A Comparative Study Of Proof of Stake Algorithms

Antonio Saaranen
dat11asa@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Martin Hell

Examiner: Thomas Johansson

December 22, 2020

© 2020
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Cryptocurrencies are growing at a rapid pace. It is well established that the mining operation fueling the cryptocurrency Bitcoin consumes a significant amount of energy. This study aims to determine if there is an alternative to the consensus model, maintaining security, integrity, and decentralization. It includes a deep dive into the Bitcoin network and the consensus model Proof of Work (PoW). Additionally, four cryptocurrencies running the alternative consensus model Proof of Stake (PoS) are examined. To test if a PoS consensus model implementation could be a viable option, four distinct PoS implementations are described. Bitcoin is compared to four PoS implementations in the different aspects: *security, scalability, and decentralization.* How the different implementations differ and how the various properties are affected when changing the consensus algorithm is the main target of this thesis. The results show that the Bitcoin network suffers from a high fraction of the mining consumption originating from a small concentration of mining entities. Upon dissecting the PoS implementations, the largest attack vectors stem from a problem called the long range attack vector involving the lack of cost for proposing a block to the blockchain, in contrast to the Bitcoin blockchain that uses the PoW model that consumes external resources. This problem is handled by different security mechanisms depending on the implementation, mostly consisting of checkpointing schemes and finality on the network. However, despite not seeing any problems with this in practice, there is no fully trustless way to mitigate the problem. Additionally, despite PoS not being built primarily as a scaling solution, a cryptocurrency can implement it in that setting. For example, with EOS, significantly increasing the throughput compared to Bitcoin, with the cost of losing parts of the decentralization properties by maintaining 21 block producers that produce blocks for the entire blockchain. Additionally, Algorand is significantly improving throughput compared to Bitcoin, under the assumption that at least $\frac{2}{3}$ of the users are active, meaning that they are either validating blocks or delegating their vote to an active validator.

Acknowledgment

I want to thank Pontus Lindblom for proofreading the thesis and giving valuable input. Also, my supervisor Martin Hell for showing great patience and for helping me along the way. Tobias Johansson for doing the graphical illustrations. Andreas Antonopoulos for writing the book "Mastering Bitcoin" helping me get a good understanding of the Bitcoin stack and inspired me to use the same structure as in the book to explain the Bitcoin blockchain. Last but not least, I would like to thank my colleague and friend David Jaenson for guiding me through the jungle that is blockchain, giving me the knowledge base required to write this thesis.

Disclaimer

The blockchain space in general and the Proof of Stake research field in particular is new and ever-changing. The algorithms are described in the state of writing the thesis (2018-2020), due to the rapid change in the space a lot of the consensus algorithms will change after publication.

Glossary

The terminology in blockchain technology is often ambiguous and the terms often lack a formal definition.

Bitcoin address - consists of the account holders' public key hashed two times, first hashed using SHA-256 and then hashed by RIPEMD-160 hashing algorithm.

UTXO - Unspent transaction output

Blockchain - Open distributed ledger of transactions

Genesis block - The first block in the Bitcoin blockchain

P2P - Peer to peer

PoW - Proof of Work consensus algorithm

PoS - Proof of Stake consensus algorithm

DPoS - Delegated Proof of Stake

DoS - Denial of service attack vector

SPV - Simplified Payment Verification

Casper FFG - Casper Friendly Finality Gadget

Bitcoin - The name of the Bitcoin network

bitcoin - The cryptocurrency bitcoin

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The blockchain technology was proposed in 2008 as a reaction to the prevailing economic crisis. Throughout the years before the invention of the blockchain data structure, there have been many attempts to create different types of digital currencies with the common denominator being a central entity in control of monetary transactions and money supply. The financial crisis might have played a key role in the attempts to create a digital currency without a third party controlling the currency while still maintaining the network tamper-proof. It should remain impossible to alter the currency supply without revealing the change to the community. The challenge of creating a digital currency that remains decentralized was approached by trying to solve the *Byzantine Generals' Problem [1]* which simplified is where parties can not come to an agreement due to the fear of being misinformed. In order to tackle this problem, the solution lied in obtaining *consensus* in the network. Achieving decentralization and obtaining consensus in the network is a large task, not the least due to the fact that it is hard to define what decentralization means in practice. While the definition of decentralization will be examined more in detail for now we assume a network is decentralized if no single entity has control over changing the history of transactions and processing of new transactions in the network. In 2008 [2] a pseudonym called *"Satoshi Nakamoto"* published a paper on metzdowd.com on a mailing list called *"The Cryptography mailing list"* [3] and proposed the first distributed public blockchain, namely the cryptocurrency network called Bitcoin.

Satoshi claims he started writing the code 2007 [3]. He left a message in the first block of transactions on the blockchain. This block is also referred to as the genesis block. The genesis block contained the message *"The Times 3 January 2009 Chancellor on brink of second bailout for banks"* referring to the headline in the newspaper The Times on 3 January 2009. The objective of this message was to prove that the first block was not mined (created) before this date. However, the headline topics also raise ideological questions as to the objective of the new cryptocurrency called Bitcoin. Bitcoin and many of the other cryptocurrencies the came after are based on a data structure called *blockchain* that can be visualized as a growing stack of public records containing back-linked transactions secured using cryptography. To solve the consensus problem Satoshi suggested a *consensus algorithm* known as *proof of work*. All participants in the network are able to suggest their own records (blocks of transactions) to the public blockchain ledger. One is allowed to propose a block by solving something called the hash puzzle requiring the use of computing power. The main purpose of the consensus algorithm is to obtain synchronicity in the blocks of transactions on the blockchain. Reaching a collective agreement on which transactions are legit and allowed to be appended to the blockchain in other words. By proving you are consuming a physical scarce resource you are allowed to propose a new block. You are additionally entitled to a reward in bitcoin to incentivize good behavior. If the miners suggest faulty blocks to the blockchain their bitcoins are sooner or later going to be useless. Since the computing equipment used for mining nowadays is specialized hardware that consumes large amounts of energy with the sole purpose of mining bitcoin, this creates an economic incentive not to propose faulty transactions.

As of today, PoW algorithms of different types implemented in different blockchain networks is the most widely used algorithm to achieve consensus in cryptocurrencies utilizing blockchain technology. The consensus algorithm Satoshi suggested in 2008 called *HashCash* [2] is still used as of today in the Bitcoin network. The HashCash algorithm will be examined further in this thesis as a representation of a typical PoW consensus algorithm. There are multiple other PoW consensus algorithms but these are out of scope for this thesis. The primary scope of this thesis involves an alternative model for achieving consensus networks built on the blockchain data structure called *Proof of Stake* (PoS).

The alternative consensus model that is the main focus of this thesis does not rely on physical scarce, extrinsic resources but instead intrinsic resources often represented as native tokens in the network. The token holders in these systems could be seen as analogous to stakeholders in an ordinary company. The stakeholders in the system all have voting power equal to the portion of tokens they are currently possessing the private keys of. Assume that a user is holding 1% of all the tokens in circulation on the network, that user gets to append new blocks of transactions on average 1% of all the new blocks appended to the ledger. An alternative way for this consensus algorithm is that every user gets to vote with their tokens on who is allowed to produce blocks to the blockchain. PoS is often implemented with a reward mechanism similar to the reward mechanism in PoW protocols. A user receives a fixed amount of the given token upon mining the next block in the blockchain to incentivize mining, also called staking in PoS settings.

The rapid growth of cryptocurrencies in general and the Bitcoin network, in particular, has resulted in an increasing demand for computing power and specialized computing equipment called *ASICs*. *Application Specific Integrated Circuits* are in this case specialized circuits developed to hash inputs to the SHA256 hashing algorithm efficiently. This has risen the question as to if there is an alternative way to gain consensus in blockchain networks that is less power consuming and thereby a more sustainable solution.

1.2 Thesis Goals

The aim of this thesis work is to do a comparative study of different approaches to PoS algorithms in a blockchain network. Since there is a general lack of documentation regarding PoS algorithms, documenting different algorithms is also part of the goal. Generally how attacking the consensus in cryptocurrencies differs between different PoS algorithms and the PoW algorithm called HashCash. In particular how this will change the decentralization properties in both theory and practice. Four different PoS algorithms will be examined and a comparative study will be carried out. The amount of academic documentation regarding PoS algorithms are limited and one objective of the thesis is to document advantages and limitations with implementations of different PoS models.

The thesis **aim** to answer the following.

- How does different PoS algorithms differ in the security aspect?
- Will the PoS implementations in different blockchain applications differ in the decentralization perspective?
- How will the PoS algorithms scale in different blockchain settings?

Chapter 2

Fundamental Cryptographic Concepts

The Bitcoin protocol is built upon a complex data structure called the **blockchain** and in order to understand the technology, we need to address some fundamental cryptographic concepts.

2.1 Symmetric and Asymmetric encryption

Cryptography can be divided into two larger categories, symmetric and asymmetric encryption. Symmetric encryption is as the name suggests symmetric, in the sense that it uses the same cryptographic key for both the encryption and decryption operations.

Asymmetric encryption (can also be referred to as public-key cryptography) is a cryptographic encryption system that uses a key pair. Consisting of the public key which can be distributed to any entity willing to send you encrypted data. The asymmetric encryption scheme also consists of a private key that should remain secret and is the key used to decrypt the data encrypted by the public key.

2.2 Digital Signature

Digital signature schemes are designed to be the digital counterpart of handwritten signatures [6]. Using mathematical schemes digital signatures are developed to demonstrate the authenticity of digital data. If implemented correctly a valid digital signature scheme gives the receiver a reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in the transmission process (integrity) [5].
The digital signature scheme can be divided into two parts, the *secret key* of the signer consisting of a byte sequence that should remain hidden. The format of the secret key depends on the cryptographic algorithm. Additionally, there is a *public key* relating to that specific private key. An unbiased party must be able to verify the digital signature without requiring the signers' private key. Digital signatures in blockchain applications use asymmetric digital signatures with *asymmetric* meaning that each participant selects a key pair consisting of a

private key and a related public key. The participant keeps the private key secret whilst making the public key public for other participants to use it to verify the digital signature.

There are numerous types of digital signatures, one of the first is called *Digital Signature Algorithm* with security based on computational intractability of the *discrete logarithm problem* (DLP) in prime-order subgroups of $\mathbb{Z}_p$. The Bitcoin blockchain uses elliptic curve cryptosystems (ECC) in order to compute digital signatures and this protocol is called **ECDSA**. Compared to DSA, the subgroup $\mathbb{Z}_P$ in ECC is replaced by the group of points on an elliptic curve over a finite field. This results in this scheme being computationally intractable of the elliptic curve discrete logarithm problem (ECDLP) as a mathematical basis.

2.3 Cryptographic Hash Functions

The word *hash function* denotes a function that takes a string of arbitrary length as input and compresses it to a string of fixed length, allocating the output as uniformly as possible. It is often used to track the records for a file and the term has also been widely adopted into crypto as *cryptographic hash function* [8].
A hash function maps data of arbitrary size to data of a size specified by the function. This function is usually constructed by iterating a compression function on the input message.
An application for cryptographic hash functions can be to verify the authenticity of a short byte sequence or hashcode using a secret key called MAC (Message Authentication Code) and MDC (Message Detection code) for functions that do not use a secret key.
Message Authentication Code (MAC) stems from US standards and can be a bit misleading since it is not actually a code. The Message Detection Code can be divided into two classes depending on the requirements on the security, namely the *one-way hash function* or weak one-way hash function and the *collision-resistant hash function* or strong one-way hash functions. The following informal definitions of *one way hash functions* and *collision resistant hash functions* was given by R.Merkle and M.Rabbin [8]. In the following definitions, the hash function will be denoted with h, and its argument, i.e., the information to be protected with X. The image of X under the hash function h will be denoted with $h(X)$ and the secret key with K.

One way hash function (OWHF)

A ***one-way hash function*** *is a function h satisfying the following conditions:*

1. *The description of h must be publicly known and should not require any secret information for its operation.*
2. *The argument X can be of arbitrary length and the result h(X) has a fixed length of n bits*
3. *Given h and X, the computation h(X) must be "easy".*

Collision resistant hash function (CRHF)

*A **collision resistant hash function** is a function h satisfying the following conditions:*

1. *The description of h must be publicly known and should not require any secret information for its operation.*
2. *The argument X can be of arbitrary length and the result h(X) has a fixed length of n bits*
3. *Given h and X, the computation h(X) must be "easy".*
4. *The hash function must be one-way in the sense that given a Y in the image of h, it is "hard" to find a message X such that h(X) = Y and given X and h(X) it is "hard" to find a message X' $\neq$ X such that h(X') = h(X)*
5. *The hash function must be collision resistant: this means that it is "hard" to find two distinct messages that hash to the same result.*

HashCash

"HashCash was originally proposed as a mechanism to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers in May 1997" [11]. The algorithm was designed for the user to compute a CPU cost-function token. This token will be used as PoW.

The HashCash Cost Function

The term *client* is used to refer to the user participating in the HashCash protocol computing a token using a cost-function denoted **MINT()**. The term mint for the cost-function is used because of the cost of creating tokens and the analogy to minting physical money. The function used to check if the value of the token is within the correct range is called **VALUE()** which aborts the protocol if the token is not within the correct range. In order to calibrate the amount of work corresponding to the correct amount of time on average for a user to mint a token, the cost function is parameterized. There exist interactive cost-functions where centralized servers issue challenges to the client. These cost-functions are however not in the scope of this thesis. In non-interactive cost-functions, the client chooses its own challenge or random start value in the **MINT()** function.

$$\begin{cases} T \leftarrow \mathbf{MINT(s,w)} \\ V \leftarrow \mathbf{VALUE(T)} \end{cases}$$

2.4 Merkle Trees

Thirty years ago Ralph Merkle produced multiple one-time signatures associated with a single public key using complete binary trees. Since the introduction, a complete binary tree with a k bit value associated to each node such that each interior node value is a one-way function of the node of its children has been defined as a Merkle tree. [7]. The Merkle trees are specifically designed so that a leaf value can be verified efficiently assuming the knowledge of a publicly known root value and the authentication data of the leaf. To authenticate data in the form of nodes at each height is needed, where these nodes are the siblings of the nodes on the path connecting the leaf to the root. The task to solve is called the Merkle tree traversal problem and involves authenticating a transaction in the most efficient way possible.

2.4.1 Definitions

The following definitions given by M Szydlo [7] will help with the understanding of Merkle tree traversal.

Binary trees A complete binary tree T is said to have *height* H if it has (2^H) leaves, and $2^H - 1$ interior nodes. By labeling each left child node with a "0" and each right child node with a "1", the digits along the path from the root identify each node. Interpreting the string as a binary number, the leaves are naturally indexed by the integers in the range 0, 1,... $2^H - 1$. The higher the leaf index, the further to the right that leaf is. Leaves are said to have *height* 0, while the *height* of an interior node is the length of the path to a leaf below it. Thus, the root has height H, and below each node at height h, there are 2^h leaves.

Each block in the blockchain contains a *Merkle root* which is the root of all the transactions hashed together in pairs of two forming a Merkle tree. This is also known as a binary hash tree and is a data structure to efficiently verify that a transaction exists in a certain block as well as checking the integrity of data.
In the hashed binary tree the leaf nodes are transactions and all the none leaf nodes are hashed data (transactions) results from its two children. To build a hashed binary tree the pair of nodes are recursively hashed until there is only the root left. Bitcoin uses the hash algorithm SHA256 applied twice. Merkle trees are very time efficient and if you have N transactions in a Merkle tree (block) you can check to see if the transactions exist in the tree with at most $\mathcal{O}(2 \log_2 n)$ calculations. In the following example, we have four different transactions we call 1, 2, 3, and 4 which forms a Merkle tree as we can see in Figure 2.1. The transactions themselves are not stored in the tree but their data is hashed and the resulting hash is stored as H1, H2, H3, and H4.

```
H1 = SHA256(SHA256(Data of transaction 1))
```

The parent nodes of the leaves are then created by concatenating the two leaf nodes, transaction 1 and 2 for example consisting of two 32-byte hashes which result in a 64-byte string which is hashed using SHA256 **twice**.

```
H12 = SHA256(SHA256(H1 + H2))
```

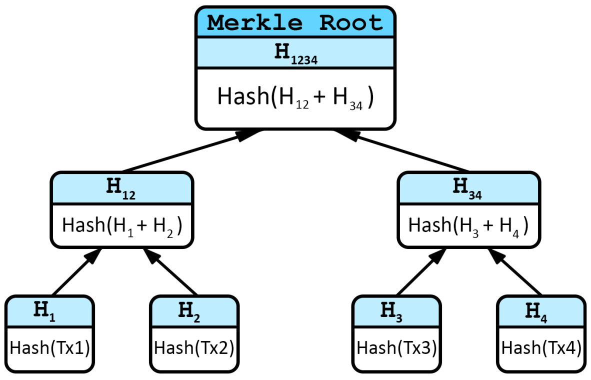

Figure 2.1: A Merkle tree with four transactions.

This process continues recursively until there is only the Merkle root left, this resulting value of a **32-byte** hash is stored in the block header and summarizes all the transactions in the block. There has to be an even number of leaf nodes since the Merkle tree is a binary tree if there is an odd number of transactions in the block the last transaction will be duplicated and hashed and concatenated with itself. Since we are using double SHA256 as the hash function, no matter how many transactions there are in the tree and how large the tree is, the Merkle root is always going to be 32-bytes.

Merkle path is used to prove that any specific transaction is included in a block, this is done by producing $\log_2(N)$ 32-byte hashes following the path to the root of the tree. The binary tree data structure is important since the base-2 logarithm time complexity increases much slower than other types of data structures resulting in proving a single transaction existing in a block is effective even though there are thousands of transactions in a block. Proving a Merkle path is simple, in our example in Figure 2.1 to prove that H3 exists in the block one would request H4, H34, and H12 from a full node.

Chapter 3

The Bitcoin Blockchain

A key challenge when building a blockchain data structure containing immutable data is distributed consensus in the network. When building a decentralized monetary network, what mechanisms are ensuring that a user can not spend her currency twice? How can we ensure that you can only spend a transaction with the currency that you are in possession of only once? The network must reach distributed consensus, meaning that after the protocol terminates all correct nodes decide on the same value (block for example) with the value being proposed by a non-malicious node. In order to understand how this problem is solved, we need to understand how Satoshi designed the transaction data structure, which is an **essential** part of the blockchain.

The blocks in the Bitcoin blockchain contains a hash pointer to the previous block in the chain as well as a timestamp and transaction data. By the use of a *merkle tree hash* the blockchain will be alerted to any modification of the data. All the blocks in the chain of transactions link back to the **genesis block** which is the first block of transactions ever created. When used as a distributed ledger a blockchain is managed by a peer-to-peer network collectively choosing the next block of transactions to be validated and appended to the blockchain. A consensus algorithm in the blockchain aims to achieve distributed consensus. To put this in more trivial terms, collectively choosing the next valid block of transactions among the proposed blocks propagated to the network. This is the challenging part of the system, to achieve consensus in the system regardless if there are malicious nodes on the network. Consensus algorithms are a huge part of the blockchain ecosystem and despite that, there are limited documentation and research on the subject. To understand the Bitcoin blockchain architecture and understand it is down to the core, some key components need to be covered.

3.1 Bitcoin Transactions

The Bitcoin blockchain is structured as a back-linked list of blocks of transactions ordered chronologically with the oldest blocks in the beginning. Each block in the chain links back to the previous block. This can be seen as a stack with the genesis block on the bottom building up to the most recent mined block. This can be visualized as stacking plates on top of each other. The genesis

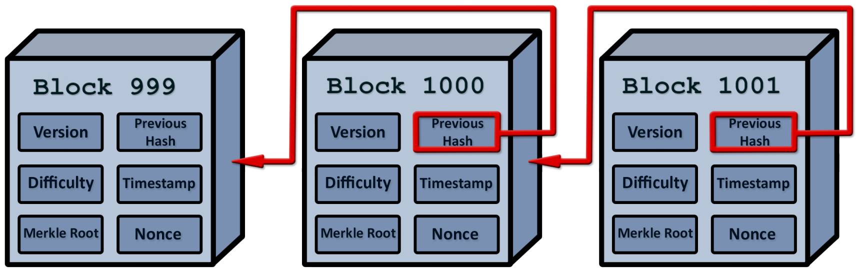

Figure 3.1: The bitcoin blockchain.

block is the initial plate at the bottom with the transactions engraved on the plate. As you stack more plates on the stack, the weight of the stack will increase. It is not possible to alter the engraving of transactions on a plate but it is, however, possible to replace one plate with another plate with transactions engraved. Since the transactions are back-linked if you replace one plate, you also need to replace the plates on top of that plate. The more plates that are stacked on top of one plate, the more weight that plate meaning that the security for that plate of transactions increases and the probability that it will be replaced decreases. Each block header has the following fields:
Version: This number keeps track of proposed upgrades to the blockchain.
Prev_Hash: Pointer to the hash of the previous block header in the blockchain.
Merkle root: The hash of the blocks Merkle root which contains all transactions in the block.
Timestamp: Seconds from Unix epoch indicating when the block was created.
Difficulty target: The PoW difficulty for this specific block
Nonce: A counter for the PoW Hash Puzzle

The most elementary function of the Bitcoin blockchain is the possibility of sending bitcoins to other users of the network. These transactions are data-structures encoding the data of the transfers. All the transactions and data of the transactions in the Bitcoin blockchain is public and can be accessed by anyone connected to the Bitcoin network. The transactions in the bitcoin protocol are created in blocks, then signed with the private key corresponding to the address, and broadcasted on the bitcoin network in order to be propagated to every node. Finally verified in a block by the miner that solves a mathematical computing problem called *hash puzzle*.

Once the transactions have been appended to a block on the blockchain and verified by a subsequent number of blocks it will with high probability stay on the blockchain forever and is accepted as **valid** by all participants. Building blocks of the Bitcoin protocol needs to be explained further before it is possible to go into more depth about the technical aspect of transactions.

3.1.1 UTXO - Unspent Transaction Output

The US dollar is divisible into smaller units called cents (1$ = 100cent). The same applies to bitcoins with the smallest unit of the Bitcoin currency is called *satoshi* where 1 BTC = 100,000,000 sat. In the blockchain, the Bitcoin full nodes keep track of every single transaction in the form of available transactions i.e. tokens that certain addresses are able to spend. This set of transactions is known as UTXO - unspent transaction output and these are made up by the sum of all incoming transactions. This can be seen as the balance on the Bitcoin address which can be accessed by the use of Bitcoin wallet clients. When making a transaction, one address can contain one or multiple UTXO and in order to make a transaction, you have to spend a whole UTXO. For example, if Alice has one UTXO of 15 BTC and wants to send 10 BTC to Bob, this transaction needs to be split into at least two outputs with one transaction containing 10 BTC to Bob and the other transaction 5 BTC back to herself called the change address.

3.1.2 Script

Both input and output addresses are in fact scripts, in the simplest case the output script specifies the public key and the input script specifies the signature that can only be made if the creator knows the private key corresponding to the output script's public key. In the input, there is a reference to an output from a previous transaction. In the validation processes, the two scripts get concatenated together and if the script can run without any errors it's considered a valid transaction. Let's look at a simple script in bitcoin (this is the same script generated in the transaction example) where a sender Alice specifies the public key of the recipient Bob and in order for Bob to redeem the coins has to specify a signature using that particular public key.

```
Script example: <sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>
    OP_EQUALVERIFY OP_CHECKSIG
```

The first two elements namely the <*sig*> and <*pubKey*> are values that are pushed onto the stack which contains the digital signature and the public key in conjunction with the private key to generate that particular digital signature. All data instructions are pushed onto the stack. OP_DUP (0x76) duplicates the top element on the stack resulting in the stack now containing from bottom to top: <sig><pubKey><pubKey>. OP_HASH160 pops the top value on the stack and computes a HASH160 of it so the top value becomes <pubKeyHash>. The next data instruction <pubKeyHash> was specified by Bob (the sender of the coins) and this is pushed onto the stack.

OP_EQUALVERIFY(0x88) makes sure that the hashes of the two elements on the top of the stack are equal and OP_CHECKSIG (0xac) pop the public key and the signature in order to validate the signature for the transaction returns TRUE if there is a match.
There are no loops or flow control since there are no states in the bitcoin scripting language. Therefore the bitcoin scripting language is often referred to as *Turing incomplete*. Meaning that we know that the scripts have limited execution time and do not loop forever, this is to mitigate DoS attacks on the network.

3.1.3 Serialization

Transmitting data structures bit for bit in a byte stream requires serialization and this is the conversion process that makes this transmission possible. Since storing the transactions as byte streams would require parsing every time you needed to access a single field, data structures are instead used. The backward process that converts the byte stream representation to an internal data structure of a library is called *deserialization*.

3.1.4 Generating a transaction

A Bitcoin transaction is visualized by generating our own transaction in the bitcoin client. Since we do not need to interact with random nodes in this setting, we will use Bitcoin Core's **Regression Test Mode**. This will create a private blockchain disconnected from the real Bitcoin network where the user decides the exact time of mining a new blocks of transactions by the command line.

```
./bitcoin-cli -regtest generate 1000      #Mined 1000 blocks
./bitcoin-cli -regtest getnewaddress
New bitcoin address: muRgEHUxooLxDhDHbXybGugMY536hcSToy
```

Primarily an arbitrary number of blocks are mined in order to create the blockchain and build a chain of blocks. This is done in regression mode by *generate 1000*, 1000 blocks are being mined. A bitcoin address is created with the command *getnewaddress* corresponding to the address *muRgEHUxooLxDhDHbXybGugMY536hcSToy.*

```
./bitcoin-cli -regtest sendtoaddress
    muRgEHUxooLxDhDHbXybGugMY536hcSToy 1 #Sends 1 btc to the newly
    generated address
./bitcoin-cli -regtest generate 1         #Mine a block in order
    to send the tx
[
  "2428cbcc46d81e19ecfc0df8b783ebb00079d47dc3a1ba7e4ea10a2acefa740a"
]
#The hash of the mined block
{
  "hash":
      "2428cbcc46d81e19ecfc0df8b783ebb00079d47dc3a1ba7e4ea10a2acefa740a",
  "confirmations": 1,
  "strippedsize": 419,
  "size": 455,
  "weight": 1712,
  "height": 1001,
  "version": 536870912,
  "versionHex": "20000000",
  "merkleroot":
      "59d653253b32b18da6d4232a277f48b5e5d3a7a9016cc3eacf306fa5e21b11da",
  "tx": [
    "6e13c47d312bea786b8cc176ef31626576717bce5d14365bcf41a345809a79fb",
        #Coinbase tx
    "e10357f144262c9a3b60ae908a51be797bf9d3a5152e6e0fa814d589308643b4"
        #Our tx
  ],
  "time": 1506261025,
  "mediantime": 1506260920,
  "nonce": 0,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000
  000000000000000000000007d4",
  "previousblockhash": "3c3e2b5d3819f25a7089ce163db8f056f040a92
  707223088a948d919ae147033"
}
```

A new transaction is created and one bitcoin is sent to the newly generated address. This transaction is broadcasted to the network but in order to process the transaction the next block **#1001** has to be mined (otherwise it will not be included in the blockchain) and this is done with the command *generate 1*. This returns the hash of the mined block. Displaying the different data fields of the block, indicating for example block number 1001 since the height is 1001. In the "tx"-data structure there is an array with 2 transactions, the first transaction being the *coinbase* transaction, this is the reward for the miner that solved the hash puzzle which is going to be covered more extensively later in the thesis. The second transaction is the 1 bitcoin that was sent using the command line.

```
./bitcoin-cli -regtest getrawtransaction
    e10357f144262c9a3b60ae908a51be797bf9d3a5152
e6e0fa814d589308643b4 1
```

```
#Returns the transaction with corresponding tx-id, 1 returns the
    transaction decentralized

{
  "txid":
      "e10357f144262c9a3b60ae908a51be797bf9d3a5152e6e0fa814d589308643b4",
  "hash":
      "e10357f144262c9a3b60ae908a51be797bf9d3a5152e6e0fa814d589308643b4",
  "version": 2,
  "size": 191,
  "vsize": 191,
  "locktime": 1000,
  "vin": [
    {
      "txid":
          "5c5159d8d421fbee26ed3fcff6b934ee68f1ef5317699623bd90558548df003a",
      "vout": 0,
      "scriptSig": {
        "asm":
            "304402202ad79eaed6c10ad9ddc519b0c1df80d1525c533cbc1a9dc80a6911795e1b
        41c402201f83a8c63847e330195ffbf03ba5f001a
        541e17df03ac9e92b537a6ec674927a[ALL]",
        "hex":
            "47304402202ad79eaed6c10ad9ddc519b0c1df80d1525c533cbc1a9dc80
        a6911795e1b41c402201f83a8c63847e330
        195ffbf03ba5f001a541e17df03ac9e92b537a6ec674927a01"
      },
      "sequence": 4294967294
    }
  ],
```

To dig deeper into the transaction a *getrawtransaction* with corresponding transaction hash returns data structures of the whole hash. The script is made up of two components, a signature, and a public key. In the **scriptSig-script** there is an assembly decoding of the script as well as a hex representation. Since the input script is the hash of a public key, this will not make sense in assembly.

```
  "vout": [
    {
      "value": 1.00000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160
            989245529326b5246f4fcfeabd16525850a5101f OP_EQUALVERIFY
            OP_CHECKSIG",
        "hex": "76a914989245529326b5246f4fcfeabd16525850a5101f88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "muRgEHUxooLxDhDHbXybGugMY536hcSToy"
              #BTC-address format base58
        ]
      }
    },
    {
      "value": 0.56246160,                          #This is the
          change-address part
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160
            9bb56f525804c39760d5d59d1d44084dfd6d879f OP_EQUALVERIFY
            OP_CHECKSIG",
        "hex": "76a9149bb56f525804c39760d5d59d1d44084dfd6d879f88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "muiGP6spyRZnf3t4w5TksJKdPXitkFvh4H"
        ]
      }
    }
  ],             #Unserialized tx in hex
  "hex":
      "02000000013a00df48855590bd2396691753eff168ee34b9f6cf3fed26eefb2
  1d4d859515c000000004847304402202ad79eaed6c10ad9ddc519b0c1df80d1525c53
  3cbc1a9dc80a6911795e1b41c402201f83a8c63847e330195ffbf03ba5f001a54
  1e17df03ac9e92b537a6ec674927a01feffffff0200e1f505000000001976a9
  14989245529326b5246f4fcfeabd16525850a5101f88ac903f5a0300000000
  1976a9149bb56f525804c39760d5d59d1d44084dfd6d879f88ace8030000",
  "blockhash":
      "2428cbcc46d81e19ecfc0df8b783ebb00079d47dc3a1ba7e4ea10a2acefa740a",
  "confirmations": 1,
  "time": 1506261025,
  "blocktime": 1506261025
}
```

3.1.5 Coinbase

The **coinbase transaction** is the first transaction in every block. This amount in the Coinbase transaction started out to be 50 bitcoins and is reserved as the reward for the miner that solves the hash puzzle. When a miner solves the challenge(hash puzzle), the miner needs to claim the reward in the Coinbase transaction. This is done by creating a transaction of the current amount of mining reward with the vin pointing to a null transaction with hash 0x0 and n=0xFFFF. . . .

3.2 The Bitcoin Network - Peer to Peer architecture

All nodes in the network are peers to each other and equal in the sense that there exists no *super node* that has more influence over the network than the other nodes. The network is interconnected in a mesh network with a flat topology. No server, no centralized service and, no hierarchy in the network. The decentralization properties of the blockchain can only be achieved by the flat P2P consensus network. The collections of nodes running the bitcoin P2P protocol makes up the *bitcoin network*. There are four different function nodes in the bitcoin network can support. These functions are *wallet*, *miner*, *full blockchain node*, and *routing*.

The nodes in the network can have any number of these functionalities but all nodes in order to participate in the network have the routing functionality. All nodes propagate and validates blocks in the blockchain as well as connect to other nodes. In order for a transaction to be relayed the transaction must be valid according to the current blockchain. It also exists a whitelist where scripts that are legal to execute are listed, in order for the nodes to relay the transaction the corresponding script has to exists in this whitelist.

Full nodes stores a complete up to date copy of the public blockchain making the full nodes responsible to verify incoming transactions.

Mining nodes exists with the purpose of solving the PoW hash puzzle challenge and then propose a new block to the blockchain. Mining nodes are often built up by specialized equipment with the sole purpose to solve the given hash puzzles as quickly as possible.

Wallets are made to keep track of the balance corresponding to the sum of the UTXO of every user's Bitcoin address, this can be a full node but is often a light-client called SPV.

When a node wants to connect to the Bitcoin network it needs to find at least one other node in the Bitcoin network and connect to it, this node can be selected at random. In order for a node to connect to another node a TCP connection has to be established, usually to port 8333 as this is the port known to be used by bitcoin. When the nodes have established a TCP connection a "handshake" will be initiated by transmitting a version message containing the basic information for identification. In order to connect to peers, a node can use *DNS seeds* which are IP addresses of bitcoin nodes in the form of DNS servers.

3.2.1 SPV - Simplified Payment Verification

Because of storage and power constraint on different devices (the blockchain is approximately 173 GB late June 2018 [22]) and therefore a *Simplified Payment Verification* (SPV) method is used which mathematically derives the transactions in the form of a Merkle path in order to significantly decrease required storage. These clients are often referred to as lightweight clients or SPV clients. Only the headers of each block are downloaded by the SPV nodes instead of all the transactions resulting in a chain of blocks without the actual transactions that in size are in the magnitude of 1000 times smaller than the full-size blockchain. To verify a transaction, the SPV provides a Merkle path making it possible to verify that a given transaction is in a specific block. Since the SPV nodes only download the headers of each block they cannot construct a full picture of all the UTXOs in the blockchain instead they rely on peers to provide partial information on the relevant parts of the required information on demand.

Full blockchain nodes construct the full blockchain by their height downloading every single transaction in the blockchain. An SPV node instead verifies all the blocks but not every single transaction, this is done with every block header and link that chain to the desired transaction. A full node can create a database of UTXO and can thereby see the validity of each transaction, an SPV node on the other hand mathematically derives a link between the transaction and the block that contains it by providing certain Merkle nodes as described in the *Merkle Tree* section.

3.2.2 Memory Pool

Full nodes on the bitcoin network often keep a list of unconfirmed transactions propagated on the blockchain, called the *mempool* or transaction pool [9]. This temporary list contains transactions propagated to the network but not yet included in any block on the blockchain. Wallet clients keep track of the mempool to track incoming transactions to the wallet that are sent by the user making the payment but not yet confirmed and included by a miner on the network.

A separate pool of orphaned transactions is implemented in some full nodes. If the input of a transaction reference a transaction that is unknown, this transaction is stored temporarily in the orphan pool. The reason for this is that the input transaction referenced may not have been propagated and included in the network yet. When transactions are added to the transaction pool the orphan pool is checked for any orphans referencing to a parent transaction recursively and matching orphans are validated. The orphan transactions are removed from the orphan pool and added to the transaction pool upon matching and validation.

When investigating the workload of the bitcoin network, the mempool is a good place to look. Obviously, when the mempool contains a large number of transactions it indicates that newly mined and produced blocks on the

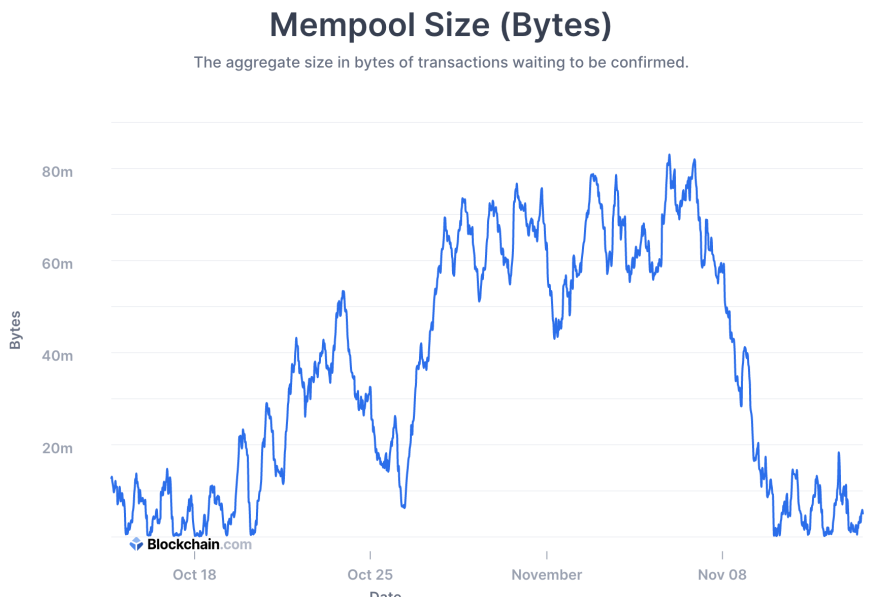

Figure 3.2: Mempool in the bitcoin network in november, source:blockchain.com

blockchain are at capacity, and the transaction is chosen to be a part of the next block will likely be chosen by the transaction fee included in the transaction and transaction fees will increase. In Figure 3.2 we can clearly see that the mempool has a high fluctuation.

3.2.3 Scalability in the Bitcoin network

In 2010 Satoshi added a block size limit of $1MB$ [20] with the motivation to prevent miners from appending large blocks of spam. In recent years as the bitcoin network has grown, the blocks have more frequently reached maximum transaction capacity (as can be seen in Figure 3.2), increasing the average transaction fees on the Bitcoin network. This has started an ongoing debate regarding the block size in the bitcoin blockchain and whether it should be increased (or even decreased as some propose). As previously referenced the bitcoin blockchain was approximately 177 GB in late June 2018 with the block size of 1MB. If the block size were to be increased to 2MB the blockchain would in worst-case increase in size with two-fold the speed. Larger storage requirements for running full nodes would be the result of this and the blockchain would potentially not scale to every user of the network if the storage requirements for running a full node is larger than a typical user in the network is able to provide.

The scaling debate can be divided into two parts. Namely, the number of transactions per second and the number of users able to run full nodes since every transaction on the blockchain is validated by every full node in the network to keep the protocol in a decentralized manner. The limit on blocks in size of data storage and not in transactions, therefore it is hard to pin an exact amount of transactions a 1MB block can contain but upon looking at the average number of transactions in a block around 2017-09-26 (since there were a large number of transactions at that date) [23] the number of transactions lies around 2000. Given that a block is produced on average every 10 minutes, the number of transactions is calculated by:

$$\frac{Transactions}{Second} = \frac{2000}{10 * 60} = 3.333\ldots \tag{3.1}$$

An easy solution proposed to increase the throughput in blockchain applications is an increase in block size, by doubling the size of the blocks the number of transactions per second relayed on the blockchain will on average also be doubled. Here the second part of the scalability term enters, upon increasing the block size from 1MB to 2MB the blockchain will also increase two-fold in size making it more expensive to run a full node. Originally the bitcoin protocol was designed to scale to more or less every user on the internet, if you had a normal computer you were supposed to be able to run a full node. The full nodes are the only true validators of the network.

There are a number of proposed *side* and *off-chain* solutions to tackle this scaling problem but these are out of the scope of this thesis. [17]

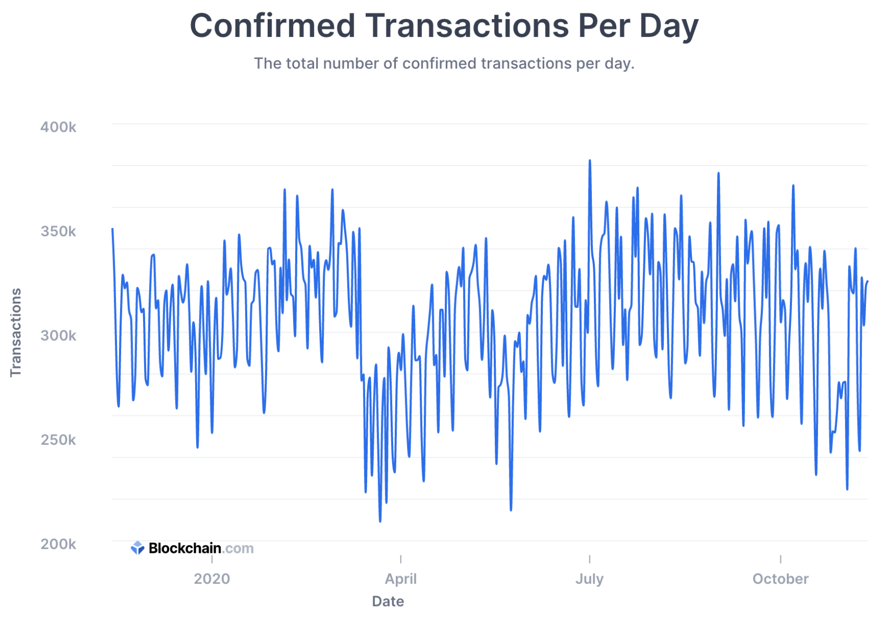

Figure 3.3: Confirmed transactions 2019 source:Blockchain.com.

Chapter 4

Consensus Algorithms In Blockchain Applications

4.1 The Byzantine Generals Problem

To understand how consensus is achieved in distributed networks namely blockchain applications a scientific model to theorize this problem is dissected [10]. This research is built upon that reliable computer systems must be able to handle the failure of one or more of its components without the whole system crashing. This field of research came to be particularly valuable for consensus algorithms in blockchain applications. A failed component may send conflicting information to different parts of the system and coping with this type of problems is expressed abstractly as the *Byzantine Generals Problem.* The problem is formulated in the following way:

Image several divisions commanded by their own general is camped outside an enemy city with the only mean of communication being oral messages. They must decide upon a common strategy and plan of action after observing the enemy assuming some of the generals might be traitors. The traitors are trying to prevent the loyal generals from reaching an agreement. The algorithm of the generals must guarantee that:

1. All honest generals decide upon the same plan of action.

The traitors can do as they wish but all the honest generals must do what the algorithm tells them to do and the algorithm must guarantee condition 1 regardless of what the traitors do. The loyal generals should reach an agreement as well as agree upon a reasonable plan therefore we want to ensure that:

2. A small number of traitors cannot cause the honest generals to adopt a bad plan.

Since it is very hard to tell precisely what a bad plan is, condition 2 is hard to formalize but instead of figuring out what a bad plan is our scope is how generals reach a decision.

The enemy is observed by each general and he communicates his observation to the other generals. v(i) is denoted as the **information** communicated by the ith general and each general uses some method for combining all the information **v(1),v(2)...v(n)** into a single plan of action where n is the number of generals. By using the same algorithm of combining the information we can achieve condition 1 and condition 2 is achieved by using a robust method.

If the scenario is that the general has the binary option to attack or retreat, $v(i)$ is General i's option after observing the enemy of whether to attack or not and the final decision can be based upon a majority vote among them. In this case, the traitors can affect the decision only if the decision among the honest generals was a close call beforehand in this case neither decision could be called bad. That the generals communicating their values $v(i)$ to each other is assumed by this method. One general sending $v(i)$ messages to each other general will not work because condition 1 states that every honest general obtain the same values $v(1) \ldots v(n)$ and a traitorous general may send conflicting values to different generals. For condition 1 to hold the following must be true:

1.1 Every honest general must obtain the same information $v(1) \ldots v(n)$

This implies that a general cannot use a value $v(i)$ obtained directly from general i since a traitorous general i may send different values to different generals. This means that condition 1.1 must be approached with caution so that the generals do not choose a value $v(i)$ different from the value that was sent by general i despite of general i being honest. This can not occur if condition 2 is to be met and we, therefore, have the following requirement:

1.2 If the ith general is honest, then the value that he sends must be used by every honest general as the value of $v(i)$.

This can be phrased in the following way:

The Byzantine Generals Problem

A commanding general must send an order to his n-1 lieutenant general such that

1: All loyal lieutenants obey the same order. Interactive consistency
2: If the commanding general is honest, then every honest lieutenant obeys the order he sends.

4.2 Proof of Work

Proof of work is a widely used consensus algorithm in blockchains as of today and this thesis will focus on the Bitcoin PoW algorithm called HashCash [11]. The idea behind PoW is that it is a piece of data that is difficult to produce in terms of computing power and time but easy to verify by the use of a cryptographic backdoor. Producing the PoW can be done for example by hashing a large number of block headers.

HashCash in the Bitcoin Blockchain

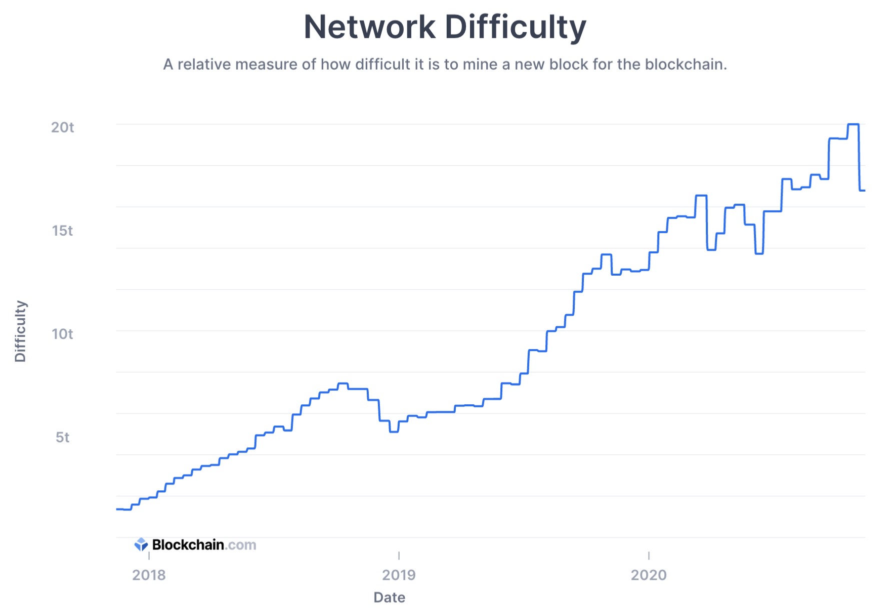

Figure 4.1: Mining difficulty over the last three years. source:blockchain.com.

The miners in the blockchain aim to find a hash value input which corresponds to a hash function output value smaller in magnitude than the challenge (also called the target) and this is called a *hash puzzle*. Each proof will be generated on average every 10 minutes and the difficulty to find the proof is adjusted according to how fast the miners are finding the proof. The mining difficulty is recalculated every two weeks on average (every 2015 block), every node recalculates this difficulty independently of each other. Figure 4.1 show the difficulty in the Bitcoin network over the last three years.

```
next_difficulty = previous_difficulty * (2 weeks) / (time to mine
    last 2016 blocks)
On average 10 minutes per block in 2 weeks results in 2016 blocks.
```

How does this Hash Puzzle work?
The miner is hashing the current block header as can be seen in Figure 4.2 with the hash of the previous block and the Merkle tree for all transactions as well as with a nonce starting on 0x0000... and iterating through all possible values for the 32-bit integer. However, this is often not enough for finding the solution to the hash puzzle. In this case, you have to change a parameter in the Coinbase transaction, there is a nonce parameter in the Coinbase transaction that can be incremented thus changing the whole hash given you a range of 32-bits more nonce values in the block header to try in the search for the solution.

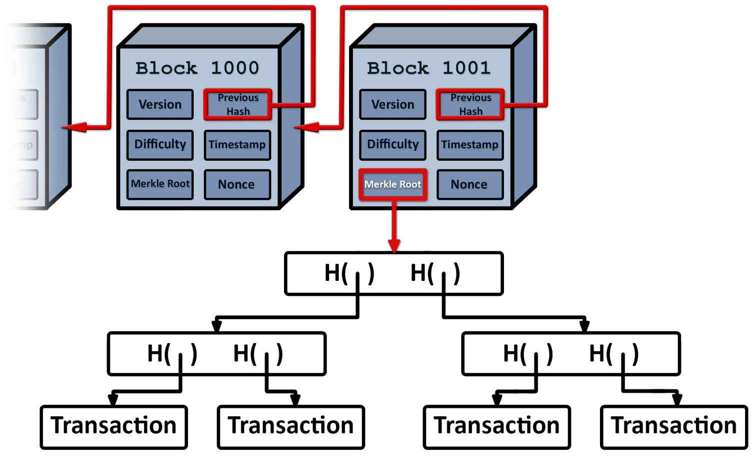

Figure 4.2: Hash Puzzle.

If two or more miners solve the puzzle at the same time a *fork* occur in the network. Since there is a delay in relaying the blocks to all full nodes in the network, two full nodes in the network might have distinct ledgers where both ledgers are legit with a correct proof to the Hash Puzzle. The reason for this that propagating a block of transactions to the network involves a latency before reaching every full node in the network as there is no way to do this instantaneously. Note that this is not necessarily malicious users attacking the network but merely two or multiple miners solving the mathematical puzzle at the same time, and since the transactions need to be propagated to every full node on the network it is not uncommon that the whole ledger is not fully synchronized. The result of this is that the ledger is split into two or more branches, called forks. In order to solve this problem and determine the "right" fork, the Bitcoin client chooses the fork that has proven the most accumulated work. Initially, to incentivize block proposers, the miners are given a reward if their block ends up in the public blockchain. The reward was 50 bitcoins when the Bitcoin protocol launched but is halved every four years and is currently 6.25 bitcoins. Since the mining reward is halved every four years an additional incentive is required. The second incentive is the transaction fee the miner receives for each transaction in the block. Each transaction contains a list of inputs and a list of outputs and the transaction fee is the sum of the input minus the sum of the outputs.

4.2.1 Mining Pools

The development of specialized mining equipment (ASICs) has increased the difficulty factor, rendering ordinary computers practically obsolete when it comes to mining due to the relatively small return in contrast to electricity cost.

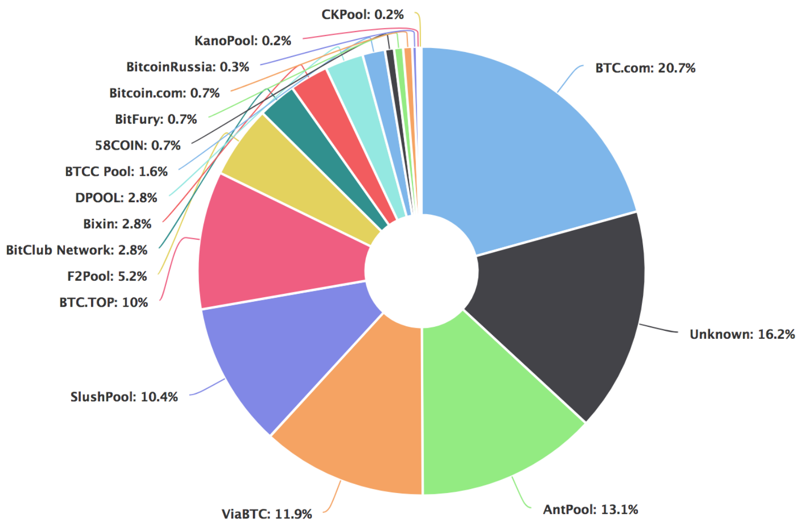

Figure 4.3: Miningpool Hashrate Distribution In The Bitcoin Network in 2017, source:https://www.blockchain.com

Therefore, mostly ASICs and specialized mining rigs (usually composed of several GPUs) are profitable mining equipment. With a large number of actors mining and the corresponding large difficulty factor, it will be hard for a small miner to predict when they will receive block rewards. Possibly making it difficult to plan the extrinsic costs of the mining operation (electricity, storage, etc) in a periodic manner. Therefore, mining pools were created where actors accumulate their computing hardware whilst splitting the reward proportionally to the amount of computing power you provide to the pool. The larger amount of accumulated computing power, the more frequent block rewards making the income for the miners more predictable. As can be seen in Figure 4.3 the four largest mining pools (BTC.com, AntPool, ViaBTC and SlushPool) accumulate over 50% of the hashrate in the bitcoin network.

Chapter 5

PoS - Proof of Stake

5.1 Introduction

Proof of Stake is an alternative consensus algorithm which rather than burning extrinsic resources (computing power) as in *PoW*, involves *staking* intrinsic resources. The intrinsic resources that are put up as stake are often the native token of the blockchain where the consensus algorithm is implemented. There are different types of approaches to PoS with, the common denominator in these being the users of the network staking intrinsic value (tokens) to ensure the blocks produced are valid. PoS algorithms often implement a reward scheme for producing blocks to incentivize users to participate in the staking/block producing process. Simplified this could be compared to a bank account with an interest rate, where the user locks up an amount of stake and is slightly compensated by the block production reward. The actual staking process differs between various PoS algorithms while the actual stake almost exclusively is the native token of the blockchain network where the PoS algorithm is implemented.

As this consensus field is relatively new, the terminology of this thesis will be the following. The PoS algorithms will be divided into two categories. The first one in a historical context is called *chain-based* PoS. The chain-based approach simulates the functionality of the PoW mechanisms in the Bitcoin protocol-setting. The aim is to achieve probabilistic irreversibility within the blocks in the blockchain. It is possible to reorganize the blocks in the blockchain but in practise there is a large cost involved with it.

The second approach is the *Byzantine Fault Tolerance* approach. There are generally a number of different ways of implementing this approach but the aim is to reach **finality** within the blocks of transactions on the blockchain. Finality guarantees that the order of blocks of transactions that have reached finality historically will never be changed. The following terms are defined.

- The validators are the **active** users in the system running a full node, validating block proposals and reject block proposals violating the consensus rules.
- The stakers in the protocol are the active users involved in staking in form of locking up native tokens to ensure the integrity of the block produced.

- The block producers are the users allowed to append blocks of new transactions to the blockchain.

In the subsection PoS algorithms, there are additionally two different categories. There is *PoS* (sometimes referred to as Native PoS) where the user is in possession of the tokens are involved directly in the staking and block proposing/production procedure. The other category is called *Delegated PoS* (dPoS) where the users of the network (token holders in this case) are delegating their stake to a third party staker. Simplified this could be described as each token is proportional to one vote with the total number of tokens in circulation being the full number of votes. The user delegates their tokens (votes) to a third party that she considers trustworthy to maintain the integrity of the blockchain. Delegated PoS is often implemented in conjunction with the BFT-approach to obtain finality in the blockchain. Additionally, despite the stakers being the subset of users in the network participating in the staking process by locking up native tokens to ensure the integrity of the blockchain, the actual staking process and what is actually called staking differs between the PoS-algorithms. There is no universal definition of what exactly a stake is, therefore the staking process and what a stake is will be covered in each PoS-algorithm. The algorithms covered in the thesis were decided based on relevant projects that used PoS as a consensus algorithm, which also had at least minimal documentation as of late 2020. Within the *BFT*-styled PoS algorithm, the largest cryptocurrency implementing this approach is Ethereum, and therefore Casper FFG is covered in this thesis. Ethereum is also at the time of writing the second largest cryptocurrency in the world [4] making it suitable to cover. The EOS-project is one of the largest projects [4] by marketcap using Delegated PoS and the EOS project also had to this point the largest initial coin offering (raising of funds) [21]. Qtum is the largest project by marketcap [4] using pure chain-based PoS, a mimic of the Bitcoin PoW consensus model and is therefore relevant to cover in the thesis. The Qtum PoS model is called Mutualized PoS and is a fork from PoS 3.0 [16]. Additionally, Algorand is a promising PoS algorithm with heavy research behind the architecture of the consensus model and also introducing new cryptographic derivatives that are new to the PoS consensus models at the time of writing this report (verifiable random functions for example), therefore Algorand will be covered as well. The algorithms will be covered to gain a deeper understanding of the architecture of the protocols in various settings. Attack scenarios, misaligned incentive structures, and centralization issues can be formulated and highlighted to perform the comparative analysis.
Common attack vectors on PoS protocols are going to be covered briefly before dissecting the PoS algorithms.

5.2 Nothing at stake

Essentially the *nothing at stake* problem is the root to most attack vectors against PoS protocols. The nothing at stake problem should not be seen as an attack vector but instead a misaligned incentive structure within the actors of the network. When mining a block in a blockchain utilizing a PoW protocol, extrinsic resources are burned in the form of computing power. A block proposer is allowed to append a block to the blockchain upon proving the use of computational power making it expensive for an adversary to fork the original chain and build an alternative chain longer chain. Particularly in chain-based PoS protocols, there is no extrinsic cost involved, instead, the block proposer proves an intrinsic stake in the form of tokens on the blockchain and the computational cost namely the cost of generating a signature is negligible. Due to the lack of extrinsic cost, there is no cost involved with staking on multiple blockchains when a fork occurs. When a fork occurs regardless of wheter the fork was a coincidence or a malicious attempt to rewrite the history, the optimal strategy for any validator is to stake on every chain resulting in the validator getting their potential reward either way regardless of which fork that accumulates the most difficulty. There are different approaches of implementing PoS schemes and some implementations involve locking your stake up to ensure that you are doing right by the network, in that case, if you act maliciously, your stake might get slashed.

5.3 Long Range Attacks/History Revision Attack

One of the most debated attack vectors to PoS algorithms is the *Long Range Attack* or *History Revision Attack*. The Long Range Attack vector can be divided into two subgroups. *Synchronized Long Range Attack Vectors* assumes the user is synchronized to the legit blockchain meaning that the full node already has the correct blockchain of transactions. These types of attacks are often mitigated in a more trivial manner than the second subgroup.
The second subgroup is unsynchronized users constitutes the users not synchronized to the current legit blockchain and the focus will mainly lie on mitigating the attack on this subgroup.
The objective of a Long Range attack for an adversary is to use a large portion of private keys to fork the blockchain and create an alternative blockchain in an attempt to deceive users. The deceiving part here is that the private keys can consist of old keys that are already spent since unsynchronized nodes will accept any ledger that has proven the most difficulty. The most difficulty in this sense means where the most private keys have validated since the genesis block. When the adversary has collected enough private keys to building an alternative chain with more difficulty (derived from the weight of the stake), new users connecting to the network can be fooled to connect to the malicious chain.
A large challenge with PoS protocols involves the validator set. There is no objective way for unsynchronized nodes to determine mutations in the validator set after genesis, without the use of an anchor of trust. Depending on the setting, a malicious user could bribe a large stakeholder to move all his tokens to a new address and acquire the private key to the old address which is now empty. Assume a user with malicious intent acquired the private keys of a large percentage of the tokens in circulation at any time in history, the user is able to fork the current blockchain and start revising the history of the blockchain effectively creating an alternative blockchain. Since the adversary is in control of

a large portion of the coins in circulation and there is almost no cost involved in producing blocks (due to the lack of extrinsic cost linked to proposing a block in a PoS protocol), the adversary is able to fork the blockchain and even create a longer blockchain with larger accumulated difficulty than the original chain. Upon succeeding in this attack, it might be possible to trick new nodes (or even existing ones in some settings) connecting to the network that a maliciously produced blockchain is the real one.

5.4 Denial Of Service

A Denial of Service attack is when an entity is denied access to a service, often by the use of flooding attacks where a malicious user sends high volumes of data to a user/server often resulting in an overload of the server. The definition of a DoS attack vector is broad and multiple different uses of DoS attacks in different settings will be covered.

5.5 Sybil attack

In a permissionless distributed network any node can connect and participate in the network. If all nodes in such a system would have equal voting power of which transactions to include in the blockchain an adversary could flood the network with malicious nodes voting in favor of the adversary.
This attack is mitigated by the use of staking weight. The probability for a validator of proposing a block is proportional to the stake in the network.

5.6 Stake grinding attacks

In the setting where stake grinding attacks are possible is where stakes consist of a transaction header analogous to the Bitcoin header in the Bitcoin setting. The stake here can be viewed seen exactly like a block proposal that a Bitcoin PoW miner is calculating, however in this case instead of iterating over a large number of nounces, one stake can be composed by one UTXO and that calculates a target that is being increased by the number of tokens that are being staked. If for example, a user has a UTXO with the weight of 5, the target is 5 thus increasing the chance of hitting the target and be allowed to stake a block. Stake grinding attacks is when an adversary customize their stake and precompute the outcome of the stake in order to end up with a stake that meets the target. This attack generally only applies to chain-based PoS algorithms and is performed in various ways depending on the implementation. However, to understand the stake grinding attack, a generic chain-based PoS algorithm approach will be explained. Let us assume the outcome of a stake only contains the UTXO being staked, the staking process can be visualized as a lottery with each UTXO being one ticket. The target value is calculated based on the difficulty of meeting the target in the last round and calibrated to obtain an average block producing time of a fixed time. The target is multiplied by a factor consisting of the amount of token, if five tokens are being staked the target is multiplied by five and the user is on average five times more likely to reach the target. A stake grinding attack involves customizing the inputs to the stake to obtain an output as small as possible. Ideally, less than the target value and this can be done by *grinding* through the possible staking opportunities.

5.7 Mutualized Proof of Stake - Qtum

Mutualized Proof of Stake is the consensus algorithm the blockchain network Qtum utilizes. Mutualized PoS (MPoS) can be visualized as a lottery with the set of *mature* UTXOs as lottery tickets where every ticket consists of a hash, namely the *kernel hash* and the **weight** is made up of the amount of *QTUM* staked, the weight is proportional to the stake. To be allowed to stake tokens, the validator must have tokens that have been confirmed for a specific amount of blocks called the *staking maturity*, the tokens in a given wallet must have been confirmed by at least 500 blocks in order to be used in staking, this is to prevent so-called *stake grinding attacks*. The data of the current block being proposed by the validator is not affected by the kernel hash. In a similar way to the Bitcoin protocol, there is a target. The target is a numerical representation of the difficulty of the staking protocol, adjusted to propose a block every 128 seconds on average. The Qtum blockchain is a fork of Bitcoin and also backward compatible with Bitcoin and therefore have large similarities with the Bitcoin blockchain [14].

What exactly is a stake in MPoS?

The **stake** in MPoS is denoted by outcome of the kernel hash, which in turn is dependent of the UTXO set the staker is currently in possession of. The staking is usually performed directly in the core client but to understand the staking process, we are assuming that the staker is in charge of every step of the way of the staking. Here is a simplified step by step explanation of the staking process:

1. The staker picks a mature UTXO. The staker is required to control the private keys of the UTXOs being staked.

2. A kernel hash is created by double hashing the current block timestamp, the previous block stake modifier, and attributes of the UTXO being staked. The outcome is called *hashProofOfStake*

3. The *hashProofOfStake* is compared to the target value multiplied by the number of tokens in that UTXO.

4. If the *hashProofOfStake* is smaller than the target multiplied by the weight, the staker has a “winning lottery ticket” and is allowed to propose a block to the blockchain.

5. The staker propagates a block of transactions to the blockchain together with a coinstake transaction to collect the staking reward.

5.7.1 The Coinstake Transaction

The stakers of the MPoS protocol are incentivized by something called *the coinstake transaction.* The coinstake transaction is defined as the second transaction in each block. A coinbase transaction is the first transaction in each block and this exists in the Qtum setting to be compatible with the Bitcoin protocol which is something that goes outside the scope of this thesis.

A **coinstake transaction** is defined as a transaction that:

- *Have at least one valid vin being a mature UTXO*
- *The first vout must be an empty script*
- *The second vout must not be empty*

Furthermore, there are some rules that the coinstake transaction must abide to.

- *The second vout must be either a pubkey script, or an OP_ RETURN script that is used to make the stake transaction unspendable*
- *output* $\leq$ *(input + block reward +tx fees)*
- *The coins staked needs to be mature*
- *Even though you can use more than one UTXO in a staking transaction the first vin is the only one that affects the weight of the target i.e the probability to mint a block.*

When a staker manages to find a UTXO that solves the staking puzzle, the staker is allowed to create a staking transaction following these rules to collect the staking reward. The staking reward is four tokens and is halved every fourth year. Due to the compatibility with Bitcoin, the scripts are identical and the coinstake transaction is generating the reward for the miner, divided into portions where the reward is sent in fractions of 10 blocks after coinbase maturity time (500 blocks). The coinbase transaction can either be marked as invalid or a pubkey script that is being described earlier in the thesis, that computes if the transaction is valid and allowed to send the reward.

5.7.2 Kernel Hash

The Kernel Hash is a double SHA256-hash that concatenates these following fields into one hash and can be seen in figure 5.1. The output the kernel hash results in is also the value that is being compared to the difficulty to determine if the staker solved the staking puzzle. The output of the kernel hash is therefore denoted as the stake.

Previous Block Stake Modifier - The stake modifier of the previous block
Transaction Timestamp - The Timestamp of the UTXO
Staking Transaction Hash - The Hash of the UTXO
Staking Transaction Output Index - Index of the UTXO in the block

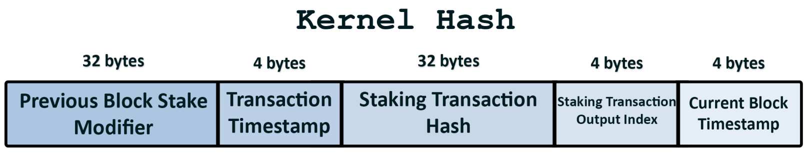

Figure 5.1: Kernel Hash.

Current Block Timestamp - Current Timestamp for the block the staker proposes. The four least significant bits in this field are zero to decrease granularity. The lower the granularity the more work is required. The four least significant bits are set to zero called the timestamp mask meaning that the block time can only be represented in 16-second timeslots and a new kernel hash (*nTarget*) can be calculated every 16 seconds. Setting the last four bits to zero results in less work for each staker, only computing on average $\frac{128}{16}$ computations per UTXO staked per block leading to the Kernel Hash recomputed on average 8 times per mature UTXO before a solution is found and a block proposed to the network.

The algorithm is designed so that at least one staker is likely to find a valid stake within the 128 seconds, if we were to think of these 128 seconds as time slots of 8 iterations (there will on average be $\frac{128}{16} = 8$ iterations). Since there in this scenario exist eight time slots, at least one staker will likely be able to propose a block in of the 8 iterations, which is every 16 seconds. If the algorithm were to be adjusted so that the frequency of computations decreased so that one computation is done every 32 seconds. The difficulty is still set to one block every 128 seconds, meaning that there will be $\frac{128}{32} = 4$ times the *hashProofOfStake* will be calculated (iterations) until there on average will be a user that is able to propose a block. Since there are 4 iterations instead of 8 on average, there will be more stakers that obtain a *hashProofOfStake* proof allowing them to propose a block per iteration. When two or more stakers are allowed to propose a block within one iteration, the probability of a fork increases meaning that the probability of a fork increases when the iteration frequency decreases (time between the calculation of new Kernel Hash) and the probability decreases when the iteration frequency decrease. The calibration of the current block timestamp mask is set as a trade-off between the probability of forks as well as how many computations a staker is expected to make.

The outcome of the kernel hash operations is called *hashProofOfStake* and is represented by a 256-bit integer. If the hashProofOfStake is smaller than $bnTarget * nWeight$ the validator has generated an *eligibility proof* required for the proposed block to be appended to the blockchain.

```
hashProofOfStake = hash(nStakeModifier + blockFrom.nTime +
    txPrev.vout.hash + txPrev.vout.n + nTime)
hashProofOfStake <= bnTarget * nWeight
```

5.7.3 Stake modifier

The purpose of the stake modifier is to scramble the target and prevent stake grinding attacks. This links the target directly to the previous block in the blockchain. The only part of the kernel the staker has control over is the staking transaction and since the coin maturity time is 500 blocks it is difficult to determine which exact block that is going to be appended to the blockchain 500 blocks ahead.

The stake modifier of a mined block is linked with the stake modifier of a block 500 blocks ahead. Without the stake modifier, the PoS algorithm could in practice be transformed into a PoW algorithm. The stake modifier of a block is a hash of exactly:

- *The hash of the prevout transaction*
- *The previous block's stake modifier (the genesis block's stake modifier is 0)*

5.7.4 Scalability

The difficulty is adjusted every block to produce a block every 128 seconds, producing five times more blocks than the Bitcoin protocol in a given time period and the block size is 2MB. The MPoS algorithm in the Qtum setting is similar to the Bitcoin protocol in a scalability perspective with other input parameters (block size and average time) for producing a block.

5.8 Casper - The Friendly Finality Gadget

Ethereum as one of the largest cryptocurrencies is migrating to PoS and the first step of this process is appending a PoS-layer onto the existing PoW-layer. This means that in the current state of Ethereum, the block producer is identical to Bitcoin with the difference being the hashing algorithm of Ethereum belongs to the Keccak family [13], otherwise the block proposing and Hashpuzzle is identical being PoW. The *Friendly Finality Gadget* (FFG) is an additional layer to the consensus algorithm governed by an Ethereum smart contract providing *finality* (i.e. guarantee that blocks will not be reversed). Casper utilizes rewards as well as penalties to incentivize good behavior. Validators put tokens at stake and are rewarded slightly to compensate for locking up their capital analogous to savings accounts at an ordinary bank. But the cost of reverting transactions comes from penalties that are a hundred or thousands of times larger than they got in the meantime. In theory, a large number of validators can collude and start acting maliciously. If implemented correctly however there is a possibility to minimize the damage these assailants can produce. If the malicious users try to prevent new validators from joining or perform a 51% attack the network will be forced to do a hard-fork and delete the offending validators.

Casper follows the **BFT-traditions**[18] with some modifications. Casper is called a *Friendly Finality Gadget* and can be described as a protocol on top of the block proposal mechanism with the proposal mechanism in this case being built on a PoW algorithm. The first version of Casper is a hybrid between PoW and PoS. The Casper FFG protocol proposes checkpoints and is responsible for finalizing these checkpoints.

Casper has this following **properties**:

- *The proof statement consist of "if there is a safety failure, then at least 1/3 of validators violated some protocol rule"*
- *Accountability is added. The protocol is able to detect who violated the rule and allows to penalize malfeasant validators, solving the nothing at stake attack vector. By making it very expensive to break protocol it's greatly disincentivized to break protocol.*
- *A provably safe way for the validator set to change over time is introduced.*
- *A new introduction to recover from attacks where more than 1/3 of validators drop offline are introduced at the cost of a very weak trade-off synchronicity assumption*

The first version (*simple version*) works under the assumption that there already exists a block proposal mechanism with a set of validators with the proposal mechanism in the early stages being PoW. The Casper incentive logic lives in a smart contract operating on the Ethereum network. Under normal circumstances, the proposal mechanism would suggest blocks that almost all resulting in forming a chain. However, if the proposal mechanism is faulty or acting in a malicious way there might be multiple divergent chains being extended at the same time. Casper FFG proposes a solution for determining the fork that should be finalized.

A chain of blockhashes into a sequence of an append only datastructure is called **B**. Within this blockchain **B** there is a subset of checkpoints **C** and *h(x)* is the height of checkpoint x.

The following terms are *defined* [18]:

- A *supermajority link* is an ordered pair of checkpoints (a, b), also written $a \Rightarrow b$, such that at least $\frac{2}{3}$ of validators (by deposit) have published votes with a source a and target b. Supermajority links *can skip checkpoints*, i.e it's perfectly okay for $h(b) > h(a) + 1$.
- Two checkpoints a and b are called *conflicting* if and only if they are nodes in distinct branches, i.e neither is an ancestor or descendant of the other.
- A checkpoint c is called *justified* if (1) it is the root, or (2) there exists a supermajority link $c' \Rightarrow c$ where c' is justified.
- A checkpoint c is called *finalized* if it is justified and there is a supermajority link $c \Rightarrow c'$ where c' is a direct child of c. Equivalently, checkpoint c is finalized if and only if: checkpoint c is justified, there exists a supermajority link $c \Rightarrow c'$, checkpoints c and c' are not conflicting, and $h(c') = h(c) + 1$

$$\mathbf{B} \equiv (b_0, b_1, b_2, b_3...)$$
$$\mathbf{C} \equiv (b_0, b_{49}, b_{99}, b_{149}...)$$

The **notation** are as follows:

C_0 is the genesis block.
Epoch is defined as the sequence of the discrete number of blocks between two checkpoints, including the later checkpoint but not the earlier one. In this case for example the epoch index is increased by one every 50 blocks.

The set of validators in the system is all the participants with a deposit choosing to participate and stake. Their stake is the number of coins that they have deposited and the deposit is increased with rewards and decreased with penalties. Validators only need to broadcast one message called the vote message. A vote has three parameters namely the checkpoint hash, epoch number, and epoch source. Using this notation for two votes.
<**vote1**, h1,e1, s1> <**vote2**, h2,e2, s2>

Notation	Description	Notation	Description
h_1	Checkpoint hash	h_2	Checkpoint hash
e_1	Epoch number	e_2	Epoch number
s_1	Epoch source	s_2	Epoch source

5.8.1 Slashing conditions

There exist two slashing conditions that are called **Casper Commandments** and it is impossible for two conflicting checkpoints to be *finalized* unless 1/3 of the validators violated one of these considering the two votes:

- *Don't vote twice in the same epoch e1=e2*
- *Be consistent in your votes. e1>e2>s2>s1*

The first slashing condition is illustrated in Figure 5.2

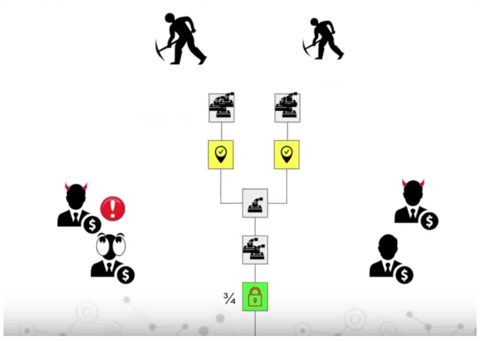

Figure 5.2: No Double Vote, under permission by the creator Karl Floersch

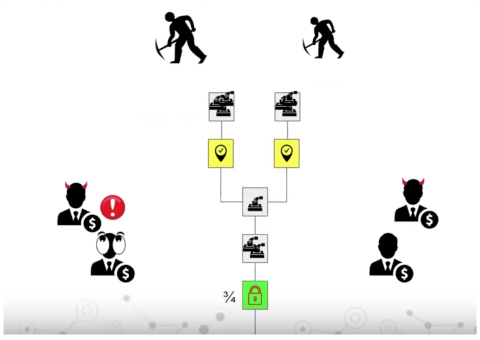

Figure 5.3: No Surround Vote, under permission by the creator Karl Floersch

The second slashing condition that can be seen in Figure 5.3, assumes that you firstly vote *(e1, s1)* and secondly *(e2, s2)*, the epoch counter is an implicit clock so we also assume e2>e1 and therefore s2≥s1.

A checkpoint is defined as *justified* if:

- *The genesis is justified*
- *A chain has accepted $\frac{2}{3}$ validators for a checkpoint C, all using the same ancestor (source checkpoint) C' and C is justified*

A checkpoint is defined as *finalized* if:

- *The genesis is finalized*
- *A chain has accepted $\frac{2}{3}$ validators for a checkpoint C, all using the same ancestor (source checkpoint) C' and C is justified*

5.8.2 Leak mechanism

Consider a scenario in which a fraction larger than $\frac{1}{3}$ of validators went offline or stops voting on forks. This event could occur in a collaboration with a malicious set of validators or simply by a large network partition error that cuts off large parts of the validating nodes in the network. In this scenario no finality can be reached therefore a *inactivity leak* was implemented slowly draining the deposits of any validator that does not vote for checkpoints [18] until the inactive users are drained enough so that the active validators who are voting are a supermajority. A validator with deposit size D that fails to vote will result in the following voting power: $D \cdot p$ (for $0 <$ p < 1) Where (1-p) is the fixed percentage of voting power the validator loses every epoch.

5.9 EOS BFT-DPoS

Byzantine Fault Tolerant - Delegated Proof of Stake is another algorithm utilizing the PoS algorithm approach using delegates called *block producers* to produce blocks. The description of this algorithm will be shorter than the rest of the algorithms covered in the thesis due to the fact that despite extensive search it was not possible to locate more documentation regarding the consensus algorithm. In the delegated PoS approach the users staking in the network votes on a block producer which they trust will produce honest blocks and maintain integrity on the ledger. EOS is as of today one of the largest cryptocurrencies and whilst there are several different implementations of DPoS schemes, this thesis will limit the scope to the EOS BFT-DPoS. Upon comparing Bitcoin to EOS, there are few similarities, in EOS there are a few numbers of nodes that produce and validate blocks and the cost of running such a node is quite high. Also, the throughput of transactions is significantly higher in EOS than in Bitcoin and the data structures of both blockchains are distinct with different code bases.

Users in possession of tokens on the EOS blockchain may select block producers through a continuous approval voting system [24] where the users sign their vote with the private key of their possessed tokens. Anyone in the network may participate in block production, where they will be selected to produce blocks if they are able to persuade the community to vote for them. Currently one staked EOS token can be used to vote for up to 30 different block producer candidates. Votes are weighted based on the total number of staked tokens, and the 21 block producers [26] with the largest weight (i.e most votes), will be elected as active block producers. The voting is a continuous process where the votes are recalculated approximately every 2 minutes. Token holders also have the opportunity to delegate or proxy their vote to another EOS account and tokens will be locked up 3 days after not participating in the staking process anymore.

Scheduling allows the average block time in the EOS protocol 0.5 seconds between blocks. Exactly one block producer is authorized to produce a block at any given timeslot, if the block producer decline to produce a block in the given timeslot, that particular block is skipped. Blocks are produced in rounds of 126, there are 21 block producers who each get to produce 6 blocks. The block producing ordering are scheduled in an order that must be agreed upon by 15 or more producers. If a block producer declines to produce a block within 24 hours, that block producer is removed from consideration until they notify the blockchain that they are live and ready to start producing blocks again. This feature will minimize the number of empty slots. Since the network is reliant on that block producers are online, they need to be shielded against DOS-attacks. The block producers are responsible for executing and validating the transactions and contracts the users in the network are propagating [25]. The valid transactions are processed by the block producers and included in the storage layer. When at least 15 block producers have validated and signed the data set, the transactions are finalized. If a block producer decline to produce a block during their assigned timeslot, the other block producers will confirm a set of empty transactions and move on to the next slot. Every transaction contains a hash value that references a previous block in the blockchain

5.9.1 EOS Nodes

There are two different types of nodes on the EOS network [15], namely *Producing nodes* and

Non-Producing Nodes

A non-producing EOS node does not produce blocks, instead, it receives and validates the blocks from the block producer making it synchronized with other peers from the EOS blockchain.

Producing Nodes

The main purpose of the producing nodes is to produce blocks of transactions to the EOS ledger. Since there are only 21 block producers, producing every block of transactions on the blockchain with every block coming every $\frac{1}{2}$ second, the hardware requirements to handle all the transactions are high.

Byzantine Fault Tolerance is added to traditional DPoS by allowing all producers to sign all blocks so long as no producer signs two blocks with the same timestamp or the same block height. Once 15 producers have signed a block the block is deemed irreversible. Any byzantine producer would have to generate cryptographic evidence of their treason by signing two blocks with the same timestamp or block height. Under this model, an irreversible consensus should be reachable within 1 second.

5.10 Algorand

Algorand invented its own protocol called Byzantine Agreement (BA) to obtain consensus in the network. A novel mechanism based on *Verifiable Random Functions* (VRF) with the main purpose to scale to a large set of users. Algorand emphasizes the importance of the fact that users never have divergent views of a confirmed transaction, even if some of the users are malicious and the networks are temporarily partitioned, meaning that the Byzantine Agreement consensus protocol is as could be guessed by the name, a BFT PoS algorithm. This mechanism allows users to anonymously check if they are selected to participate in the BA to decide and propose the next set of transactions by including their proof of selection in their network messages. The purpose of the VRFs is to randomly select users in a non-interactive and private way to avoid Sybil attacks.

Algorand is designed to confirm transactions on the order of one minute. The core of Algorand uses a Byzantine Agreement protocol denoted **BA*** designed to reach consensus with low latency without the possibility of forks scaling to a large set of users. Verifiable random functions are used in Algorand to randomly select users in a private and non-interactive way. Algorand is designed to avoid *Sybil attacks*, scale to millions of users and be resilient to DOS-attacks and keep operating regardless of nodes in the system are disconnected. These challenges are addressed using the following techniques:

Weighted users As with most other PoS algorithms, there is a weight assigned to each participant with the weight being the proportion of the number of coins the user is staking to the total number of coins being staked. As long as $\frac{2}{3}$ or more of the users are honest this prevents forks and double spending.

Consensus by committee
Scalability is achieved by randomly selecting a small set of stakers from the total set of the stakers in the system called a committee to run each step of the Byzantine Agreement protocol. The protocol messages are observed by all the users allowing them to learn the agreed-upon block. By using a committee for the block proposal protocol the network is exposed to a targeted attack against the chosen committee members.

Cryptographic sortition
The committee members are selected in a private non-interactive way to prevent targeted attacks. Each member calculates independently if they are selected to

be on the committee by computing a VRF of their private key and public blockchain information. The function is deterministic and if it indicates that the user is chosen it returns the proof as a string which is communicated to the rest of the network included in the users' network messages. Since this is computed independently of each other an adversary does not know which users belong to the committee until it is broadcasted and the users start running each step of the protocol.

Participant replacement
Since an adversary may target a committee member once the member broadcasts a network message and start running the steps in the BA*, committee members are required to speak only once. Meaning that once a committee member has broadcasted the network message, there is no use of targeting that particular user since that particular committee member is going to be replaced. This is achieved by avoiding any private state except for every users' private key making every user equally capable of participating as well as electing new committee members for each step of the BA* protocol.

In Algorand the user's weight is proportional to the number of coins you stake, this is similar to other PoS algorithms. The key difference here is that while other PoS algorithm uses the weight in a lottery to pick the next block proposer, making it possible to create a fork Algorand is using the weight to select a committee member in the BA* protocol not giving the user the possibility to make a fork in the network. The weight here is used to prevent a user from amplifying his power by the use of pseudonyms. As long as the attacker holds less than 1/3 of the network Algorand ensures that the probability for a fork to appear is negligible.

5.10.1 Cryptographic sortition

The purpose of the cryptographic sortition algorithm is to choose a random subset of users according to the user's weight in proportion to the total amount of coins staked. $W = \sum\limits_{i=1} w_i$ being the weight and the probability that the user i is selected is proportional to w_i/W. The randomness in the cryptographic sortition is introduced from a publicly known random seed. In order to check and verify that a user was chosen the sortition requires that each user has a public key and corresponding private key. This cryptographic sortition is implemented using Verifiable Random Functions. On any input string x, $VRF_{sk}(X)$ returns two values, a hash, and a proof. The hash consists of a hashlenbit-long value that is uniquely determined by the secret key and the input string x but is indistinguishable from random to anyone without the knowledge of the secret key. The proof enables anyone with the public key to verify that the hash corresponds to the string x without requiring the corresponding secret key, like a hashing function we require VRF to provide these properties even if the public and private key are chosen by an attacker.

5.10.2 Gossip Protocol

The gossip network implemented in Algorand is similar to the Bitcoin protocol where users select a predefined number of random peers to gossip the messages to. This is to ensure the integrity and that messages cannot be forged, every message is signed by the private key of its original sender, and the receiver checks that the signature is valid before relaying it. The gossip protocol is implemented over TCP and weighted peer selection based on the amount of tokens the gossiper is in possession of to mitigate pollution attacks.

5.10.3 Block Proposal

Cryptographic sortition is executed for users in the network to determine if they are selected as a block producer in a given round. Sortition ensures that randomness is introduced when picking a small fraction of users, weighted by their number of tokens in proportion to the total amount of tokens in circulation. It also provides each selected user with a priority to determine the order of the committee and a proof of the chosen user's priority. The users of the network execute cryptographic sortition locally to determine if they are selected to propose a block in a given round, sortition ensures that a small fraction of users are selected at random, weighted by the number of tokens they possess gives them a priority. Since the sortition process is random, multiple users may be selected to propose a block and the priority between the selected proposers is compared and the proof with the highest priority is selected.

5.10.4 Algorand Nodes

There are two distinct types of nodes that can be run by the user [12], relay nodes and non-relay nodes. Relay nodes are used for communication routing to a set of connected non-relay nodes. Route of blocks is being relayed to non-relay nodes by relay nodes that communicate with other relay nodes. Non-relay nodes can connect to several relay nodes (but never another non-relay node) and are also the nodes that participate in consensus.

In addition, there are two node types, archival nodes that store the entire ledger and the indexer that if turned on, the search range via the API REST endpoint is increased.

Chapter 6

Consensus Algorithm Analysis

6.1 Introduction

In this section an analysis of the different consensus protocols are going to be covered from three different perspectives. Each algorithm covered is going to be analyzed from a *security*, *decentralization* and *scalability* perspective discussing strengths and weaknesses.

6.1.1 Analysis from the Security Perspective

The analysis from the security perspective will mainly discuss the most common attack vectors against PoS algorithms covered in chapter 5 and how they are mitigated or handled in the different algorithms and settings, additionally, other security threats might be covered. Note that chapter 5 introduced the most common attack vectors against PoS protocols and depending on setting and implementations all the attack vectors will not be relevant for each algorithm, some algorithms might just have one relevant attack vector.

6.1.2 Analysis from the Decentralization Perspective

Comparing different blockchain applications from the decentralization perspective is both a controversial subject and tends to get subjective. There is no clear definition of a decentralized blockchain network, therefore the decentralization perspective will be inspected upon a number of different criteria. All questions might not be answered depending on the protocol but it is used as a framework to evaluate.

- Are there any barriers of entry to propose a block of transactions to the blockchain?
- How many nodes validate all the transactions on the blockchain?
- What is the cost involved in running a full node/validating all transactions on the blockchain?
- Can anyone validate the chain?
- Is there any entry barriers to participate in the staking process?

6.1.3 Analysis from Scalability Perspective

Scalability in a blockchain context usually refers to throughput. Throughput and user experience are highly prioritized, therefore **transactions per second**, is the most commonly used concept referring to how many transactions the blockchain network is able to process per second. In this sense, the bitcoin network has a mean block producing time of 10 minutes with a constraint of 1MB per block averaging 3 transactions per second on the bitcoin network assuming that SegWit is not used, if SegWit is used the block size is closer to 2MB. This number is however highly volatile since the blocks are constrained by bytes instead of the actual number of transactions.

6.2 Mutualized Proof of Stake

6.2.1 Analysis from security perspective

Attack vectors against Mutualized PoS are going to be covered in this section.

Stake Grinding Attack

The stake modifier is used to prevent block proposers from precomputing future proofs of given UTXOs, effectively transforming the PoS algorithm into a PoW algorithm. In order to grasp this attack vector, the staking process must be reviewed. The stake in MPoS is dependent upon the input parameters of the kernel hash. The kernel hash is derived from a mature transaction making up the input parameters *transaction timestamp*, *staking transaction hash* and *staking transaction output index*. One input parameter that the kernel hash is derived from that can be altered is the *current block timestamp*. However, the protocol is designed to be iterated every 16 seconds. Meaning that timestamps must be multiples of 16 seconds to be accepted by the client and the timestamp must be greater than the timestamp of the previous block plus 15 seconds to be accepted. It is designed this way to mitigate the stake grinding attack by iterating through a large number of timestamps to find a kernel hash that meets the target. The staking transaction needs to be mature and is therefore required to be created at least 500 blocks before the staking occurs, however, this does not itself give any security guarantees. Since the UTXO picked to stake could be customized to minimize the outcome of the kernel hash if the only other parameter would be the timestamp (the future timestamp is predictable). Therefore to scramble the kernel hash and make the stake grinding attack hard to perform in practice, the stake modifier is introduced and included in the kernel hash.

This is due to the fact that the stake modifier is the hash digest of the previous stake modifier and the last stake transaction hash. The attacker, therefore, needs to know the chain of every UTXO that will stake and append a block of transactions to the blockchain. Since this chain is at least 500 blocks, the attack is hard to perform in practice.
Without the stake modifier, the block proposer can *grind* through different UTXOs and wait for a suiting time (specific timestamp that favors the outcome)

when the bnTarget becomes large (i.e the staker preforms well and will likely meet the target). Since kernel hashes can not be computed without stake modifiers and stake modifiers can not be computed for a maturity period after the staker receives the UTXO this attack is mitigated.

Long Range/History Revision Attacks

This attack vector can be divided into synchronized vs unsynchronized users of the network. For synchronized users of the network, the MPoS algorithm in the Qtum client setting rejects blockchains where the difference is more extensive than the most previous 500 blocks. Meaning that changes in the most recent 500 blocks are accepted by the client but changes to the transactions on the blockchain exceeding the most recent 500 blocks will be discarded. This mechanism will mitigate the long range attack vector for synchronized nodes. However new nodes, which in turn are the unsynchronized users connecting to the network for the first time accept the chain with the largest accumulated difficulty. Since the checkpoints are consensus critical meaning that to get rid of these, a user would in practice have to hardfork the blockchain. Meaning, in order to perform this attack, a user is required to develop his own version of the Qtum source code where the checkpoints are bootstrapped. Additionally, the user has to convince the community that the new client is the proper Qtum client and follow that chain instead. Each block contains a difficulty parameter indicating the difficulty to mine a block, all these difficulty parameters will be accumulated and compared to other blockchains being proposed.

Nothing at Stake

Essentially the nothing at stake problem is not quite an attack vector but instead misaligned incentives that could potentially result in an attack. There is nothing preventing or disincentivizing the *Nothing at Stake* attack but there are mechanisms to mitigate the most common attack vector involved in the problem such as History Revision Attacks. However, the problem with misaligned incentives might result in problems. Imagine the following scenario: A malicious user forks the blockchain and creates an alternative version of the public blockchain. The users of the network do not have anything to lose by staking on both chains, maximizing the probability to retrieve a potential staking reward.

Denial of Service

The set of UTXOs is known due to the public and permissionless nature of the Qtum blockchain. It is possible to predict the set of validators that are allowed to propose the next block. This could in theory allow an attacker to launch DoS attacks against those validators. The attack is hard to perform in reality due to the lack of information given by a UTXO. A UTXO is not itself enough to perform a DoS flooding attack, a validator IP-address is required to perform the attack.

6.2.2 Analysis from the Decentralization Perspective

Users running a full node are responsible for processing and validating all transactions propagated to the network. In a similar way to the Bitcoin network, anyone with a computer capable of running a full node is able to validate the network.

6.2.3 Analysis from a Scalability Perspective

The throughput of the MPoS algorithm in a Qtum network setting is theoretically 10 times the throughput of the Bitcoin protocol. The amount of Qtum transactions is theoretically two-fold in size compared to the amount of Bitcoin transactions per block due to the increase in block size. Since MPoS mimics the HashCash protocol implemented by the Bitcoin network, the scalability of the MPoS consensus algorithm in the Qtum setting is almost identical to the scalability of HashCash in the Bitcoin setting. The difference merely lies within the settings of the different networks being average block production time and the size of the block of transactions.

6.3 Casper FFG Analysis

6.3.1 Analysis from the security perspective

The Casper FFG is merely an extra layer on top of the existing PoW algorithm where the PoW algorithm is the block proposer, therefore some of the attack vectors covered will not apply to the Casper FFG. Additional Casper specific attack vectors will therefore be covered. *What attack vectors can be performed without violating the slashing conditions?*

Delayed Double Finality Attack

In this attack scenario a malicious validator that has accumulated approximately 50% of the total stake in circulation, votes persistently on an alternative chain to the other 50%. To prevent chains from finalizing, Casper FFG has a built-in leak mechanism that decreases the deposit of validators that does not participate in voting. The attacker is able to persistently vote on the malicious chain until the deposit comprises $\frac{2}{3}$ of the total stake. Since the votes are allocated equally on both forks, the leak mechanism will progress at an equal rate at both chains. This will result in both chains finalizing simultaneously, both being valid chains, making two distinct finalizing without violating any slashing condition. This however assumes that both forks also have equal mining power since the leaks occur per epoch.

Small Stake Fork Attack

The adversary is required to have mining power and a stake high enough to participate in the voting process. In this attack scenario, the adversary votes on a different fork distinct from the rest of the validators. If the adversary persistently votes on the alternative fork then the rest of the validator network whilst also mining on the alternative fork. The mining is required to maintain liveness in the fork. This will eventually result in the malicious fork reaching finality. No slashing conditions need to be violated to finalize the malicious fork. The malicious fork will most likely finalize long after the fork the rest of the validators voted on but this will nonetheless result in two conflicting finalized forks.

6.3.2 Analysis from the decentralization perspective

Since the underlying block proposing algorithm will remain PoW and Casper is merely running on-top of the consensus algorithm as a check-pointing system that is handled by a smart contract, the decentralization properties will almost remain the same as in the Bitcoin protocol. However, there will be a threshold amount of tokens to be allowed to participate in staking. This threshold is still unknown but in order to keep the smart contract efficient, this threshold will likely be high. Setting the threshold to a large number of ether tokens would result in a small number of entities in the network actually being able to stake and thereby influence what checkpoints will be set.

6.3.3 Analysis from scalability perspective

Since the block proposing algorithm in the Casper FFG setting is the underlying PoW, the scalability will be identical to the previous pure PoW Ethereum setting.

6.4 EOS BFT-DPoS

6.4.1 Analysis from the security perspective

Denial of Service

The number of block producers in charge of producing the entire set of transactions is fixed in a given period of time. In such setting, the up-time of the block producers is crucial for EOS network's liveness. There are several ways to obfuscate the IP addresses of the block producing servers, but since they need to interact with the blockchain in a peer-to-peer protocol, there is no known way to keep the servers completely anonymous. The block producers in the EOS blockchain are vulnerable to DoS attack vectors. There are a number of different ways these attacks could be performed, with the most trivial being a flooding attack. A malicious entity sends a large volume of information to the block producer, essentially making the block producer unable to receive additional information, resulting in the block producer being disconnected from the blockchain, unable to produce blocks of transactions thereby halting the network.

6.4.2 Collusion Attack

Forks in the DPoS blockchain occur when block producers are in disagreement of which is the latest valid block. The forks are designed to automatically resolve themselves by switching to the longest chain. To understand how a fork can occur, let us assume a scenario consisting of five malicious block producers while the other block producers act according to protocol. The five malicious block producers will collude and build their own version of the blockchain, including malicious transactions conflicting. These transactions will be appended sequentially by the five colluding block producers while the rest of the block producers will append their valid transactions on top of each other. At the end of round 126, the chains are compared, and the largest chain gets appended to the public blockchain, therefore, there must be more colluding malicious users than active, honest users for a malicious fork to get finalized. However, since block producers are public and a relatively small number needs to be hacked/persuaded to collude, this attack may be possible in practise.

6.4.3 Analysis from the decentralization perspective

The EOS blockchain is reliant upon 21 block producers to produce valid blocks for the whole network. These block producers are in charge of validating, processing, and propagating valid transactions to the entire network. Since there are only 21 block producers in charge of all transactions on the network and with the throughput promised by EOS, these servers' demands are high.

6.4.4 Analysis from scalability perspective

Since there are only 21 block producers tailored to process transactions, the EOS network has reportedly processed over 3000 transactions per second [27]. Giving the EOS network a throughput greatly exceeding the Bitcoin network, purely looking at the scalability from a transactions per second point of view.

6.5 Algorand

6.5.1 Analysis from the security perspective

Attack vectors against Algorand are going to be covered in this section.

6.5.2 Denial of Service

The Algorand blockchain relies on synchronicity in the network, stating that at least $\frac{2}{3}$ of the users of the blockchain needs to be good actors. This also implies that at least $\frac{2}{3}$ of the blockchain users needs to participate in the blockchain consensus model at all time to reach finality in the network. One should note that any user can choose to delegate their validator weight to a third party that then, in turn, handles the block validation/production. Regardless, due to the high threshold of participation required, the network is highly dependent on the users of the network participating in consensus to keep the blockchain up and running. It makes it vulnerable to denial of service attacks striking out a portion of the network larger than $\frac{1}{3}$ of the blockchain users that are not delegators.

However, the cryptographic sortition algorithm is used for choosing a random subset of users chosen to participate in the block production process. This process is designed to mitigate targeted denial of service attacks since the proof is calculated upon each user's private key returning a hash and a proof to prove that the user, in fact, possesses the private keys and is therefore eligible to participate in the block producing process. A user calculates these values locally to determine if they are eligible to stake, and when they are certain, they propose a block with the proof and the hash. This is the only time the user interacts with the blockchain in this block producing process, and since one can assume adversaries are not in possession of the private keys of the users, the block will already be proposed when the producer reveals their identity rendering a targeted denial of service attack useless.

6.5.3 Analysis from scalability perspective

Besides many of the other consensus algorithms covered in this thesis, Algorand scales at the core level, thus greatly increasing the number of transactions. From the architecture it is clear that Algorand scales better by their own reference [19] 125 times faster than Bitcoin. Putting this into perspective, Bitcoin does not require any user to be online to avoid the network from halting, possibly resulting in that it might take a longer time to achieve the same security/consensus of the transactions of the network.

6.5.4 Analysis from the decentralization perspective

The cost involved in running an Algorand participation node is quite low. According to Algorand's website [19], the cost is lower than hosting a Bitcoin node, which in that sense makes Algorand even more decentralized than Bitcoin. At least $\frac{2}{3}$ users must keep the nodes running or delegate to a user that keeps the node running at all time.

Chapter 7

Discussion

7.1 General Analysis of Proof of Work vs Proof of Stake

The largest problems with PoW identified throughout the thesis relates to the concentration of computing power in larger mining pools, where a majority of the hash power (65%) in Bitcoin is concentrated in China [30]. One could easily imagine that profits from a mining operation increase at a non-linear rate as the mining operation expands, thus benefiting larger mining farms due to economies of scale. There could be a number of different reasons for these advantages in scale. One relating to the development of specialized mining equipment (ASICs). Another one being that larger corporations manage to negotiate better deals on equipment and storage related to the mining operation as well as the electricity cost which is the main price factor.

One of the most highlighted arguments against PoW is the vast energy argument, namely that the mining operation uses a vast amount of energy without producing "anything". While it is true that the computing procedure itself does not produce anything of value, the same can not be said of the whole mining operation. The purpose of mining is to secure the Bitcoin network and maintain the integrity of the transaction ledger, i.e., preventing double spend attacks. As long as the mining operation secures the ledger of transactions, the operation itself could potentially not be considered wasted energy. It is merely a question of energy efficiency within securing a financial system. The mining operation demands energy analogous to how the conventional financial system requires energy and material in the money printing process. The US Federal Reserve System 2020 had allocated a budget of $877.2M in currency budget [31], and it might not be fair to compare the Bitcoin monetary network to the US financial ecosystem. However one could compare the energy efficiency of both monetary systems weighted by their market cap or size. The rhetoric could potentially instead be shifted towards: *What is the most energy-efficient financial system?*

One of the key arguments of PoS involves the low cost involved in running the consensus protocol. This is a common misconception since despite the mining process itself does not consume large amounts of energy, extrinsic resources. The participants in the consensus protocol still needs to lock up stake in form of intrinsic resources. Upon examining these intrinsic resources today among all those cryptocurrencies covered in this thesis, they have real value [4]. The cost, in this case, could be seen as an opportunity cost of locking up capital, which is, in fact a real cost. One could view this as more energy-efficient since there is practically no energy burned while obtaining consensus in PoS networks while there are large amount burned in PoW network. In order to really get a grasp of this comparison, one would have to examine the energy sources for the mining operation and also compare that to the philosophical question where the capital locked into PoS ecosystems could potentially benefit the environment if it wasn't locked up. This is a controversial and philosophical question not relating to the consensus algorithms' technology and was not be covered in this thesis.

7.2 Analysis From The Scalability Perspective

It becomes more and more evident throughout the thesis that PoS is intended as a consensus algorithm and not primarily a solution to the scalability problems blockchain applications are facing. With that being said, there are also blockchain solutions implementing scalability on the core layer as Algorand and EOS as covered in this thesis. The pure chain-based approach MPoS algorithm in the Qtum network is theoretically allowing for ten times greater transaction throughput. However, the reason for this is due to settings and trade-offs within the protocols. The increase in throughput of increasing the block size from 1MB to 2MB is done with the increasing blockchain storage trade-off. Theoretically, the number of transactions per second will be doubled, but the blockchain will also double in size at twice the pace assuming all blocks of transactions are filled each iteration of blocks. MPoS in the Qtum network also chose to have a difficulty adjustment algorithm calibrated to produce blocks five times faster than the bitcoin protocol and the same applies here, this will increase the throughput per second of transactions but will also increase the size of the blockchain with the same factor. However, there are consensus algorithms covered in this thesis that scales the network at the core layer. EOS uses 21 validators that require expensive, robust hardware for block producers capable of handling a significant number of transactions and, therefore, greatly beats Bitcoin and similar consensus algorithms in the sense of transaction throughput. The result of this increased throughput and increased scalability will be covered in the decentralization perspective. Algorand also enables improved scaling at the core layer compared to Bitcoin and similar consensus models due to Algorand's design and relatively new cryptographic primitives (ZK-snarks). Casper FFG is not a scalability solution. It merely operates on-top of the underlying block proposal scheme. Which is PoW (for the time of writing) and it will therefore have the same scalability properties as PoW.

7.3 Analysis From The Decentralization Perspective

Using the definition and context of decentralization used in this master thesis, decentralization boils down to how many users can validate the entire blockchain of transactions. This, in turn, boils down to different kinds of topics, with the main more relates to cost and efficiency. What is the cost of running a full node and proposing or producing blocks to the blockchain? This can relate to storage but also memory and bandwidth. One of the questions that is often asked in the blockchain community is: *can an average middle-class family run a full node?*. This could potentially be a problematic way of formulating the question but if we refrain from the political side of it and merely exemplify what it could mean is *can a family with an average income in the US run a full node?*. Let us illustrate what is meant by that. Imagine, for example, that the full node itself is not expensive but that it occupies up a vast amount of storage and would require a server hall to operate it. Could an average American family operate that?. The other aspect is cost. What does it cost to run a full node or a block proposer in the network? In these regards, specific costs will not be covered since all blockchains grow in size over time and the costs changes. Instead, let us look into the technical details and see how the cost change over time for each protocol. Upon dissecting Bitcoin and Qtum, one could quickly identify similarities in their consensus algorithms despite being PoW and PoS. Both Bitcoin and Qtum grow at a linear pace where the Qtum blockchain, in theory, could grow ten times faster than the Bitcoin network due to consensus settings. However, neither Qtum [32] nor Bitcoin [33] are currently too large in storage or memory for an average American family to run a full node. It is not especially costly nor time consuming to verify the validity of all transactions on the network from the genesis block to the most recent block. If simplified a lot, the archival nodes of the Algorand network as described in the Algorand section "Algorand Nodes" is similar to the Qtum and Bitcoin full nodes in the sense that they store and validate the entire blockchain (depending on the parameters), and, additionally does not require any special hardware to run. It is also similar in the EOS setting, any user can run an archival node to validate and verify the transactions on the blockchain, and despite the block time being $\frac{1}{2}$ second, the blockchain will increase in a linear pace.

The other aspect is the threshold to produce blocks and facilitate block producing or block proposing nodes. Can anyone in the network do it, or is it some process to select these users/entities and who handles this process? For the case of Qtum and Bitcoin, the block proposing mechanism is completely trustless, meaning that anyone with the smallest amount of computing power in the case of Bitcoin or the smallest amount of Qtum are allowed to participate in the consensus algorithm in the task of proposing blocks. In Bitcoin, there are advantages in running large mining operations in scale, as discussed before, whilst in Qtum the reward is linear. In the case of Algorand, the network is similarly completely trustless in the sense that any user can participate in the block production scheme. The difference here is that a large percentage of the users must either participate in the block producing scheme or delegate to a user that participates, if $\frac{2}{3}$ of users fails to participate or delegate, the blockchain will halt.
In the EOS case, the initial 21 block producers were picked by the EOS foundation after a transition was made so that token holders delegated to block producers and initiated new block producers while other left, and this process still take place. Regardless, it is debatable how decentralized 21 block producers controlling the whole blockchain and transactions are. One argument here is that

the increased throughput comes with the cost of centralization and that if a block producer behaves maliciously, they will be voted out. One argument is also that a few large mining pools are producing the majority of blocks in Bitcoin and the majority of stake in Qtum while the counterargument to that is that anyone is able to use their computing power to propose a block and there is no threshold to entering the block producing scheme. Also pools are often made up of several computing entities where the entities are free to mine by themselves or point to another mining pools. Initially, a chart of all these properties was to be made. However, due to the highly controversial nature of the subject and also since this is a sliding scale and hard to determine in such a binary manner, this was not done. Casper FFG does not change the decentralization properties of Ethereum.

Chapter 8
Conclusions

We can observe that the Bitcoin network suffers from a high fraction of the mining consumption originating from a small concentration of mining entities. Upon dissecting the PoS implementations, the largest attack vectors stem from a problem called the long range attack vector involving the lack of cost for proposing a block to the blockchain, in contrast to the Bitcoin blockchain that uses the PoW model that consumes external resources. This problem is handled by different security mechanisms depending on the implementation, mostly consisting of checkpointing schemes and finality and the network. Qtum manages this problem by a checkpointing mechanism coded into the releases. Meaning that if an adversary were to fork the source code and convince the entire network to run their software instead, a long range attack could be possible, however, this attack appears impractical and very hard to carry out. The rest of the blockchains similarly mitigates this by obtaining finality among the synchronized nodes on the blockchain; however, the attack vector remains. There is no fully trustless way to mitigate the attack.
Additionally, despite PoS not being built primarily as a scaling solution, a cryptocurrency can implement it in that setting. For example, with EOS, significantly increasing the throughput compared to Bitcoin, with the cost of losing parts of the decentralization properties by maintaining 21 block producers that produce blocks for the entire blockchain. Additionally, Algorand is significantly improving throughput compared to Bitcoin, under the assumption that at least $\frac{2}{3}$ of the users are active, meaning that they are either validating blocks or delegating their vote to an active validator.

References

[1] L.Lamport, R. Shostak M. Pease, *The Byzantine Generals Problem*, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/The-Byzantine-Generals-Problem.pdf

[2] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, https://bitcoin.org/bitcoin.pdf

[3] James A. Donald, *Re: Bitcoin P2P e-cash paper 2008-11-17 16:33:04 UTC*, http://satoshi.nakamotoinstitute.org/emails/cryptography/15/#selection-7.0-11.23

[4] CoinMarketCap *Coinmarketcap.com* https://coinmarketcap.com/

[5] Eliza Paul *What is Digital Signature- How it works, Benefits, Objectives, Concept* http://www.emptrust.com/blog/benefits-of-using-digital-signatures

[6] D. Johnson A. Menezes S. Vanstone *The Elliptic Curve Digital Signature Algorithm (ECDSA)* https://link.springer.com/content/pdf/10.1007\%2Fs102070100002.pdf

[7] Michael Szydlo *Merkle Tree Traversal in Log Space and Time* https://link.springer.com/content/pdf/10.1007\%2F978-3-540-24676-3_32.pdf

[8] Bart PRENEEL *Analysis and Design of Cryptographic Hash Functions* https://www.esat.kuleuven.be/cosic/publications/thesis-2.pdf

[9] Andreas M.Antonopoulos *Mastering Bitcoin* https://github.com/bitcoinbook/bitcoinbook

[10] Leslie Lamport , Robert E. Shostak, Robert Shostak, Marshall C, Marshall Pease *The Byzantine generals problem* https://dl.acm.org/doi/abs/10.1145/3335772.3335936?casa_token=k3HQ_l5-diwAAAAA\%3AP7vAUji_bezaIxTFlwLXdOY6DpeZGnebqjPIIItpuaYRc7Rjqqy5b-uJZgZz9VcSdmJCRzB-L17v

[11] Adam Back, *Hashcash - A Denial of Service Counter-Measure* ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf

[12] Algorand Node Types *Run a Node* https://developer.algorand.org/docs/run-a-node/setup/types/

[13] Vitalik Buterin *Ethereum Whitepaper* https://ethereum.org/en/whitepaper/

[14] Patrick Dai, Jordan Earls, Neil Mahi, Alex Norta *Qtum Whitepaper* https://qtum.org/user/pages/01.home/Qtum\%20whitepaper_en\%20v0.7.pdf

[15] EOS Developer Documentation *NodeOS* https://developers.eos.io/manuals/eos/latest/nodeos/index

[16] Jordan Earls, *The missing explanation of Proof of Stake Version 3*, http://earlz.net/view/2017/07/27/1904/the-missing-explanation-of-proof-of-stake-version

[17] Zhou, Qiheng & Huang, Huawei & Zheng, Zibin. (2020). *Solutions to Scalability of Blockchain: A Survey* IEEEAccess.PP.10.1109/ACCESS.2020.2967218.

[18] Vitalik Buterin and Virgil Griffith, *Casper the Friendly Finality Gadget* https://github.com/ethereum/research/blob/master/papers/casper-basics/casper_basics.pdf

[19] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*, https://people.csail.mit.edu/nickolai/papers/gilad-algorand-eprint.pdf

[20] Bitcoin wiki, *Block size limit controversy*, https://en.bitcoin.it/wiki/Block_size_limit_controversy

[21] *The First Yearlong ICO for EOS Raised 4 Billion. The Second? Just 2.8 Million* https://www.coindesk.com/the-first-yearlong-ico-for-eos-raised-4-billion-the-second-just-2-8-million

[22] Blockchain.info *Blockchain size* https://blockchain.info/sv/charts/blocks-size

[23] Blockchain.info *Average Number of Transactions per Block* https://blockchain.info/sv/charts/n-transactions-per-block

[24] EOS *EOS Technical Whitepaper* https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md#consensus-algorithm-bft-dpos

[25] Consensys *EOS Test Report* https://www.whiteblock.io/library/eos-test-report.pdf

[26] EOS *EOS Block Producers* https://medium.com/coinmonks/eos-block-producer-voting-guide-fba3a5a6efe0

[27] EOS Throughput *EOS Transaction per second* https://coincodex.com/article/2052/eos-achieves-3000-transactions-per-second/

[28] Polkadot Nominated Proof of Stake *Nominated Proof of Stake* https://wiki.polkadot.network/docs/en/learn-staking

[29] Algorand Hardware Requirements *Algorand Hardware Requirements* https://algorand.foundation/network

[30] Jeffrey Gogo *65% of Global Bitcoin Hashrate Concentrated in China* https://news.bitcoin.com/65-of-global-bitcoin-hashrate-concentrated-in-china/

[31] US Federal Reserve *2020 Currency Budget* https://www.federalreserve.gov/foia/files/2020currency.pdf

[32] Qtum.info *Qtum.Info* https://qtum.info

[33] Blockchain.info *Blockchain.info* Blockchain.info https://blockchain.info