Blockchain Consensus Layer as a Plug-in
Tools for Realtime Analysis and Monitoring

Guilherme João Bidarra Breia Lopes

Master's disseration
Computer Science and Engineering
(2nd degree cycle)

Supervisor: Prof. Doctor Simão Melo de Sousa

October 2023

Contents

List of Figures

Acronymms

ASIC	Application-specific integrated circuit
BFT	Byzantine Fault Tolerance
DAG	Directed Acyclic Graph
DSL	Domain-Specific Language
FLP	MJ **F**ischer, NA **L**ynch, MS **P**aterson -
PoS	Proof of Stake
PoW	Proof of Work
PoA	Proof of Authority
RPC	Remote Procedure Call
SMR	State Machine Replication
TEE	Trusted Execution Environments

Chapter 1

Introduction

Blockchains are a revolutionary technology that have the potential to transform various industries by providing a secure and transparent platform for storing and exchanging information and assets. Decentralized in nature, blockchains use cryptography and consensus algorithms to create a shared ledger system, eliminating the need for a trusted intermediary.

In recent years, the demand for secure and transparent systems has led to the growing popularity of blockchains, especially in finance, supply chain management, digital identity, and digital assets. The decentralization of blockchains reduces the risk of single points of failure and ensures the integrity of data stored on the network.

Consensus algorithms play a critical role in the functioning of blockchains, determining the process of validating transactions, adding new blocks to the chain, and other crucial aspects like the selection of the next block producer. Different consensus algorithms offer varying levels of security, scalability, and decentralization, making the choice of the right algorithm for a specific use case crucial for the success of a blockchain network.

The main objective of this dissertation is to develop tools and methods to easily swap, test, develop, and compare different consensus algorithms in a live environment using actual nodes on a network. This study aims to provide insight into the field of blockchains and consensus algorithms, contributing to the advancement of more secure, scalable, and efficient blockchain systems. The Tezos [1] blockchain node will serve as a testbed for this project, and the focus will be on examining the potential of consensus algorithms to be swapped and changed on demand. Through this work, we introduce innovative tools that facilitate these processes, setting the stage for future advancements in the field.

This dissertation will provide valuable insights into the field of blockchains and consensus algorithms, contributing to the development of more secure, scalable, and efficient blockchain systems. From tools and methods to test consensus algorithms to the development of algorithms and their integration into existing blockchain nodes, this dissertation will leave a lasting impact on the future of blockchains.

1.1 Motivation

Blockchain technology has rapidly gained popularity in recent years, yet the development and implementation of consensus algorithms for blockchains remains a complex and time-consuming task. The lack of tools and methods for easily swapping and testing different consensus algorithms in a live environment presents a significant barrier to the adoption and evolution of blockchains, limiting the ability of developers to experiment with new and innovative consensus algorithms. This is particularly concerning given the importance of

consensus algorithms in achieving optimal balance between scalability, security, and decentralization, also known as the "blockchain trilemma".

This document aims to address these challenges by exploring the potential for consensus algorithms to be plugged and changed on demand. The study will provide valuable insights into the most effective methods for testing consensus algorithms in a live environment, and contribute to the development of a domain-specific language for describing consensus algorithms. The results of this research will benefit both the academic community and industry stakeholders, who will be able to make informed decisions about which consensus algorithm is best suited for their specific use case.

The objective of this study is to advance the field of blockchains by improving the security, scalability, and decentralization of blockchain networks. The importance of this research cannot be overstated, as the ability to easily swap and test consensus algorithms is critical for the continued growth and evolution of blockchains.

In light of these challenges and opportunities, it becomes evident that there is a pressing need for a solution that simplifies the process of developing, testing, and swapping consensus algorithms. The current landscape is fraught with complexities and limitations that hinder the full potential of blockchain technology. The following section will delve into these problems in greater detail, setting the stage for the proposed solutions and contributions that this research aims to provide.

1.2 The Problem

The development of blockchain technology has led to the creation of a large number of consensus protocols, each with its unique characteristics, from the selection of the leader, to the type of network, data structure, block structure, time between each block, and multiple ways to have a chain, to name just a few. Newer and innovative protocols are designed "everyday" with totally different characteristics compared to previous protocols. There's a lot of "fine tunning" to be done, even when comparing protocols of the same type. These small changes are enough to start a totally different network. For example, just increasing the original block size of Bitcoin was enough to start a new network, Bitcoin Cash [2].

This abundance of choices has made it difficult for individuals or organizations looking to start their own blockchain to determine which protocol is the best fit for their needs, as there's a lot to account for.

Furthermore, there is no easy way to test and compare these consensus protocols in a live, non-simulated environment. There is no blockchain node software that allow for a seamless swap of the consensus algorithm with testing and benchmarking, making it challenging to properly evaluate the performance of each protocol. This presents a major obstacle for researchers and developers looking to experiment with and improve upon existing consensus protocols.

Additionally, the process of designing and coding a new consensus algorithm is also challenging and requires a significant amount of expertise and resources. There is currently no platform that allows for the easy design, swap, and testing of new consensus algorithms.

All these problems contribute to the lack of standardization in the field of consensus protocols, a lot of attributes and characteristic to take in, making it difficult for individuals and organizations to adopt blockchain technology effectively. It is crucial to find a solution that allows for the easy and efficient comparison and testing of consensus protocols, as well as the design and implementation of new protocols. This will pave the way for the further development and widespread adoption of blockchain technology.

1.3 Proposed Solution

The proposed solution aims to tackle the challenges mentioned in the previous section by providing a method of easily swapping consensus algorithms, so they can then be tested. To achieve this, the solution leverages the advantages of a pre-existing blockchain that is suitable for both swapping and testing.

By using an already built blockchain, the focus of this solution is solely on the consensus protocol/layer, the crucial aspect of any blockchain. The need for implementing other parts of the blockchain node such as the Peer-to-Peer layer, client, validation layer, storage of the blockchain state, etc., is eliminated. This benefits the solution by making use of already tested and industrial-grade blockchain software, freeing up time and resources to concentrate solely on the consensus.

One of the ideas for implementing this solution is to use the Tezos blockchain. The reason for using Tezos is its inherent adaptability, which allows for the removal of its current consensus algorithm and the implementation of a new one. This approach provides a layer of adaptability to the Tezos blockchain, making it suitable for testing and swapping various consensus algorithms with ease.

As of the writing of this document, there's no knowledge of projects or documents that describe a similar solution to this one proposed. There exists the concept of consensus testing when talking about a blockchain node implementation in specific, that is, for example, tools to test the Ethereum network, or the Tezos network, but there are no tools to test specifically the consensus part of these projects and compare them, or even, a tool to compare and test consensus protocols in general.

An experiment will be demonstrated to showcase the feasibility and practicality of our objective. This first study is designed to validate our ideas and provide a foundation for future development. While we did not develop a full-fledged platform/toolset, this experiment serves as a crucial first step in demonstrating that there's a potential for consensus algorithms to be easily swapped, tested, and compared in a live environment.

In summary, this thesis presents a comprehensive approach to address the challenges outlined in the problem statement. We have developed tools that facilitate the easy swapping, testing, and comparison of consensus algorithms in a live environment, using the Tezos blockchain as a base. This work serves as a pioneering step in standardizing the evaluation and development of consensus protocols, thereby making blockchain technology more accessible and customizable. The following section will elaborate on the main contributions of this research, detailing the tools and protocols developed to realize this solution.

1.4 Main Contributions

The work presented in this thesis addresses several challenges in the development and testing of blockchain protocols, leading to the following main contributions:

- Development of Bootstrapper: A tool that simplifies the process of creating new blockchain protocols by providing a template-based approach, reducing the amount of code that needs to be written.
- Creation of Live Testing Tool: A tool designed for real-time testing and debugging of blockchain protocols, offering various metrics for performance evaluation.
- Implementation of a Proof of Work Protocol: A fully functional PoW protocol developed from zero, serving as a practical example of the tool's capabilities.
- Implementation of a Proof of Authority Protocol: A PoA protocol developed to showcase the versatility of the Bootstrapper and Live Testing Tool, featuring a round-robin mechanism for validator selection.
- Open Source Availability: All tools and protocols developed are open-source and publicly available, encouraging further research and development in the field.

1.5 Document Organization

MUDAR:

This document is organized as follow:

1. **Core Concepts and State of the Art** - This chapter aims to provide an overview of the Core Concepts necessary to understand the inner workings of a Blockchain network, as well as to explore the most commonly used consensus algorithms and a Domain-Specific Language used that describes this type of algorithms.
2. **Problem Statement** - In this chapter we will describe the problem we are solving and a solution proposed to solve it. In it we will also elaborate on a experiment done to further sustain that the solution is a feasiable one and, to finish the chapter, the tasks for the continuation of the development of this dissertation will be uncovered.

This is the organization of the document, where firstly we describe how this type of system works, how they can differ and then we elaborate on the problem and solution that will be worked on in this dissertation.

Chapter 2

Core Concepts and State of the Art

The consensus mechanism is a critical component of any blockchain network. It determines the rules of the system, like validating transactions and adding them to the ledger. In order to understand the current development and innovation in this area, it is important to have a solid understanding of the core concepts of blockchains as well as the current state of the art in consensus algorithms. This chapter aims to provide an overview of both of these key topics.

2.1 Core Concepts

This section focuses on the fundamental building blocks of decentralized systems like blockchains.

This section covers State Machine Replication (SMR), Blockchain Data Structure and Networks, Network Models, and the Concept of Consensus. Understanding these concepts is crucial for understanding how blockchains networks function in their core, the different trade-offs and design considerations involved in creating and operating a blockchain network. The section starts with an overview of SMR and how it provides consistency and availability in decentralized systems. Then, the concept of Blockchain Data Structure and its application in decentralized networks is explored. The different Network Models in distributed systems, such as synchronous, asynchronous, and partially-synchronous are also discussed. Finally, the section concludes with a exploration of the topic of consensus and it's importance in blockchain networks.

State Machine Replication

State Machine Replication is the concept of replicating data in a distributed system, where nodes in a system maintain the same state of a machine. That is, where multiple entities maintain the same information.

In this approach, a node broadcasts an update to its state using a total order broadcast, and all other nodes receive the same updates in the same order and apply it to their own state.

Total order broadcast refers to notion of broadcasting messages in a specific order, such that all nodes in the network receive the messages in the same order, even if the messages are sent from different nodes at different times.

The main advantages of using SMR, like mentioned before, are that it can provide both **consistency** and **availability** (in some cases it only provides availability), making it a popular choice for use in blockchain technology, databases, file systems, and other decentralized systems.

Consistency, because it guarantees that every node contains the same state of the machine (not necessarily at the same time), and availability is also ensured because, in case of a node failure, there are other nodes in the system that have the information.

Overall, the concept of State Machine Replication is a fundamental component of decentralized systems that plays a crucial role in ensuring the consistency and availability of shared data. In case of Blockchains, to ensure that every node has the same chain and that there's no single point of failure of the whole system. This makes Blockchain Networks reliable, since in case multiple nodes fail in the network, it can still operate, and that no single node has the power over the whole information, making the network decentralized.

Blockchain Data Structure and Networks

Blockchain is a data structure made of blocks that is often compared to a linked list (to which blocks are the nodes of the list), that are connected to a previous block in the chain by a refernce to it, where usually is the hash of the block it is pointing to.

That is, every block, except for the first block (also called "Genesis" block), contains the hash of it's previous (or parent) block.

Once a block is added to the blockchain, it cannot be altered or deleted, making the ledger tamper-resistant and immutable, since, in order to change a block's information, one would need to change every child's block hash. It's also possible to say that, the older the block, the more "immutable" it is, or the bigger the number of child blocks a block has, the harder is the tampering of such block. That is because the hash of a block depends on the hash of the previous block, and so on.

Figure 2.1 illustrates the blockchain datastructure, where it's to possible to observe that the last block (Block N) points to it's parent, and so on. This data structure is often separated by two parts, the header of the block, that contains it's metadata, such as the previous block hash, and the data part, where, in most blockchain networks, this data is where the transactions are stored and visible to everyone that has access to it.

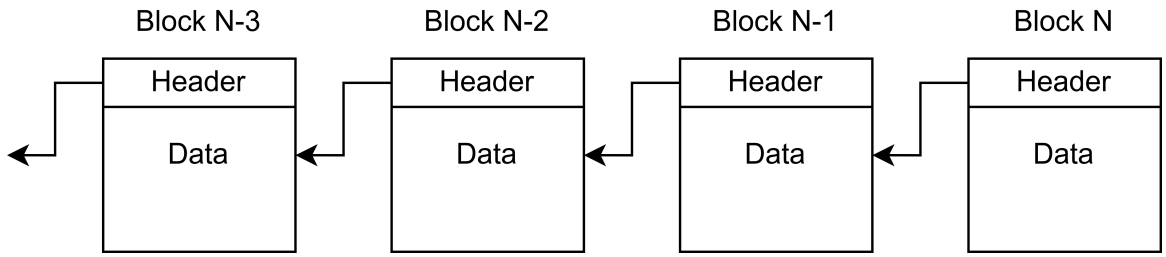

Figure 2.1: Blockchain Datastructure.

Since it is used to record transactions, a blockchain can be considered a "ledger", and it's transparent, as all transactions are recorded.

There are networks, like Bitcoin [3] and Ethereum [4] that store and replicate the blockchain (where the chain is the State Machine), with no central authority or entity controlling the network, like mentioned in the previous section. This makes the state of the blockchain highly available, since there are multiple nodes that contain it's information.

The network reaches "common consent/decision" on the contents of a new block through the use of consensus mechanisms, such as Proof of Work or Proof of Stake, which will be described in the next section (2.2). The blockchain is then a distributed ledger, as all nodes in the network have a copy of the ledger, ensuring availability of the state.

The blockchain can be considered the state of a machine, where multiple copies of the state of this machine are stored in multiple nodes, and newer states, that is, the process of appending a new block to the chain, are replicated to every machine in the network.

Beyond cryptocurrency, blockchains have a wide range of potential applications in various industries, like supply chain management, voting systems, and identity verification. The transparent nature of blockchains ensures that all transactions are publicly accessible and verifiable.

In conclusion, blockchain is a data structure that leverages the concepts of state machine replication to provide availability, leverages the concepts of cryptographic concepts to make it secure, immutable and tamper-resistant ledger where all transactions are recorded, making it transparent. The decentralized nature of blockchains networks and the use of consensus mechanisms ensure that all nodes in the network have the same state of the chain, providing the availability to access of this information.

Network Models

The concept of network types is a fundamental aspect of distributed systems and plays a crucial role in their design and implementation of these systems. There are three main models of networks in distributed systems: synchronous, asynchronous, and partially-synchronous. Each has different properties and characteristics that affect the behavior of distributed algorithms and protocols, and it is important to understand these differences in order to design effective and efficient systems.

Synchronous networks are characterized by the fact that the entities in it have a common notion of time and upper bounds on message delay. That is, in the Synchronous model, a finite time limit Δ is established and known. The adversary can only delay the delivery of a message sent by at most Δ time. This means that the network can guarantee that messages will be delivered within a certain amount of time and that all nodes in the network have a consistent view of the current time. This type of network is relevant when strict timing constraints are necessary, such as in real-time systems.

Asynchronous networks [5], on the other hand, neither have a common notion of time nor guarantee upper bounds on message delay. In this type of network, messages may be delayed indefinitely, and nodes in the network may have a different view of the current time. Asynchronous networks are relevant when timing constraints are not strict. The unpredictability of message delays makes asynchronous networks more challenging to design and implement, but they are also more robust and can handle failures and network partitions more effectively.

Partially-synchronous networks [6] are a hybrid of synchronous and asynchronous models. They have a common notion of time but only provide partial guarantees on message delay. In this type of network, some messages may be delivered within a certain amount of

time, while others may be delayed indefinitely. Partially-synchronous networks are relevant in scenarios where some timing constraints are necessary, but not all and is often used in blockchain systems, such as Tendermint [7](used in Cosmos Network [8]and others), and Tezo's blockchain take on this protocol, Tenderbake [9], which balance the need for speed and reliability with the need for resilience and robustness.

In conclusion, the concept of network types is critical in the design and implementation of distributed systems. Understanding the differences between the three networks can help protocol designers make informed decisions about the type of network that best suits their needs and can ensure the success of their systems.

Consensus

Consensus in Distributed Systems refers to the process of achieving agreement/common consent among all participants in a network on the state of a shared information.

While State Machine Replication is the notion of broadcasting the update of a state and multiple entities maintaining a copy of it, consensus is the concept of how the nodes replicate and decide on a replicated value.

One way to implement total order broadcast, is by sending messages via a designated leader node, but if that leader becomes unavailable, the approach fails.

So, nodes have to reach an agreement together, in a decentralized manner, on what is the next state of the replicated machine (or who's going decide that next state), and should not be decided by a single node.

Consensus protocols are **critical** to the blockchain context because they are the mechanisms that allow the network participants to agree on the state of the distributed ledger. In this type of network, transactions are submitted and processed by nodes in a decentralized manner, and the consensus protocol ensures that all nodes have the same view of the ledger and agree on which transactions are valid and should be included in the next state. The construction of a block is used to make the system more efficient, by "packaging" multiple transactions together, but it's possible to reduce this idea to a single transaction being the update to the state. Also, the algorithms enable the selection of the next leader, that is, the node that takes the transactions, builds the block and broadcasts it to the networks.

This is important for maintaining the integrity and reliability of the blockchain, as well as for enabling trust in the network among participants who may not necessarily trust each other. A well-designed consensus protocol is essential for ensuring that the blockchain is able to process transactions efficiently and securely, even in the face of network failures, attacks, or other challenges.

Theorems

There are several important theorems related to consensus algorithms that are relevant to the context of Blockchain Networks, as these are used to attribute characteristics and properties to this type of networks (not only), which include the **FLP** Theorem [10] and the **CAP** Theorem [11].

The **FLP** Theorem states that, in an asynchronous network where messages may be delayed but not lost, it is impossible to achieve both agreement and termination in the presence of process failures. In other words, it's only possible to guarantee two of the following properties at the same time:

- Finality or Agreement, that is, if (functioning) have to decide on a value, they all decide one specific one.
- Fault Tolerance or Integrity, the system still functions in case of node failures.
- Termination or Liveness, where all the functioning nodes decide on value.

On the other hand, the **CAP** Theorem states that, in an asynchronous network where messages may be lost, it is impossible for a distributed system to simultaneously provide the following:

- Consistency - all nodes see the same data at the same time
- Availability - every request receives a response, without guarantee that it contains the most recent version of the data
- Partition Tolerance - the system continues to operate despite arbitrary partitioning due to network failures.

Figure 4.1 illustrates these theorems/propreties, where it's possible to observe that the area where the 3 parts meet it's impossible to achieve.

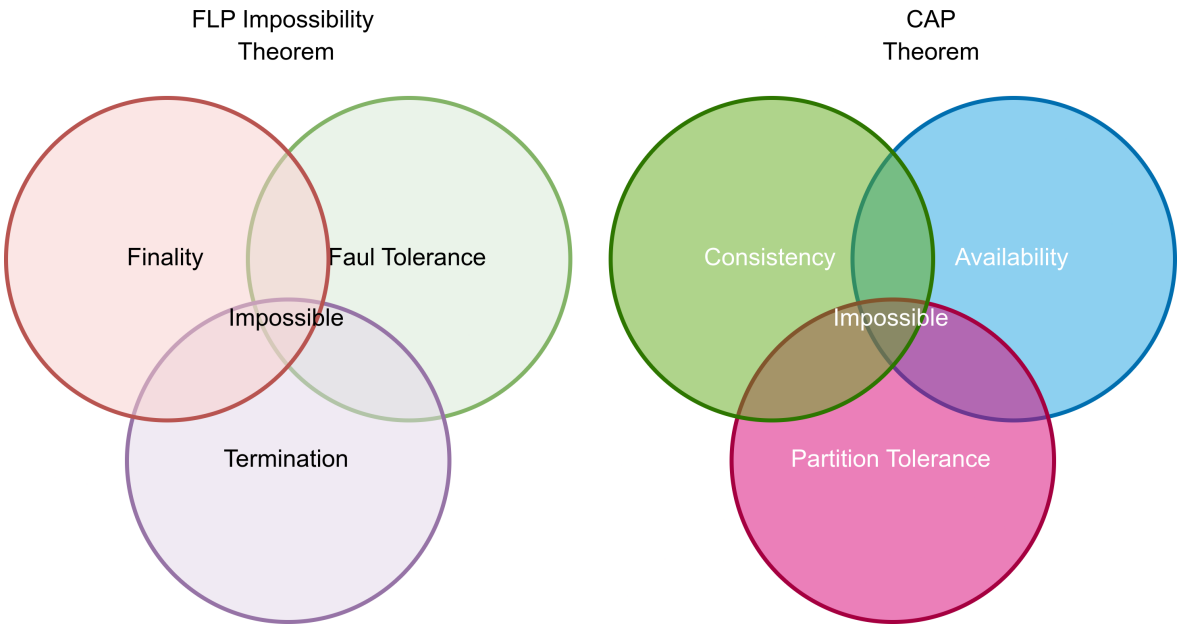

Figure 2.2: FLP and CAP theorems visualized

These (and other) theorems provide limits and trade-offs in decentralized systems, and serve as a useful reference for designers and researchers in the field. In the following sections, more insight about these properties will be provided and why they are relevant to the topic

of Blockchain networks, such as the fact that networks like Bitcoin chooses Availability over Consistency (while being Partition Tolerant) and chooses Termination over Finality (while being Fault Tolerant and having a Probabilistic Finality).

Permissioned and Permissionless Blockchains

Blockchains are a type of decentralized system that uses consensus algorithms to maintain the integrity of its data. In the context of blockchains, the terms "permissionless" and "permissioned" refer to the way in which participants in the network are allowed to participate in the validation of transactions and the creation of new blocks. These terms are critical in understanding the trade-offs between security, scalability, and decentralization in blockchains. In this section, we will delve deeper into what "permissionless" and "permissioned" mean and the advantages and disadvantages of each approach.

Permissionless blockchains, also known as public blockchains, are open to anyone and anyone can participate as a node. No central authority or entity is in control, making it a completely decentralized system. Permissionless blockchains are the most popular type of network and are most often used for cryptocurrencies such as Bitcoin, Tezos and others, where anyone can participate in the network and validate transactions, contributing to the security and reliability of the network.

Permissioned blockchains, are restricted to a set of trusted participants. Only approved entities are allowed to participate as nodes and validate transactions, making it a partially decentralized system. This type of blockchain is often used in business environments where the participants are known and trusted. Usually this networks are faster and more secure (yet, lack decentralization), since the number of participating nodes is way smaller and the fact that nodes are "handpicked", they are already trusted.

Known examples are Hyperledger-Fabric [12], an enterprise-grade network and Binance Smart Chain [13], a cryptocurrency network that's a fork of Ethereum.

Processes of Block creation

In the context of blockchain networks, there's a need for nodes selecting the next update to the state, or the next block to add to the blockchain structure. At each round/heartbeat of the network, a proposer decides on the structure of the block (in a cryptocurrency blockchain, the proposer selects the transactions to include in the block).

In this subsection, we will explore the two main ways of reaching consensus/selecting a leader in blockchain networks: proof-based consensus and committee-based consensus.

Proof-based consensus protocols rely on the concept of proof of leadership. Nodes are selected to generate new blocks through a cryptographic random algorithm. The selection of the new leader/proposer is similar to the idea of a lottery. The node that wins such lottery has the right to propose a new block. Nodes validate transactions, generate a Merkle Tree Root of these, and package them into a new block. The leader node broadcasts the new block and its proof of leadership to the network, that validates the new block, and upon validation, it appends it to the blockchain.

In **committee-based consensus** protocols, nodes vote to decide the next block to be appended to the blockchain. The proposer multicasts a preparation block request to other participants, who reply with their status. If the proposer receives a sufficient number of ready messages, it enters the pre-commit phase. Participants then broadcast their votes to commit the proposed block. If the number of commit responses agreeing to the new block exceeds a threshold, the block is appended to the blockchain.

Blockchain "trilemma"

Another subject that is usually used to compare different blockchain networks, with much discussion, is the topic of the "Blockchain trilemma" [14] that refers to the trade-off between scalability, security, and decentralization. This topic was first introduced by Ethereum's founder *Vitalik Buterin*.

In other words, it is hard, if not impossible to achieve all these three characteristics at the "same time" in a public/permissionless blockchain network:

- **Decentralization**: refers to the distribution of power and decision-making authority across all participants in the network. In blockchain, this means that there is no central authority or single point of control, allowing for a more equal and democratic system.

- **Security**: refers to the robustness and reliability of the network, ensuring that data and transactions are protected from tampering, hacking, or other forms of malicious activity.

- **Scalability**: refers to the ability of the network to handle a growing number of transactions and users, without sacrificing performance or speed.

Even though that in the context o distributed systems the topic of decentralization isn't relevant, when talking about blockchain networks it's important, as it's one of the main reasons why people use these systems in the first place.

For example, Bitcoin is a highly decentralized and secure blockchain, as there are many nodes (more than 40 thousand [15], as of the writing of this document), yet lacks on scalability, since the creation of a block takes 10min and a transaction takes about 1 to 1.5 hours to complete/be confirmed [16].

Therefore, when designing a blockchain network, it is important to understand the trade-offs between scalability, security, and decentralization and to make trade-offs based on the specific goals and requirements of the network.

Figure 2.3 illustrates this trilemma. It's common to assign blockchain network as a point in the area of this triangle. For example, since Bitcoin is secure and decentralized, yet lacks scalability, one could say that it stands near the line between "Secure" and "Decentralized".

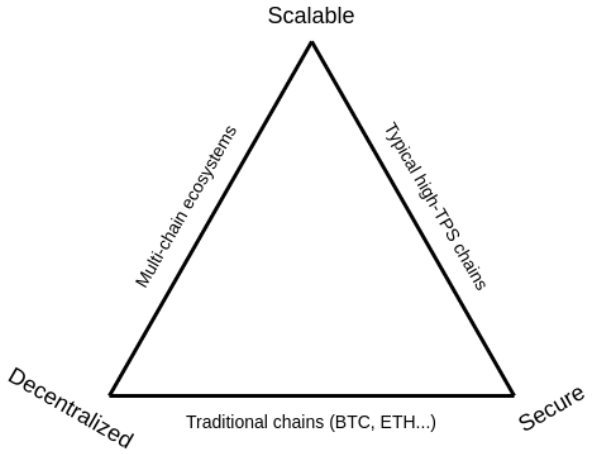

Figure 2.3: Blockchain trilemma

2.2 State of the Art

In the world of blockchain, consensus algorithms play a crucial role in maintaining the integrity and security of the network. They are the backbone of a blockchain's ability to ensure that all nodes agree on the state of the network, even in the presence of malicious actors, and they are of big importance not only in the matter of ensuring the continuation of the network, but they also play a political and philosophical role in the network. As blockchain technology continues to evolve, so do the consensus algorithms used to secure them. Not only they differ on the question of "Blockchain trilemma", but each blockchain network has specific characteristics and functionalities that make it unique.

In this State of the Art section, we will delve into the most commonly used consensus algorithms in blockchain technology. We will examine the different strengths and weaknesses of each algorithm and discuss their suitability for different use cases and will also explore milestone and popular protocols.

This will further expose that there's way too many blockchain consensus protocols to choose from when creating a new network. The reader can obtain additional information about the topics covered in this section at the reference [17] and [18].

Proof of Work

Proof of work (PoW) [19] was originally introduced as a solution to the problem of spamming and abuse on the Internet. In a proof of work system, a computational problem is designed such that it requires a significant amount of computational effort to solve, but its solution can be easily verified. The idea is that, by requiring a party to perform a certain amount of computational work in order to perform a certain action, such as sending an email or posting a message, the system can ensure that the action is performed in a responsible and controlled manner.

The first use of proof of work was in the context of a system for fighting email spam, designed by computer scientist Adam Back in 1997. In this system, an email sender was

required to perform a certain amount of computational work, in the form of solving a cryptographic puzzle, to send an email. The idea was that the cost of performing the work would make it unfeasible for spammers to send large quantities of unsolicited email, while still allowing legitimate users to send email without significant difficulty.

Proof of Work was later adapted as a consensus algorithm used by many blockchain networks, Bitcoin [3] being the most popular network to do so. Bitcoin or "Nakamoto Consensus" was also the first instance of the Proof of Work Consensus and of a blockchain network.

The idea behind PoW (applied to blockchain networks) is that, in order for a block (containing a list of transactions) to be added to the blockchain (and replicated), at least one node in the network has to package together a list of transactions into a block and solve a cryptographic hard problem in order for it to be valid. That is, the block has to contain a stamp or a proof that actual computational work was put in to create a block. That work/computational power is provided by nodes called "miners", and these miners compete to be the first to solve the cryptographic puzzle, where the first to solve it is selected as the leader and has the ability to add the their created block to the blockchain.

In a permissionless environment, it's complex to make a democratic system where each entity could cast a vote, since a malicious entity could create many fake identities or nodes to manipulate the network, known as a "sybil attack". In blockchain networks, entities make use of Public-key cryptography to be identified, so a malicious entity could create as many pairs of Public-Private keys as it wished to cast as many votes as it wanted.

The "Nakamoto Consensus" (where PoW is used) was the first to be tolerant to sybil attacks in a permissionless setting. It prevents these attacks by making it computationally expensive for an attacker to try to disrupt the network, since, instead of voting power being concentrated on the number of votes for an election, it's replaced by the idea of computational power, where an attacker would have to do as much work as the rest of the network (assuming that it's not malicious) in order to gain a majority, and that could imply problems to the malicious entity, like large electricity costs in case of mining. By requiring a significant amount of computational power to mine blocks, PoW ensures that only legitimate nodes with real computational resources will be able to participate in the network. Mining is what is called coequally to the process of using computational power to solve cryptographic puzzles and adding blocks to the blockchain.

In other words, instead of entities voting for a block to be added to the blockchain, the "vote" is done by solving puzzles, where the first to solve such puzzle is the one deciding the block to be added. Another analogy can be that the winner of the "lottery" get's the prize of deciding the next block (solving the puzzle is actually a random chance process, where increasing the mining power also increases the chance of winning).

Mining has several purposes in PoW, such as leader selection and preventing sybil attacks, like mentioned before, but also enforcing block timing with difficulty adjustment and, in case of cryptocurrency blockchains, to add new value/mint new currency into the system and rewarding those who do the computational work.

The difficulty adjustment ensures that the time between blocks is consistent (in Bitcoin, that is 10min), and the computational difficulty to mine new blocks adjusts accordingly.

When in a epoch (in Bitcoin, that is 2016 blocks), the median time taken to mine a block was lower than it should've been, the difficulty is increased proportionally to make it harder to mine, so it actually takes the pretended time, and vice versa.

The mining also introduces/mints new currency into the system, since itself doesn't have any value in it initially (it's a closed economic system, no more currency can be put into it), and that currency is awarded to the participants of the mining process.

Properties of Proof of Work Consensus

The properties of PoW consensus can be analyzed through the FLP theorem and the CAP theorem, which were discussed in the previous section.

Regarding the FLP theorem, PoW exhibits Probabilistic Finality [20]. This means that all functioning nodes eventually agree on a single block, but the process is not deterministic and involves a certain degree of probability. The waiting for confirmations adds an additional layer of security to the consensus process. Like mentioned before in this example (2.1) in Bitcoin this takes about 6 confirmations, or about 1 hour, since a block usually takes 10 minutes to created. This happens because mining is a process independent from the state of the network and independent from other nodes, and two or more nodes can solve the cryptographic puzzle/build a block "at the same time" (until the whole network agrees on the same block). When this happens, a node can receive more than one valid candidate extending the same chain, and each protocol handles this in a specific way. This is what is called a "Fork". Once again, taking Bitcoin as an example, a node handles this case by using the rule of the longest chain (Longest Chain Rule), where, when a node has two competing blocks, where each makes a branch, it waits until one of the branch is bigger than the other. This is done because, the branch with the most computational power is the one with the bigger chance of being extended.

PoW also demonstrates Fault Tolerance, making it a form of Byzantine Fault Tolerance (BFT). This is achieved through the complex and expensive mining process, which makes it economically infeasible for a malicious node to alter the data in a block and catch up with the rest of the network. The incentive to follow the longest blockchain also adds to the fault tolerance of the consensus mechanism.

Finally, PoW also exhibits Termination, meaning that every functioning node reaches a decision, even if it's not the final one (because of the Probabilistic Finality).

Regarding the CAP theorem, PoW may not provide Consistent results, as a node might not have the latest block when requested, and like mentioned before, forks can happen, and so different nodes can contain different chains (for a period). However, it does provide Availability, as miners are continuously trying to mine new blocks, the difficulty level of mining is adjusted to ensure the steady addition of blocks to the blockchain and there are multiple nodes in the network. PoW is also Partition Tolerant, meaning that even if a portion of the nodes stop functioning or the network splits, the blockchain can still function and recover.

Nakamoto Consensus

In the previous subsection, we discussed the consensus mechanism used in blockchain technology where participants in the network compete to solve complex mathematical puzzles in order to validate transactions and generate new blocks. We introduced the concept of miners, who play a crucial role in this process, and how they are incentivized through rewards for their efforts.

Building on this foundation, we now delve into the specifics of the Nakamoto Consensus, named after the pseudonym used by the unknown creator(s) of Bitcoin (Satoshi Nakamoto). The Nakamoto Consensus is a milestone in the history of blockchain technology and a major innovation in the field of consensus algorithms.

The Bitcoin network is a Peer-to-Peer network, where blocks and transactions are gossiped by the nodes and where they are also responsible for generating blocks, just like mentioned previously.

This acts as a barrier to entry for malicious actors, as it becomes extremely difficult for them to launch Sybil attacks by generating many fake nodes. The puzzle in Bitcoin involves computing a nonce (basically, a number) that meets certain conditions, as outlined by the equation $H(hi1, nonce, tx, par) < Target$, where "hi–1" is the hash of the previous block, "tx" represents the set of validated transactions, and ''par" represents other parameters such as blockchain version and cryptographic parameters, and the "Target" is a value decided by the network (that encoded the current difficulty).

The usual workflow of a node is the following:

- Node gossip transactions in the network (They are temporally stored in a data structure that is called "Mempool").
- When a node receives a block from the network, it appends it to it's own blockchain stored.
- To start the mining process, a node selects multiple transactions for the transaction part of the block. There's a size limit in bytes for this part, and the selection criteria is irrelevant, they only have to be valid.
- It creates a new block header with information, like, the hash of the previous block, a nonce (a integer number) and a Merkle Tree Root Hash of the transactions and other information (that is irrelevant to this).
- The node tries out every possible value for the nonce until the resulting hash (SHA256 hash function) of the header has a value bellow the current target.
- If a node receives a new block from the network before it finds itself the nonce, it stops the mining and appends such block. If it finds the nonce, it shares the block with the network and it's rewarded.

When a entity creates a transactions, it can also offer a fee. This is used to prevent transactions spamming and also acts as a "place in the block" auction. Besides block creation reward,

the miner also receives these transactions fees, so miners will select transactions based on that fee, and other attributes of the transactions.

The difficulty/target of the puzzle is adjusted periodically (every 2016 blocks like mentioned before), depending on the actual block generation intervals and the expected block generation intervals (of around ten minutes in Bitcoin). This allows Bitcoin to maintain a consistent block generation time of around ten minutes, regardless of the computational power in the network.

Improving Proof of Work

The Nakamoto Consensus protocol, despite being the first consensus mechanism for blockchains and widely adopted, has limitations when it comes to scalability, security, and decentralization.

There are multiple ways of improving the Nakamoto Consensus protocol, and doing so has resulted in the creation of new blockchain networks. These changes move the consensus protocol within the triangle formed by the "trilemma", but they also come with new challenges compared to the original consensus for blockchains.

This topic is relevant to this dissertation since it's these characteristics that differentiate consensus even further.

Improving Scalability

Blockchain networks have faced scalability issues since their inception. A consensus mechanism must maintain security and decentralization, so it's challenging to scale the network and improve the transaction processing speed. To overcome this challenge, several solutions have been proposed, including decoupling blockchain functions, parallel chains, and DAG-based protocols.

Decoupling of Blockchain Functions: The functions of blocks in a blockchain network can be separated into key blocks for leader election and microblocks for transaction packing. By doing so, the miner who successfully solves the puzzle becomes the leader of an epoch and generates key blocks and microblocks. This method helps to improve the throughput of the network, but it doesn't significantly reduce the transaction confirmation latency. Also, it comes with problems such as the fact that the leader can be compromised during the epoch.

Parallel Chains: The parallel chains method involves miners extending parallel chains simultaneously to improve the blockchain throughput. In this method, miners compete to solve puzzles, and the generated block is appended to one of the chains based on the random hash value of the puzzle solution. While this improves throughput, a malicious entity could still target a single chain.

DAG-based Protocols: instead of using a traditional blockchain structure, DAG-based protocols utilize a tree-like structure, called a Directed Acyclic Graph (DAG). The DAG structure allows for concurrent block generation and operates differently than blockchain structures. Unlike parallel-chain protocols that have multiple independent genesis blocks at ini-

tialization, there is only one genesis block in DAG-based protocols. If the DAG-based blockchain follows the longest chain rule, there will only be one longest chain, instead of multiple chains in the parallel-chain scheme, and blocks on the forks will be pruned. This structure provides a unique approach to improving scalability in blockchain networks

Improving Security

Blockchain networks are vulnerable to various security attacks, such as selfish mining attacks [21] and double-spending attacks [22]. To improve the security of blockchain networks, various solutions have been proposed, including changing the incentive mechanism in the chain and how it's shared.

The FruitChain [23] protocol is a blockchain protocol that improves on this while still providing the same consistency and liveness properties as Nakamoto consensus. With FruitChain, any honest set of nodes controlling a certain fraction of computational power is guaranteed to get at least that fraction of the block generation and rewards in any length segment of the chain, making it harder for coalitions between miners to happen.

Improving Decentralization

To improve decentralization, several solutions have been proposed, including de-incentivizing centralized activities like pool mining and incentivizing decentralized/solo mining using methods like eradicating ASIC (Application-Specific Integrated Circuits) mining.

Pool mining is a point of centralization in Proof of Work networks. The idea of pool mining is that there's a single node (connected in the network) that communicates with multiple miners, that is, entities which their only purpose is to find a nonce. These miners join to mine a single block as if they were only a single node, so they have more computational power together, and inherently having a bigger change of finding a block. When a block if found by the pool (a node, as viewed from the network), the reward is shared with all the miners, depending on how much effort they put into the mining. The problem comes with the fact that most mining nowadays, like in Bitcoin, is done by groups like this, and so, the owners of these pool nodes have a lot of power over the network, as they can make decisions on the block creation, like black-listing transactions from someone, or selecting transactions based on unfair criteria.

Also, with the increase of hashing power, that is, the computational power of blockchains, miners have improved the mining process by developing highly specialized hardware, called ASICs.

By doing so, it's infeasible for anyone (without this type of hardware) to join the mining process, and centralizing the power on the owners of ASICs and the it's manufacturers. So there are multiple efforts to erradicate this type of mining like by using memory-hard puzzles, that are harder for ASIC hardware than for Personal Computer Hardware (like GPUs or CPUs), or using hash functions that are hard to implement as ASICs.

This is the case of Monero [24], that besides providing total privacy (the ledger isn't transparent like bitcoin), is also designed to promote equality between miners by using an egal-

itarian approach based on the use of a different hash function (compared to Bitcoin, that uses SHA256) called “CryptoNight”, which is said to be memory-hard, making it hard for the development of ASIC hardware and also hard for mining using GPUs.

Proof of Stake and Its Variants

Proof of Stake (PoS) has emerged as a popular alternative to Proof of Work (PoW). In PoS, nodes are selected to validate transactions and create new blocks based on their stake in the network. The core idea is similar to PoW but replaces ”computational power” with ”stake,” making it more energy-efficient [**?**].

Chain-based Proof of Stake (ChainPoS): Focuses on incentivizing stakeholders to act honestly by staking assets. It offers two main approaches for block creator selection: randomized stakeholder selection and a hybrid between PoW-like selection and randomized stakeholder selection. Peercoin [25] and Casper FFG [26] are notable examples.

BFT Proof of Stake (BFT PoS): Developed to tackle Byzantine faults in decentralized systems. It operates in a multi-round consensus process involving validator selection and the use of quorums of super-majority. Tendermint and Tenderbake in the Tezos blockchain are examples [27].

Delegated Proof of Stake (DPoS): Combines elements of PoS and democratic governance. It aims to balance decentralization with efficiency by having a smaller number of delegates. Validators are selected based on reputation scores or other metrics, and stakeholders can vote for delegates.

In conclusion, PoS and its variants offer a range of options that balance security, scalability, and efficiency, each with its own set of trade-offs and benefits.

Proof of Authority and Its Variants

Proof of Authority (PoA) is a consensus algorithm commonly used in permissioned blockchain networks, where decentralization is not the primary concern. In PoA, a limited set of pre-approved validators are responsible for block creation and transaction validation. This approach offers a high degree of efficiency and scalability while eliminating the need for resource-intensive calculations. Interestingly, our implementation of PoA has certain similarities with PoW, a topic that will be explored further in this dissertation.

Clique PoA: This variant aims to improve the decentralization aspect of PoA by allowing multiple authorities to propose blocks in a round-robin manner. This reduces the risk of a single point of failure and enhances network security [28].

Aura PoA: Known for its flexibility, it allows validators to take turns in a defined order to create blocks. It also includes a mechanism for validators to communicate and come to a consensus if a block is valid or not [29].

IBFT PoA (Istanbul BFT): A Byzantine-fault-tolerant PoA algorithm that can handle malicious nodes and network splits. It uses a voting mechanism among validators to reach consensus and is known for its robustness [30].

In conclusion, PoA serves as an effective choice for networks where rapid transaction validation is crucial, and where a permissioned structure is acceptable. Its variants offer different ways to balance efficiency with security, making PoA adaptable to various specific needs.

Other Consensus Protocols

Apart from Proof of Work and Proof of Stake, there are other consensus algorithms that are used in the blockchain industry. However, they are not as popular as PoW and PoS and are mainly used in niche applications or not used in the popular cryptocurrency blockchains. In this section, we will explore and explain some of these alternative consensus algorithms.

In this subsection we will enumerate and give a overview of a few of these consensus algorithms.

Proof of Burn [31] shares some characteristics with Chain-based Proof of Stake. In this algorithm, instead of staking, the node sends some tokens to an address that is difficult to access, essentially ”burning” them. The node can then mine or validate blocks proportionate to the amount of tokens that were burned. The idea behind Proof of Burn is that a node is willing to sacrifice tokens, making it less likely that they would act maliciously as they would lose their investment.

Proof of Research [32] algorithm is used in some cryptocurrencies that aim to provide an environmentally friendly alternative to Proof of Work. In this algorithm, nodes perform research tasks that can range from solving mathematical problems to annotating data. The first node to complete the task is allowed to mine the block, providing a solution to the computational waste that is inherent in PoW.

Proof of Capacity/Space/Storage [33] shares some characteristics with Proof of Work. In this algorithm, nodes are required to dedicate a certain amount of disk space for storage. The more disk space a node dedicates, the more chances it has of being selected to mine the next block. The algorithm is energy-efficient compared to PoW, but it still suffers from centralization problems as nodes with more disk space are more likely to be selected to mine blocks. These kinds of protocols are often used in conjunction with other blockchain networks (Proof of Work or Proof of Stake) to store large amounts of information that wouldn't be feasible to store directly in those blockchains.

Trusted Computing [34] algorithms use the secure hardware provided by Trusted Execution Environments (TEEs) to secure the consensus mechanism. One example of such an algorithm is POET, used in the Hyperledger blockchain platform. In POET, a trusted party generates a wait time for each node and the first node to complete the wait time is allowed to mine the next block. The use of TEEs ensures that the wait time cannot be tampered with, providing a secure mechanism for consensus.

There are many alternative consensus algorithms that are used in the blockchain industry, each with its strengths and weaknesses. While some, like Proof of Burn and Proof of Research, offer environmentally friendly alternatives, others, like Proof of Capacity/Space/Storage, offer a totally different concept of proof. Trusted Computing Algorithms, like POET, provide secure consensus mechanisms, but rely on trusted hardware that may not be readily available and making it more centralized, as it relies on specific companies to build this kind

of hardware. Ultimately, the choice of consensus algorithm will depend on the specific use case and requirements of the blockchain platform.

Lupin Language

The Lupin Language is a Domain-Specific-Language (DSL) that is currently in its early stages of development. The process of compilation of this language results in OCaml language code. It is designed to simplify the process of designing and coding consensus algorithms, which can be a challenging task that requires a significant amount of expertise and resources.

Not only that, but it can also be used to perform rigorous formal analysis and verification on these algorithms, bringing a whole new level of capability to analyzing consensus algorithms and will be an important tool in the future.

This project provide will provide feedback for the development of Lupin and may also serve as a layer of execution for the results of its compilation, and is in collaboration with the Lupin Language project and its results will be further exposed in a later section.

2.3 Conclusion

This chapter provides a comprehensive overview of the core concepts of blockchains and the state of the art in consensus algorithms. With so many different consensus protocols available, it is essential to have a clear and concise way to compare and evaluate them. As the field of blockchain continues to evolve, the development of more effective consensus algorithms will be a critical factor in determining the success of blockchain networks. By gaining a deeper understanding of the current state of the art, researchers and developers can work towards developing new consensus algorithms that are more secure, efficient, and scalable than ever before.

Chapter 3

Preliminary Case Study

The Tezos blockchain has proven itself to be a flexible and adaptable platform, capable of accommodating a wide range of applications and use cases. This makes it a suitable choice for an experiment that aims to test and demonstrate the implementation of a consensus protocol. In this chapter, we will elaborate on why Tezos is an ideal fit for this experiment and explore the inner workings of the platform that make it so.

The experiment at hand involves the implementation of a Proof of Work consensus protocol, similar to the one used by the Bitcoin blockchain. As an exercise, this protocol was implemented on the Tezos blockchain and its workings will be discussed in detail. This will include a discussion of the requirements for a protocol, its integration with the Tezos node, the implementation of the protocol itself and the tools used to interact with it.

We will also delve into the implementation of a client and a baker, which are integral components of the protocol. The client is used to access information within the network, while the baker is responsible for the creation and mining of blocks. This includes the process of mining the block by finding the nonce that makes the block's hash lower than the target, which is a key point in the execution of a Nakamoto like consensus.

In conclusion, this experiment will serve as a demonstration of the flexibility and adaptability of the Tezos blockchain and its ability to accommodate different consensus protocols. The implementation of the Proof of Work consensus protocol will serve as a proof of concept and highlights the potential of the platform to support a wider range of consensus protocols, even those different from the orginal protocol implemented.

3.1 Tezos Blockchain as a tool for the Experiment

The choice of Tezos as the blockchain network for this experiment is a deliberate one, as it aligns well with the goals of the solution presented in the previous section.

To understand why Tezos was chosen, it is important to first explain what Tezos is. Tezos is a blockchain network that was launched in 2018, and it's history began with the idea of creating a generic and self-amending crypto-ledger, which could be improved and upgraded over time without causing any disruption to its community. This makes Tezos stand out from other blockchain networks, which often struggle with the challenges of upgrading their underlying technology. The idea behind it is to provide a blockchain network that is not only secure, but also flexible and upgradable. The network is designed to be self-amending, meaning that changes to its protocol can be made without the need for a hard fork. Tezos originally operates on a Delegated Proof of Stake consensus, but the specifics of this consensus are irrelevant since we are taking it out and swapping with different consensus protocols.

A hard fork is a permanent divergence in the blockchain of a cryptocurrency, leading to

the creation of two separate chains. It occurs when a node's existing code is altered, resulting in a change to the rules that govern the network. This can happen for a variety of reasons, including a desire to add new features to the network, fix security vulnerabilities, or resolve a disagreement within the community about the direction of the project. The disagreement of the size of a block in Bitcoin was the cause of a creation of Bitcoin Cash's network (mentioned in example 1.2), where the Bitcoin blockchain was splitted in two (and later in more chains).

When compared to other blockchain networks, Tezos stands out as a really good fit for this experiment. It is commercially, well-tested and industry-grade blockchain network, and its upgradability makes it suitable for the replacement of the consensus algorithm. Additionally, the protocol in the codebase is independent and the whole project is designed for the replacement of such. The idea of the protocol/consensus layer in Tezos is that it's stateless, meaning that it is independent from other parts of the node like the Peer-to-Peer layer, the client, the validation layer, and the storage of the blockchain state on disk. This makes it easy to focus on writing the actual protocol and eliminates the need to worry about other components.

In conclusion, using Tezos for this experiment was a strategic decision as it offers the necessary features and characteristics to carry out the experiment successfully. Its history, design, and upgradability make it an ideal candidate for this experiment, and the stateless nature of the protocol part of the codebase allows the focus to be on the consensus, rather than other components of the node. The results of this experiment will provide valuable insights into the potential of Tezos and its adaptability, making it an exciting step in the development of a blockchain network that can easily swap consensus algorithms.

3.2 How Tezos is structured and How it allows self-amending

Tezos has a unique structure, which separates the protocol, or how the Tezos project calls it, "Economic Protocol" from the rest of the node, known as the shell. The diagram 3.1 illustrates the Tezos architecture, and it's explained in greater detail here [35]. The protocol is responsible for interpreting transactions and administrative operations, as well as detecting erroneous blocks, everything else is handled by the shell.

The shell includes the part of the validation code, the peer-to-peer layer, the disk storage of blocks, and the versioned state of the ledger. The distributed database abstracts the fetching and replication of new chain data to the validator.

The economic protocol on the chain is subject to an amendment procedure, which allows for the on-chain operations to switch from one protocol to another.

Finally, the RPC (Remote Procedure Call) layer is an important part of the Tezos node, allowing clients, third-party applications, etc. to interact with the node and introspect its state using the JSON format and HTTP protocol.

In other words, the outside world of the node communicates with the shell, via RPC layer calls, and the shell communicates with the protocol. Like said previously, the protocol is stateless, and is only used to preform the logic part of the consensus, that is, to verify transactions, blocks and to describe what the shell should do when receiving newer information.

The communication between the shell and the protocol in Tezos is based on an interface called “Updater.PROTOCOL”. The protocol component is restricted to a specific environment that restricts the access to a defined set of OCaml modules. This is to improve the security of the protocol. The protocol must implement the “Updater.PROTOCOL” interface (defined in the environment, and later explained in more detail) in order to interact with the shell. The interface requires the protocol to define protocol-specific types for operations/transactions and block header, along with encoders/decoders for the types.

The protocol must also provide functions for processing blocks and updating the context, which represents the protocol state. The context is stored as a disk-based immutable key-value store. The block header and operations also have a shell header (protocol-independent) and a protocol-specific header. For example, a Block can have two levels, one corresponding to the overall/shell level, and other is the level of the block since the current protocol is started.

Blocks in Tezos have a generic part, understood by any protocol, and a protocol-specific part.

A Tezos node can contain multiple economic protocols, but only one of them is activated at any given time, and the it always starts with the “genesis” protocol.

Some protocols are linked to the node at the time of compilation, that is, the node already has information about it when starting, and some protocols can also be registered dynamically at runtime via an RPC, allowing for protocol updates without the need of manual intervention to do so.

Protocol activation in Tezos is a two-step process.

Firstly, a command injects an “activation block” to the blockchain. This block contains the hash of the protocol to be used, and only one operation to activate it. The activation block is the only block using the genesis protocol, as this protocol doesn’t contain any other functionality besides the activation of a different protocol.

Secondly, the next block in the blockchain will be the first block using the activated protocol.

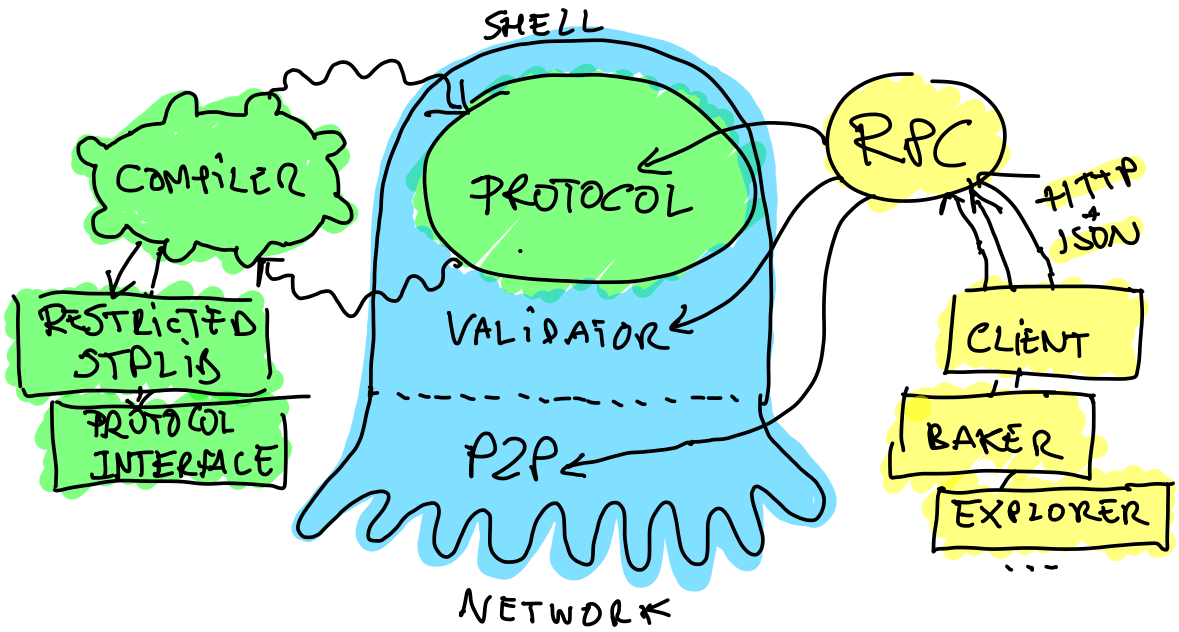

Figure 3.1: Architecture of Tezos (Source: Tezos Documentation)

In conclusion, the structure of Tezos is unique in that it separates the protocol, also known as the "Economic Protocol", from the rest of the node, called the shell. The protocol is responsible for interpreting transactions and administrative operations, while the shell includes the validator, peer-to-peer layer, disk storage of blocks, and versioned state of the ledger. The economic protocol on the chain is subject to amendment procedures, allowing for on-chain operations to switch from one protocol to another. The RPC layer allows clients, third-party applications, and daemons to interact with the node and its state. The communication between the shell and the protocol is based on the Updater.PROTOCOL interface, and the protocol is restricted to a specific environment to improve security. A Tezos node can contain multiple economic protocols, but only one of them is activated at any given time through a two-step activation process.

3.3 Implementation of a Protocol

Implementing a Proof of Work consensus algorithm as a protocol for Tezos was motivated by several reasons. Firstly, it was an opportunity for hands-on learning about how protocols are implemented and structured. This understanding is crucial for later work, such as testing and adding an adaptive layer to the DSL mentioned in previous sections. It was necessary to understand how these protocols work before attempting to create tools related to them. It also allowed for a deeper understanding of how Tezos works as a whole.

The implementation of Proof of Work was also significant because it provides a contrast to Tezos' original Proof of Stake protocol. This demonstrated that Tezos can be used to implement other types of consensus algorithms, including those that are different from the original. The implementation of Proof of Work also has the concept of mining, which is not present in the Proof of Stake protocol.

Additionally, implementing a protocol in Tezos is a big accomplishment. The fact that

only a few people do so makes this achievement even more significant, especially considering that it's a different type of consensus. This demonstrates a deep understanding of consensus algorithms and the ability to put that knowledge into practice.

In conclusion, the implementation of Proof of Work as a protocol for Tezos was a valuable learning experience that allowed for a deeper understanding of how protocols are structured and executed in Tezos. It also reinforced the concept of consensus algorithms and demonstrated the ability to put that knowledge into practice by implementing a unique protocol in Tezos.

Requirements of a Protocol In Tezos

In order to implement the experiment of a Proof of Work consensus algorithm for Tezos, a "lib_protocol" module in the protocol folder had to be created, where this is the main entry to the protocol and is executed by the node as the new amendment/consensus protocol. This is the only part that has to be implemented in OCaml and has to be compiled to work with the node.

To do so, a "TEZOS_PROTOCOL" file must be included in this module, that is used to specify the version of the environment that the protocol is to be compiled against, which in this case the version 6 was the one used, the hash of the protocol folder and the set of modules implemented in the folder.

There's the possibility to implement a newer environment for the protocol to accommodate the protocol, yet, for this experiment, this wasn't necessary.

Like said previously, the environment is an interface that provides a set of OCaml modules that the protocol can use, and the interface used to interact with the Tezos Shell.

The most important functions and types in that had to be implemented for environment "v6" were:

- *block_header_data*: This is the protocol-specific part of the header.
- *block_header*: This is the combination of the protocol header and the shell header.
- *operation_data*: This is the protocol-specific part of an operation/transaction.
- *operation*: This is the combination of the protocol operation part and the shell operation part.
- *validation_state*: This is the state of the validation of a block or operation, passed between functions.
- *init*: This function is executed to prepare the chain to start executing the new protocol.
- *begin_application*: This function is used when validating a block received from the network.
- *begin_partial_application*: This function is used when the shell receives a block more than one level ahead of the current head.

- *begin_construction*: This function is used by the shell when instructed to build a block and for validating operations as they are gossiped on the network.
- *apply_operation*: This function is called after *begin_application* or *begin_construction* and before *finalize_block* for each operation in the block or in the mempool, respectively. It validates the operation and updates the intermediary state accordingly.
- *finalize_block*: This function represents the last step in a block validation sequence.

In summary, the "TEZOS_PROTOCOL" file is used to specify the protocol environment to be used by the Tezos node, and the environment provides a set of functions and types that the protocol must implement to be able to operate within the Tezos network.

Implementation of the module "lib_protocol"

The implementation of a new protocol in the Tezos codebase is a significant challenge. In this particular case, the goal was not only to implement the new protocol, but to also learn how existing protocols are structured and implemented within the Tezos codebase. The approach was to study and follow closely the implementation of existing protocols, rather than taking an easier approach that may also have resulted in a working protocol but would not have offered the same level of learning.

The protocol that was implemented includes a few functionalities, such as transactions between two entities, where "Tez", the currency used in Tezos, is transferred from one account to another.

Another aspect of the protocol is the concept of mining and verifying the proof of work, like mentioned in a previous section about Proof of Work. This is done by hashing the header whole header, both the shell part and the protocol part, this last one containing the nonce, and verifying that the hash value is lower than the target where this is one of the main mechanism behind proof of work.

The environment that the protocol is compiled against requires a definition of the protocol header, which contains three elements: "target", "nonce", and "miner".

The **target** is used for comparison with the target value that the node calculated it should be (the specifics of how this is done is explained in a later point). The **nonce** is used to achieve a header hash with a value lower than the target, which is a key part of the Proof of Work mechanism. Finally, the **miner** field stores the address of the miner who found the nonce/hash, which is used to reward the miner. These elements combined make up the protocol-specific part of the header.

Implementation of Information Representation and Storage Logic

The implementation of the protocol in Tezos involves defining and representing the various types of information that are necessary for the protocol to function. This includes things like accounts, constants, headers, time, target, currency (Tez), and operations.

The information stored in the protocol is abstracted, meaning that the protocol itself is stateless. The context in which the protocol is applied maps to the actual storage of the

blockchain, but this is hidden from the protocol. The protocol only sees the storage as a generic key-value store and the functions that access this are defined in the ”raw_context.ml” file (in reality, they are accessed as a Tree, not just as a generic key-value). These functions allow the protocol to read, write, and update the storage, but the actual details of how this is done are not relevant to this document.

Also, as mentioned previously, the shell part of Tezos doesn't necessarily "understand" the protocol, as the protocol is somewhat independent from it. So the way that Tezos handles this fact is by representing the protcol parts of data (either from Blocks, Storage or Operations) as bytes, then these bytes are decoded by the protocol.

The encoding and decoding process of this information is done using the "data-encoding" library in Tezos. This library is also used to encode and decode OCaml types to and/from bytes or JSON, so it can also be sent to other nodes in the network.

The Tezos Protocol is implemented through a series of files that represent various components and functionalities. In this section, we will take a closer look at the information stored and the logic behind each component in the protocol.

The protocol contains representations/definitions and operations of the following types/ information, which are stored in files that end with "repr.ml". The Tezos documentation calls this the "representation layer" of the protocol. Unless the representations are only used within the protocol code, they also implement the encoding and decoding logic so this information can be sent/received, either fromto the network or the shell part of the node.

- Accounts/Manager: This represents the concept of an account in the protocol. An account is just a Publick Key Hash, which is used to fetch information from the storage.
- Constants/Parameters: This file contains information that is constant throughout the execution of the protocol. Some constants can be defined when activating the protocol. The constants include:
  - *block_time*: Defines the time between blocks.
  - *initial_target*: Defines the first target value.
  - *difficulty_adjust_epoch_size*: Defines the number of blocks to wait to then readjust the target.
  - *halving_epoch_size*: Defines the number of blocks to wait to then halve the mining reward.
  - *reward_multiplier*: The initial reward, which then gets halved.
  - *Header*: Defines the Protocol header, as mentioned before.
  - *block_time*: Defines the time between blocks.

  These parameters are crucial for achieving the project's objective, allowing for more precise tuning of the protocol.
- Time: Type definiton of "time"' in the protocol.

- Target: Defines how the target, used in Proof of Work is represented, in this case, it's a 256-bit number.
- Tez: Defines the type and operations for the currency.
- Operations: Representations and functionalities of the available operations. These include Transactions (between two entities) and Reveal (maps a Public Key to a Public Key Hash, like the ones used in the Accounts).

Some of these representations where designed by following the existing protocols implemented in the Tezos codebase. These representations ensure that the creation, conversion, encoding and decoding and other functionalities are done correctly by the protocol.

The logic behind storing information in the Tezos Protocol is an integral part of its functionality. The protocol uses an abstraction over the blockchain as its storage mechanism.

This is really important to the implementation of the protocol. While implementing a protocol, the developer doesn't have to understand how the blockchain will be stored, what optimizations for inserting and retrieving information are done, etc.

The files that are responsible for storing information have names ending with "-storage.ml", and they store the following key pieces of information:

- Accounts: It stores information about each account in the protocol. It maps the account's key to the account's balance and other important information and how defines how this information should be stored, retrieved and other checks.
- Target: It stores the current target, which is used for comparing the value of the header hash in the proof of work process.
- Epoch Time: It stores the timestamp of the latest difficulty adjust epoch, which is used to determine when it's time to adjust the target value. This is an important part of maintaining the security and integrity of the network, as it allows the network to dynamically adjust the difficulty of mining to ensure a stable rate of block production.

By storing these types of information in the blockchain, the Tezos Protocol provides a secure and transparent method of tracking the state of the network, which is crucial for the proper functioning of its operations. The protocol also provides a well-defined structure for the information it stores, which ensures consistency and maintainability of the code.

From the Protocol side of view, it sees the Blockchain as a Map/Tree, where it can fetch and append information to it using the typical concept of a Map (in programming).

Implementation of the main entry points to the protocol

The implementation of the main functions of the protocol plays a crucial role in ensuring the correct execution of the protocol.

These functions are designed to take the necessary steps to apply the information and verify the validity of the information being applied. The functions presented in the previous

section, such as *begin_application*, *begin_partial_application*, etc. are all a part of the main functions of the protocol.

These functions perform various checks and updates to the context, which is an immutable representation of the state of the protocol. The context contains information such as the accounts, the target, and the epoch time, among others. The functions return some form of validation state and updated context, if applicable.

Once again, the protocol is independent from the rest, so these functions are implemented with this concept. That is, they are called by the shell part of the node, by passing information such as the current "context" and some information about the previous block (to the one being built or validated).

- *begin_application* and *begin_partial_application*:
  - These functions prepare the current raw context, which contains information such as the context and more. The concept of raw context will be explained later.
  - Checks are performed, such as verifying if the current target is the correct one and if the header has a valid hash, that is, if the hash of the whole blockheader has a value lower than the target.
  - They update information such as the target for the next epoch, if an epoch has elapsed.
  - These functions are called when receiving a block from the network. The partial variant is used when the block received doesn't have the successor level to the current block.
- *begin_construction*:
  - Does the same preparation to the context as *begin_application* and *begin_partial_application*.
  - Performs the same checks, but also rewards the miner if the block is valid.
  - This function is called when the node is building a node. For example, if the node is a mining node.
- *apply_operation*:
  - Verifies if the operation contains a valid signature.
  - Verifies if the operation has a positive result, for example, if the account has enough funds to transfer to another account.
  - If the operation is positive, it executes the operation on the current context.
  - This function is used after both *begin_application* and *begin_construction*.
- *finalize_block*:
  - Commits the change to the shell.
  - Returns a receipt or result log that exposes what happened with the application of either a block or operation.

- *init:*
  - Initializes everything needed to start the protocol, such as the constants and storage.
  - Also creates the first blocked of the protocol to be appended to the chain.

In high-level view, the way that these functions are executed is the following:

We assume that every function is independent and purely functional. Each call of any of these functions receive and return (back to the shell) an updated-copy of a value called "context", in other words, it updates the state of the blockchain, unless something goes wrong, which then the updated context gets ignored. The shell handles the flow of execution. It starts with *begin_application*, *begin_partial_application* or *begin_construction*, depending of the conditions mentioned above. They handle the initial validation of the block header.

Then, for every operation included in the block, *apply_operation* is executed, copy-updating the context and passing it to subsequent calls of this function, until the last operation. Here, operations update the state and are validates.

After the last operation is validated, the context is finally passed to *finalize_block*, where the last updates to the context are performed and the last validations are done. After this function is called (and depending on the result of this process), the block is commited back to the shell, which is then added to the underlying blockchain.

Implementation of Alpha and Raw Contexts

The implementation of the protocol relies heavily on two core abstractions: **Alpha_context** and **Raw_context**. These two modules play crucial roles in ensuring the separation of concerns and allowing the protocol to be implemented over a generic key-value store.

Alpha_context, defined in the "alpha_context.ml" module, serves as the consensus view of the context. This module enforces the separation between mapping the abstract state of the ledger to the concrete structure of the key-value store, and implementing the protocol over the state. The Alpha_context defines a type "t" that represents the abstracted state of the ledger and can only be manipulated through the use of selected manipulations, which preserve the well-typed aspect and internal consistency invariants of the state.

The abstracted state is read from the disk during the validation of a block and is updated by high-level operations that preserve consistency. Finally, the low-level state is extracted to be committed to disk. This abstraction provides a well-separated structure in the code, with the code below Alpha_context handling the ledger's state storage, and the code on top of it implementing the protocol algorithm using plain OCaml values.

Raw_context, defined in the "raw_context.ml" module, is the information view used by Alpha_context. It serves as the raw, storage, or non-abstract view of what is actually done in the background by the consensus/protocol. Raw_context provides the abstraction used by Alpha_context to access the raw part of the protocol, that is, the information that is actually stored and the representation layer.

In conclusion, the implementation of the protocol relies heavily on Alpha_context and Raw_context to enforce separation of concerns and to abstract away the underlying imple-

mentation details, allowing the protocol to be implemented over a generic key-value store in a readable and well-separated manner.

Features that weren't implemented

In this feasibility study, we have focused on the implementation of consensus protocol in the Tezos node. However, there are a few features that have not been included in this study but could be considered as future work.

One of the missing features is the support for smart contracts. This was not the main focus of the experiment, but smart contracts can be added on top of the protocol to provide more functionality. Currently, the protocol only supports peer-to-peer value transactions, but with the addition of smart contracts, more complex operations can be performed.

Another missing feature is the lack of upgradability in the protocol. This was intentional as this experiment was focused on a standalone Proof of Work consensus algorithm, and there is no previous protocol nor will there ever be a next protocol that is an amendment to this one. Upgradability could be added to the protocol in the future to provide more flexible and dynamic updates, but once again, for this kind of experiment, doing so would be useless.

It should be noted that the absence of these features was not due to technical limitations, but rather a deliberate decision to focus on the implementation of a protocol in the Tezos.

3.4 Tools developed to interact with the protocol

In this study, tools related to the protocol were also implemented to enable interaction with the network. These tools could be developed independently from the Tezos project, since it's possible to communicate with the tezos node through JSON RPC. However, implementing these tools using OCaml and Tezos modules has the added benefit of being automatically integrated with the Tezos node client (and much more). Upon activation of the protocol, the client would automatically adapt to accommodate it, enabling commands that are specific to the protocol, such as "Transfer" and "Reveal", which is the case of the protocol covered here.

One of the tools implemented was the client, which can be found in the "lib_client" folder. The client is mostly used to access information on the network, such as an account's balance or the current target. It serves as a better user interface for accessing the information through JSON RPC services. The client also has the capability of injecting blocks and operations into the network, so it can perform the two operations included in the protocol.

Another tool that was implemented is the baker, which can be found in the "lib_baker" folder. The baker uses functionalities implemented in both the client module and the "Shell Services" module, this last one being already part of the Tezos codebase. The baker performs the task of baking (how block creation is called in Tezos), or mining, a block by taking multiple operations, pre-applying the block in the protocol (to check if the block header is valid), and then performing the work to find the nonce that makes the block's hash lower than the target, similar to what a miner would do in a Proof of Work blockchain. Once the block is mined, it is pushed to the chain (and spread to the network).

3.5 Conclusion

MUDAR: In conclusion, this experiment proved to be a valuable exercise as it allowed us to gain a deeper understanding of how consensus protocols work and how they can be implemented within a blockchain network. The implementation of the protocol allowed us to test its capabilities, limitations and to identify areas for improvement. Furthermore, it demonstrated the versatility and flexibility of the Tezos network in accommodating custom consensus protocols, making it an ideal platform for experimentation and innovation.

Chapter 4

Tool Development

This chapter focuses on the development of two essential tools designed to streamline the process of creating and evaluating blockchain protocols, particularly on the Tezos platform. The first tool, known as the Bootstrapper, automates the initial setup and coding tasks, providing a quick start for developers. The second tool, the Live Testing Tool, offers a real-world testing environment equipped with a variety of features for performance metrics and network control. Together, these tools aim to address the complexities and challenges inherent in blockchain protocol development, offering a more accessible and efficient pathway for both novice and experienced developers.

4.1 Bootstrapping Tool

The Bootstrapper was conceived with the aim of simplifying the development process for creating new protocols in Tezos. Our objective was to minimize the amount of code a developer needs to write to get a protocol up and running. To achieve this, we carefully selected code components that are generic enough to be applicable across multiple protocols. Additionally, we automated certain aspects of the protocol's integration with both Tezos and our Testing tool. In essence, the Bootstrapper serves as a scaffolder or starter pack for protocol development in Tezos.

Protocol Generalizations and Common Code

As previously mentioned in the document, there are specific requirements that every protocol must meet to be compatible with Tezos:

- Must reside in the ‘src‘ folder within the Tezos codebase.
- Must contain a ‘lib_protocol‘ folder and a ‘main.ml‘ file.
- Requires a ‘dune‘ file, as Tezos utilizes the Dune build system.
- Must include a ‘TEZOS_PROTOCOL‘ JSON file with attributes:
  - ‘expected_env_version‘: Expected environment version. Defaults to version 6 for simplicity but can be changed.
  - ‘hash‘: Protocol's hash, serving as its identifier within the node.
  - ‘modules‘: Names of the OCaml Modules within the folder.

Our tool automates the creation of these required components, eliminating the need for manual setup.

Common Code Components

Here we outline the elements that are common across most protocols.

Main.ml File: Since we use environment v6, a ‘main.ml‘ file is included with the essential layout. This is one of the few components that can’t be generalized, as it contains the protocol’s logic.

Account and Tez Representation: Most protocols have the concept of users/accounts and balances. Therefore, we include representations for accounts and Tez, the currency in this system.

Storage Logic, Functors, and More: We include abstractions for storage in Tezos, utilizing the concept of Maps for key-value storage, which is common to most protocols.

Block Header Representation: Every protocol needs to define how it will represent the header part of its blockchain. This is a common requirement, although the specific characteristics may vary.

Given these commonalities, our tool serves as a starter kit, providing the essential components that can be generalized across different protocols. The tool aims to fill in as much as possible, leaving only the unique logic to be implemented by the developer.

Functionalities

The Bootstrapper is a command-line program designed to streamline the initial setup of a new protocol in Tezos. It functions as an ”Init Protocol” command, performing the following tasks of **folder creation**, where it automatically sets up the necessary folders, **template copy**, as it copies a predefined template (created by us) into the new folder, but are left in the folder so the developer can change it for their needs, **text replacement and integration**, as it replaces specific text in the template and ensures integration with Tezos’ Dune build system and other required code components, and finally it **includes a pre built script** to build the protocol, so there’s no need for the developer to learn how this process is done.

To further ease development and reduce boilerplate code, we enhanced a pre-existing meta-programming library, *ppx_deriving_encoding* ADICIONAR REFERENCIA. This library allows developers to automate the definition of encoding functions, by appending a meta-programming tag *[@@deriving encoding]* to a type, the encoding function for that type is automatically generated, and this is particularly useful for type representations, as a big part of the code in protocols are these functions. We also include a block header functor, that auto-generates the entire block_header file functionality by merely providing a type. This not only reduces the code volume by threefold (as experienced while improving the previously mentioned Proof of Work) but also alleviates the developer’s need to delve into the intricacies of the Data-encoding library.

For the user, this is abstracted with the following command

./bootstrapper -name my_really_fast_protocol -env 6

By incorporating these functionalities, the Bootstrapper serves as a comprehensive tool for initiating and easing the development of new protocols in Tezos.

4.2 Live Testing Tool

The Live Testing Tool is central to the main objective of this thesis, which aims to facilitate the development and evaluation of blockchain protocols in a live setting. After developing a protocol in a standardized manner, particularly using Tezos as the platform, the next crucial step is to observe its real-world behavior. This tool is designed to not only run the protocol in a live environment but also to provide valuable metrics about the network's performance. Additionally, it offers the capability to control various aspects of the network, allowing for a comprehensive evaluation of the protocol under different conditions, and can also be used during the development of the protocol as part of the development workflow, as with it it's possible to write tests and get a network running in way fewer steps as with manually executing commands and operations.

Overall, the Live Testing Tool serves as an essential component for both the development and assessment phases of blockchain protocols.

Functionalities

The Live Testing Tool operates as a web server, making it language-agnostic for test development. Any programming language capable of making HTTP requests can interact with it. The server exposes a variety of endpoints that facilitate different aspects of testing, such as initiating a protocol test, controlling node operations, and gathering network metrics.

In its architecture, the server acts as a coordinator, maintaining a list of all node endpoints and ensuring seamless interaction among them. This enables the tool to have a centralized point for controlling and monitoring the network.

While it's executing, the tool also logs information about the tests it performed, from the protocols, to the start time of the nodes and more.

To provide real-time metrics and insights, the tool employs a monitoring mechanism for each node. Referred to as "watchers," these components continually request information from their respective nodes, focusing on blocks and transactions. This allows the tool to perform various calculations and assessments, offering a comprehensive view of the protocol's behavior in a live setting.

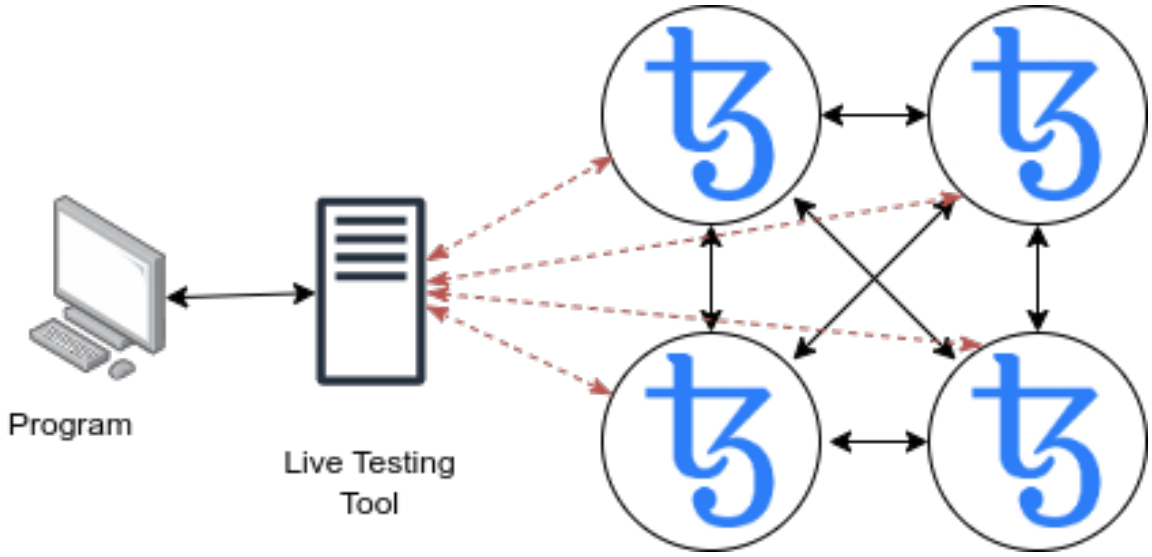

Figure 4.1: Live Testing Tool Architecture Visualized, with 4 Tezos Nodes Running

Endpoints Provided by the Tool

Generic Endpoints

- `/start-test`: Initiates a test network given the *protocol name*, *number of nodes*, and *initial parameters*.
- `/stop-test`: Halts all ongoing tests and activities.
- `/status`: Displays the current status of the test.
- `/protocols`: Lists the protocols that the tool has detected.
- `/protocol-parameters/:name`: Retrieves the parameters of a specific protocol.
- `/nodes`: Returns information about each node, including *IP*, *ports*, *process ID*, and *directory*.

Protocol-Related Endpoints

- `/swap-protocol`: Replaces the current protocol with another.
- `/change-parameters`: Alters the existing parameters of the network's protocol.

Network Control Endpoints

- `/start-node/:id`: Starts a node with a specific *ID*, if provided.
- `/stop-node/:id`: Stops a node with a specific *ID*.

Split Brain/Fork-Related Endpoints

- `/split-network`: Splits the network into smaller networks, given a list of nodes.
- `/rejoin-network`: Rejoins the fragmented networks into a single network.

Metrics Endpoints

- `/tps`: Provides the max, min, and average *Transactions Per Second*.
- `/time-to-consensus`: Measures the time taken for the majority of the network to agree on a block.
- `/propagation-times`: Calculates the time for a block to propagate across the network.
- `/first-and-last-block-times`: Gives the time of the first and last blocks.
- `/blocks-per-second`: Offers the max, min, and average blocks per second.
- `/discarded-blocks`: Indicates the number of blocks discarded by the network.

Example Usage

In this section we showcase how this tool can be used from a Python script.

Listing 1 Example function that starts a test

```
def start_test(protocol_name, nodes, fitness, parameters):
    data = {
        "protocol_name": protocol_name,
        "n_nodes": int(nodes),
        "fitness": int(fitness),
        "parameters": parameters
    }
    print(data)
    response = requests.post(ADDRESS + "/start-test", json=data)
    if response.status_code == 200:
        print("Success:", response.text)
    else:
        print("Failure:", response.status_code)
        exit(1)
```

Listing 2 Multiple utility functions to fetch information about the network

```
# Function that returns the current status of the network
def get_status():
    return requests.get(ADDRESS + "/status").json()
# Function that returns information about all nodes
def get_nodes():
    return requests.get(ADDRESS + "/nodes").json()
# Function that returns the hash of the current head
def get_head_hash(node_rpc):
    return requests.get(f"http://127.0.0.1:{node_rpc}/chains/main/blocks").json()[0][0]
# Function that returns the info of a specific block, provided a hash (block_id)
def get_block_info(node_rpc, block_id):
    return requests.get(f"http://127.0.0.1:{node_rpc}/chains/main/blocks/{block_id}").json()
# Function that returns the info of the head
def get_head_info(node_rpc):
    return get_block_info(node_rpc, get_head_hash(node_rpc))
```

Listing 3 Functions that request information in a loop

```
# Loop that keeps requesting the latest info about the tps of the system
async def fetch_tps():
    while True:
        async with requests.get(ADDRESS + "/tps") as response:
            tps = await response.text()
            print(f"Current TPS: {tps}")
        await asyncio.sleep(1)
# Loop that waits until the test is running
async def wait_for_start():
    last_status = 'stopped'
    while last_status != 'running':
        status_resp = get_status()
        last_status = status_resp["status"]
        sleep(1)
    return
```

Listing 4 Main Function that tests how the TPS changes with the Block Time

```
# Main function that preforms the test
# In this test we check out what happens to TPS when we duplicate the Block Time
def test():
    blocks_to_wait = 200 # Blocks to wait until we increase the Block Time
    protocol_name = "demo" # Name of the protocol
    current_block_time = 60 # In seconds
    parameters = get_protocol_parameters(protocol_name) # Fetch parameters
    number_of_nodes = 20 # Number of nodes to start the network
    fitness = 0
    # we change the default parameters so it has the block time specified
    parameters["constants"]["block_time"] = current_block_time

    # We start the test and wait until the network is ready
    start_test(protocol_name, number_of_nodes, 0, parameters)
    wait_for_start()

    # Ommited function, that fetches information about the nodes
    nodes = get_nodes()
    first_node = nodes[0]

    # Ommited function that makes it so the first node has an baker executing
    # MINER_ADD is the address of the miner
    start_mine(first_node, MINER_ADD)

    # Ommited function that spams the network with transactions
    spam_transactions(nodes, MINER_ADD)

    # We show the current TPS in the command line
    fetch_tps()

    while True:
        last_block_info = get_head_info(first_node)
        level = last_block_info["header"]["level"]

        # In case we reached the level of blocks to wait, we increase the block_time
        if level % blocks_to_wait == 0:

            current_block_time = current_block_time * 2
            parameters["constants"]["block_time"] = current_block_time

            print("Increasing the Block Time to " + current_block_time)
            change_parameters(parameters)
        sleep(current_block_time)
```

The Live Testing Tool serves as a comprehensive platform for both deploying and evaluating blockchain protocols in a real-world setting. Its language-agnostic design and extensive set of endpoints make it a versatile tool for developers. By providing real-time metrics and control over network parameters, it offers a robust environment for rigorous protocol assessment.

4.3 Conclusion

This chapter introduced two pivotal tools aimed at simplifying the development and evaluation of blockchain protocols: the Bootstrapper and the Live Testing Tool. The Bootstrapper serves as an initial setup utility, automating much of the boilerplate code and integration tasks. On the other hand, the Live Testing Tool provides a live environment for protocol testing, complete with real-time metrics and network control. Together, these tools form a cohesive ecosystem that significantly eases the challenges of blockchain protocol development and assessment.

Chapter 5

Use Case: Developing and Testing a New Protocol

Building on the previous chapter's conclusion, the focus of this chapter is to demonstrate the practical utility of the tools we've developed—namely, the Bootstrapper and the Live Testing Tool—in a real-world development scenario. The objective is to provide a comprehensive view of how these tools can be seamlessly integrated to facilitate the development and testing phases of a blockchain protocol.

To achieve this, we chose to implement a Proof of Authority protocol. While this protocol shares some similarities with the previously implemented Proof of Work, its core consensus mechanism is fundamentally different. This serves as an excellent case study to showcase the capabilities of our tools. The Bootstrapper was used to automate the initial setup and coding tasks, while the Live Testing Tool provided a live environment for performance evaluation and fine-tuning. This chapter shows how these tools work together to make creating a protocol faster and easier.

5.1 Goals and Approach

The primary aim of this chapter is to implement a Proof of Authority protocol. We intend to start the development process using the Bootstrapper tool to set up the initial codebase. Following this, we will write tests specifically designed to evaluate the protocol's performance using our Live Testing Tool.

To meet these objectives, we have a multi-faceted approach. First, we aim to demonstrate the efficiency gains by using the Bootstrapper, highlighting the reduced number of steps needed to get the protocol up and running. Second, we plan to show how the Live Testing Tool can be an invaluable asset during the development phase, particularly for debugging and iterative testing. Lastly, we will present example tests that can be run on the new Proof of Authority protocol, showcasing the tool's versatility and utility in a real-world scenario.

5.2 Boostrapping and Implementing Proof of Authority

In this chapter, we delve into the implementation of a Proof of Authority (PoA) protocol. Unlike public, permissionless protocols like Proof of Work, PoA is a permissioned protocol. This means that only a select group of nodes, known as authorities or validators, are allowed to create new blocks.

Proof of Authority operates on the principle of reputation and trust. A small number of trusted nodes are given the authority to validate transactions and create new blocks. In our

implementation, we replace the concept of mining with a list of these authorities. These validators are responsible for block creation and operate in a round-robin fashion.

The sequence in which these validators create blocks is determined by hashing. Specifically, for each round (or level, for simplicity), we hash the address (or Public Key hash) of each validator concatenated with the current round number. The hash function used is $H(address + round)$. These hashes are then used to sort the order of the validators, to decide the order of validators for the next round.

In case a validator fails to create a block, the protocol is designed to move on to the next validator in the sorted list. This process continues until a block is successfully created. This mechanism ensures both fairness and liveness in the protocol. Every node gets an opportunity to create blocks, and the protocol continues to function even if some nodes fail.

Notice: the way we order the validators is somewhat deterministic (unless validators fail). This has real-world implications and isn't necessarily a well designed protocol. To improve on this protocol, some sort of "randomness" should be added that is sourced from an outside entity. For example, using previous block information to sort the validators (for example, using the hash of the operations, or timestamp) wouldn't be a good way as the validator creating the block could have influence that information and in some way force the next block validator (to himself). This protocol was implemented just for demonstration purposes.

5.3 Methodology

To kickstart the development of our Proof of Authority protocol, we used the Bootstrapper tool with the command *./bootstrapper -name poa -env 6*. This command sets up a new protocol, ensuring it's integrated into the Tezos build system. It also creates a folder with pre-made templates for various files, saving us from starting from scratch.

Since the Bootstrapper already provides files for account logic, storage, and monetary representations, our focus was mainly on filling in the protocol-specific logic. We began by defining the block header, which in our case only includes the "validator" and "authority_list" values, along with the block's signature by the validator.

Next, we utilized the pre-provided storage file to implement the storage, logic, and representation of the validator set. This was followed by the core logic of the protocol, which is executed when a block is either received from the network or built by the node itself. This logic includes:

- Verifying if the validator address in the header is authorized.
- Confirming if the block proposer is the expected one based on our round-robin logic.
- Checking the validity of the block's signature.

When the node receives a block from the network at a level above the expected one, it only verifies if the signature is correct and if the block validator/proposer is the expected one based on the information on the header.

We also implemented the logic for round selection and created an executable for the block creator, similar to the miner executable in the previous example. That is, in order for a node to be a validator one, it must have a baker attached to it, which is just an executable that keeps track of new blocks in the network and upon new blocks it checks if the account address associate to it is of the current proposer.

By following these steps, we were able to implement the protocol with minimal effort, focusing only on the protocol-specific logic. We only had to implement the logic behind the verification of the block header as well as the round-robin mechanism.

While developing the protocol, we used the Live-testing tool with the following script, to test whether the network was executing and if the protocol works no matter the number of validators:

Listing 5 Script to live test the Proof of Authority during development

```
protocol_name = "poa"
parameters = get_protocol_parameters(protocol_name)

number_of_validators = 5 # This can be changed to account for more/less validators

validators = []
for _ in range(number_of_validators):
    account = create_account() # Returns account with address, private and public keys
    accounts.append({key: account[key] for key in ('address', 'public_key')})
return accounts

start_test_response = start_test(
    protocol_name, number_of_validators, 0, parameters)

nodes = get_nodes()

for i in range(number_of_validators):
    # Adds public and private key to node and starts baker
    add_key_and_propose(nodes[i], validators[i])
```

In summary, the Bootstrapper and Live Testing Tool significantly streamlined the development process of our Proof of Authority protocol. The Bootstrapper's template provided a strong foundation, allowing us to focus solely on the protocol-specific logic. This was evident in the number of lines of code written; out of the 6166 lines in the complete protocol, only 1700 were written by us. That's just 27

This not only reduced the coding effort but also lowered the barrier to entry, as we didn't have to delve deep into the intricacies of protocol development. The Live Testing Tool further aided in debugging and validation, making the entire development cycle more efficient. Therefore, these tools prove to be invaluable assets for anyone looking to develop and test blockchain protocols with ease and efficiency.

5.4 Evaluation and Insights: PoA vs PoW

In this section, we aim to compare the key metrics of Proof of Authority (PoA) and Proof of Work (PoW) protocols. Specifically, we focus on the metric of Time-to-Consensus, which measures the time taken for a newly created block to be accepted by a majority of nodes or validators in the network. This comparison serves as a practical use-case for our developed tools and provides insights into the efficiency of different consensus algorithms.

To carry out this comparison, we used a script that simulates various network conditions for both PoA and PoW. This script is designed to measure the Time-to-Consensus under different configurations of block time and validator set size.

Listing 6 Script to execute the tests for Time-to-Consensus

```
def start_protocol_test(protocol, n_validators, n_nodes, blocks_to_wait, blocks_to_run, block_time):

    print(f"Starting {pretty_name} test")
    parameters = get_protocol_parameters(protocol)
    parameters["constants"]["block_time"] = str(block_time)
    baker_name = "./octez-baker-custom-"
    accounts = create_multiple_accounts(n_validators)
    reveal_accounts = False

    if protocol == "poa":
        parameters["constants"]["initial_validator_set"] = [
            {key: account[key] for key in ('address', 'public_key')}
            for account in accounts
        ]
        baker_name = baker_name + "poa"
    else:
        reveal_accounts = True
        baker_name = baker_name + "demo"

    print("Starting the network")
    start_test(protocol, n_nodes, 0, parameters)
    wait_until_status("running")

    nodes = get_nodes()

    print("Starting the baker")
    processes = start_bakers(protocol, baker_name, accounts, nodes)

    print("Running the test....")
    wait_for_blocks(nodes[0]['rpc'], blocks_to_run)
    ttc = fetch_time_to_consensus()
    print(f"Final TTC: {ttc}")

    print("Stopping the test....")
    stop_test()
    wait_until_status("stopped")
    stop_bakers(processes)

    print(f"{pretty_name} test complete.")


tests = [
    {"validators": 10, "nodes": 10, "block_time": 30},
    {"validators": 20, "nodes": 20, "block_time": 30},
    {"validators": 10, "nodes": 20, "block_time": 20},
    {"validators": 30, "nodes": 30, "block_time": 20}
]


blocks_to_wait = 5
blocks_to_run_for = 25

for config in tests:
    for protocol in ["poa", "demo"]:
        start_protocol_test(protocol, config["validators"],
                            config["nodes"], blocks_to_wait, blocks_to_run_for, config["block_time"])
```

Expectations, Results and Observations

Based on prior knowledge and the inherent characteristics of PoA and PoW, we expect the following outcomes:

- PoA should exhibit a more consistent average Time-to-Consensus, irrespective of the block time and the size of the validator set.
- In PoW, the Time-to-Consensus is likely to increase with the number of validators, as a larger majority is needed to resolve forks and discard invalid blocks.
- Smaller block times in PoW are expected to result in longer Time-to-Consensus due to increased frequency of forks.
- The need for larger block times in PoW is evident, as shorter block times can lead to network instability and increased security risks.

We present the resulting statistics in the following table:

Table 5.1: Time-to-Consensus Comparison between PoW and PoA

Parameters			Metrics (Time-to-Consensus)		
Validators	**Nodes**	**Block Time**	**Max**	**Min**	**Average**
Proof of Work					
10	10	30	117	10	50.959
20	20	30	143	9	70.312
10	20	20	143	10	65.229
30	30	20	191	20	72.332
Proof of Authority					
10	10	30	35	2	13.827
20	20	30	38	1	14.233
10	20	20	30	4	14.101
30	30	20	42	2	13.994

Upon analyzing the results, we find that our expectations hold true:

- PoA maintains a consistent average Time-to-Consensus (of about 14 seconds) across different configurations.
- PoW's Time-to-Consensus increases with the number of validators and decreases with larger block times, confirming its sensitivity to these parameters.

These observations validate the efficiency of PoA in achieving quicker consensus and highlight the limitations of PoW, especially in networks with smaller block times and larger validator sets.

5.5 Conclusion

In this chapter, we've demonstrated the power and utility of our developed tools—Bootstrapper and Live Testing Tool—in a real-world scenario. Through the lens of implementing and com-

paring two distinct consensus algorithms, Proof of Authority and Proof of Work, we've showcased how these tools can significantly streamline the protocol development and testing process.

The Bootstrapper tool allowed us to implement a fully functional PoA protocol with minimal effort, requiring us to write only 27% of the total lines of code. This not only reduces the development time but also lowers the entry barrier for newcomers in the field of blockchain protocol development.

Our Live Testing Tool provided invaluable insights into key metrics like Time-to-Consensus, enabling us to make data-driven decisions and optimizations. The tool's flexibility and adaptability were evident as we easily configured it to test different network conditions, thereby obtaining a comprehensive understanding of the protocols' performance under various scenarios.

The comparative analysis between PoA and PoW further emphasized the importance of choosing the right consensus algorithm, and how our tools can aid in making that decision based on empirical data.

In summary, our tools offer a robust, efficient, and user-friendly environment for developing and testing blockchain protocols. They not only save time and effort but also provide the kind of deep insights that are crucial for the development of secure, efficient, and scalable blockchain networks.

Chapter 6

Conclusion

Blockchain technology has emerged as a transformative force in various sectors, from finance to healthcare and beyond. However, the development and testing of blockchain protocols remain complex and time-consuming tasks. This thesis addresses this critical issue by providing insights and tools designed to simplify and expedite these processes.

We have developed two distinct tools: the Bootstrapper and the Live Testing Tool. The Bootstrapper serves as a foundational tool for protocol development, allowing developers to focus on the unique aspects of their protocols by automating much of the boilerplate code. On the other hand, the Live Testing Tool offers a dynamic environment for testing and analyzing blockchain protocols under different network conditions. Together, these tools form a comprehensive suite that can significantly reduce the time and effort required to develop and test new blockchain protocols.

To demonstrate the utility of our tools, we implemented two consensus algorithms: Proof of Work and Proof of Authority. These implementations served as real-world test cases, allowing us to showcase the capabilities of our tools in a practical setting. The comparative analysis between these two protocols provided valuable insights into the importance of selecting the appropriate consensus algorithm for specific use-cases.

Our work contributes to the broader goal of making blockchain technology more accessible and efficient. By lowering the barriers to entry for protocol development and offering a robust testing environment, we believe our tools can play a pivotal role in the advancement of this technology.

In conclusion, this thesis presents a significant step forward in the field of blockchain protocol development and testing. The tools and insights provided herein not only facilitate the development process but also contribute to the optimization and scalability of future blockchain networks. As blockchain technology continues to evolve, the methodologies and tools developed in this thesis offer a solid foundation upon which further advancements can be built.

6.1 Future Work

While this thesis provides a comprehensive toolset for blockchain protocol development and testing, there are several avenues for future work to further enhance its utility and scope:

- **Extend to Non-Blockchain Protocols:** One of the most immediate extensions would be to adapt the tools for non-blockchain based protocols, such as Directed Acyclic Graphs (DAGs). This would make the toolset more versatile and applicable to a broader range of decentralized technologies.

- **Additional Metrics:** The current version of the Live Testing Tool focuses on a limited set of metrics. Future work could involve the implementation of more comprehensive metrics to evaluate protocols from various angles, such as security, scalability, and fault tolerance.
- **Bootstrapper Enhancements:** The Bootstrapper tool could be evolved into a full-fledged development suite. This could include libraries, helper functions, and perhaps even a graphical user interface to assist developers in creating robust and efficient protocols more easily.
- **Integration with Lupin DSL:** There is potential for integrating the tools developed in this thesis with the Lupin Domain-Specific Language (DSL). This could streamline the development process further, allowing for more efficient code generation and testing.

By addressing these points, future work can build upon the strong foundation laid by this thesis, contributing to the development of more efficient, secure, and scalable decentralized systems.

Bibliography

[1] L. Goodman, "Tezos—a self-amending crypto-ledger white paper," *URL: https://www. tezos. com/static/papers/white paper. pdf*, vol. 4, pp. 1432–1465, 2014. 1

[2] "Peer-to-peer electronic cash." [Online]. Available: https://bitcoincash.org/ 2

[3] S. Nakamoto, "Bitcoin whitepaper," *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008. 6, 13

[4] E. Foundation, "Ethereum whitepaper." [Online]. Available: https://ethereum.org/en/whitepaper/ 6

[5] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, 1985. 7

[6] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988. 7

[7] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016. 8

[8] T. Inc, "Internet of blockchains." [Online]. Available: https://v1.cosmos.network/resources/whitepaper 8

[9] L. Aştefanoaei, P. Chambart, A. Del Pozzo, T. Rieutord, S. Tucci, and E. Zălinescu, "Tenderbake–a solution to dynamic repeated consensus for blockchains," *arXiv preprint arXiv:2001.11965*, 2020. 8

[10] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985. 8

[11] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, no. 10.1145. Portland, OR, 2000, pp. 343 477–343 502. 8

[12] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15. 10

[13] B. Bnb-Chain, "Whitepaper/whitepaper.md at master · bnb-chain/whitepaper," Oct 2022. [Online]. Available: https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md 10

[14] V. Buterin, "Why sharding is great: Demystifying the technical properties." [Online]. Available: https://vitalik.ca/general/2021/04/07/sharding.html 11

[15] Bitnodes. [Online]. Available: https://bitnodes.io/ 11

[16] C. Gondek, "Here is why bitcoin transactions take so long." [Online]. Available: https://originstamp.com/blog/here-is-why-bitcoin-transactions-take-so-long 11

[17] J. Xu, C. Wang, and X. Jia, "A survey of blockchain consensus protocols," *ACM Computing Surveys*, 2023. 12

[18] M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and A. Colman, "Blockchain consensus algorithms: A survey," *arXiv preprint arXiv:2001.07091*, 2020. 12

[19] A. Back *et al.*, "Hashcash-a denial of service counter-measure," 2002. 12

[20] E. Anceaume, A. Pozzo, T. Rieutord, and S. Tucci-Piergiovanni, "On finality in blockchains," *arXiv preprint arXiv:2012.10172*, 2020. 14

[21] C. Grunspan and R. Pérez-Marco, "On profitability of selfish mining," *arXiv preprint arXiv:1805.08281*, 2018. 17

[22] K. C. Chaudhary, V. Chand, and A. Fehnker, "Double-spending analysis of bitcoin," in *Pacific Asia conference on information systems*. Association for Information Systems, 2020. 17

[23] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM symposium on principles of distributed computing*, 2017, pp. 315–324. 17

[24] "The monero project." [Online]. Available: https://www.getmonero.org/ 17

[25] P. Foundation, "The pioneer of proof-of-stake." [Online]. Available: https://www.peercoin.net/ 18

[26] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017. 18

[27] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the works of leslie lamport*, 2019, pp. 203–226. 18

[28] P. Szilágyi. (2017, 3) Eip-225: Clique proof-of-authority consensus protocol. Ethereum Improvement Proposal. [Online]. Available: https://eips.ethereum.org/EIPS/eip-225 18

[29] Proof of authority - poa - identity digital. Aura is one of the Blockchain consensus algorithms available in OpenEthereum (previously Parity). [Online]. Available: https://www.poa.network 18

[30] H. Moniz, "The istanbul bft consensus algorithm," 2020. 18

[31] K. Karantias, A. Kiayias, and D. Zindros, "Proof-of-burn," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 523–540. 19

[32] “Gridcoin wiki home.” [Online]. Available: https://gridcoin.us/wiki/ 19

[33] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, “Proofs of space,” in *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Springer, 2015, pp. 585–605. 19

[34] M. Bowman, D. Das, A. Mandal, and H. Montgomery, “On elapsed time consensus protocols,” in *Progress in Cryptology–INDOCRYPT 2021: 22nd International Conference on Cryptology in India, Jaipur, India, December 12–15, 2021, Proceedings 22*. Springer, 2021, pp. 559–583. 19

[35] N. Labs, “Tezos software architecture¶.” [Online]. Available: https://tezos.gitlab.io/shell/the_big_picture.html 22