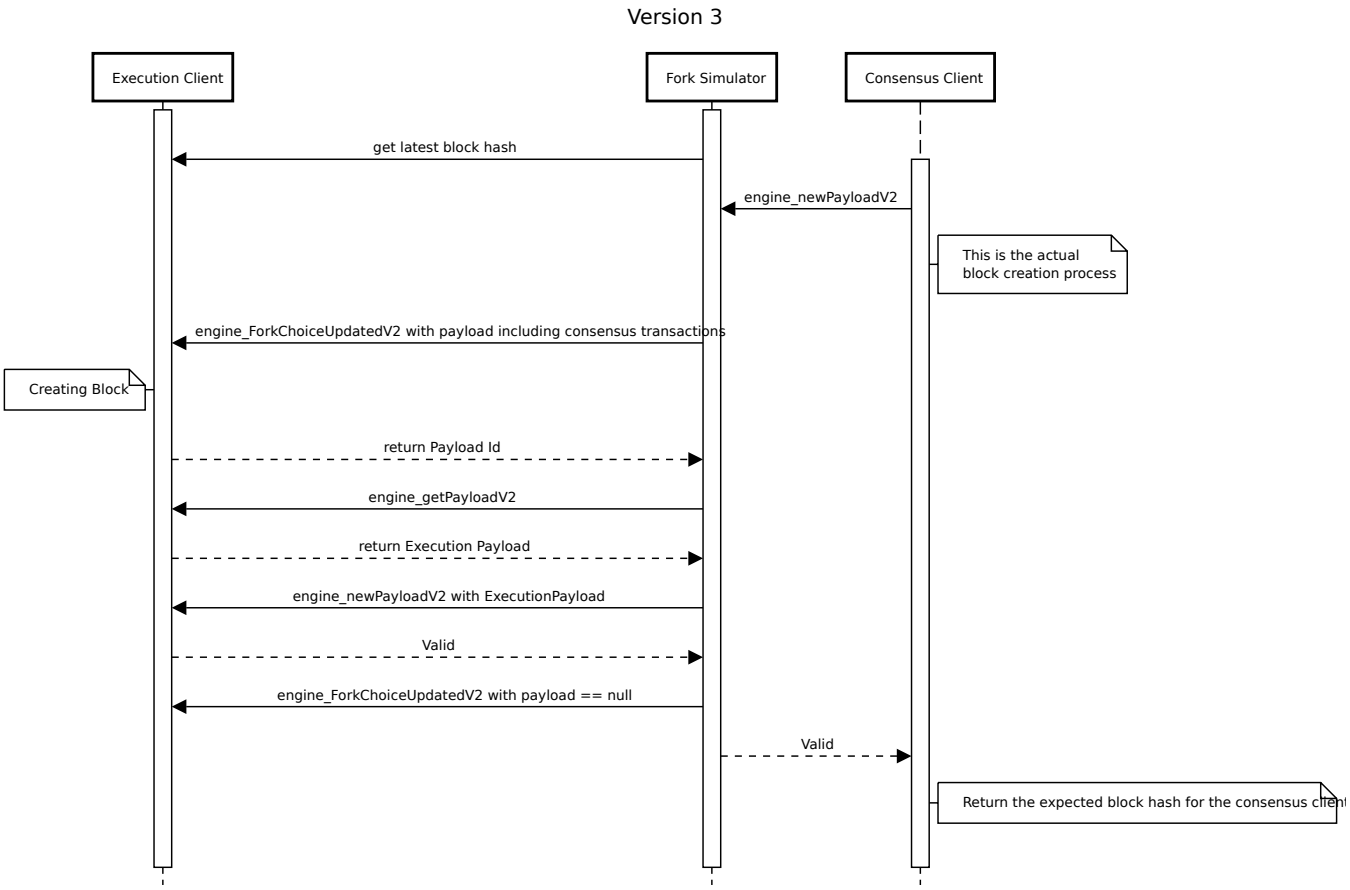


Figure 6.4.: Visualization of all necessary function calls between the components for the third version of the Fork Simulator

6.2.4. Version 3.1

Version 3.1, as visualized in Figure 6.5, has been introduced to enable and analyze forks at any desired block height. In this iteration, the fork simulator retains the same modified execution client as in version 2 since including transactions in the block creation process is necessary. To enforce a blockchain fork at a specific block height n, the execution client itself must be reverted to establish block n as the new head of the chain. This is achieved through RPC calls, ensuring that the blockchain is effectively forked at the targeted block height. Listing 6.7 shows the implementation of the code used to simulate a blockchain fork at a specific block height, including the retrieval of block data and processing of RLP-encoded transactions in Version 3.1.

```python
def init_past_fork(block_number):
    node = web3.Web3(web3.Web3.HTTPProvider(
        "http://127.0.0.1:8546",
        request_kwargs={'timeout': 3600}
    ))

    block = node.eth.get_block(block_number)
    transaction_count = len(block["transactions"])
    timestamp = hex(block["timestamp"])
    fee_recipient = block["miner"]
    withdrawals = [node.to_json(i) for i in block["withdrawals"]]
    parent_hash = block["parentHash"].hex()
    transaction_rlps = []

    for index in range(transaction_count):
        transaction_rlps.append(node.eth.get_raw_transaction_by_block(
    block_number, index).hex())

    create_block(parent_hash, fee_recipient, withdrawals,
    transaction_rlps, timestamp)

init_past_fork(17366687)
```

Listing 6.7: Code for simulating a blockchain fork at a specific block height in Version 3.1, including block data retrieval and RLP-encoded transaction processing.

Additionally, a second fully synchronized execution client is required to perform blockchain forks at any block height. It retrieves all the vital block information necessary for the fork simulation, such as the miner's address, timestamp, withdrawals, and RLP-encoded transactions. Once all the essential block parameters have been successfully retrieved, the subsequent block creation process executed by the fork simulator closely resembles the one implemented in version 3.

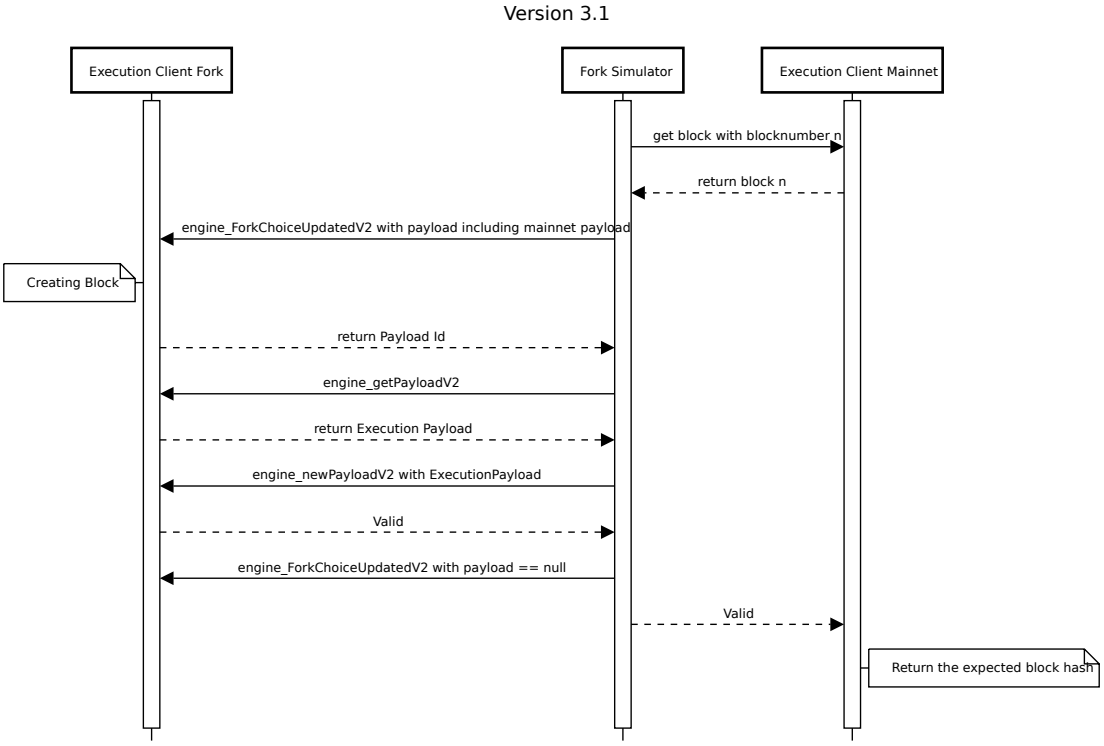


Figure 6.5.: Visualization of all necessary function calls between the components of the Fork Simulator version 3.1

7. Validation

With the Ethereum 2.0 fork simulator's capability to simulate forks, the next crucial step is to validate our simulation model. Simulation model validation is essential to ensure the accuracy and reliability of the simulation, confirming that it accurately represents the behavior of real-world blockchains. To achieve this, we tested our proposed simulation model on various test networks, including Goerli, Sepolia [64], and the Ethereum 2.0 mainnet. It is worth noting that as of April 2024, the Goerli test network has been discontinued. To define a sufficient framework demonstrating and validating the block-creation process, we must understand when a block is valid, and to do so, we have to look closer at the general conceptual design of the Ethereum 2.0 blockchain, its structure of blocks and transactions, and the concept of Merkle trees and blockchain state.

7.1. Definitions

As already mentioned, our simulation model is tested against both the main blockchain and test environments. In order to verify our results, we need to showcase a reproducible block-creation process, therefore the following block body and header fields are critical:

- **Block number**: The block number in Ethereum is an atomic identifier assigned to each block in the blockchain, representing its sequential position within the chain. [9] The block number of a simulated forked block should, therefore, adhere to the sequential order of the incremented block count in the blockchain. It is essential to mention that the protocol would not allow the creation of a block with an invalid block number anyway.

- **Block hash**: The block hash in Ethereum is a cryptographic hash function applied to the block header of each block in the blockchain. It is a unique identifier and a security measure that ensures the integrity and immutability of the block and its associated data. [9] Therefore, identically setting all block header fields in a simulated forked block to its original main chain block should result in the same hash. However, by doing so, we would not create a forked blockchain, we would produce an exact copy of the correct chain. Therefore, altering at least one block header field is crucial to acquiring a different block hash and, consequently, a different block. In our simulation model, we choose the *extra Data* field. The extra data field is an arbitrary 32-byte value that stakers or previously miners used to add additional information or metadata to a block. Furthermore, the block headers field *parent hash* should also be replaced and set correctly if the fork is deeper than 1. [9]. We

will further on call blocks with altered block hashes but otherwise identical fields **similar block**.

- **State root**: The state root in Ethereum is the Merkle Patricia trie's cryptographic hash that represents the Ethereum Virtual Machine state after executing all transactions within a block. It is included in the block header, ensuring its integrity and consistency. The state refers to the collective state of all accounts, including their balances, contract codes, and storage. The Merkle Patricia trie structure is an adopted and optimized version of Patricia tries, a tree-like data structure used for efficient key-value storage. Nodes in the network can reconstruct the state trie by traversing the transactions within the block and computing the resulting state root. By comparing the computed state root with the one specified in the block header, nodes can, therefore, confirm the accuracy of the state and reach a consensus on its validity. [9] To validate our simulation model, we, therefore, have to ensure that all transactions of the forked block are present and executed in the same order. Minor reordering of transactions would result in a different state root hash. In theory, by generating blocks with identical transactions and state inputs, the simulator should accurately replicate the original chain's state. However, it is crucial to ensure that changes to the block header (extra Data) necessary for creating, i.e., similar blocks, do not affect the system state. This is complicated by specific Ethereum Virtual Machine (EVM) OPCODEs that access data from block headers. Ethereum OPCODEs are the backbone of the EVM, a quasi-Turing-complete virtual machine responsible for executing code on the Ethereum blockchain. Understanding that the system's state is determined by executing all transactions on the previous state is critical. In Ethereum, this includes executing code where the smallest units of computation are these OPCODEs. Smart contracts, initially written in high-level languages like Solidity, are translated into bytecode consisting of OPCODEs. These OPCODEs define the contract's logic and behavior. When deployed, the EVM executes these OPCODEs, handling operations like token transfers and data validation. [9] Smart contracts can access block-related data from the current and previous blocks using specific OPCODEs, such as BLOCKHASH (0x40), NUMBER (0x43), and TIMESTAMP (0x42). [14]

With the knowledge gained, we can outline an approach to validate our fork simulator. The first step involves creating a block replica to verify the accuracy of our block creation process. This step does not involve forking, as it maintains all block data, including its header, and does not alter the blockchain state. Next, we proceed to create an actual fork using our simulator, aiming to produce blocks that closely resemble the original blocks. This is achieved by modifying the *extra Data* field in the block's header, which changes the block hash without affecting the blockchain state, as the *extra Data* field is not accessible by smart contract code. Consequently, we can generate a fork with different block hashes but identical state roots, except in one case. Since our fork simulation generates blocks similar to those on the original network, any transactions that utilize the block hash OPCODE can cause divergence. Thus, we can only ensure that the state

root hash matches the initial forked block and any subsequent blocks, provided that the transactions do not act upon the block hash OPCODE.[14]
To sum it up, we can define a validation model for our presented fork simulator: The block number has to be an increment of its predecessors' block number. The *extra Data* field should have been changed to acquire a different block hash. The parent hash should also be adjusted accordingly if the fork level is deeper than n = 1. All transactions of the original block should be included in the correct order to have the same state root hash. Although this might seem to be a reasonable framework to verify our simulation, we experienced multiple limitations in our experiment that need to be observed and analyzed.

7.2. Experiments

As already mentioned, we are testing our fork simulation on both the Ethereum 2.0 mainnet and the Sepolia Testnet, which both utilize the new Proof-of-Stake consensus mechanism and include both an execution client and a consensus client. In the following section, we will present a detailed experiment conducted on the Sepolia Testnet. This experiment involves simulating a fork until we produce a block whose state root differs from its counterpart on the original chain.

7.2.1. Ethereum 2.0 Testnet

We simulated a blockchain fork at block height 5557840 in the Sepolia testnet. At fork depth *m=1*, for block number $n = 5557840$, the state root of block B is $0x25f8a6f0b06e7a9005e150c290fe3a943e3c5148367991f5df8b72661eca1bd9$, which matches the state root of its forked block B'. The block hash of block B is $0x7411ca94a6d9b4e8bb99fc900e18267521f9c6ba039b3c6c2a904ff0a6cc3a87$, whereas the block hash of its forked block B' is $0x37819c1abdb92b720c7d55a118a627311a4112ab48afaadb5c31b37f1fe8b17b$. At fork depth *m=2*, for block number $n = 5557841$, the state root of block B is $0x20d06bb8dfdaeb3b29844a01291ca6903232fdf9937dd81e765e106791e7c35d$, which matches the state root of its forked block B'. The block hash of block B is $0x641da45356f7b57253f2ce30a252df8d54b1185ac9dfeb5fd3d094b051c9c210$, whereas the block hash of its forked block B' is $0xcc89afa0e489d145c2b1c1da8f009a440f425d30f0667af400ee7468aafc30ab$.

Block Number	Original Block Hash	Original Block State Root	Forked Block Hash	Forked Block State Root
5557840	0x7411...	0x25f8...	0x3781...	0x25f8...
5557841	0x641d...	0x20d0...	0xcc89...	0x20d0...
5557842	0x2c2e...	0xdbdb...	0x2da0...	0xc8b2...

Table 7.1.: Comparing the block hash and state root of a block sequence in the sepolia test network and a fork we simulated at block height 5557840 with a fork depth of 3

At block height 5,557,842, the state root of the forked block differs from its original

counterpart. After closely analyzing this block, we discovered that, as described earlier, a transaction acted upon the OPCODE BLOCKHASH 0x40, leading to this divergence. Precisly, it was transaction *0x5feb216b5c973bd59981bdbe3114d53deb09257e6908654bb11f00f408ef4a1f* [1] acting upon the OPCODE.

7.3. Oberservations

In this section, we explain what concepts of transactions and smart contracts may or may not alter the block structure or influence the blockchain state further.

7.3.1. Transaction Nonce

A nonce is a sequential number tied to an account that is used to track the order of transactions, serving as a crucial security feature in Ethereum's blockchain. It prevents replay attacks, ensures the correct order of transaction execution, and functions as a transaction counter. Each transaction from an account must have a unique nonce, and transactions are executed in strict sequential order based on this counter. A transaction with an invalid nonce will be excluded from any block, protecting the network's integrity. [9] During our fork simulator's development and testing phase, we encountered several anomalies related to nonce handling, underscoring this feature's importance. Ethereum's protocol requires that transactions are validated before being included in a block, which includes checking the nonce value. When the nonce is out of sequence, the transaction is rejected. During our experiments, we tried to create a block after skipping several blocks from the original chain, only to find that these blocks were almost empty of transactions. After a thorough debugging process, we identified that nonce mismatches were the cause. Our custom execution client was passed a set of RLP-encoded transactions directly, bypassing the usual transaction selection process from the mempool. During pre-processing, the EL identified nonce mismatches, which excluded those transactions and resulted in underfilled blocks.

7.3.2. Problematic OPCODEs for the Fork Simulation

During our experiments, we found that identifying a sequence of blocks without transactions that used the BLOCKHASH opcode during execution was both challenging and time-consuming. We traced all transactions, including internal transactions, across various random block ranges, searching for blocks where BLOCKHASH was not invoked. However, sequences of more than two blocks without BLOCKHASH were rarely encountered. Consequently, it was difficult to demonstrate a sequence of similar blocks with identical state roots to the original chain.

[1] https://sepolia.etherscan.io/tx/0x5feb216b5c973bd59981bdbe3114d53deb09257e6908654bb11f00f408ef4a1f

7.4. Discussion of the Validation Process

While a reasonable starting point, the validation approach we adopted has limitations. Specifically, our method is limited to validating blocks based solely on their state root, which either confirms complete equivalence or indicates divergence. A more rigorous approach would involve analyzing and comparing the Merkle Patricia tries of the original and forked blocks to trace differences in the state roots to the precise changes within the trie structure. Conducting such an analysis would require considerable effort, as there are no readily available tools or well-established algorithms in the literature for comparing the similarity between Merkle Patricia tries. The time and resources required to develop and validate this capability were not feasible within this thesis. Thus, we leave this as a potential area for future research. Another potential validation method involves using empirical data from real-world forks to evaluate the accuracy of our fork simulator. For instance, we could replay significant historical forks, such as the deep fork that occurred during 'The Merge,' by feeding the same block data into our simulator. By doing so, we can compare the state roots produced by our simulator with those recorded on the actual blockchain during these events. This would provide an empirical basis for evaluating the simulator's accuracy, though even this approach must account for the impact of opcodes like BLOCKHASH, which could lead to differences in the resulting state roots.

8. Future Work

In this chapter, we will critically examine the limitations of the proposed fork simulator, identify areas that remain unexplored, and outline potential further investigations. Forks are a complex challenge that demand careful attention and innovative solutions. Despite the progress made in this thesis, numerous aspects of the problem remain unaddressed, indicating the need for continued research and development in this area.

8.1. Limitations

Our proposed fork simulator was designed to work primarily with a specific execution and consensus client combination, Besu and Teku. Therefore it would be interesting to explore the extent of modifications necessary to adapt our simulator for other Ethereum clients, as differences may exist that do not manifest in regular operation but could become apparent during testing. Moreover, we have yet to examine whether our approach could be applied to blockchains that build upon the Ethereum codebase but diverge in protocol or client implementation. While these blockchains theoretically adhere to the same foundational specifications, there may be unique characteristics or variations in their execution and consensus mechanisms that could impact the effectiveness of our simulator. Testing our simulator with these alternative blockchain protocols would provide valuable insights into its versatility and potential limitations.

8.2. Open Research Questions

Several considerations should be addressed in future work to enhance the utility and applicability of our proposed fork simulator. One critical aspect is further validating our simulation model to ensure its accuracy. This could involve inputting actual fork data into the simulator to assess whether the simulated fork aligns with the expected outcomes. Additionally, it is crucial to develop a robust mechanism for efficiently identifying and quantifying state divergences between forks and the mainnet. While detecting differences in the state root is relatively straightforward, the framework should be capable of quantifying these differences using various metrics. This would provide a more detailed understanding of the extent and nature of the divergences, enabling more precise analysis and comparison of fork states with the mainnet.
As thoroughly discussed in Section 3.6, the role and behavior of the former coin exchange FTX, as well as the activity of MEV bots during forks, should be critically examined. Although we have conducted an extensive analysis of the blocks and transactions that emerged from the PoW Merge chain, the absence of certain transactions in the main

chain requires further investigation. This task is notably time-consuming, and we were only able to examine transactions on a random basis. Given the potential reasons for transaction invalidity in a fork, as outlined in Section 2.3, a promising direction for future research could be identifying specific transaction patterns.
A point we have not addressed is the impact of forks on Layer 2 (L2) blockchains concerning transaction finality. L2 solutions for Ethereum are secondary frameworks built on top of the main Ethereum blockchain to enhance scalability and efficiency. [9] Although there are various types of L2 solutions, scientific research on the effects of forks on L2 transaction finality is lacking.

9. Conclusion

This thesis addressed several critical research questions concerning forks in the Ethereum blockchain. We examined the potential impact of deep forks, revealing how they can immensely alter the blockchain state, particularly during the deep fork we named the PoW Merge Fork that occurred with Ethereum's transition from Proof-of-Work to Proof-of-Stake. The empirical analysis of the PoW Merge Fork, more precisely, the examination of the 28 monitored blocks, was the first extensive analysis of a deep fork to our knowledge. Our literature review assessed the state-of-the-art research on fork behavior, identifying gaps such as the limited discussion on state divergence and transaction finality and highlighting various potential applications for a blockchain fork simulator. As mentioned, we did not find empirical analyses of deep forks, which may be attributed to the challenges of obtaining fork data, necessitating active and ongoing monitoring.

The core of our thesis is the implementation of a fork simulator designed to work with real-world data. This required an in-depth understanding of execution clients to replicate the block creation process accurately. Our implementation evolved through multiple iterations, beginning with a preliminary approach that involved generating random blocks, followed by developing our own *fork client* and making adjustments to the existing execution client to enable the replication of blocks at any height within the chain.

To validate our fork simulation model, we conducted tests under various conditions, ensuring the accuracy of the simulated blocks. We compared these similar blocks against their corresponding counterparts on the main chain to verify correctness.

Lastly, we discussed the limitations of our proposed fork simulator, including its applicability with different execution and consensus client combinations other than Teku and Besu. Additionally, we identified specific research gaps, like the impact of forks on the base layer on layer 2 protocols.

Bibliography

[1] R. Nourmohammadi and K. Zhang, "Modeling the fork probability of blockchains: Did eip-1559 improve ethereum?," in *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)*, pp. 33–40, 2022.

[2] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013 Proceedings*, pp. 1–10, Sept. 2013. ISSN: 2161-3567.

[3] A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottenbelt, "A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice," in *Financial Cryptography and Data Security* (A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 31–42, Springer, 2019.

[4] A. M. Antonopoulos, *Mastering Ethereum : building smart contracts and DApps*. Beijing Boston Farnham Sebastobol Tokyo: O'Reilly, first edition. ed., 2018.

[5] P. McCorry, E. Heilman, and A. Miller, "Atomically Trading with Roger: Gambling on the Success of a Hardfork," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology* (J. Garcia-Alfaro, G. Navarro-Arribas, H. Hartenstein, and J. Herrera-Joancomartí, eds.), Lecture Notes in Computer Science, (Cham), pp. 334–353, Springer International Publishing, 2017.

[6] E. Shi, "Analysis of Deterministic Longest-Chain Protocols," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pp. 122–12213, June 2019. ISSN: 2374-8303.

[7] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol: Analysis and Applications," 2014. Publication info: A major revision of an IACR publication in EUROCRYPT 2015.

[8] A. Judmayer, N. Stifter, P. Schindler, and E. Weippl, "Estimating (Miner) Extractable Value is Hard, Let's Go Shopping!," 2021. Publication info: Preprint. MINOR revision.

[9] "Ethereum." `https://ethereum.org/en/`. Accessed: 2023-01-09.

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," May 2009.

[11] J. Garay and A. Kiayias, "SoK: A Consensus Taxonomy in the Blockchain Era."

[12] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2567–2572, Oct. 2017.

[13] S. Tikhomirov, "Ethereum: State of Knowledge and Research Perspectives," in *Foundations and Practice of Security* (A. Imine, J. M. Fernandez, J.-Y. Marion, L. Logrippo, and J. Garcia-Alfaro, eds.), (Cham), pp. 206–221, Springer International Publishing, 2018.

[14] N. Stifter, A. Judmayer, P. Schindler, and E. Weippl, "Opportunistic Algorithmic Double-Spending: How I learned to stop worrying and hedge the Fork."

[15] D. G. Wood, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER,"

[16] V. Gatteschi, F. Lamberti, and C. Demartini, "Blockchain Technology Use Cases," in *Advanced Applications of Blockchain Technology* (S. Kim and G. C. Deka, eds.), Studies in Big Data, pp. 91–114, Singapore: Springer, 2020.

[17] "The Ethereum merge explained: what's all the fuss about?." https://theface.com/life/ethereum-merge-explained-2.0-ether-cyrptocurrency-bitcoin-web3, Sept. 2022.

[18] "The Merge Countdown & Node Tracker - ethernodes.org - The Ethereum Network & Node Explorer." https://web.archive.org/web/20220915060652/https://ethernodes.org/merge, Sept. 2022.

[19] D. . S. Lutz, "The Ethereum Merge Is Almost Here. What Could Go Wrong?." https://decrypt.co/108791/ethereum-merge-almost-here-what-could-go-wrong, Sept. 2022. Section: Deep Dives.

[20] "Erigon | Efficient Ethereum Client." https://erigon.tech/, June 2020.

[21] "Project Jupyter Documentation — Jupyter Documentation 4.1.1 alpha documentation." `https://docs.jupyter.org/en/latest/`.

[22] A. O. Mahony and E. Popovici, "A systematic review of blockchain hardware acceleration architectures," in *2019 30th Irish Signals and Systems Conference (ISSC)*, pp. 1–6, 2019.

[23] J. Kubal and L. Kristoufek, "Exploring the relationship between Bitcoin price and network's hashrate within endogenous system," *International Review of Financial Analysis*, vol. 84, p. 102375, Nov. 2022.

[24] V. Dhillon, D. Metcalf, and M. Hooper, *The DAO Hacked*, pp. 67–78. 11 2017.

[25] "coinwarz." `https://www.coinwarz.com`. Accessed: 2023-01-09.

[26] "Why Ergo - ErgoDocs." `https://docs.ergoplatform.com/dev/protocol/why/`.

[27] "2miners." `https://2miners.com/erg-network-hashrate`. Accessed: 2023-01-09.

[28] "ETHW - EthereumPoW (ETHW) Official." `https://ethereumpow.org/`.

[29] L. Kiffer, D. Levin, and A. Mislove, "Stick a fork in it: Analyzing the ethereum network partition," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, (New York, NY, USA), p. 94–100, Association for Computing Machinery, 2017.

[30] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, jun 2020.

[31] "Home - Ethermine - Ethereum (ETH) mining pool." https://ethermine.org/.

[32] "Bitcoin Mining Pool | Bitcoin Mining Contracts | Crypto Mining Pool | Binance." https://pool.binance.com/en.

[33] A. P. Ozisik, G. Bissias, and B. N. Levine, "Estimation of Miner Hash Rates and Consensus on Blockchains,"

[34] F. Reid and M. Harrigan, "An Analysis of Anonymity in the Bitcoin System," in *Security and Privacy in Social Networks* (Y. Altshuler, Y. Elovici, A. B. Cremers, N. Aharony, and A. Pentland, eds.), pp. 197–223, New York, NY: Springer, 2013.

[35] F. Victor, "Address Clustering Heuristics for Ethereum," in *Financial Cryptography and Data Security* (J. Bonneau and N. Heninger, eds.), Lecture Notes in Computer Science, (Cham), pp. 617–633, Springer International Publishing, 2020.

[36] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," *CoRR*, vol. abs/1904.05234, 2019.

[37] S. Yang, F. Zhang, K. Huang, X. Chen, Y. Yang, and F. Zhu, "Sok: Mev countermeasures: Theory and practice," 2023.

[38] "Built-in tracers." https://geth.ethereum.org/docs/developers/evm-tracing/built-in-tracers.

[39] A. Maitz, *Ethereum Attack Simulator. A Discrete Event Simulation Engine for Attacks against the Ethereum Blockchain.* Thesis, Wien, 2023. Accepted: 2023-01-16T12:09:12Z.

[40] A. Maria, "Introduction to modeling and simulation," in *Proceedings of the 29th conference on Winter simulation*, WSC '97, (USA), pp. 7–13, IEEE Computer Society, Dec. 1997.

[41] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation (3rd Edition)*. Prentice Hall, 3 ed., 2000.

[42] R. Paulavičius, S. Grigaitis, and E. Filatovas, "A Systematic Review and Empirical Analysis of Blockchain Simulators," *IEEE Access*, vol. PP, pp. 1–1, Mar. 2021.

[43] C. Faria and M. Correia, "BlockSim: Blockchain Simulator," in *2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 439–446, July 2019.

[44] M. Alharby and A. van Moorsel, "BlockSim: A Simulation Framework for Blockchain Systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, pp. 135–138, Jan. 2019.

[45] S. M. H. Bamakan, A. Motavali, and A. Babaei Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Systems with Applications*, vol. 154, p. 113385, 2020.

[46] C. Mohan, "State of Public and Private Blockchains: Myths and Reality," in *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, (New York, NY, USA), pp. 404–411, Association for Computing Machinery, June 2019.

[47] S. Solat, P. Calvez, and F. Naït-Abdesselam, "Permissioned vs. Permissionless Blockchain: How and Why There Is Only One Right Choice," *Journal of Software*, vol. 16, pp. 95–106, Dec. 2020.

[48] E. Anceaume, A. Pozzo, T. Rieutord, and S. Tucci-Piergiovanni, "On Finality in Blockchains," Dec. 2020. arXiv:2012.10172 [cs].

[49] S. Sankagiri, X. Wang, S. Kannan, and P. Viswanath, "Blockchain cap theorem allows user-dependent adaptivity and finality," 2021.

[50] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[51] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma," Feb. 2021. arXiv:2009.04987 [cs].

[52] N. C. K. Yiu, "An Overview of Forks and Coordination in Blockchain Development," 2021. arXiv:2102.10006 [cs].

[53] C. Chen, X. Chen, J. Yu, W. Wu, and D. Wu, "Impact of Temporary Fork on the Evolution of Mining Pools in Blockchain Networks: An Evolutionary Game Analysis," *IEEE Transactions on Network Science and Engineering*, vol. 8, pp. 400–418, Jan. 2021. Conference Name: IEEE Transactions on Network Science and Engineering.

[54] F. J. Couto da Silva, S. B. Damsgaard, M. A. Mousing Sorensen, F. Marty, B. Altariqi, E. Chatzigianni, T. K. Madsen, and H. P. Schwefel, "Analysis of Blockchain Forking on an Ethereum Network," in *European Wireless 2019; 25th European Wireless Conference*, pp. 1–6, May 2019.

[55] M. Neuder, D. J. Moroz, R. Rao, and D. C. Parkes, "Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders," Feb. 2021. arXiv:2102.02247 [cs].

[56] Y. Kwon, D. Kim, Y. Son, E. Vasserman, and Y. Kim, "Be Selfish and Avoid Dilemmas: Fork After Withholding (FAW) Attacks on Bitcoin," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 195–209, Association for Computing Machinery, Oct. 2017.

[57] H. Kang, X. Chang, R. Yang, J. Mišić, and V. B. Mišić, "Understanding Selfish Mining in Imperfect Bitcoin and Ethereum Networks With Extended Forks," *IEEE Transactions on Network and Service Management*, vol. 18, pp. 3079–3091, Sept. 2021. Conference Name: IEEE Transactions on Network and Service Management.

[58] S. N. Mighan, J. Mišić, V. B. Mišić, and X. Chang, "An In-Depth Look at Forking-Based Attacks in Ethereum With PoW Consensus," *IEEE Transactions on Network and Service Management*, vol. 21, pp. 507–516, Feb. 2024. Conference Name: IEEE Transactions on Network and Service Management.

[59] "Client Diversity | Ethereum." `https://clientdiversity.org/#switch`.

[60] "Besu documentation." `https://besu.hyperledger.org/`.

[61] "Teku documentation." `https://consensys.net/knowledge-base/ethereum-2/teku/`.

[62] Alchemy, "What are Ethereum commitment levels?."

[63] "Flask documentation." `https://flask.palletsprojects.com/en/2.3.x/`.

[64] "Sepolia Resources." https://sepolia.dev/.

A. Appendix

Number	Block Created	Block Arrived	#Transactions	#Failed	#Success
15537393	2022-09-15 06:42:42	2022-09-15 06:44:15	37	2	35
15537393	2022-09-15 06:42:42	2022-09-15 06:42:50	1	0	1
15537394	2022-09-15 06:42:51	2022-09-15 06:42:53	76	4	72
15537395	2022-09-15 06:42:53	2022-09-15 06:43:07	140	8	132
15537396	2022-09-15 06:43:07	2022-09-15 06:43:35	178	42	136
15537397	2022-09-15 06:43:34	2022-09-15 06:43:47	244	11	233
15537398	2022-09-15 06:43:46	2022-09-15 06:45:10	281	40	241
15537399	2022-09-15 06:45:09	2022-09-15 06:47:04	188	13	175
15537400	2022-09-15 06:47:04	2022-09-15 06:47:05	263	14	249
15537401	2022-09-15 06:47:05	2022-09-15 06:49:18	345	98	247
15537402	2022-09-15 06:49:17	2022-09-15 06:59:13	320	41	279
15537403	2022-09-15 06:59:11	2022-09-15 07:00:09	305	3	302
15537404	2022-09-15 07:00:09	2022-09-15 07:02:13	180	3	177
15537405	2022-09-15 07:02:12	2022-09-15 07:07:11	76	7	69
15537406	2022-09-15 07:07:10	2022-09-15 07:09:31	164	6	158
15537407	2022-09-15 07:09:31	2022-09-15 07:12:28	57	2	55
15537408	2022-09-15 07:12:28	2022-09-15 07:18:31	95	6	89
15537409	2022-09-15 07:18:31	2022-09-15 07:28:49	298	34	264
15537410	2022-09-15 07:28:48	2022-09-15 08:07:48	410	15	395
15537411	2022-09-15 08:07:40	2022-09-15 08:23:01	430	3	427
15537412	2022-09-15 08:23:00	2022-09-15 08:27:46	156	0	156
15537413	2022-09-15 08:27:45	-	185	1	184
15537414	2022-09-15 09:26:51	2022-09-15 10:06:59	84	1	83
15537415	2022-09-15 09:28:37	2022-09-15 10:07:02	123	3	120
15537416	2022-09-15 10:07:00	2022-09-15 11:14:48	408	17	391
15537417	2022-09-15 11:14:48	2022-09-15 11:35:27	182	0	182
15537418	2022-09-15 11:35:26	2022-09-15 11:44:55	271	1	270
15537419	2022-09-15 11:44:52	2022-09-15 12:53:59	368	11	357
15537420	2022-09-15 12:53:58	2022-10-07 17:23:03	843	84	759

Table A.1.: Overview of Block Creation, Propagation, and Transaction Details
This table provides a comprehensive summary of the blocks, including their creation and arrival times, the total number of transactions, and the success and failure rates of these transactions. The data illustrates variations in block propagation delays, particularly for the last block, which exhibited a significant delay in arrival time.

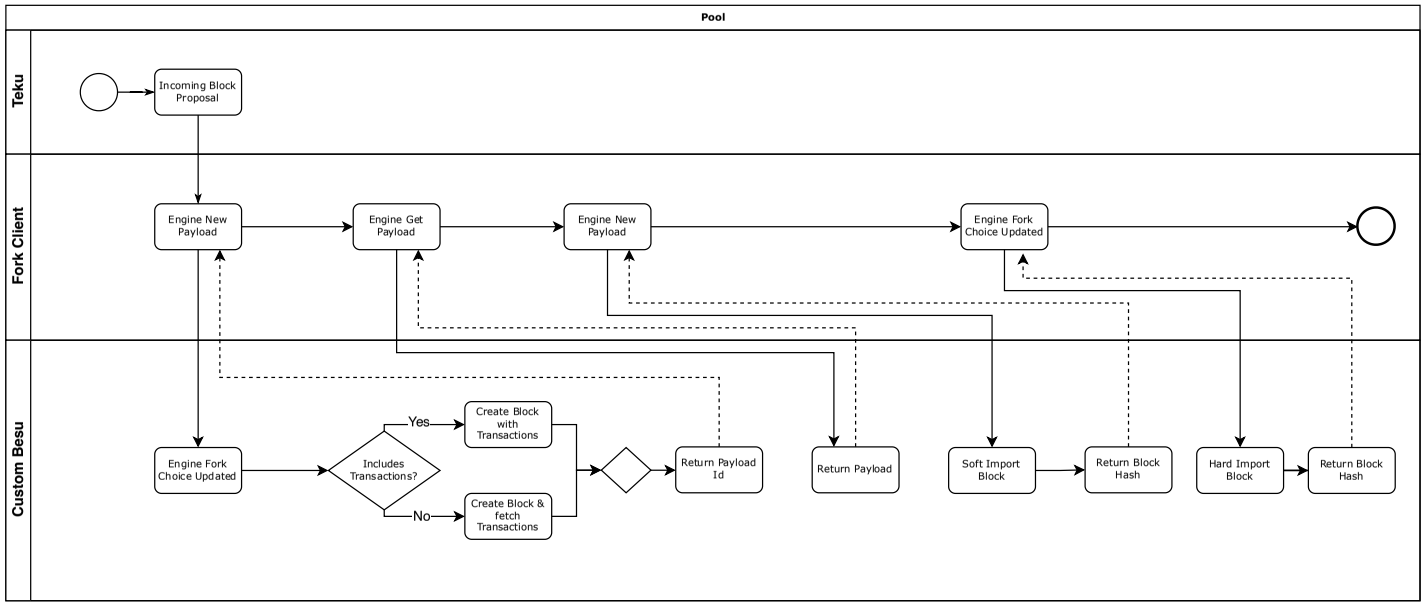


Figure A.1.: Block Creation Process initiated by Consensus and Fork Clients for 1 Block