UNIVERSIDADE BEIRA INTERIOR

# Blockchains
## Subtitle

**Guilherme João Bidarra Breia Lopes**

Master's disseration planning
**Engenharia Informática**
(2nd degree cycle)

Supervisor: Prof. Doctor Simão Melo de Sousa

**janeiro de 2023**

# Contents

# Acronymms

| | |
|---|---|
| NFT | Non-fungible token |

# Chapter 1

# Introduction

## 1.1 Intoduction

Blockchains have gained significant attention in recent years for their potential to revolutionize various industries by providing a secure and transparent method for storing and transferring information and assets. At their core, blockchains are decentralized and permissionless ledger systems that leverage cryptography, consensus algorithms and State Machine Replication to maintain a shared history of all transactions on the network.

The increasing popularity of blockchains can be attributed to the growing demand for secure and transparent systems in various industries, particularly finance, supply chain management, digital identity and, recently, digital assets with the use of Non-Fungible Tokens (NFT).

In addition, the distributed nature of blockchains enables them to operate in a trustless environment, reducing the risk of single point of failure and ensuring the integrity of data stored on the network.

Consensus algorithms play a crucial role in the functioning of blockchains, as they determine the process by which transactions are validated, new blocks are added to the chain, and other details such as the selection of the next block producer.

Different consensus algorithms offer varying levels of security, scalability, and decentralization, and choosing the right algorithm for a specific use case is critical for the success of a blockchain network.

The main objectives of this dissertation are to compare different consensus algorithms for blockchains, evaluate the trade-offs between security, scalability, decentralization and explore the potential for consensus algorithms to be plugged and changed on demand. Additionally, the dissertation will cover the implementation and testing of a blockchain network with a pluggable consensus protocol and a thorough examination of the Tezos(REFERENCIA) blockchain as a testbed for consensus algorithms.

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, such as tools and methods to test consensus algorithms, to develop algoritms and to include them in already developed blockchain nodes.

## 1.2 Motivation

Despite the growing popularity of blockchains, there is currently a lack of tools and methods for easily swapping and testing different consensus algorithms in a live environment. This makes it difficult to determine which algorithm is best suited for a specific use case, particularly in the context of the blockchain trilemma, where it is challenging to achieve optimal balance between scalability, security, and decentralization.

Additionally, developing consensus algorithms can be a complex and time-consuming task, and there is currently no standardized method for describing and implementing them. This presents a significant barrier to the adoption and evolution of blockchains, as it limits the ability of developers to experiment with new and innovative consensus algorithms. Also, the decentralization being one of the strenghts points of blockchain networks falls in the hands of a few selected developers.

This dissertation aims to address these challenges by providing a comprehensive analysis of different consensus algorithms for blockchains, and exploring the potential for consensus algorithms to be plugged and changed on demand. The results of this study will serve as valuable feedback for the development of a domain-specific language for describing consensus algorithms, and will provide insights into the most effective methods for testing consensus algorithms in a live environment.

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. This will not only benefit the academic community but also industry stakeholders, who will be able to make informed decisions about which consensus algorithm is best suited for their specific use case.

In conclusion, this dissertation is motivated by the growing need for a comprehensive analysis of consensus algorithms for blockchains, and the desire to explore new and innovative methods for swapping and testing protocols in a non-simulated way. The objective of this study is to contribute to the development of a standardized method for describing and testing consensus algorithms, ultimately advancing the field of blockchains and improving the security, scalability, and decentralization of blockchain networks.

## 1.3 Problem Statement

Consensus algorithms, being a fundamental aspect of blockchain technology, are subject to a range of emerging problems. Hardforks, network Denials of Service attacks, centralization of networks, and other relevant issues are challenging the stability and security of these systems. The choice between scalability, security, and decentralization, as described by the Blockchain Trilema, can lead to trade-offs in the design of consensus algorithms.

One of the challenges of consensus algorithms is the difficulty of changing the rules of the game after a blockchain network has been established. The hard forks of the Bitcoin, Bitcoin Cash, and Bitcoin SV networks have highlighted this issue. Similarly, the Ethereum network has also undergone several hard forks, leading to a split in its community.

Scalability issues are another challenge faced by some blockchain networks, including Bitcoin and Ethereum, which have struggled to accommodate increasing demand. In addition, some networks, such as Solana, have experienced outages, further highlighting the need for a more robust and reliable consensus system.

This dissertation aims to address these challenges by exploring the development of consensus algorithms and ways to test them. The Tezos blockchain provides a suitable platform for this work, as it has a pluggable consensus system and allows for the implementation of different consensus algorithms. The goal is to learn how to develop consensus protocols in Tezos, develop the main entry points for a possible connection with a Domain Specific Language, and develop ways to test consensus algorithms. The first subject of such tests will be a proof-of-work consensus algorithm.

The question we want to answer in this dissertation is the following: How can we make a blockchain network that is flexible in terms of consensus algorithm selection and provides a platform for testing and comparison of different consensus algorithms? By addressing this question, we aim to provide a solution that will allow blockchain networks to be more scalable, secure, and decentralized, thereby addressing the challenges posed by the trilema. Additionally, this dissertation will serve as input/feedback for later work on a Domain Specific Language that will be used to describe consensus algorithms, making it easier to develop and test consensus algorithms in the future.

## 1.4 Document Organization

# Chapter 2

# Core Concepts and State of the Art

INTRODUÇÃO

## 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, 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 discussion on the importance of consensus in blockchain networks.

### State Machine Replication

State Machine Replication (SMR) is the concept of replicating data in a distributed system, where nodes in a system maintain the same state of a machine.

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, the network 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 the blocks are the nodes), that are connected to a previous block in the chain by referencing to it, that is usually 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.

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 and Ethereum 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 consensus on the contents of a new block through the use of consensus mechanisms, such as Proof of Work or Proof of Stake. The blockchain is a distributed ledger, with all nodes in the network having a copy of the ledger, ensuring availability of the state.

The blockchain can be considered the state of a machine, where copies of the state of this machine is 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. The cryptographic functions used in blockchains, such as hashing and consensus mechanisms, provide a high level of security to the ledger and its transactions, and the replication of it ensures the availability of the information.

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, imutable 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 types of networks in distributed systems: synchronous, asynchronous, and partially-synchronous.

Each type of network 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 having a common notion of time and upper bounds on message delay. That is, in the Synchronous model, a finite time limit $\Delta$ is established and known. The adversary can only delay the delivery of a message sent by at most $\Delta$ 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**, 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, such as in data-centric systems. 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** 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 (used in Cosmos Network and others), and Tezo's blockchain take on this protocol, Tenderbake, 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 among all participants in a network on the state of a shared database or system.

While State Machine Replication is the concept of broadcasting the update of a state, consensus is the concept of how the nodes replicate and decide on a replicated value.

One way to implement total order broadcast, that is, to broadcast messages or updates to the state machine in a specific order, is by sending messages via a designated leader node. But if the 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 to be the next leader), and should not be decided by a single node, and this being the coequally the concept of consensus.

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 a blockchain 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 block. Also, consensus algorithms enable the selection of the next leader, that is, the node that takes the transactions, builds the block and broadcasts the new block 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 to this type of networks, which include the **FLP** Theorem and the **CAP** Theorem.

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

- 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.

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 theorems will be provided and why they are relevant to the topic of Blockchain networks.

## 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 most often used for cryptocurrencies such as Bitcoin, Tezos and most of popular networks, where anyone can participate in the network and validate transactions, contributing to the security and reliability of the network.

**Permissioned blockchains**, also known as private 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, and where confidentiality and privacy are important considerations. Usually this networks are faster and more secure, 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, an enterprise-grade network and Binance Smart Chain, 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 “trillema”

Another subject that is usually used to compare different blockchain networks, with much discussion, is the topic of the “Blockchain trilemma” that refers to the trade-off between scalability, security, and decentralization. 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 (Around 40 thousand (TODO: Add citation)), yet lacks on scalability, since the creation of a block takes 10min and a transaction takes about 1 to 1.5 hours to complete (TODO: Add citation).

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.

## 2.2 State of the Art of Consensus Algorithms in Blockchain Networks

### 2.2.1 Proof of Work

Proof of Work (PoW) is a consensus algorithm used by many blockchain network, Bitcoin being the most popular network to do so. Bitcoin or “Nakamoto Consensus” was the first instance of a Proof of Work Consensus and the first instance of a blockchain network.

The idea behind PoW 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 impossible 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.

PoW or “Nakamoto Consensus” 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 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.

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.

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), 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. 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. For example, in Bitcoin this takes about 6 confirmations (or 60 minutes (TODO: add reference)). 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 does 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. 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 previousky.

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(hi−1,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. The usual workflow of a node is the following:

- Node gossip transactions in the network (They are 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 parameters are 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 starts trying out every possible value for the nonce until the resulting hash 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 reward.

When a entity creates a transactions, it can also offer a fee. This is used to prevent transactions spamming and also acts as a “spot in the block” auction. Besides block creation reward, the miner also receives the transactions fees, so miners will select transactions based on that fee, and other attributes of the transactions.

The difficulty 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. 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 “trillema”, 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 traditional blockchain structures. Unlike parallel-chain protocols that have multiple independent genesis blocks at initialization, 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, double-spending attacks, and liveness attacks. 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.

For example, some blockchain networks have adopted a new consensus algorithm that uses random incentives or that the reward of mining is shared.

## 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 basic 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 get together to mine a single block as if they were only a single node, so together they have more computational power, and inherently having a bigger change of finding a block. When a block if found by the pool (the 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, in Bitcoin, is done like this, and so, the owners of these pool nodes have a lot of power over the network.

Also, with the increase of Hashing power, that is, the computational power of blockchains, in specific “Bitcoin”, miners have improved the mining process by developing highly specialized hardware, called ASICs.

By doing so, it’s infeasible for anyone to join the mining process, and centralizing the power on the owners of ASICs. 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.

### 2.2.2 Proof of Stake

### 2.2.3

# Chapter 3

# Problem Statement, Experiments and Work Plan

## 3.1 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. 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, 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.

## 3.2 Solution Proposed

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, 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, customizable 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 this projects and compare them.

In conclusion, the solution proposes to overcome the challenges mentioned in the previous section by providing a method of testing and easily swapping consensus algorithms. By leveraging the advantages of a pre-existing blockchain and its adaptability, the focus remains solely on the consensus algorithm, the crucial aspect of any blockchain, allowing for efficient and effective testing and comparison of different consensus algorithms.

## 3.3 The Experiment

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 section, 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.

## 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. The history of Tezos 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 Tezos 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 Proof of Stake consensus, but the specifics of this consensus are irrelevant since we are taking it out and swapping with different consensus protocols.

When compared to other blockchain networks, Tezos stands out as a really good fit for this experiment. It is commercially and industrially used and its upgradability makes it suitable for the replacement of the consensus algorithm. Additionally, the consensus in the codebase is independent and the whole project is designed for the replacement of such. The idea of the protocol in Tezos is 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 addition to its technological strengths, Tezos is also a well-tested and industry-grade blockchain network, used by millions of users. This makes it a good fit for the experiment described in this dissertation, as it can be relied upon to provide a stable and secure platform for testing new consensus algorithms.

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.

### How it’s 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 protocol is responsible for interpreting transactions and administrative operations, as well as detecting erroneous blocks.

The shell includes the validator, which selects the valid head with the highest absolute score, 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, and daemons to interact with the node and introspect its state using the JSON format and HTTP protocol. This component is fully interoperable and auto-descriptive, using JSON schema.

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) in order to interact with the shell. The interface requires the protocol to define protocol-specific types for operations/transactions and the 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.

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. The protocols are linked to the node at the time of compilation, and some protocols can also be registered dynamically at runtime via an RPC.

Protocol activation in Tezos is a two-step process. Firstly, a command injects an "activation block" to the blockchain. This block contains only one operation, which is to activate the next protocol. 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. The activation command requires the hash of the protocol to be activated, and the protocol must be registered.

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.

## 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 also allowed for a deeper understanding of how Tezos handles protocols and how they are executed in the Tezos node.

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 whole 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 the idea of a transaction 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.

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.

- Accounts/Manager: This represents the concept of an account in the protocol. An account is just a key, which is used to fetch information from the storage. It is a Public Key Hash (of Ed25519, Secp256k1, P256).
- 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.

- Time: Defines the type of time used throughout the protocol.
- Target: Defines how the target should be, 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 account repr).

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 the blockchain as its storage mechanism. 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.

### 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", "begin_construction", "apply_operation", "finalize_block", and "init", 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.

- 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.
  - They perform checks, 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.
- begin_construction:
  - Does the same preparation as begin_application and begin_partial_application.
  - Performs the same checks, but also rewards the miner if the block is valid.
- 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.
- 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.

The extensive logic involved in these functions will not be presented in this paper, but it is important to understand the role they play in ensuring the correct execution of the protocol. The protocol is stateless, meaning that the context is immutable and can only be updated in a controlled manner through the application of the functions.

## 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 implementation 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 project, 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.

#### 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. 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.”

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.

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 it's called in the whole Tezos project), or mining, a block by taking multiple operations, preapplying 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 can be pushed to the chain.

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.

## 3.4 Future Tasks

# Bibliography