Warsame Mohamed Hersi

Network Impact on the Creation of Temporary Forks in Bitcoin

Master's thesis in Communication Technology and Digital Security
Supervisor: Yuming Jiang
Co-supervisor: Befekadu Gezaheng Gebrselase
June 2022

Warsame Mohamed Hersi

Network Impact on the Creation of Temporary Forks in Bitcoin

Master’s thesis in Communication Technology and Digital Security
Supervisor: Yuming Jiang
Co-supervisor: Befekadu Gezaheng Gebrselase
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Title:	Network Impact on the Creation of Temporary Forks in Bitcoin
Student:	Warsame Mohamed Hersi

Problem description:

When two miners independently find and publish a new block on the same height in a blockchain system, what can occur is something called a temporary fork. When a fork occurs, transactions are usually discarded, which will strain the network. Accordingly, we want to understand the impact that temporary forks have on the network. The objective of this thesis project is to investigate how different network parameters may affect the creation of forks in Bitcoin. Specifically, the investigation will be conducted by running simulations on a Bitcoin Core testnet testbed and gathering data from the simulations to answer the following research questions:

- How will different network parameters such as topology and delay affect the number of temporary forks?
-To what extent will the temporary fork affect transaction waiting/confirmation time?

Date approved:	2022-03-14
Responsible professor:	Yuming Jiang
Supervisor(s):	Befekadu Gezaheng Gebrselase

Abstract

Bitcoin was created in 2009, and it is a digital currency which relies on volunteer nodes rather than central authorities, compared to traditional currencies. Temporary forks are a known issue with blockchains that uses proof of work (PoW). When two miners find a block at the same height referencing the same previous block, a fork is created. Additionally, a fork can occur due to different reasons such as latency, block propagation or block size. Temporary forks cause inconsistencies and increase the strain on the network. Therefore it should be avoided. For this thesis, we have created a Bitcoin Core testbed to run our experiments. Using the testbed, we analyze how latency and Peer-to-peer (P2P) topology affects the number of forks created in the Bitcoin network. We then gather the blocks that have created forks and analyze how their waiting/confirmation time has been affected. This thesis differs from other work in that we have created our testbed where we can control the delay and topology and see how this affects the creation of temporary forks. We run nine experiments with different configurations, each lasting for 4-6 days. Temporary forks are hard to observe, and therefore each experiment runs for multiple days to be able to see forks occurring. Due to the timeframe given for this master thesis, we cannot replicate experiments multiple times to get the confidence interval we would like. The analysis shows that the delay and topology affect the number of forks, and some of the transactions in these blocks have delayed confirmation time.

Sammendrag

Bitcoin ble opprettet i 2009, og er en digital valuta som er avhengig av frivillige noder i stedet for sentrale myndigheter, sammenlignet med tradisjonelle valutaer. Temorary forks er et kjent problem med blokkjeder som bruker PoW. Når to miners finner en blokk i samme høyde som refererer til den samme foregående blokken, vil dette føre til en fork. I tillegg kan en fork oppstå på grunn av forskjellige årsaker som latency, block propagation eller blokkstørrelse. Temporary forks forårsaker inkonsekvenser i nettverket og øker belastningen på nettverket. Derfor bør det unngås. For denne oppgaven har vi laget et testnett for Bitcoin Core for å kjøre eksperimentene våre. Ved å bruke testnettet analyserer vi hvordan latency og topologi vil påvirke antall forks som opprettes i Bitcoin-nettverket. Vi samler deretter blokkene som har laget forks og analyserer hvordan vente/bekreftelsestiden deres har blitt påvirket. Denne oppgaven skiller seg fra annet arbeid ved at vi har laget vårt eget testnett testbed hvor vi kan kontrollere forsinkelsen og topologien og se hvordan dette påvirker opprettelsen av temporary forks. Vi kjører ni eksperimenter med forskjellige konfigurasjoner, som hver varer i 4-6 dager. Temporary forks er vanskelige å observere, og derfor kjører hvert eksperiment i flere dager for å kunne se forks oppstå. På grunn av tidsrammen som er gitt for denne masteroppgaven, kan vi ikke replikere eksperimenter flere ganger for å få det konfidensintervallet vi ønsker. Analysen viser at forsinkelsen og topologien påvirker antall forks, og noen av transaksjonene i disse blokkene har forsinket bekreftelsestid.

Preface

Upon delivering this master thesis, it concludes my five years master's degree in Communication Technology and Cyber Security at Norwegian University of Science and Technology (NTNU). These last five years have indeed been a fantastic experience filled with learning and growing. I want to thank my professors and my classmates, all the friends I have gathered during my stay at Trondheim, and those I had before moving to Trondheim. The time at NTNU is something I will cherish and remember for the rest of my life.

Before starting working on this master thesis, I had a fundamental understanding of Bitcoin and blockchain. However, I got a more profound understanding by taking a class on the subject and doing research. Undoubtedly, this thesis would not be possible without the help of my supervisors, Yuming Jiang and Befekadu Gezaheng Gebrselase. I want to especially thank Befekadu for being patient and always willing to help answer any questions I had during this process. I thank both of you a lot for this process.

Lastly, I would like to thank my family for always supporting and motivating me. Especially my mom and dad.

Contents

List of Figures

List of Algorithms

List of Acronyms

BTC Bitcoin.

CDF Cumulative Distribution Function.

DNS Domain Name System.

ECDSA Elliptic Curve Digital Signature Algorithm.

KB Kilobyte.

Mempool Memory pool.

MS Millisecond.

NTNU Norwegian University of Science and Technology.

P2P Peer-to-peer.

PoW proof of work.

SegWit Segregated Witness.

SSH Secure Shell.

TxID Transaction ID.

UTXO Unspent transaction output.

Chapter 1 Introduction

Bitcoin was created in 2009 and is a digital currency system [BitcoinDigital]. Its goal is to decentralize the global currency system and facilitate transactions. In contrast, today's economy relies upon a central authority responsible for the currency and the validation of transactions. Bitcoin issues its own currency and uses a P2P network to store and validate transactions [Nak09]. The P2P network consists of volunteer nodes that process the transactions and agrees upon their validity. Each node has a copy of all transactions [VJR18]. This helps replace the central authority that exists in traditional economies.

In Bitcoin, we have two main components blocks and transactions. The blocks are generated through a consensus algorithm called PoW. As a rule, nodes have to solve a cryptographic puzzle, and the node that solves the puzzle is rewarded with a block. Furthermore, a block is used to store the user-generated transactions. This puzzle usually contains some hash calculations. The process of solving this puzzle is referred to as mining. The miner that solves the puzzle is rewarded with an amount of Bitcoin. When miners solve the cryptographic puzzle almost simultaneously, something referred to as a temporary fork can occur. This happens because the blocks found were at a similar time and referencing the same previous block.

Consequently, the result of this fork is a split in the blockchain with two competing chains. Usually, this is resolved within the following blocks. When a temporary fork occurs, it results in the abandonment of one of the blocks that created the fork [NH20] [CCY+20]. This causes transactions to be delayed, impacts the performance and puts a strain on the network.

1.1 Motivation

Temporary forks have been garnering more and more attention. The picture below is taken from this paper [NH20], and it shows the number of forks in different Bitcoin Core versions. Bitcoin Core is the most popular way to run a node on the Bitcoin

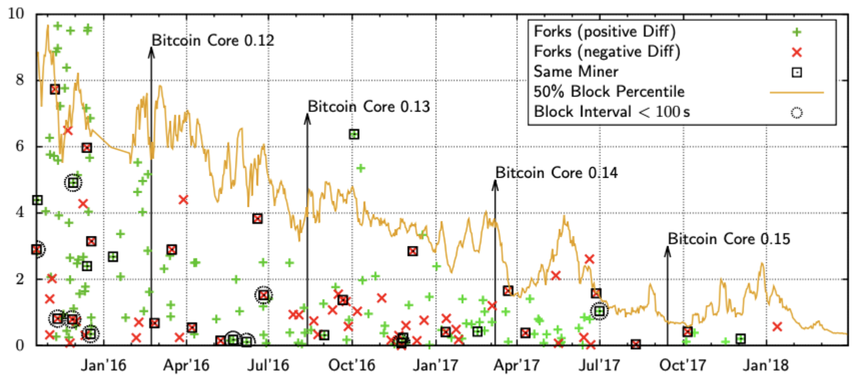


Figure 1.1: Study on block that led to forks [NH20]

network [Bitcoin Core]. The paper analyzed the Bitcoin network from 2015 to 2018 and analyzed the blocks that led to temporary forks. As we can see from the picture, temporary forks do transpire in Bitcoin. At the start, there were many occurrences of forks, and with improvements in the Bitcoin Core, we can see that the amount has decreased over the years. Every cross and plus sign indicates a fork, and the yellow line shows the average propagation time for a block throughout the network. When the propagation for the block decreases, the amount of temporary forks also decreases, as seen from this graph.

Certainly, temporary forks are an issue in Bitcoin and harm the network. One of the negative impacts of temporary forks is that some of the abandoned transactions in the block need to be redone. In turn, this increases the load on the network. Secondly, in this paper [GWR+16], it was shown that the likelihood of double spending occurring is higher when blocks are discarded because temporary forks cause inconsistencies on the blockchain. Double spending is a phenomenon that can happen in digital currencies, and what happens is that someone can spend the same digital currency twice. Thirdly, all the energy that goes into the mining process is wasted.

Temporary forks are not just something that affects Bitcoin, but every cryptocurrency that uses some variation of PoW. Therefore it is an important topic to understand. Researchers have been doing some research in this area. For now, one area which has seen some research is how the blockchain parameters such as block size and block propagation time affect the creation of forks [GWR+16] [VJR18] [CCY+20]. However, one area that has not been researched is how network parameters such as topology and delay affect forks creation.

This thesis explores how delay and topology affect the number of temporary forks. We have done experiments with 19 nodes running the Bitcoin Core, deciding the topology and delay for every node, seeing how the network behaves, and the number of forks created. We achieved this by using our Bitcoin Core testnet testbed. We then analyzed transactions included in a forked block to see how their confirmations/waiting time was affected.

We ran nine experiments with different configurations, each lasting 4-6 days. Temporary forks are hard to observe, and therefore we had to run each experiment for multiple days to be able to see forks occurring. In the beginning, we ran the experiment for up to 6 days. This resulted in observing more forks and transactions, which allowed us to see how forks might impact transactions. Due to the timeframe of the master thesis, only one run for each experiment run was done. Moreover, the timeframe for each run was shortened to 4 days.

1.2 Research objectives

Research objectives and questions are maintained from the pre-project in spring. The objective of this thesis is to find out how different network parameters affect the creation of forks. We achieve this by running experiments in a Bitcoin Core testbed and afterwards gathering the output from the experiments to answer the research questions below. When a fork occurs, transactions are usually discarded, which strains the network. Accordingly, we want to understand the impact that temporary forks have on the network.

Research questions:

– How will different network parameters such as topology and delay affect the number of temporary forks?

– To what extent will the temporary fork affects transactions waiting/confirmation time?

1.3 Thesis structure

In chapter 2, background theory is presented which includes an overview of Bitcoin, the consensus algorithm, Memory pool (Mempool) and the P2P. Chapter 3 deeply delves into temporary forks and the related work in this field. Chapter 4 explains the methodology used in this thesis. Furthermore, the nodes and the configurations for the topology and delay are elucidated in chapter 5. In chapter 6, the result is presented and discussed. Lastly, the further work that can be done is presented in chapter 7, and the conclusion of this thesis comes in chapter 8.

Chapter 2 Background

For Bitcoin to be used as a currency in today's economy, it needs to provide security in line with how central banks add physical features to disallow fraud. Since it is only a digital currency, Bitcoin needs other measures to prevent defrauding. Accordingly, different cryptographic techniques are used to help secure Bitcoin. This chapter begins by explaining the cryptographic fundamentals of Bitcoin. Next, the different Bitcoin components, such as addresses, transactions and blocks, are presented. Other technologies that Bitcoin uses are explained, such as what a p2p network is and what blockchain is. Lastly, there is a discussion on the security of Bitcoin.

2.1 Cryptographic fundamentals

To understand how Bitcoin functions, it is vital first to understand some cryptographic primitives. These are hash functions and digital signatures, which lay the foundation for the cryptography used in Bitcoin.

2.1.1 Hash functions

A hash function is a mathematical function that deterministically maps a message of any size to a digest of fixed size. It also needs to compute this efficiently. This means that the hash function's output is in a reasonable amount of time for a given message. Nevertheless, it must be cryptographically secure to be used in a cryptocurrency like

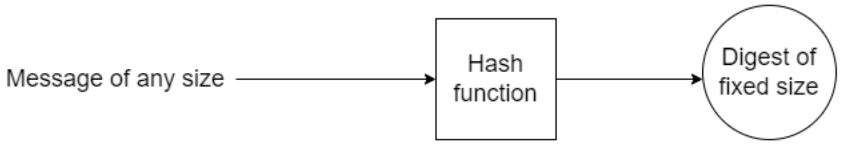


Figure 2.1: Hash function

Bitcoin. By this, we mean that the hash function needs to have these properties: collision-resistance, hiding, and puzzle friendliness [NBF+16].

1. **Collision-resistance** means that two different inputs cannot produce the same output. Therefore, a hash function is collision-resistant if nobody can find such a collision. For example, in Bitcoin, the hash that is used is SHA-256. Accordingly, since the hash function maps every message to 256 bits, collisions exist since the input space is bigger than the output space. However, users should not be able to find such collisions.
2. **The hiding property** asserts that for a given output, there should be no feasible way to assert something about the input for the function. If the hash function is y = h(x), there should be no way to figure anything out about x.
3. **Puzzle friendliness** is a property more specific for cryptocurrencies. If someone wants a particular output y from the hash function, there should not be any more effective strategy than trying random values.

2.1.2 Digital signatures

Digital signatures are supposed to be the equivalent of handwritten signatures. There are two properties we want from a digital signature. First and foremost, only you can make your digital signature, but the signature is verifiable by everyone. Secondly, your signature should not be used for other documents that you have not signed. This is achieved in Bitcoin by using these three algorithms [NBF+16].

1. (sk, pk) := generateKeys(keysize) This method generates two keys a public key (pk) which anyone can use to verify the signature, as well as a private key (sk) which are used as a signing key.
2. sig := sign(sk, message) This method takes in a message and a sk and outputs a signature for the message.
3. isValid := verify(pk, message, sig) The verification method. Takes in a pk, message and a signature as input and returns true if sig matches the public key.

The method used in digital signatures is very similar to asymmetric encryption, which also has a public and a secret key.

2.2 Bitcoin

Blockchain technology is relatively new, and Bitcoin is one of its first implementations. By using technologies such as blockchain and P2P network, the creators of Bitcoin

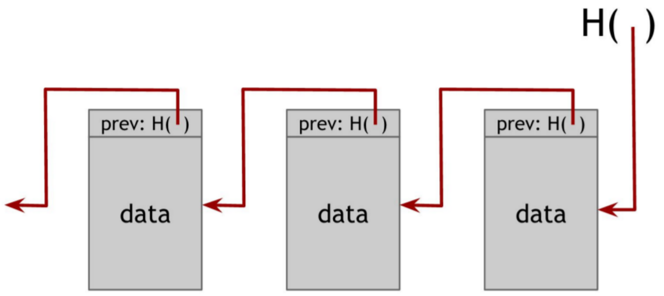


Figure 2.2: Hash pointer chained [NBF+16]

have created a decentralized economy. This chapter explains the different components of Bitcoin and how by using blockchain, Bitcoin has created a digital ledger that is hard to tamper with. Bitcoin is a decentralized system that uses a P2P network, and this chapter also explains how this is done.

2.2.1 Blockchain

Blockchain uses a data structure called a hash pointer. Each block has a hash value calculated from the block's data. Generally, a hash pointer is a data structure pointing to where some information is stored with some cryptographic hash of the information. In addition, what is unique with a hash pointer is that it allows observing that the information in a block has not changed, which is very useful in Bitcoin [NBF+16].

The Bitcoin blockchain is built using a linked list of hash pointers. This linked list of blocks is also referred to as a ledger. Essentially, with a regular linked list where each block has a pointer to the previous block, the blockchain has a hash pointer pointing to the previous block's hash. This makes it so that we know the value of the previous block and allow us to see that the value has not changed. Conversely, if someone changes the previous block's data, the hash changes since the hash is derived from the block's data. Therefore, if someone changes the data in one of the previous blocks, this would be detected. As a result, this makes Bitcoin and other cryptocurrencies secure and very tamper-proof.

2.2.2 Addresses

In Bitcoin, an address is a unique identifier that allows for money transfer. It serves as a virtual location where the money is stored until spent. Bitcoin addresses can either be associated with a script or a public key. The most straightforward

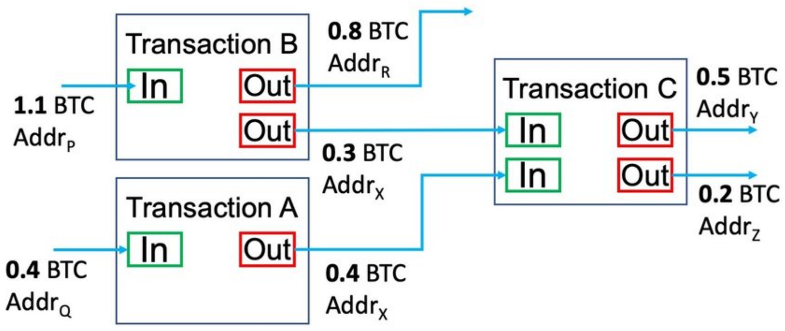


Figure 2.3: Transaction input and outputs [ZJMA16]

addresses are hashed Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA is a digital signature scheme from the USA [NBF+16]. Addresses can contain an amount of Bitcoin (BTC), the same as a bank account would contain money. BTC is an abbreviation for Bitcoin when talking about the Bitcoin currency. The BTC moves from one address to another, and anyone can create an address. The addresses can be used once or multiple times. However, using a Bitcoin address for multiple transactions would not be ideal for anonymity reasons. Usually, the more someone uses an address, the higher the likelihood of linking that address with a person increases.

2.2.3 Transactions

A Bitcoin transaction contains one or more inputs and one or more outputs. These inputs and outputs are addresses. The address used in the input must be an output address from a previous transaction. Moreover, an output address is referred to as Unspent transaction output (UTXO). In the transaction, the input must spend all the previous UTXO [NBF+16]. Suppose Alice received five BTC from one address and wanted to send three BTC to Bob. In the input, she would use the UTXO where she got sent the five bitcoins, and in the output, Alice would have one address that was Bobs and another output where she would send the remaining two bitcoins to herself. Alice would need to create a new address if she did not have one from before.

Furthermore, the number of bitcoins in the input must be greater than the output. The difference is the transaction fee the sender must pay for the node handling the transaction. This fee is to incentives nodes to keep verifying and handling transactions. Every transaction has Txid, which works as a unique identifier for a transaction. Figure 2.3, illustrates an example of a transaction and shows how previous output has been used as input in a newer transaction.

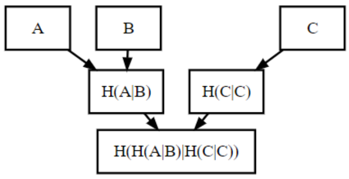


Figure 2.4: Block header data [BitcoinHeader]

2.2.4 Mempool

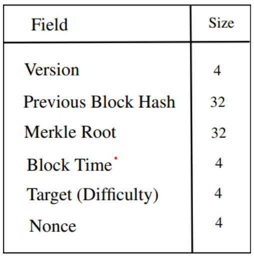
Every node in the Bitcoin network has a Mempool. The Mempool consists of unconfirmed transactions. These transactions are stored locally, and the Mempool of different nodes do not have to be the same. So, when a transaction is broadcasted to the network, the nodes take the transactions and save them to their Mempool. After that, they send the transactions to their peers to whom they are connected. Transactions are in the mempool until a block is found. The mempool can be configured differently for each node[Mempool].

2.2.5 Blocks

In Bitcoin, a block consists of the values seen in figure 2.5. The block size could not exceed 1 MB when Bitcoin first was created. However, the limit was increased to 4MB. Indeed this size increase was due to a soft fork called Segregated Witness (SegWit). What a soft fork is are explained in section 3.1.2. How SegWit works is that it separates witness data such as signatures from the input and output in transactions.

A block is generated on average every 10 minutes. The transactions in the Mempool are stored on this block and propagated throughout the network. The other nodes verify this block and add it to the blockchain. How many blocks have been generated in Bitcoin is referred to as the block height.

The first field in the header is version, and it tells us which Bitcoin Core version this block belongs to. Since Bitcoin launched in 2009, there have been updates, and this version tells us which version this block uses. The previous block hashes reference the previous block in the blockchain. Merkle root is constructed using all the Transaction ID (TxID) from transactions that are a part of this block. The coinbase transaction is always put first. It contains how much the miner takes as a transaction fee and the block reward. It also has an input field. The Merkle tree is an effective way to store and look up transactions in a block. It is constructed by hashing transactions together. An example of a Merkle tree can be seen in this figure



Field	Size
Version	4
Previous Block Hash	32
Merkle Root	32
Block Time*	4
Target (Difficulty)	4
Nonce	4

Figure 2.5: Block header data [NBF+16]

2.4. In the figure, A is the coinbase transaction. The rest of the transactions come after and are paired. Two and two transactions are hashed with a hash function until arriving at the top of the tree, where we are left with a hash that consists of all the transactions previous [BitcoinHeader].

Block time is when the miner starts mining for a block according to themself. This time has to be greater than the median time of the previous 11 blocks, and nodes do not accept blocks that contain a block timer more than two hours into the future. [BitcoinHeader]. The nonce is a random value the miner can change. The target is a 256-bit unsigned, and the header hash must be equal to or below this number to be valid.

2.3 Peer to peer network

The machines connected to the Bitcoin network and participating are called nodes. There are two types of nodes in the Bitcoin network: full and lightweight nodes. A full node downloads the whole blockchain, so it has all the transactions and blocks. While the lightweight nodes only download the headers to verify the authenticity of the transactions. These nodes are connected through a P2P network. In a P2P network, the nodes push/pull updates from their peers. The nodes push updates to their peers based on flooding. Flooding is a term used in computer science, which means that the node sends the incoming packet to every outgoing link except the link from where the packet arrived [ECP21].

A node typically has eight outbound connections to push transactions and blocks to Bitcoin. These eight nodes then validate transactions and blocks and forward the transactions and blocks to 8 more nodes. Even though every node is not directly connected, every node should have a path through the whole network through their neighbours. The nodes are also able to accept 117 incoming connections. This is a total of 125 connections. If one of the eight outbound connections drops, the node tries to replace it with one of the other connections [BKP14].

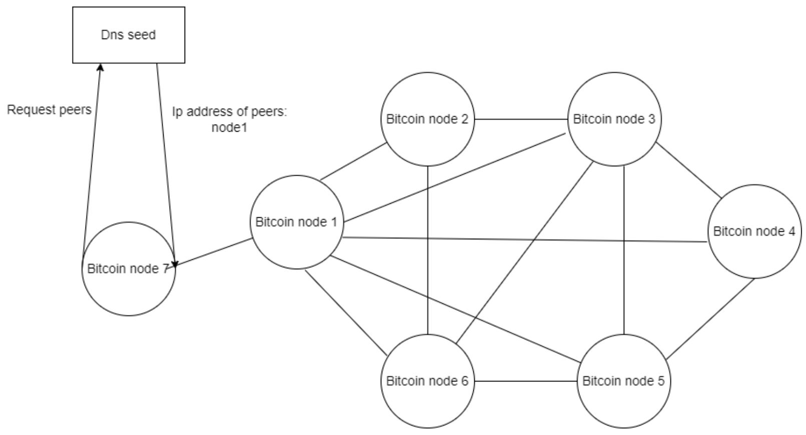


Figure 2.6: A simplified example of the P2P in bitcoin

2.3.1 The creation of Bitcoin topology

The way that node connects to the P2P network is by first querying a Domain Name System (DNS). This is so it can synchronize its local chain with the mainchain that the P2P network agreed upon. When a node first wants to join the network, it is not aware of any of the nodes in the network, and therefore it has to query the DNS. A DNS is continuously crawling the network, adding active nodes to their list. Usually, a DNS to query is hardcoded into the Bitcoin client. When the node has queried the DNS it receives nodes to connect to [ECP21].

Next, the node has connected to either one or multiple peers. After that, its peers can send address messages containing the IP address and port number of other nodes on the network. This is how nodes discover other nodes on the network with the help of their peers. When a node has discovered a peer, it connects to them by sending a version message. The version message contains a version number, block and current time to the remote node. Afterwards, the remote node responds with its version message. Both nodes send a verack message to confirm that the connection has been made. After the connection is made, the nodes can send each other getaddr to gain more peers [P2PNetwork].

An example of this can be seen in the figure 2.6. Node 7 is a new node that is not yet connected to the network. It queries the DNS seed for peers. The DNS responds with the IP address of node 1. Node 1 then sends address messages containing the IP address and port number of the other nodes after they connect. This is the way that Bitcoin forms the topology dynamically.

2.3.2 Communication in Bitcoin

After the nodes are connected, they can use pre-defined messages to communicate. The messages below are taken from this site [BitcoinDeveloper].

1. **getblocks** request an inv message that provides block header hashes starting from a specific point in the blockchain. This allows connecting nodes to get up to date with the blockchain.
2. **inv** A node can send this message unrequested to inform of a new block or transaction, or it can be a reply to getblocks or getmempool. This message is used to minimize network traffic. Instead of flooding transactions and blocks, a node simply sends an inv message to inform about a transaction or block. Then if a node does not have this data, it can request it.
3. **mempool** this message is used to ask for transactions another node has verified but not yet been added to a block. These are transactions that are in the other node's memory pool. The response to this message is usually several inv messages containing TxID.
4. **getData** a request for data objects such as transactions and blocks from another node.

2.4 Putting it altogther

Now we have looked at some of the fundamentals of Bitcoin. This section discusses how these fundamentals interact and how decentralization is achieved. The way Bitcoin works can be put into five steps taken from this book [NBF+16].

1. New transactions are broadcasted to all nodes.
2. Each node collects new transactions.
3. Each node works on solving a puzzle at each block height.
4. Other nodes accept the block only if the block is valid.
5. Nodes keeps building the next block on top of the latest block.

2.4.1 Transactions broadcasted

The transactions that are broadcasted come from users of the Bitcoin network. Users send money from one address to another. When a node receives a transaction, it broadcasts it to its neighbours. Afterwards, the neighbouring nodes store this transaction and propagate it to their neighbours.

2.4.2 Saving transactions

Immediately upon receiving a transaction, it takes the aforementioned transaction and saves it to their Mempool. It also checks if the transactions are valid by looking at the digital signatures to see if they are correct and have enough funds. The transaction remains in the Mempool until the node has found a block. When the block is found, it records the data onto the block and propagates the block throughout the network.

2.4.3 Proof of work

A block is generated every 10 minutes in Bitcoin. This is generated by nodes solving a cryptographic puzzle called PoW. PoW is a consensus algorithm that makes the network agree upon the ledger's state. The idea of PoW is that a node suggests a block every 10 minutes, and the process is randomized so that any node in the network can suggest a block if they can solve the puzzle. How it works is that nodes solve a heavy computational cryptographic puzzle. This puzzle is based on the hash functions described earlier, and solving this is called mining. The puzzle that the miners are solving is shown below.

$$SHA256(SHA256(header)) < target$$

As we can see, the formula for the puzzle is a double SHA256 hash of the header. The miners find a valid block by changing some of the inputs in the Bitcoin header until it finds a block below the target. The header is the fields in 2.5. The miner can change nonce how they want, and change block time a little. Furthermore, they also can change coinbase transactions in the Merkle root. Coupled with that, the miner can use the input field in the coinbase transaction as another nonce, but changing the input field in the coinbase requires recomputing the Merkle tree, which is a more expensive process than just changing the nonce in the header. The way miners solve the proof of work can be summed up into these five steps [NBF+16].

1. First, the nodes have to choose which transactions from the Mempool are a part of this block.

2. Decide the input for the coinbase transactions.

3. chooses a nonce, then compute a hash from the formula above.

4. If the hash is bigger than the target, they return to step 3. If all nonces have been tried, then go to step 5.

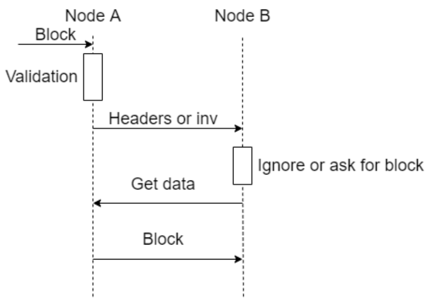


Figure 2.7: How blocks are propogated through the network [BlockRelay]

5. Chooses a new input for coinbase data, recompute Merkle root, then return to step 3.

For now, blocks are generated every ten minutes. Since the nodes solve hash functions, the more CPU power a node has, the faster they are likely to solve the hash function and find a block. Bitcoin has to consider the CPU power of every node combined to create puzzles that only generate one block every 10 minutes. Because if the puzzle is too easy, the block generation time is less than 10 minutes, which is not the intended block generation time in Bitcoin. For this reason, the formula below is used to determine the difficulty of the puzzle.

$$NewDifficulty = oldDifficulty \times \frac{2016 \times 10}{minutesUsedToMineLast2016Blocks}$$

This formula is used to control the difficulty of the puzzle. Depending on the time it took to mine the previous 2016 blocks, the difficulty can increase or decrease. Usually, it increases, but at certain times the difficulty decreases. Since having more CPU power makes it more likely to solve the puzzle, miners go into groups called mining pools.

2.4.4 Block relay

After a node finds a block, it saves the transactions on this block and propagates it through the network. The other nodes then receive this block. After they have received it, they validate that the block is correct. This is done by checking transactions and signatures. Afterwards, it sends an inv message or the block header to neighbouring nodes. Neighbouring nodes check if they have this block already. If they do, they ignore this inv message or reply with a getdata message if they do not have this block. Then the node replies with a message containing the block [BlockRelay]. This interaction can be seen in this figure 2.7.

2.4.5 Transaction confirmation

When a block is found, the transactions are stored upon this block. Nodes propagate this block to their neighbours. Immediately after, the node starts to find the next block in the chain and stops to work on finding the block on the height that was just found. It is very natural to think that a transaction is deemed confirmed when it is a part of a block, but this is not the case. Transactions in the block just found are said to be at depth 1. For each subsequent block found, the depth of these transactions increases by one. To be secure from double-spending, transactions are usually not considered confirmed until a transaction is a specific block deep. Transactions are shown as validating and waiting for more confirmations. The transaction is shown confirmed when the transaction is six blocks deep in the classic Bitcoin client [BitcoinConfirmation].

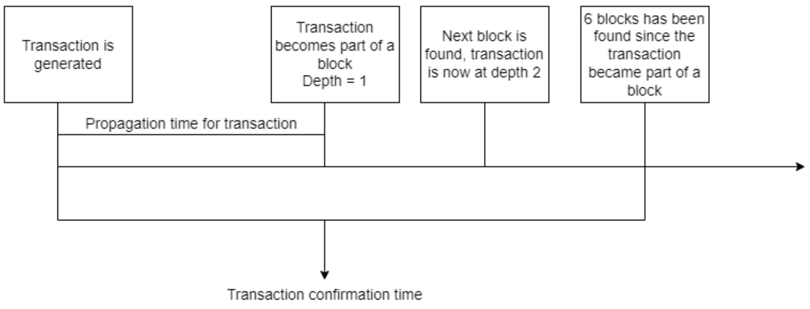

Transaction is generated
Transaction becomes part of a block Depth = 1
Next block is found, transaction is now at depth 2
6 blocks has been found since the transaction became part of a block
Propagation time for transaction
Transaction confirmation time

Figure 2.8: Transaction confirmation in Bitcoin

2.5 Bitcoin incentives for nodes

As mentioned earlier, nodes are volunteers. As long as the device meets the technical requirements, there is no restriction on which devices can become a node in the Bitcoin network. However, what are the nodes gaining from this? Firstly, the node that solves the PoW is rewarded with BTC. At the beginning of Bitcoin, the reward was 50 BTC for the node that found a block. For every 210 000 blocks that are found, this reward is halved. Currently, the reward is 6.25 BTC. Ultimately, the reward will reach zero. Nevertheless, nodes are also rewarded a transaction fee for including transactions in blocks. When the block reward reaches zero, the transaction fee will be the most prominent incentive to be a node in Bitcoin. These are the main reason for operating a node in the Bitcoin network [NBF+16].

2.6 Security and protection

As said in the beginning, if Bitcoin is to be used as a currency, it must be a secure payment. In this section, we cover how the components of Bitcoin provide security from adversaries. We are looking at three scenarios: stealing BTC, denial of service and double-spending. Denial of service and double-spending are problems more known with the use of the internet compared to regular currencies [NBF+16].

Can person A steal Bitcoin belonging to person B? No. The reason for this is the protection Digital signatures give. With a digital signature scheme, only the owner of those addresses can sign off on the transaction. If someone else tried, this would be figured out upon inspection of the transaction.

Denial of service is an attack where someone tries to shut down a service making it inaccessible for users. In Bitcoin, someone can refuse to add transactions from specific users onto the block. So, person A can refuse to add transactions from person B when suggesting a block. In this case, person A has refused person B of service. However, person B must wait for the following block to get their transaction on. This attack is inefficient because block generation is randomized, so one miner can not indefinitely refuse service for person b.

Double spending is a problem in electronic cash. Is when a user can spend the same coins in different transactions. How this can occur in Bitcoin is if person A sends money to person B, this transaction propagates throughout the network and becomes a part of the next block. However, the next block that is being proposed, if this is proposed by person A, they can ignore the previous transaction they did to B, and spend the output they used in the previous transaction and send the money to themself.

2.7 Summary

Bitcoin is a P2P network consisting of volunteer nodes. It uses hash functions and digital signatures as a foundation. The basic component of Bitcoin is addresses, transactions and blocks. Addresses work like bank accounts. They are used as storage for BTC. In addition, they are used in transactions as inputs and outputs. Every transaction has TxID. When a user creates a transaction, this is broadcasted to all nodes in the network. The nodes save these transactions in their Mempool. At the same time, they are solving the PoW. After solving the PoW, they receive a block, and the node is rewarded with BTC. The node saves the transaction from the Mempool upon this block. Subsequently, they propagate the block throughout the network. The other nodes verify that transactions are correct and add this block to the blockchain. In turn, every node keeps building upon this latest block.

Chapter 3

Temporary fork and related research

This section presents the topic of forks and the related research done in the field of temporary forks in Bitcoin.

3.1 Forks

There exist three types of forks in Bitcoin; hard, soft and temporary. Even though this thesis focuses on temporary forks, soft and hard forks are also explained.

3.1.1 Hard fork

A hard fork in Bitcoin is when radical changes happen to the network protocol, and the rules of the Bitcoin protocol change so much that it creates a new protocol. This can happen intentionally but also unintentionally. Since when a hard fork happens, it is not backwards compatible, so every node must update its protocol to this new protocol [HardForks]. This kind of fork is usually created by developers and crypto community members who believe something can be upgraded. For example, in 2017, Bitcoin Cash was created due to disagreement with some Bitcoin protocol changes. Some developers did not like the new changes introduced to Bitcoin, so they made other changes to the Bitcoin protocol, resulting in a new cryptocurrency called Bitcoin Cash. The picture 3.1 in the top section shows what a hard fork does. It results in two different coins.

3.1.2 Soft fork

In contrast, a soft fork results from more minor changes to the network protocol. It has been used in Bitcoin to add new functionality that is backwards compatible. Changes like pay to script hash and SegWit are implemented using a soft fork. SegWit was created to circumvent the block size limit in the Bitcoin protocol. Pay to script hash allows for backwards compatibility with new transaction types such as SegWit [softForkTerms]. More than 50% of the nodes need to adopt the changes

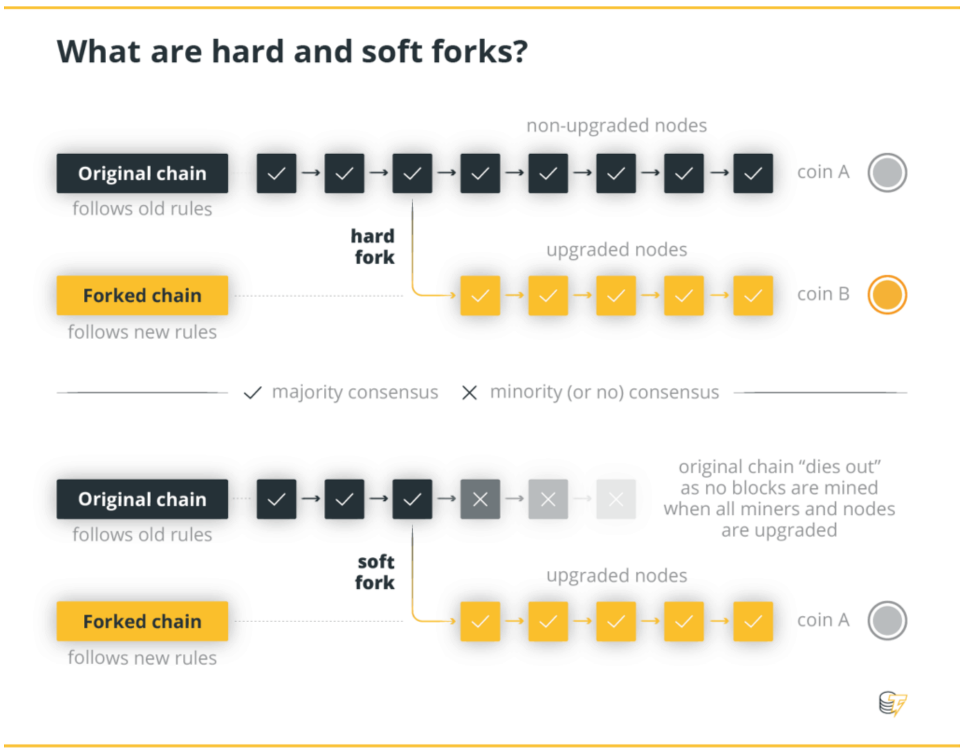


Figure 3.1: Hard and soft forks [HardAndSoftForks]

for these changes to be implemented. Opposite of a hard fork where all the nodes that want to participate in that network have to implement the changes. Blocks that come from nodes using the old rules are ignored by nodes that have implemented the new rules. However, the old nodes accept blocks from nodes running the new rules. The picture in 3.1 in the bottom section of the image shows what a soft fork does to a cryptocurrency.

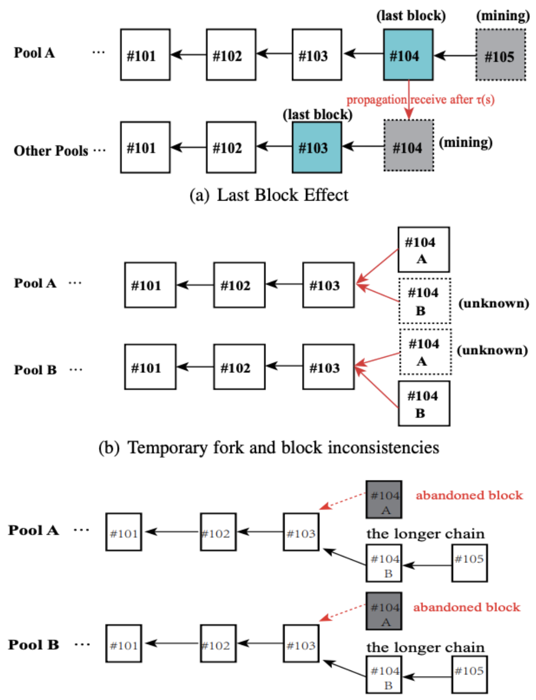


Figure 3.2: Example of an temporary fork [CCY+20]

3.2 Temporary forks

In the background section, we explained how PoW works and how nodes compete to solve this puzzle. This process of finding a block is randomized. Sometimes, two miners find blocks at the same height at an almost similar time. What then happens is a split in the blockchain with two competing forks. The process of how temporary forks are created and resolved might be easiest to explain with an example. As we can see, pool A mines block 104. The mining pool are miners that share processing powers and divide the block rewards between themself. Accordingly, this example works similarly for single miners.

When mining pool A finds block 104, it propagates this block throughout the network. While block 104 is propagating through the network, other mining pools still work on finding block 104 since this has not been found from their point of view. This is because propagating a block throughout the network takes time, depending on factors such as network delay and speed. While block 104 propagates, one of the other miners finds block 104 and propagates this block throughout the network. Now we have a split in the chain with two conflicting blocks. In this situation, we have split because both blocks were found at height 104, and they reference the same previous block. The reason for forks happening is because of the time it takes a block to propagate the network. In this period, another block might be found.

When nodes receive a block, they validate it and try to find the next block. What transpires is that some nodes closest to mining pool A get block 104A while nodes further away from node A receive the block from mining pool B. The nodes keep building on the block they received first. Forks are solved by miners choosing which chain the subsequent block is formed upon. In this example, block 105 was formed on the chain containing block 104B, which prevailed. The other fork was abandoned [CCY+20]. Abandoned blocks are referred to as stale blocks. In Bitcoin, the consensus algorithm work, such as the longest chain of blocks prevails, and the others are abandoned.

Nodes usually push blocks to their peers. A Bitcoin node has eight peers that it pushes blocks to. When a node gets two valid blocks at the same height referencing the previous block, both have valid transactions; a fork has occurred. When a node has two conflicting blocks, it stops pushing blocks to its peers since it does not know which of them is going to be part of the mainchain. This mechanism makes it so that some nodes are unaware of the forks in the Bitcoin network. From the example above, when a node first gets block 104A, it pushes this out to its peers, but when that same node gets block 104B, it does not push this block to its peers since this is a conflicting block. Therefore when wanting to see the number of forks, it can be tricky since every node might have a different perspective of the number of forks that as has occurred.

What happens to the transactions in block 104 A? These transactions are discarded. Most likely, blocks 104A and 104B would have similar transactions stored inside them. This is due to, as we know, transactions are broadcasted through the whole network, and the two mining pools therefore have very similar Mempool. However, there might be differences for the transactions that are a part of block 104A but not included in block 104B. These transactions are sent back to mining pool A's Mempool in the wait for a new block to be found. These transactions are undoubtedly delayed by a certain amount. In Bitcoin, as explained previously, a transaction is confirmed after being six blocks deep. Since these transactions have to wait to be a part of another block, they are delayed by one block. The transactions that were part of both blocks might not see much delay.

3.3 Releated research

Bitcoin is a new technology that was released in 2009. To understand what Bitcoin is, the white paper first written about Bitcoin was used [Nak09]. To better understand temporary forks, several research papers have been read.

3.3.1 Blockchain parameters and their effect on forks

The first [GWR+16] paper studies the security and performance of PoW blockchains. This paper looks at the different security aspects of Bitcoin and different attacks. However, their research into temporary forks was most relevant for this thesis. They studied how an attack called selfish mining can increase the likelihood of a temporary fork. As well as blockchain parameters such as block size and block generation time and how they impacted the creation of temporary forks. The paper found that when the block size increases, the percentages of temporary forks increase. Furthermore, when block generation time decreases, the number of forks also increases. The reason for this is that when the block size increases, it takes a longer time for nodes to validate the block, which causes a more extended propagation throughout the network. While this block is propagating throughout the network, another node can find a block and propagate it as well. When the block generation time decreases, there are more opportunities for a fork. Every block that is generated can cause a potential fork.

Selfish mining is an attack where a miner finds a block but withholds this block. So, the miner does not publish the block they found to the network. This is done so that the miner can get a head start working on the next block at the next height. The miner withholding the block has created a fork in the Bitcoin since they are working on their private chain. If the miner finds additional blocks, they create a lead on the public chain. When the public chain catches up to the miner, they reveal the private chain they had kept hidden. Resulting in other miners working on the public chain wasting their effort. This strategy has been shown to give miners more BTC reward than they probably should based on the mining power they are holding [EG13]. In the paper [GWR+16] they showed that the number of temporary forks increased when nodes used the tactic of selfish mining.

3.3.2 Statistical analysis on forks

The second paper [NH20] made an empirical analysis of the announcement and propagation of blocks that caused a fork in Bitcoin. The study was based on measurements from the Bitcoin network since 2015. They analyzed the time difference between the first announcement of competing blocks and the average block propagation delay. Additionally, they analyzed the effect of a block having a head start over a competing block and what was the probability of the block that was announced first becoming a part of the main chain. Some of the results from their research can be seen in figure 1.1. What this figure shows are forks and the average propagation delay in Bitcoin. The green positive signs are occurrences of forks where the block that was announced first becomes part of the main chain, whereas the red x's are forks where the block that was announced second becomes a part of the main chain. The yellow line shows the block propagation of a block, and when this was higher, the amount of fork occurring was also more.

3.3.3 Delay is the main reason for forks

The third paper [DW13] analyzes how Bitcoin uses multi-hop broadcast to propagate transactions and blocks. Then they take the information they gathered to verify the conjecture that the primary cause of forks in blockchain is due to propagation delay in the network. They implemented a node that listened to the Bitcoin network and collected timing information from blockchain height 180 000 up to 190 000. The timing information contained the hash of the block, the announcing nodes IP and a local timestamp for the block. They noticed that the size of the message and the propagation delay have a strong correlation. When the size increased, so would the delay. For blocks more significant than 20 kilobytesKilobyte (KB), each kilobyte would increase the propagation delay by 80 Millisecond (MS) until most of the network was aware of the block. From the 10 000 blocks, they observed that the stale block rate was 1.69 %. This would amount to 169 temporary forks observed. Since a fork occurs when a block is found when another propagates throughout the network, they also suggested methods to decrease the propagation time. The ideas they came up with to improve propagation time were; minimizing verification, pipelining block propagation, and connectivity increase.

3.3.4 Objectives

Temporary forks have garnered research attention, as these three papers show. However, these papers do not control the delay on the Bitcoin network, nor have they looked at the topology during their research. We intend to create a testbed where we control the Bitcoin network and have complete autonomy over the delay and topology. In the next's chapters, we go through the whole setup for our experiments.

Chapter 4 Methodology

For this thesis, the objective was to show how delay and topology affected temporary forks, and to what extent temporary forks affect waiting/confirmation time. To answer these questions, empirical research methodology was applied. For this thesis, we had set up an experiment from which we collected and analyzed data to answer our research questions. This chapter presents our research methodology, research design, data collection and analysis, and how the literature study was done. Difficulties and limitations are discussed at the end.

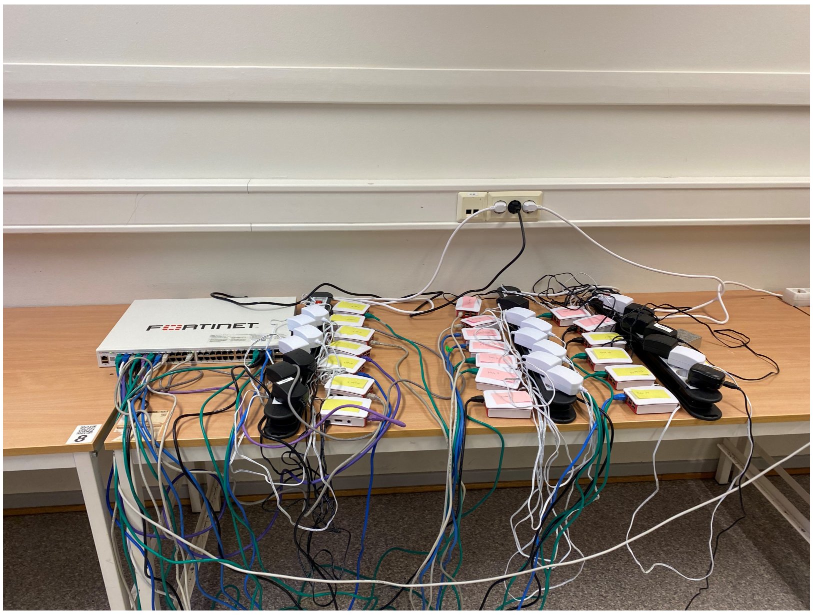


Figure 4.1: Testbed setup (20 Raspberry pis and 1 router

4.1 Research methodology and design

Empirical research methodology is research where the conclusion is drawn from the results of the data collected. This data can either be numerical or non-numerical. The empirical evidence is the data gathered from research. This can be done by either using quantitative research or qualitative research. Quantitative research gathers information through numerical data, while qualitative research is done by gathering non-numerical data [Empirica]. These methodologies use the data they gathered as empirical evidence to form their opinions.

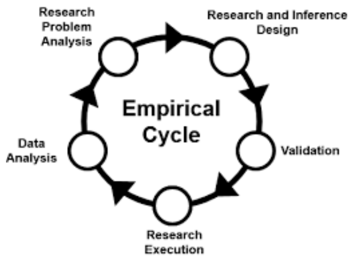


Figure 4.2: The empirical cycle [Hag20]

In this thesis, we performed experimental research. The reason for this is that we wanted to test how the two network condition parameters delay and topology affected the number of forks. We did this by creating our testbed with full autonomy of these variables.

This breakdown of the empirical cycle is partly taken from my pre-project. The empirical cycle consists of five steps. These five steps are as following:

1. **Research problem analysis**
   - This first step was mainly done during the pre-project last semester. We received a problem description, and we had to start by doing a literature study. At first, the literature was provided by thesis supervisors. Later on, literature was found more independently. It started with papers describing the workings of Bitcoin and then papers about temporary forks. By reading papers, we came to the research questions that were mentioned in chapter 1.
2. **Research and inference design**
   - The next step was to find a way to answer the research questions. The chosen research design was experimental research with data collection. This was achieved by having 19 raspberry pies that were running Bitcoin

Core. These nodes created transactions and blocks with the help of scripts presented in the next chapter. Node 1 was not creating transactions or blocks. Node 1 was the node we used to control the other nodes and run scripts from by using Secure Shell (SSH) we connected to node 1. There were multiple runs with variations of delay and topology. Three different delays and three different topologies were used during the experiments. By changing the delay and topology, we want to see its effect on the creation of forks.

The chosen delays were based on the monitoring of the Bitcoin network. Where the lowest amount of end to end delay that was monitored was 11 MS. The average was 130 MS, and the most significant delay was 200 MS [GHJ21]. The network generated a block every ten minutes, which is the same as Bitcoin, and transactions were set to appear once every 2 seconds on average. After each run of the experiment, data was collected.

3. **Validation**

   – To validate our experiment, we had several test runs. These runs had different purposes. The first run was to see that blocks were generating and that we could collect blocks and transactions from the network. Then two runs were done to see if the network would create a fork and if we would be able to see it. When we got confirmation that we could see a fork in the network, we started planning for the actual experiments.

4. **Research Execution and data collection**

   – The experiment had, in total, nine runs, each running for 4-6 days. After each run, data was collected for analysis. The configurations of the experiments can be seen below:

Experiment run	Delay	topology
1	11 MS	2
2	11 MS	4
3	11 MS	8
4	130 MS	2
5	130 MS	4
6	130 MS	8
7	200 MS	2
8	200 MS	4
9	200 MS	8

The nodes created blocks and generated transactions. These were written to two separate log files inside the nodes. The log file containing transactions contained:

a) txid: a hash of the TxID

b) date and time: when the transactions are made.

c) ntx: number of transactions in this block.

The log file containing the block information:

a) BlockId: a hash of the block.

b) date and time: when the block was generated.

c) ntx: number of transactions in this block.

We also collected from the Mempool of node 1. We did a call every 1 minute to the Mempool of node 1. We first gathered data from the Mempool almost every second, but this resulted in too much redundant information. Therefore, we changed it to every minute to make the data gathering more manageable. The Mempool of the different nodes might not be completely identical, as explained in 2, but they are very similar, so it is enough to monitor node 1s Mempool. From the Mempool, we gathered:

a) txid: a hash of the TxID

b) transaction fee: The transaction fee is the same for every transaction this is $1.41x10^{-05}$

c) epoch-time: the amount seconds since 1970.

d) height: Tells us which block this transaction belongs to.

We also collected from the main chain. This showed us which blocks became part of the main chain. What we collected was:

a) BlockId: a hash of the block.

b) epoch-time: the amount seconds since 1970.

c) height: Tells us where in the chain this block belongs to.

Lastly, we collected from the forked blocks. By using API commands from the Bitcoin Core, we got information about the blocks that caused forks and the transactions these blocks contained.

5. **Data analysis**

We analyzed the data we got from the experiments. We used the data to answer the research questions we had for the thesis. The two questions were:

a) How will different network parameters such as topology and delay affect the number of temporary forks?

b) To what extent will the temporary fork affect transactions waiting/confirmation time?

To answer the first question, we used an API command in Bitcoin Core named *getChainTips*. This gave us all the different chain tips in the Bitcoin network. We identified how many forks were created by counting the number of chain tips. However, to answer the second question, we first started looking at the transactions from the forked blocks. This was done by using another API command, *findBlock*. *findBlock* takes in a hashId for a block and returns that block with all its content. This gave us all the transactions in a forked block, but we needed to determine when these transactions were generated. We accomplished this by matching the transactions in the forked block with the log file that contained transaction information. This was done by using pandas. Pandas is an open source python package used for data analysis. We now had a table containing the transactions and the date and time generated. We now needed to find which blocks these transactions belonged to. This information was stored in the data gathered from the mempool. Therefore we did another join between the new table and the data gathered from the mempool. This resulted in the table underneath:

a) TxID

b) Date and time

c) block height

The next step was to find when the block was 6 blocks deep to find confirmation/waiting time. This was done using pandas as well. From the data gathered from the main chain, we knew when the different blocks became a part of the main chain. We then take the time when the sixth block became a part of the main chain and subtract it from when the transaction is generated. Usually, this is 1 hour, but when a fork happens, some transactions might need to be discarded and put back into the mempool, as explained in section 3. However, most of the time, the transactions in the forked block are also in the competing block that becomes part of the main chain and does not have that much delay on confirmation time. However, some transactions might have to wait for a more extended period to be confirmed. Figure 4.3 shows how the different log files are joined to find the confirmation time for a transaction. The table we were left with looked like this:

a) TxID

b) Date and time: transaction generation time

c) block height
d) time(block-height+7)
e) conf-time: block-height+7 - transaction generation time.

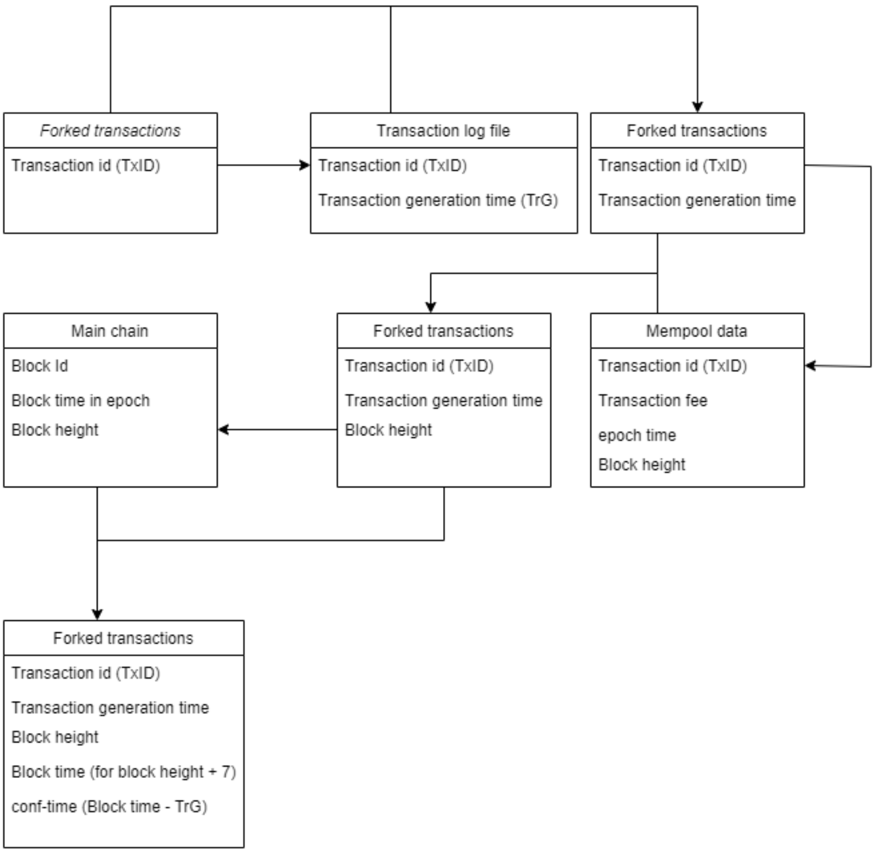


Figure 4.3: How data collection is done

4.2 Difficulties

Being able to see multiple forks has been a difficulty. In [NH20] they observed that from 10 000 blocks generated, only 169 temporary forks were created, which is 1.69%. That is not a large amount, and only having 19 nodes makes observing forks harder. Therefore we have had to increase delay and activity on the network. We increased the amount of transactions in the network to put some load on the network. Because fork occurs less frequently, and with only 19 nodes, we need to push the network so it can produce forks.

Time constraint has also been challenging. Due to that, each run takes 4-6 days. It does not leave room to do multiple runs. However, doing nine runs with different delays and topologies is still very beneficial. The experiment timeframe was from the end of March to the beginning of June. Experiments, where network parameters such as delay and topology are being controlled are new and beneficial to this research area.

Chapter 5

Design And Setup

This chapter gives an overview of the equipment used and then explains how to set up and run the testbed. The script used during the experiments is explained and a flowchart showing all the steps done before each experiment is given.

5.1 Physical devices

The physical devices consisted of 20 Raspberry Pies and one router. The router was a Fortinet router, and the 20 Raspberry Pie are connected to this with ethernet cables. In addition, the raspberry pies were running the Bitcoin protocol and are called nodes. During the setup, each node had been given a number from 1 to 20. The IP adresses for the nodes ranges from 192.168.2.1/24. Node n would refer to the node with an IP address ending with that number. When we received the nodes, they were empty. Therefore, an SD card was used to install software onto these machines. They were installed with Ubuntu 20.4 and Bitcoin Core software needed to run the simulations. To control the network, we had an MSI laptop that used Putty to connect to node 1 through SSH. We were able to run scripts and SSH to other nodes in the network from this node. The specification for the nodes was:

Specifications	Raspberry pi 3 model B
Processor	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
Memory	1GB RAM
Connection	BCM43438 wireless LAN and Bluetooth Low Energy (BLE)
Ethernet	100 Base Ethernet
Ports	4 USB 2 ports and Full size HDMI

5.2 Scripts

This section presents the scripts and API commands used to set up the experiment and answer the research questions. At the end of this section, a flow chart showing how the scripts and API commands are used before each experiment is shown.

5.2.1 Experiment setup

As explained in 5 we use a personal computer to connect to node 1. From there, we accessed the scripts to run the experiments. The first script that was run on the nodes was *RemoveChain*. This script removed the regtest directory in Bitcoin Core. The regtest maintains the wallets and the chain state. By deleting this regtest and restarting Bitcoin Core, it would create a new regtest [regTest]. This is very important so we have a baseline for the experiments. After this, we ran the *startnodes* script that started the Bitcoin Core software on all the nodes and created a new regtest since that had been deleted.

Algorithm 5.1 Topology

```
N -> List of nodes
Initialisation: Tt = number of peers

for each n in N do:
  for i = 1 less than Tt do:
      #create outbound connection for n
      n connect with n+i
      increment i
```

After the Bitcoin Core software was up and running on all the nodes, we created the topology for the network. The topology we decided to have for these experiments was either 2,4, or 8. This was achieved by the script seen above. How this script worked is that it would connect every node with either the 2,4 or 8 adjacent nodes. For example, node 2 would be connected with nodes 3 and 4 in a two-peer topology. In a four-peer topology, node 8 would be connected with 9,10,11, and 12. With an 8-peer topology node 19 would be connected to 20,1,2,3,4,5,6,7. An example of what a two-peer topology looks like can be seen in figure 5.1.

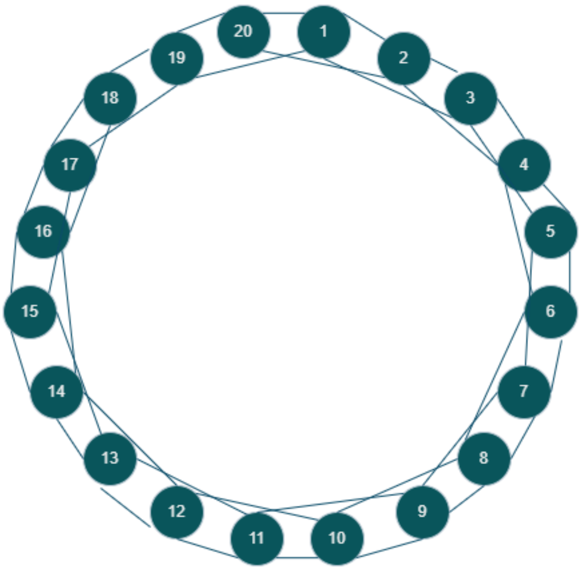


Figure 5.1: Example of 2 peers topology

After that, the topology for the network was created. We created a wallet for the nodes and generated 110 new addresses with a script called *resetChain*. *resetchain* script also loaded the nodes with BTC to use for the experiments. The nodes 17-20 would, on instances, not receive sufficient BTC to use for transactions for four days. Therefore *resetchain* script was used on these nodes separately once or twice more.

Next, the transaction fee was set to 0.00001 for every transaction. This was set so that the BTC funds would not empty during the experiment. Afterwards, the delay was configured for all the nodes. The delay was either 11, 130 or 200 MS. This delay was not constant, but it would be the chosen delay on average.

5.2.2 Generating traffic

Traffic, as in transactions and blocks, was generated after the previous steps were taken. The transaction script creates transactions in a negative exponential distribution. The script allowed us to decide the intensity of the transactions in the network. A transaction was created every 2 seconds. This resulted in 30 transactions per minute and 300 transactions every 10 min. We had 19 nodes; therefore, the number of transactions the network would create was 5700 every 10 minutes. Every node would write the transactions they created on a python log file.

Algorithm 5.2 Generate transactions

```
gen_transaction(interval)
While True:

    idle = negative_exp(interval)
    #Return a random number from this distribution.
    #This thesis interval time was 2 seconds

    sleep(idle)
    transaction = generate_transaction()
    file.write(transaction)
```

The script below was responsible for generating blocks. It used the same distribution as the transactions. This script allowed us to decide the block propagation time in the network. We can decide if we want a block generated every 2,5, or 10 min. Since we are working with Bitcoin, we choose to have a block generating every 10 min. Each node would, on average, generate a block every 11400 seconds because $11400/19 = 600$. Therefore a block would be generated every 10 min or 600 seconds. When a node generated a block, it logged this to a python log file.

Algorithm 5.3 Generate blocks

```
Generate_block(Inter-block time):
While True:

    idle = negative_exp(Inter-block time)
    #Return a random number from this distribution.
    #This thesis Inter-block time was 11400.

    sleep(idle)
    block = generate_block()
    file.write(block)
```

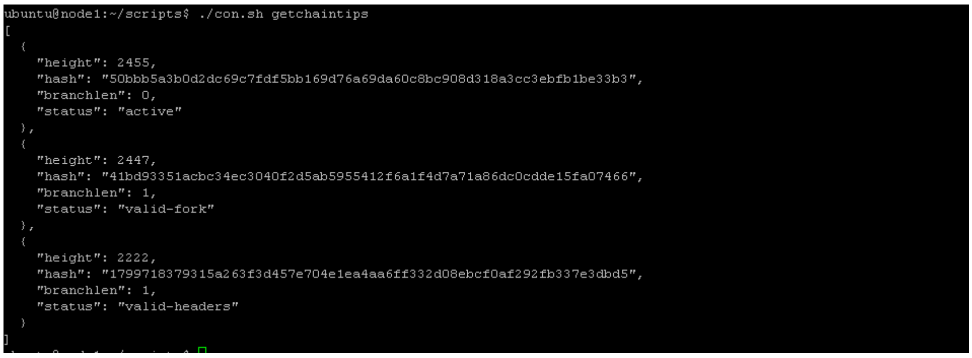
5.2.3 Data gathering

For data gathering, two scripts and two Bitcoin API commands were used. One of the scripts was used during the experiment, while the other scripts were used after. The two API calls were used after each experiment as well. The script that ran during the experiment was run after the transactions and blocks started to generate. It would poll from node 1's Mempool every 1 minute, and it would return every transaction in node 1's Mempool as a JSON file. The second script was run after the experiment and would retrieve the whole blockchain.

The first API-command *getchaintips* contained all the information about the tips of the blockchain tree. The result from this command displayed if there were forks on the blockchain [getchaintips]. Below is a list showing the different results this API command would give.

1. Active: This is the tip of the mainchain.
2. invalid: this tip contains at least one invalid block.
3. Valid fork: this tip is not part of the active main chain, but it has been validated.
4. Headers-only: not all blocks for this branch are available.
5. Valid headers: all the blocks are available, but they were not all validated.

An example of what a result from this API command would look like can be seen in figure 5.2.



```
ubuntu@node1:~/scripts$ ./con.sh getchaintips
[
  {
    "height": 2455,
    "hash": "50bbb5a3b0d2dc69c7fdf5bb169d76a69da60c8bc908d318a3cc3ebfb1be33b3",
    "branchlen": 0,
    "status": "active"
  },
  {
    "height": 2447,
    "hash": "41bd93351acbc34ec3040f2d5ab5955412f6a1f4d7a71a86dc0cdde15fa07466",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 2222,
    "hash": "1799718379315a263f3d457e704e1ea4aa6ff332d08ebcf0af292fb337e3dbd5",
    "branchlen": 1,
    "status": "valid-headers"
  }
]
```

Figure 5.2: A Result from *getchaintips* command

From the *getchaintips* command, we could see which blocks created a fork. After we found that out, we then used the second API command *findBlock.* This command would take a block's hash as an input and return that block if it existed. From figure 5.2 there is a valid-fork. Then this block would be retrieved with the API command *findBlock*. *findBlock* would return many data relating to that block. However, for this thesis, the ones highlighted are relevant [getBlock].

1. Confirmation: if this is -1, the block is not in the main chain, and it is a fork.
2. Height: the block height.
3. ntx: number of transactions in this block.
4. tx: list of the transactions in this block. They are listed by their transactions id.

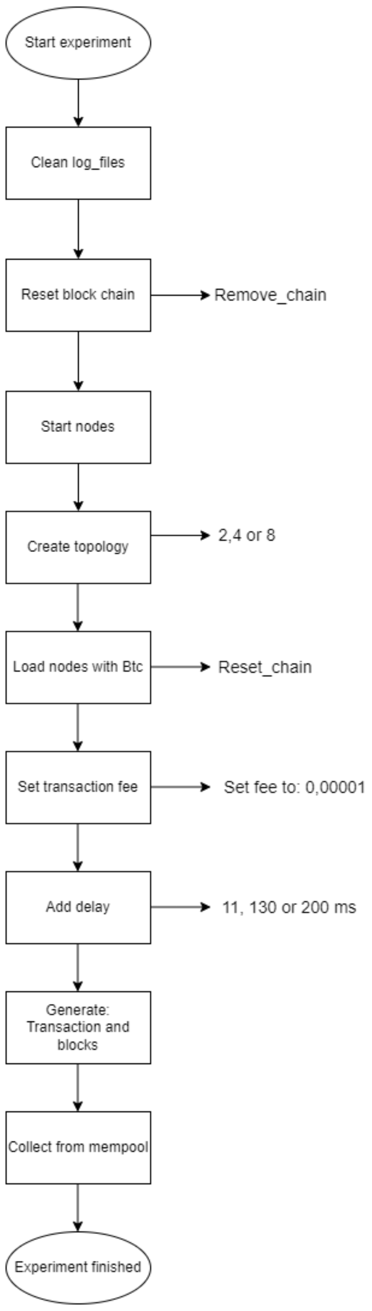


Figure 5.3: The setup for the experiment

Chapter 6 Result

This chapter presents the result of the different experiments and discusses these results. There were in total nine different configurations of the experiments. The data analyzed in the result chapter is generated from the first 500 blocks. The first 500 blocks were chosen as a benchmark to standardize the analysis. First, we present the number of forks in the whole network. After that, we show the number of forks seen from node 1's perspective. Then we show the confirmation/waiting time for transactions in a forked block from the whole network. Afterwards, we observe the transactions in a forked block from the perspective of node 1. After we present the results in each section, we discuss them. We show the perspective of what the whole network observes and what only node 1 observes because someone usually controls one node in the Bitcoin network. However, for our experiment, we control the whole network. This allows us to highlight the differences between the two perspectives.

6.1 The number of forks from the whole network

We used the API command *getChainTips* on every node to see the number of forks in the whole network. Some of the nodes would show the same forked block; if this were the case, the block would only be counted once.

6.1.1 Results from 11 MS

On the x-axis, we have different topologies, while the y-axis shows the number of forks. The x-axis shows the different topologies, and topology 2 means that each node has two peers, and topology 4 means that each node has four peers, and the same goes for topology 8. From the figure 6.1 we see that topology 2 creates the most forks with five and topology 8 with the least with just two. While topology 4 has three forks. The number of forks decreases when the number of peers a node has increases.

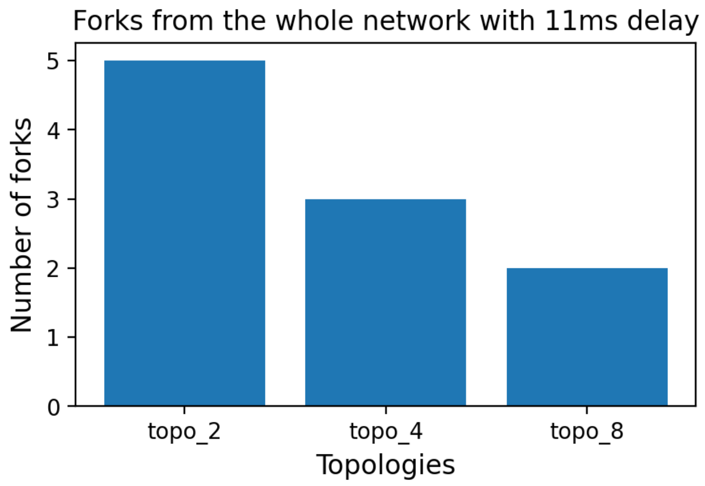


Figure 6.1: Number of forks from the whole network 11 MS

6.1.2 Results from 130 MS

From figure 6.2 we observe the number of forks created when the delay in the network was 130 MS. Topology 2 has five forks, while topology 4 has ten. Meanwhile, topology 8 has four forks.

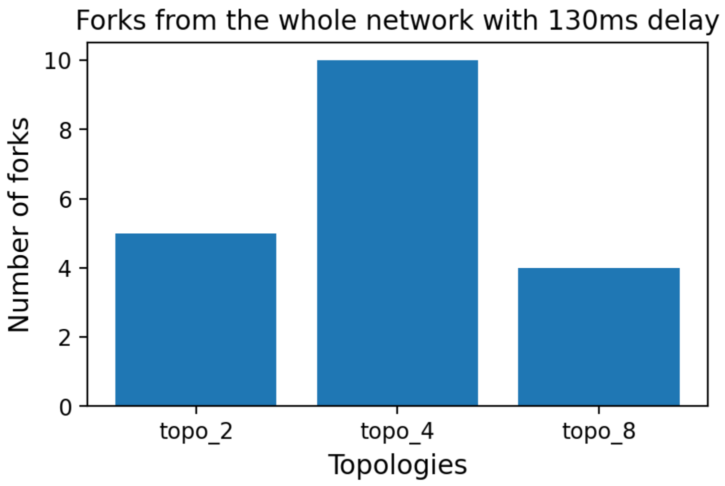


Figure 6.2: Number of forks from the whole network 130 MS

6.1.3 Results from 200 MS

Figure 6.3 shows the number of forks the different topologies creates when the network delay is 200 MS. Topology 4 creates nine forked blocks, while topology 8 creates three forked blocks. Topology 2 creates five forks.

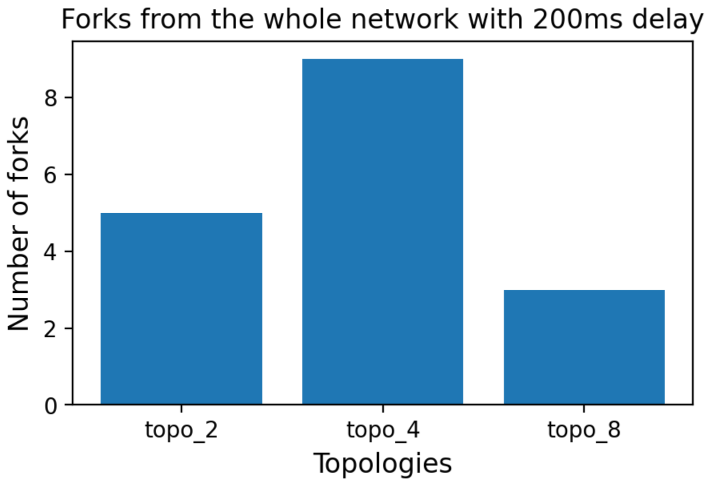


Figure 6.3: Number of forks from the whole network 200 MS

6.1.4 Discussion

From the results above, we see that topology 4 start out with creating four forked blocks when the delay is 11 MS. Then, this increases to 10 when the delay is 130 MS, and when the delay goes up to 200 MS topology 4 created nine forked blocks. There is a big jump in creating forks, more than 100%, when the delay goes from 11 MS to 130 or 200 MS. The higher delay seems to affect the creation of forks in this topology. However, when the delay goes from 130 to 200 ms, topologies 4 and 8 create one less forked block.

Topology 8 creates the least amount of forked blocks in each setup. This might be because a block is propagated throughout the whole network the quickest when having eight peers. With topology eight as well, the increase of delay increases the number of forks created. Going from 11 MS delay to 130 MS seems to have a significant impact, with the number of forks doubling. However, again like with topology 4, when the delay increases from 130 to 200 MS the number of forks goes down by one. For topology 2 the number of forks created stays steady at five for each experiment.

6.2 Number of forks from node 1 perspective

Usually, an entity only controls a small percentage of the nodes in the Bitcoin network. Therefore, showing the number of forks that node 1 can see is a realistic perspective. The x-axis shows the different topologies, while the y-axis shows the number of forks. Each figure has a different delay.

6.2.1 Results from 11 MS

From figure 6.4 we see that node 1 observes two forks when the topology is 2 or 4 and when the topology is 8 node 1 observes zero forked blocks. The delay in the network is 11 MS

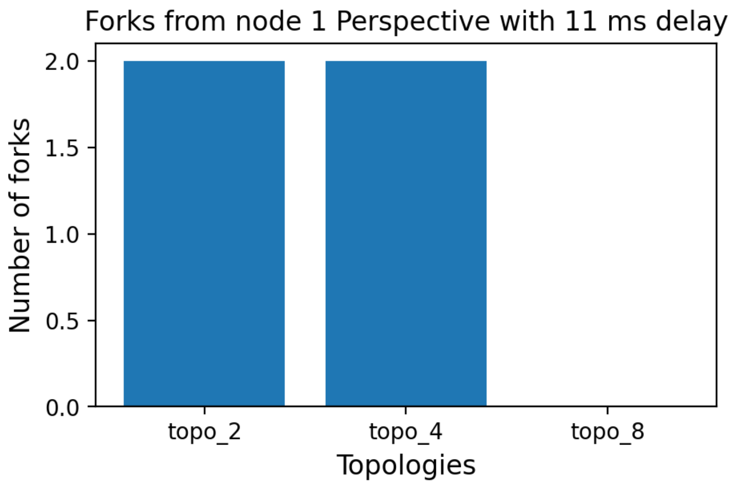


Figure 6.4: Number of forks from the perspective of node 1 with 11 MS delay

6.2.2 Results from 200 MS

Figure 6.5 shows the number of forks node 1 observed when the delay is 200 MS. Topology 8 observed 0 forks. While topology 4 and 2 observed two forks.

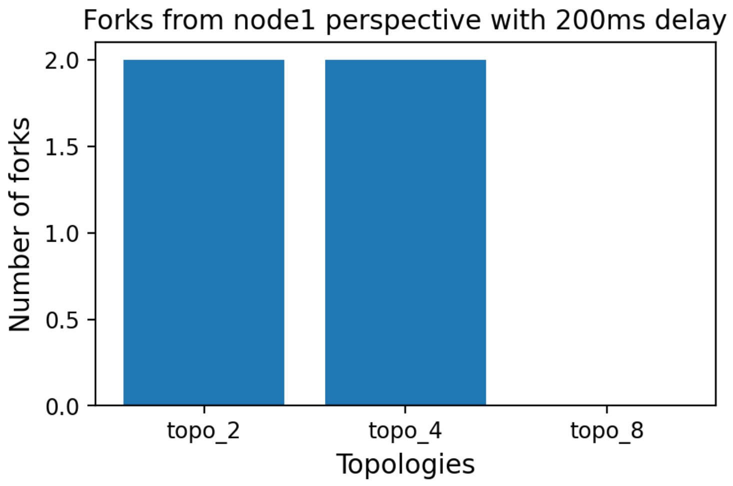


Figure 6.5: Number of forks from node 1 perspective 200 MS

6.2.3 Discussion

When the topology is 8, node 1 sees zero forks no matter what the delay is. This could be because in topology 8 blocks are propagated throughout the network the fastest, and therefore, the forked block does not reach node 1. When the topology is 8 each node has 8 peers. Therefore, when a block is generated in three steps, every node has received that block. This is a small amount of time for node 1 to receive a forked block. Node 1 does not create transactions and blocks. Hence, the forked blocks can not be generated by node 1. Because this could affect the number of forks node 1 sees from experiment to experiment.

When the topology is 2 or 4, node 1 observed two forked blocks when the delay is 11 MS and 200 MS. Even though the number of forks in the whole network increased for topology 4, node 1 still observed two of these forks. The number of forks in the whole network increased by 200% from 3 to 9, and node 1 still observed the same number in both these cases. For topology 2 node 1 observed 2 forks in both scenarios, and the network generated the same number of forks in this situation.

6.3 Transactions waiting/confirmation time from the whole network

We gathered the transactions in a forked block for each experiment configuration and analyzed how it affected their confirmation/waiting time. The result is plotted as Cumulative Distribution Function (CDF). A CDF plot, in this case, shows how many transactions that are above a specific confirmation time, as a percentage. The x-axis shows the confirmation time in seconds. The y-axis shows the percentage values. Each topology has its own graph, and the topology refers to the number of peers each node has in the network.

6.3.1 Results from 11 MS

Figure 6.6 compares the confirmation time with different topologies when the delay is 11 MS. A transaction is said to be confirmed when it is six blocks deep. This should, in theory, be after 3600 seconds. We can see from figure 6.6 that topology 4 has every transaction with less than 3500 second confirmation time. While topology 2 and 8 seem to have a more similar confirmation time, with 50 % being above 3500 seconds. The three topologies seem to have the lowest confirmation time at 1500 seconds. Topology 8 has a higher max confirmation time for a small percentage of transactions.

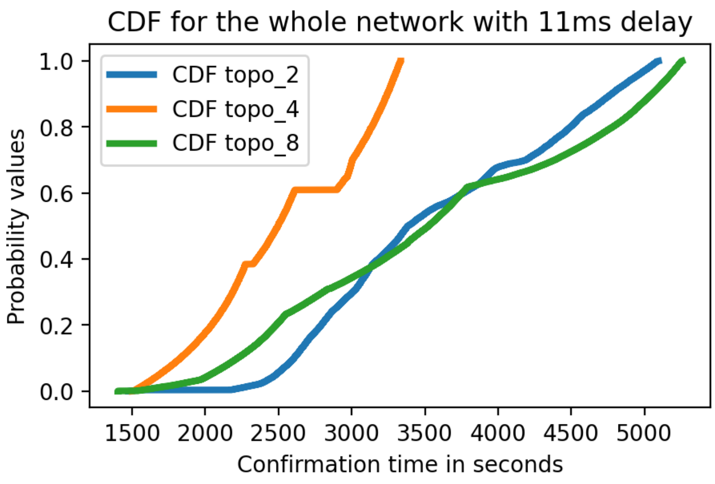


Figure 6.6: Transaction confirmation time in seconds 11 MS delay

6.3.2 Results from 130 MS

Figure 6.6 compares the confirmation time with different topologies when the delay is 130 MS. Topology 8 has the lowest max confirmation time, with all the transactions having less than approximately 5500 seconds in confirmation time. Meanwhile, topology 2 has transactions with up to 8000 seconds in confirmation time, and topology 4 has transactions that have to wait over 8000 seconds to be confirmed. For topology 4, 50% is above 4500 seconds, and for topology 8, 50% of the transactions have a confirmation time above 4750 seconds, and for topology 2, 50% are above 5000 seconds. For all the topologies, 50% of the transactions are above 4500 seconds.

This is 900 seconds more than the expected of 3600 seconds. Indeed 50% of the transactions are delayed by the time it takes to generate a block and a half. Therefore, it seems that delay impacts the transactions included in a forked block.

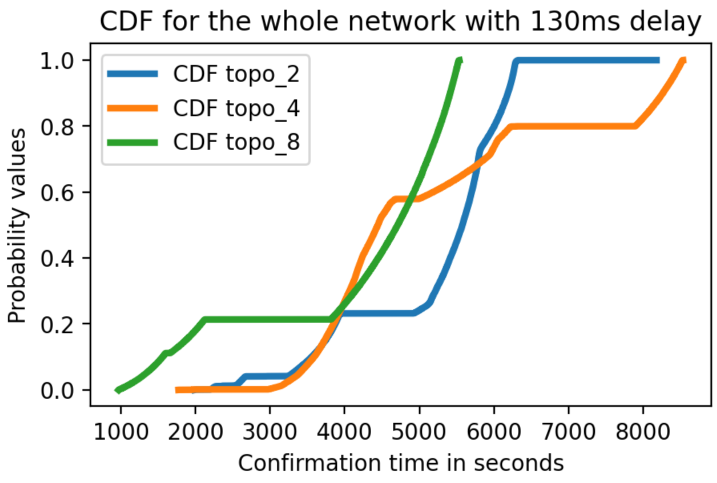


Figure 6.7: Transaction confirmation time in seconds 130 MS delay

6.3.3 Results from 200 MS

Figure 6.8 shows the transactionS in a forked block when the delay is 200 MS for the different topologies. The lowest confirmation time is 2500 seconds for topology 4, while topology 8 has 3500 seconds as the lowest confirmation time. 50% of the transactionS in topology 4 has below 3750 seconds confirmation time. While topology 8 has 50% below 5500 seconds. Topology 4 has 90% of the transactionS above 5000 seconds. Meanwhile, topology 8 has 60% of transactions above 5000 seconds in confirmation time.

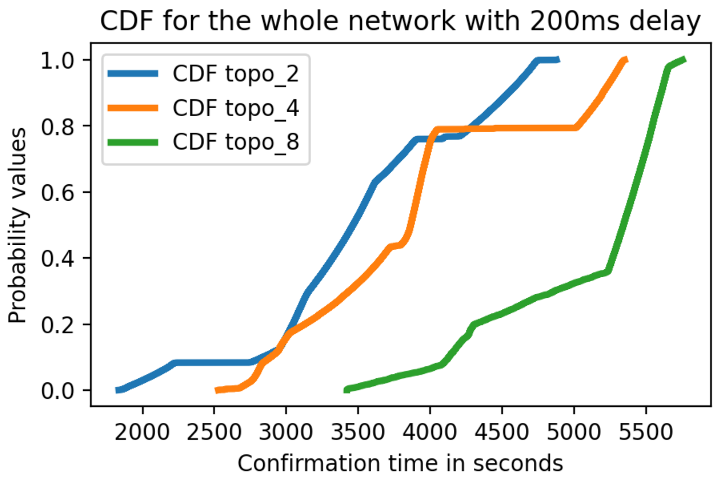


Figure 6.8: Transaction confirmation time in seconds 200 MS delay

6.3.4 Discussion

First we look at topology 2. When the delay is 11 MS, the lowest confirmation time is 1500 seconds. Meanwhile, when the delay is 130 MS, the lowest confirmation time is 2500 seconds. The difference is almost two blocks being generated. In topology 2, when the delay is 11 MS 50% of the transactions are below 3500 seconds. Meanwhile, in 130 MS delay, 50% of the transactions are below 5000 seconds, which is an increase of 1500 seconds. This is almost the average time for three blocks to be generated.

Next, if we observe topology 4 in 11 MS delay, every transaction is lower than the average confirmation time of 3600 seconds. The lowest confirmation time is around 1500 seconds. When the delay is 130 MS, the lowest confirmation time is 1750 seconds, and the max confirmation time is above 8000 seconds, which is double than when the delay was 11 MS. For topology 4, when the delay is 130 MS 50% of the transactions is above 4500 seconds in confirmation time. When the delay goes up to 200 MS, the lowest confirmation time goes up to 2500 seconds, and 50% of the transactions exceeds 4000 seconds. The max confirmation time is 5500 seconds, which is 2500 seconds less than when the delay was 130 MS.

Then if we take a look at topology 8 in 11 MS delay, we observe that the lowest confirmation time is less than 1500 seconds. When the delay is 130 MS, the lowest confirmation time is around 1000 seconds, and when the delay is 200 MS, the lowest confirmation time is almost 3500 seconds. 50% of the transactions are below 3500 seconds when the delay is 11 MS, while 50% are below 4500 seconds when the delay is 130 MS, and when the delay is 200 MS, 50% transactions are below 5500 seconds.

The max confirmation time when the delay was 11 MS was approximately 5500 seconds, and when the delay was 130 MS, the max confirmation time was 5500 seconds. For 200 MS, the max confirmation time was close to 6000 seconds.

From the results, it seems that the delay affects the confirmation time. The lowest confirmation time went up in every topology when the delay increased except for one time when in topology 8 it went down from 11 MS to 130 MS. Other than that, the lowest confirmation time increased when the delay increased. From these figures, we could see that the average went up as the delay went up for the most part. 50% of the transactions had a higher confirmation time for almost every topology when the delay increased.

6.4 Transactions confirmation/waiting time from node 1 perspective

In section 6.2 we showed the number of forks node 1 observed. We then took the transactions in the forked blocks node 1 observed and analyzed how these transactions waiting/confirmation time is affected. This is very similar to the CDF functions in the previous sections. However, the previous sections were transactions from forked blocks observed in the whole network. Meanwhile, this section presents the transactions that are in forked blocks that node 1 observed.

6.4.1 Results from 11 MS

Figure 6.9 compares the confirmation time for different topologies. Since node 1 just observed forked blocks in topology 2 and 4, confirmation time for transactions in topology 8 is not shown. As we can see from the graph, topologies 2 and 4 have less than the average of 3600 seconds. Both topologies have roughly the same maximum confirmation time, around 3500 seconds. Meanwhile, topology 2 has a steady increase, with the lowest confirmation time for a transaction being 2250 seconds and 50% of transactions being above 3000 seconds. Meanwhile, topology 4 has a confirmation time as low as 1500 seconds, and 50% of the transactions are above 2250 seconds. However, the confirmation time for the transactions jumps with 750 seconds at the 50% mark, where the transactions go from 2250 seconds confirmation time to almost 3000 seconds confirmation time.

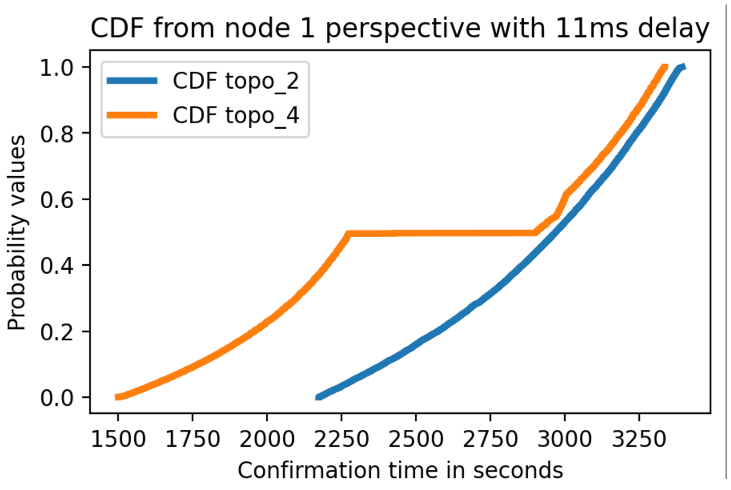


Figure 6.9: Transaction confirmation time in seconds 11 MS delay

6.4.2 Results from 200 MS

Figure 6.10 compares the confirmation time for transactions in topology 4 and 2. It seems to start at around 3000 seconds. However, topology 2 steadily increases from 3000 seconds to 3750 seconds, where 50% of the transactions are between this time frame. Meanwhile, topology 4's confirmation time for transactions starts at 3000 seconds, but there is a minuscule transaction within this time frame. 40% of the transactions in topology 4 is within the time frame of 3800 seconds and 4100 seconds. Then there is a big jump where the other 60% of the transactions is between 5000 seconds and 5500 seconds. For topology 2 the first 50% of transactions was between 3000 seconds and 3750, and the remaining 50% of the transactions was between 4250 seconds and 4800 seconds.

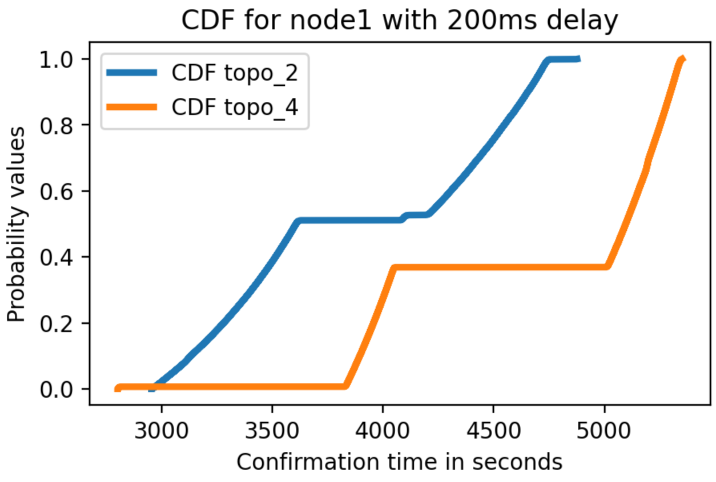


Figure 6.10: Transaction confirmation time in seconds 200 MS delay

6.4.3 Discussion

When the delay was 11 MS, all the transactions was below 3600 seconds which is the average confirmation time in Bitcoin. However, when the delay was 200 MS for topology 2, 50% of the transactions was below 3750, while the other 50% of the transactions had a confirmation time between 4250 and 4800 seconds. Meanwhile, for topology 4, 40% of transactions were between 3000 seconds and 4000 seconds. The other 60% of transactions were between 5000 and 5500 seconds.

The reason that there seems to be more delay in the confirmation time when the delay in the network increases. It might be because it takes longer to propagate the transactions the network created to every node. Therefore, the Mempool of the nodes look different. So when a block is generated, some of the transactions might be part of the forked block but not in the other block that became part of the main chain.

6.5 Independent runs

After finishing the nine experiments, we had some time left, so we did independent runs. The independent runs were to show that the experiment can be done multiple times and was to gather more information about one topology and delay combination. There were, in total, seven independent runs, each lasting for 1-2 days. These seven experiments had a 200 MS delay on the network, and the topology was 2, so each node had two peers.

The analyses done for these independent runs are similar to those done in the previous sections. However, the difference is that in this section, the averages of the seven runs combined are shown instead of the results for each run. We first show the average number of forks observed in the whole network. Afterwards, the average confirmation time between each run is shown.

6.5.1 Results for the independent runs

Figure 6.11 shows the average number of forks seen in the seven runs, and the error bars indicate a 95% confidence interval. The average number of forks seen is 2.57, and the 95% confidence interval is 0.86.

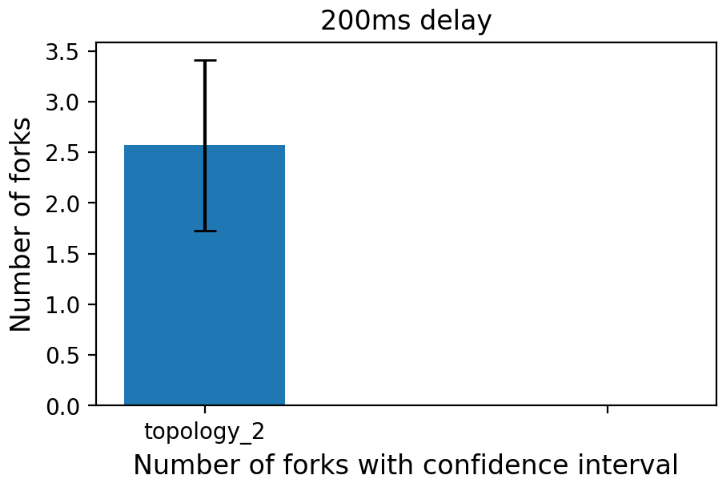


Figure 6.11: Number of forks observed in the whole network

Figure 6.12 shows the seven independent runs average confirmation time in seconds. The average confirmation time for the seven runs where 3264.105, and the 95% confidence interval was 1156.058

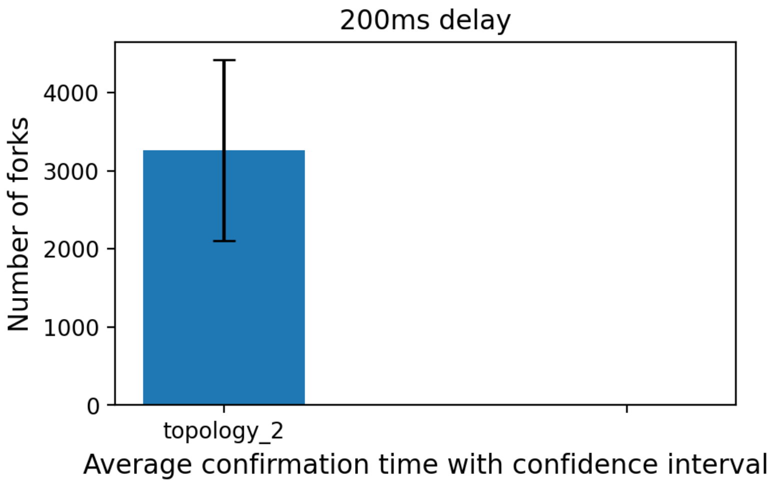


Figure 6.12: Average confirmation time

Chapter 7
Furture Work

This thesis has presented a Bitcoin Core testbed where delay and topology have been controlled to see their effects on the creation of forks. In addition, we investigated how the confirmation/waiting time is affected for transactions included in a forked block. This chapter discusses how the testbed can be used. In addition, what more can be added to the research done in this thesis.

7.1 Future work

In this thesis, each topology and delay pairing was ran once, which does not give the required confidence interval. Therefore, running a similar configuration multiple times and observing the results is something one could do. We did some independent runs at the end of the thesis, but each run lasted for 1-2 days. Each run of the experiment should ideally last for longer. The most extended amount we ran an experiment for was up to 6 days, and this is something that also could be increased to a more extended period.

There are multiple possibilities of different network parameters other than delay and topology that could be used. With the testbed we created, we only controlled the delay and topology. However, adding different network parameters to the setup might be very interesting, and seeing the effect, this has on temporary forks.

Chapter 8 Conclusion

The conclusion of this master thesis comes with this chapter. First, the research questions are going to be answered. Afterwards, our contribution to the research is reflected upon.

8.1 How did network parameters such as topology and delay affect the number of temporary forks?

From the results of the nine experiments we did. We observed that the delay did seem to affect the number of forks. When the delay was 11 MS, the number of forks created was at its minimum, and when the delay increased to 130 MS and 200 MS, the number of forks increased. For the different delays, when each node had 8 peers, the number of forks created was at its lowest, and node 1 also observed the least number of forks when it had eight peers observing 0 forks.

8.2 To what extent will the temporary fork affects transactions waiting/confirmation time?

As we explained in the background section, not all transactions that are in a forked blocked are delayed, but some of these transactions might be. How much a transaction was delayed varied, but when the delay was 11 MS, the maximum confirmation time was at its lowest and the minimum confirmation time was at its lowest. For 130 MS and 200 MS it is a bit harder to differentiate.

8.3 Contributions

For this thesis, we have created a unique Bitcoin testbed that allows us to control the delay and topology. We have done nine experiments with different configurations and gathered and analyzed these results. Additionally, we have done seven independent runs, in addition to the nine, to show that the experiments are reproducible and that this Bitcoin testbed can produce forks regularly.

Refrences

[Bitcoin Core]	What Is Bitcoin Core. [Online]. Available: https://river.com/learn/what-is-bitcoin-core/ (last visited: Feb. 20, 2022).
[BitcoinConfirmation]	BitcoinConfirmation. [Online]. Available: https://en.bitcoin.it/wiki/Confirmation (last visited: Apr. 27, 2022).
[BitcoinDeveloper]	BitcoinDeveloper. [Online]. Available: https://btcinformation.org/en/developer-reference#target-nbits (last visited: Apr. 29, 2022).
[BitcoinDigital]	BitcoinDigital. [Online]. Available: https://www.newscientist.com/definition/bitcoin/ (last visited: Apr. 27, 2022).
[BitcoinHeader]	BitocinHeader. [Online]. Available: https://developer.bitcoin.org/reference/block_chain.html (last visited: Mar. 2, 2022).
[BKP14]	A. Biryukov, D. Khovratovich, and I. Pustogarov, «Deanonymisation of clients in bitcoin p2p network», 2014.
[BlockRelay]	BlockRelay. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki (last visited: Apr. 19, 2022).
[CCY+20]	C. Chen, X. Chen, *et al.*, «Impact of temporary fork on the evolution of mining pools in blockchain networks: An evolutionary game analysis», Nov. 2020.
[DW13]	C. Decker and R. Wattenhofer, «Information propagation in the bitcoin network», 2013.
[ECP21]	J.-P. Eisenbarth, T. Cholez, and O. Perrin, «A comprehensive study of the bitcoin p2p network», 2021.
[EG13]	I. Eyal and E. Gun Sirer, «Majority is not enough: Bitcoin mining is vulnerable», 2013.
[Empirica]	Empirica. [Online]. Available: https://www.questionpro.com/blog/empirical-research/ (last visited: Mar. 27, 2022).
[getBlock]	getBlock. [Online]. Available: https://chainquery.com/bitcoin-cli/getblock (last visited: Apr. 15, 2022).
[getchaintips]	getchaintips. [Online]. Available: https://chainquery.com/bitcoin-cli/getchaintips (last visited: Apr. 15, 2022).

[GHJ21]	B. Gebraselase, B. Helvik, and Y. Jiang, «An analysis of transaction handling in bitcoin», 2021.
[GWR+16]	A. Gervais, K. Wüs, *et al.*, «On the security and performance of proof of work blockchains», Oct. 2016.
[Hag20]	S. Haga, «Towards 5g network slice isolation with wireguard and open source mano», 2020.
[HardAndSoftForks]	HardAndSoftForks. [Online]. Available: https://www.bitcoinakademiet.no/hard-fork-vs-soft-fork/ (last visited: Mar. 12, 2022).
[HardForks]	HardForks. [Online]. Available: https://www.investopedia.com/terms/h/hard-fork.asp (last visited: Mar. 12, 2022).
[Mempool]	Mempool. [Online]. Available: https://academy.binance.com/en/glossary/mempool (last visited: Feb. 24, 2022).
[Nak09]	S. Nakamoto, «Bitcoin a peer-to-peer electronic cash system», 2009.
[NBF+16]	A. Narayanan, J. Bonneau, *et al.*, «Bitcoin and cryptocurrency technologies», 2016.
[NH20]	T. Neudecker and H. Hartenstein, «: An empirical analysis of blockchain forks in bitcoin», Nov. 2020.
[P2PNetwork]	P2PNetwork. [Online]. Available: https://developer.bitcoin.org/devguide/p2p_network.html (last visited: Apr. 29, 2022).
[regTest]	regTest. [Online]. Available: https://developer.bitcoin.org/examples/testing.html (last visited: May 14, 2022).
[softForkTerms]	softForkTerms. [Online]. Available: https://river.com/learn/terms/p/p2sh/ (last visited: May 13, 2022).
[VJR18]	D. Vujičić, D. Jagodić, and S. Ranđić, «Blockchain technology, bitcoin, and ethereum: A brief overview», 2018.
[ZJMA16]	V. Zakhary, M. Javad Amiri, *et al.*, «Towards global asset management in blockchain systems», 2016.