Adaptive Information Dissemination in the Bitcoin Network

João Esteves Marçal

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors:
Prof. Luís Eduardo Teixeira Rodrigues
Prof. Miguel Ângelo Marques de Matos

Examination Committee

Chairperson:	Prof. Daniel Jorge Viegas Gonçalves
Supervisor:	Prof. Miguel Ângelo Marques de Matos
Member of the Committee:	Prof. João Carlos Antunes Leitão

October 2018

Acknowledgements

I would like to thank my advisors Professor Miguel Matos and Professor Luís Rodrigues, for their guidance throughout this work. I would also like to thank my friends for all the leisure moments we shared during this year. Last, but definitely not least, I would like to thank my parents and my sisters for the endless support.

This work was partially supported by Fundo Europeu de Desenvolvimento Regional (FEDER) through Programa Operacional Regional de Lisboa and by Fundação para a Ciência e Tecnologia (FCT) through projects with reference UID/CEC/50021/2013 and LISBOA-01-0145-FEDER- 031456.

Resumo

Os livros padrão distribuídos, que são um dos blocos fundamentais de muitos dos sistemas de cripto-moeda recentes, têm vindo a ganhar um interesse crescente, devido ao largo número de aplicações onde podem ser usados. Nos sistemas de cripto-moeda, os livros padrã distribuídos são usados para registar as transacções que ocorrem no sistema. Este registo assume tipicamente a forma de uma lista ligada de blocos, em que cada bloco regista um conjunto das transações. Uma vez que a cadeia de blocos é construída e mantida de forma descentralizada, isto obriga à troca de informação frequente entre os diversos nós do sistema. Esta comunicação pode consumir uma quantidade de largura de banda significativa. Infelizmente, tentativas ingénuas de reduzir a quantidade de informação trocada podem ter um impacto negativo no desempenho e correção do processo de propagação em função de alterações ao estado da rede.

Nesta dissertação propomos novos mecanismos para suportar a propagação de informação no livro padrão do Bitcoin. Visto que a rede é dinâmica, propomos também um conjunto de mecanismos adaptativos que ajustam a propagação de informação em resposta a alterações no estado da rede. Apresentamos uma avaliação experimental dos mecanismos propostos que mostra que é possível reduzir a largura de banda consumida em cerca de 10.2% e o número de mensagens trocadas em cerca de 41.5%, sem que isto tenha um impacto negativo no número de transações confirmadas.

Abstract

Distributed ledgers have received significant attention as a building block for cryptocurrencies and have proven to be also relevant in several other fields. In cryptocurrencies, this abstraction is usually implemented by grouping transactions in blocks that are then linked together to form a blockchain. Nodes need to exchange information to maintain the status of the chain which consumes significant network resources. Unfortunately, naively reducing the number of messages exchanged can have a negative impact in performance and correctness, as some transactions might not be included in the chain.

In this dissertation, we study the mechanisms of information dissemination used in Bitcoin and propose a set of adaptive mechanisms that not only lowers network resource usage but can also adapt to changes in the network. Our experimental evaluation shows that is possible to lower the bandwidth consumed by 10.2% and the number of exchanged messages in 41.5%, without any negative impact in the number of transactions committed.

Palavras Chave
Keywords

Palavras Chave

Livro Razão

Criptomoeda

Disseminação de informação

Adaptablidade

Utilização de recursos

Keywords

Ledger

Cryptocurrency

Information dissemination

Adaptation

Resource Usage

Contents

List of Figures

Acronyms

AS Autonomous System

BCBPT Bitcoin Clustering Based Ping Time

BGP Border Gateway Protocol

BIP Bitcoin improvement protocol

DoS Denial of Service

ip Initial push

LCM Least Common Multiple

PoW Proof of Work

PSS Peer Sampling System

P2P Peer-to-Peer

SPS Secure Peer Sampling

1 Introduction

Cryptocurrencies, of which Bitcoin is the most popular, gained a lot of exposure in recent years. Cryptocurrencies are digital assets used as a medium of exchange between users, uses strong cryptography to secure financial transactions, control the creation of additional units, and verify the transfer of assets. With their popularisation came an increase in average daily users and as a consequence of this increase, there has also been an increase in the computing and network resources spent by the systems that maintain the cryptocurrencies. For instance, at the time of this writing, Bitcoin (the most widely used cryptocurrency) has peaked at a trading value of more than 20K USD and the Bitcoin network processed more than 250K transactions per day.[1] Furthermore cryptocurrencies also have lots of advantages to the users:

- **Anti fraud** - cryptocurrencies are digital and cannot be counterfeited or reversed arbitrarily by the sender, as it happens with some other technologies.
- **Identity Theft** - traditional payment methods like credit card work on a ”pull” basis where the store indicates the payment and pulls the designated amount from the account of the user. Whereas cryptocurrencies work on a “push” mechanism that allows the user to send exactly what it wants to the recipient with no further information.
- **Lower Fees** the fees that are imposed on transactions through cryptocurrencies are much lower than the ones imposed on more traditional methods of payment.

Moreover, the Blockchain, the main technology behind cryptocurrencies has emerged as useful for a myriad of other applications, from birth, wedding, and death certificates to the monetization of music.[2]

Cryptocurrencies usually work as follows, Users exchanging digital assets between them through transactions. Transactions are then grouped together forming a block. Next blocks are linked together forming a chain, hence the name Blockchain. By linking blocks together we are establishing a chronological order over the blocks and the transactions inside these blocks, forming a ledger.

This ledger is maintained in a distributed fashion by all the nodes in the network and any node that is willing to spend their resources is then to add blocks to the distributed ledger. These nodes are known as miners.

The ledger is maintained by disseminating in the network information about transactions, blocks and other metadata. As the number of users and issued transaction grows, the process of disseminating information consumes a lot of resources. For instance, if we consider the cryptocurrency Bitcoin, at

[1] Taken from `https://blockchain.info/charts`.
[2] Taken from `https://blockgeeks.com/guides/blockchain-applications/`

the time of writing there is an average of 200K transactions processed per day. Since each transaction usually imposes an exchange of at least 3 messages then this means that for two nodes to exchange the daily amount of transactions they have to exchange 600K messages. Furthermore, this number grows even higher if we multiply it by the 10K nodes that are in the network and by the fact that each node sends a transaction to all his neighbours. Because of these reasons the process of dissemination generates a lot of duplicated messages. However, a naive reduction in the number of messages exchanged can result in some nodes not receiving, for instance, all transactions, impacting the performance and correctness of the system.

This thesis addresses the problem of the large amount of resources spent on disseminating information in cryptocurrencies, namely Bitcoin. In particular, we propose a new protocol for the dissemination of transactions.

1.1 Motivation

Our objective is to lower the amount of resources spent in the dissemination process. Since there are multiple disadvantages with nodes having to process all the extra messages that receive from their neighbours, from wasted CPU cycles to unnecessary power consumption.

However, we can not simply stop disseminating information to a fraction of the nodes without any criteria. This would not only leave the cryptocurrency vulnerable to attacks, like double-spending, but it would also lower its performance. For instance, a transaction could take a very long time to reach a miner because it was sent to nodes that had few connections to miners.

Hence we have to take advantage of already existing characteristics in these systems and study them properly in order to come up with a solution that not only lowers the resources used but also does not put at stake the current resilience of the system. In fact, in the *Bitcoin* network, only a fraction of the network (currently around 10%) spends resources generating new blocks. Hence, our strategy consists of skewing the dissemination algorithm such that transactions reach miners faster.

1.2 Contributions

This thesis studies in detail how the latest membership and information dissemination protocols of Bitcoin work. Based on this study, we proposed an improvement over the dissemination protocol that reduces the amount of information that needs to be exchanged without compromising correctness. In detail, we provide:

- A principled analysis of the limitations of the information dissemination protocol the Bitcoin network;
- A study on how the protocol has been implemented in the most recent Bitcoin releases;
- An extensible architecture that eliminates or mitigates the identified limitations and finally;
- A detailed evaluation of the proposed solution.

1.3 Results

This thesis produced the following results:

- A specification for a new dissemination protocol;
- An implementation of the proposed protocol;
- An experimental evaluation of the resulting system.
- An article accepted in INFORUM 2018 - Técnicas para Reduzir a Carga na Rede no Livro-Razão da *Bitcoin*, João Marçal, Miguel Matos, Luís Rodrigues
- Our protocol is able to save 41.5% of the messages sent and 10.2% of the information sent by the traditional Bitcoin protocol.

1.4 Structure of the Document

The remaining of this document is organised as follows. Chapter 2 provides an introduction to Bitcoin and its systems for information dissemination and membership management. It also covers some attacks the cryptocurrency and problems related with Peer-to-Peer. Chapter 3 describes the system model and our protocol. Chapter 4 presents the experimental evaluation results. Finally, Chapter 5 concludes the document by summarising its main points and discussing future work.

2 Related work

This chapter gives a general overview on how Bitcoin works as well as addressing previous work on both Bitcoin and similar Peer-to-Peer systems. Section 2.1 describes the basics of Bitcoin and Blockchain technologies, along with Bitcoin mechanisms for information dissemination. Afterwards, Section 2.2 surveys previous work developed on Bitcoin and some of its vulnerabilities.

2.1 Bitcoin and Blockchain

This section provides an overview of the operation of the Bitcoin network. Bitcoin was created in 2008 by Satoshi Nakamoto with the aim of creating an infrastructure for allowing people to make transactions without depending on a centralized third party while, at the same time, preserving some anonymity (Nakamoto 2008). For this purpose, Bitcoin creates a cryptographic currency that can be exchanged among parties. In order to exchange bitcoins, the two parties must own public-/private-keypairs and execute a protocol where the transaction is signed in such a way that serves as a cryptographic proof that the payer paid to the payee (Decker and Wattenhofer 2013). A key idea of Bitcoin is that all transactions that involve the exchange of bitcoins between two parties are registered in a serial log that cannot be tampered. This log is built by linking multiple blocks, where each block contains a set of transactions, in an infinite chain known as the *blockchain*. Bitcoin is completely decentralised, and the blockchain is maintained cooperatively by multiple nodes.

In the rest of this section, we provide a more detailed description of several modules of Bitcoin. We start by describing how transactions are represented in Section 2.1.1. Next, we describe how multiple transactions are registered in blocks, that contain the most recent transactions (Section 2.1.2). In Section 2.1.3 we discuss how these blocks are linked together to create the blockchain. Then, in Section 2.1.4 we present how the Peer-to-Peer network is built and maintained in Bitcoin. Finally, in Section 2.1.5, we describe how information is disseminated throughout the network.

2.1.1 Transactions

A transaction represents the exchange of currency between accounts. It is composed of inputs, outputs, a transaction ID and other fields not relevant for this work. The inputs are the accounts of the payers, the outputs are the accounts of the payees' and the transaction ID is the hash of the serialised transaction. The transaction ID is also what is used to identify the transaction (Decker and Wattenhofer 2013).

For an account to be able to spend bitcoins, the nodes responsible for accepting transactions have to

know the balance of that account. Nodes know the balance of an account because they keep track of all unspent transactions of that specific account (Decker and Wattenhofer 2013). Unspent transactions are transactions where an account received a payment of bitcoins. This means that the account appeared in the array of outputs of those transactions. These transactions are proof that an account has bitcoins; they work like a receipt.

In order for a transaction to be valid, the payers have to sign the transaction. This means that each owner transfers the amount to the payee by digitally signing a hash of the previous transaction and the public key of the payee as seen in Figure. 2.1 (Nakamoto 2008). The previous transaction is the transaction where the current payer received the bitcoins that he is using in the current transaction.

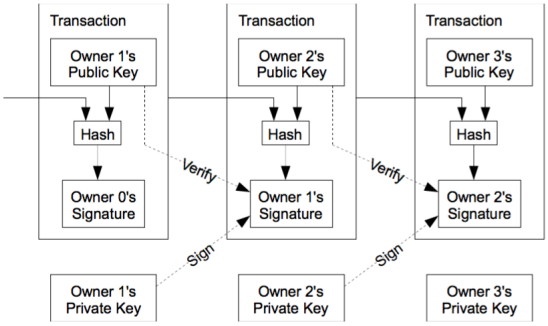

Figure 2.1: Signing mechanism. Original from (Nakamoto 2008)

Furthermore, transactions have to fulfil the following criteria regarding outputs they claim and create in order to be valid (Decker and Wattenhofer 2013):

- An output may be claimed at most once;
- Outputs are created solely as a result of a transaction;
- The sum of the values of the claimed outputs has to be greater or equal than the sum of the values of the newly allocated outputs. Claimed outputs are bitcoins the payer is trying to spend and allocated outputs are the amount of bitcoins the payee accounts are going to be able to spend. For instance, if user *A* and *B* want to buy a product to *C* that costs 3 bitcoins where each one contributes with 1.5 bitcoins in the claimed outputs will appear two entries each of 1.5 bitcoins and in the newly allocated resources have to be exactly 3 bitcoins, or less, as some amount can be claimed by the miners as a tax for including the transaction in the block.

The first criteria exist so that the user cannot double spend bitcoins. The second ensures that unspent bitcoins need to be connected to a transaction, avoiding forging of unspent bitcoins. The third guarantees that bitcoins can only be transferred and not created.

In order for payees to know that previous owners did not sign any earlier transactions, transactions are broadcasted through the Bitcoin network (Nakamoto 2008). This feature is required to ensure the payee that the payer did not already spend the bitcoins he is using to pay him. However, this feature introduces inconsistencies in the system, as transactions reach different nodes at different times:

- A node could receive a transaction that transfers bitcoins from owner *B* to owner *C*, however that node has yet to receive the transaction that transferred the bitcoins from owner *A* to owner *B*. As a result the transaction might not being accepted or it might take a longer time to be accepted;
- A node could also receive two transactions from the same owner *A* where he tries to transfer the same bitcoins to two different payees *B* and *C*. This is a case of double-spending.

As there is no guarantee that different nodes receive conflicting transactions in the same order, those nodes will disagree on those transactions and any transactions built on top of them by claiming their outputs (Decker and Wattenhofer 2013). This would have been a problem because they would not agree upon a common record. For instance, if a user *Joe* sent his transaction to node *A*, *A* would accept it because in its record *Joe* had those bitcoins to spend. But if *Joe* had sent his transaction to node *B*, *B* might not have accepted it because in its record *Joe* had never been the owner of the bitcoins he is trying to transfer. We will see how this problem is solved by Bitcoin in the next section.

2.1.2 Blocks

Since different nodes can process and even commit transactions in a different order they need to have a way to reach a consensus on the set of transactions that are considered valid. The role of the blocks is precisely to allow nodes to agree upon a set of transactions.

Each block is composed of a set of transactions, a nonce, a pointer to its parent block and other fields not relevant for this explanation. Each block is also associated with a block header that summarises the information in that block.

In order for a block to be accepted by other nodes it has to present a Proof of Work (PoW). PoW consists in finding a byte string, called nonce, that hashed with the block header results in a hash with a given number of zeros at the beginning. That number of zeros at the beginning is also called *target* (Decker and Wattenhofer 2013). As cryptographic hash functions are only one-way, discovering such nonce can only be done by trial and error. Furthermore, the difficulty of the *target* is also adjusted as follows. The Bitcoin network measures how much time it took to create the last 2016 blocks in a blockchain. If it took significantly more than 2 weeks, the PoW difficulty is reduced, meaning that there will be fewer zeros at the beginning of the next *target*. If it took significantly less than 2 weeks, the difficulty is increased.

The PoW is necessary because it adds a real-world cost to produce a block. So with the requirement of a block having to present a PoW, it becomes infeasible for people to modify the history of the system and present it as the truth to anybody else as we will see in Section 2.2.1.2.

To incentivise miners for having spent resources on finding the PoW, each time a new block is created a new Bitcoin is generated (Decker and Wattenhofer 2013). The reward transaction is only valid if it appears in a block. This transaction is also the only transaction that is the exception to the third criteria seen in the Section 2.1.1 which states that the sum of the inputs has to be greater or equal to the sum of the outputs (Decker and Wattenhofer 2013).

Today as the number of transactions increases on a daily basis and given the limited space each block has for transactions (1MB), miners also profit through fees imposed on transactions. Most miners

choose which transactions to include in their blocks based on how profitable they expect those transactions to be. If there are two transactions of equal byte size but only one of them fits in a block, then the miner will choose whichever transaction has the higher transaction fee.

Due to the very low probability of successful generation, it is unpredictable which miner nodes in the network will be able to generate the next block. Hence, when a node finds a new block, it broadcasts it to the other nodes. Upon receiving a new block, two things can happen:

- The node will rollback all tentatively committed transactions since the last block reception and then commit the ones on the new block (Decker and Wattenhofer 2013). Tentatively committed transactions are transactions that were valid and would have been committed if the node had been able to generate a new block.
- The node has already mined or received a new block and it will ignore the newly received block. The implications of this are further discussed in Section 2.1.3.

Regarding the tentatively committed transactions that were rolled back, if they were present in the new block they do not have to be re-applied. For the tentatively committed transactions that were not, they will be re-applied only if they are valid as conflicting transactions might have been present in the newly received block. Invalid transactions are transactions that conflict with one or more transactions that were present in the new block. If a transaction is flagged as invalid it will be discarded (Decker and Wattenhofer 2013). The node that created the invalid transaction will eventually receive the block with the valid transaction and it will have to rollback the invalid transaction.

Because of this feature, the creator of the block imposes which transactions are going to be committed and what is the order that they are committed (Decker and Wattenhofer 2013).

In the next section, we will see how blocks are linked together to create a distributed record of what happened.

2.1.3 Blockchain

As we have seen, blocks are used by nodes to agree on the order of recent transactions. Furthermore once a block is generated it is also linked with the block that preceded it. This creates a chronological order over all blocks and therefore transactions as seen in Figure. 2.2. This chain of blocks is called *blockchain* (Decker and Wattenhofer 2013). The first block in the chain is called the genesis block.

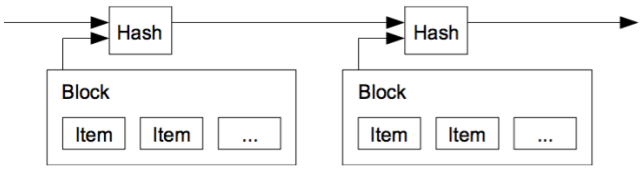

Figure 2.2: Blockchain representation. Original from (Nakamoto 2008)

The blockchain is defined as the longest path from any block to the genesis block. This definition makes the blockchain resemble a tree where the root is the genesis block and the leaves are the new

blocks. The height of a block is the distance of that block to the genesis block. The block that is the furthest away from the genesis block is called the blockchain head (Decker and Wattenhofer 2013).

As only new blocks are rewarded with new bitcoins, miners try to build on top of the blockchain head (Decker and Wattenhofer 2013). This is because if they were to build on top of previously found blocks it would require them to first reach the same height as the blockchain head and then find the new block (Nakamoto 2008). This is very hard to do because they would have to remine all the previous PoW and a new PoW while competing against the rest of the computational power present in the network. However, given the resources necessary to find only one PoW, the miners would have to have more computational power than the rest of the network when they first started the fork. Namely, for a miner to have success at doing this he would have to control at least 51% of the computational power in the network, we will further discuss this in section 2.2.1.2.

As seen in Section 2.1.2, a node can accept a new block or ignore it. Because of this, there might be multiple blocks at the same height at any point in time. This is called a *blockchain fork*. Hence, when this happens the network does not agree on which block is the blockchain head. This leads to inconsistency of the system because both block *b* and block *b'* are guaranteed to disagree on some transaction, for instance they will always disagree on the reward transaction. This will go on until one of the branches overtakes the other (Decker and Wattenhofer 2013).

When a node has as blockchain head block *b* and it receives a new block *b'* where the height of *b'* $>$*b* two things can happen depending on which branch *b* is in:

- If *b* is in the same branch as *b'*, meaning that *b* is one of the predecessors of *b'*, then the node only has to apply all transactions in the intermediate blocks incrementally, if any, and then apply the transactions in *b'*. Intermediate blocks are blocks that might have been generated after *b* but before *b'*.
- If *b* is on another branch, meaning that *b* is not a predecessor of *b'*, then the node is required to change branches. This implies that the node has to revert all transactions it has been committing until it reaches a common block ancestor. Then, it has to request all intermediate blocks apply the transactions in those blocks and finally apply the transactions on *b'* (Decker and Wattenhofer 2013).

The fork may be prolonged throughout multiple block heights *h, h+1, h+2...* where subsets of the network work on the different branches and try to find new blocks at the same time. Eventually one of the branches will surpass the others, leading to the adoption of this branch by all the nodes and therefore the end of the fork (Decker and Wattenhofer 2013). The discarded blocks are called *orphan blocks*.

As a consequence of blockchain forks, a transaction in Bitcoin is never committed permanently. Because at any point in time it could appear a branch that was not known by the nodes we were interacting with which might influence the state of our transactions. If this new branch is taller than the one where our transactions were confirmed then our transactions will be rolled back.

2.1.4 Overlay Network

Peers in the Bitcoin network are identified by their IP addresses. A node with a public IP can initiate up to eight outgoing connections with other Bitcoin nodes and accept up to 117 incoming connections. A node with a private IP only initiates eight outgoing connections. Connections are over TCP. Nodes only propagate and store public IPs (Heilman, Kendler, Zohar, and Goldberg 2015).

When a node wants to join the network there are five ways for it to connect with peers (Bitcoin.org 2008):

1. Address database - this local file contains other nodes that the node already knew about. The node will try to re-connect with those nodes. If it is the first time the node connects to the network this method doesn't work;
2. User-specified - in this method the user can specify nodes to connect to on the command line;
3. DNS seeding - this option is only used if the Address database file is empty and the user did not specify any nodes. The nodes issue DNS requests to a list of 6 DNS servers that are hardcoded in order to discover IP addresses of other peers, each DNS server can return up to 256 IP addresses;
4. Hard-coded nodes - If DNS seeding fails, the node contains a list of 1525 hard-coded IP addresses that represent Bitcoin nodes that it contacts to get addresses of other peers and then finishes the connection with them to avoid overloading those nodes;
5. From other nodes - Nodes exchange IP addresses with other nodes via the *GetAddr* and *Addr* messages which we will discuss later.

2.1.4.1 Storing Network Information

Public IPs are stored in a node's *tried* and *new* tables.

***Tried* Table** The *tried* table consists of 256 buckets, each of which can store up to 64 unique addresses for peers to whom the node has successfully established an incoming or outgoing connection. Along with each stored peer's address, the node keeps the timestamp for the most recent successful connection to this peer. Each peer's address is mapped to a bucket in the table by taking the hash of the peer's (a) IP address and (b) group, where the group defined is the /16 IPv4 prefix containing the peer's IP address. We insert the IP in a bucket by first getting a bucket using Algorithm 1 and then getting a position inside the bucket using Algorithm 2.

Thus, every IP address maps to a position in a single bucket in *tried*, and each group maps to up to eight buckets. When a node successfully connects to a peer, the peer's address is inserted into the appropriate position in the *tried* bucket. If the position already has an address then the node will briefly attempt to connect to the older address, and if connection is successful, then the older address is not evicted from the *tried* table; the new address is stored in *tried* only if the connection fails. If the peer's address is already present in the bucket, the timestamp associated with the peer's address is updated. The timestamp is also updated with the local time when an actively connected peer sends a *Version, Addr, Inv, GetData or Ping* message and more than 20 minutes elapsed since the last update.

Algorithm 1 Get bucket in *tried* table for a specific IP

```
1: function GET_TRIED_BUCKET
2:     SK = random value chosen when node is bootstrapped.
3:     IP = the peer's IP address and port number.
4:     Group = the peer's group
5:
6:     i = Hash( SK, IP ) % 8
7:     Bucket = Hash( SK, Group, i ) % 256
8:     return Bucket
```

Algorithm 2 Get position of IP inside a specific bucket

```
1: function POSITION_IN_BUCKET(Bucket)
2:     SK = random value chosen when node is bootstrapped.
3:     IP = the peer's IP address and port number.
4:     New = Either 'N' or 'K' if we are looking for a position in New or Tried bucket.
5:
6:     PosInBucket = Hash(SK, New, Bucket, IP) % 64
7:     return PosInBucket
```

***New* Table** The *new* table consists of 1024 buckets, each of which can hold up 64 addresses for peers to whom the node has not yet initiated a successful connection. A node populates the *new* table with information learned from the DNS seeders, or from *Addr* messages. Addresses in the *new* table also have an associated timestamp; addresses learned from DNS seeders are stamped with a random timestamp between 3 and 7 days old. Whereas addresses learned from *Addr* messages are stamped with their timestamp from the *Addr* message plus two hours. Every address *a* inserted in *new* belongs to (1) a *group*, defined in the description of the *tried* table, and (2) a *source group*, the group that contains the IP address of the connected peer or DNS seeder from which the node learned address *a*. We insert an IP in a bucket by first getting a bucket using Algorithm 3 and then getting a position inside the bucket using Algorithm 2.

Algorithm 3 Get bucket in *new* table for a specific IP

```
1: function GET_NEW_BUCKET
2:     SK = random value chosen when node is bootstrapped.
3:     Group = /16 containing IP to be inserted.
4:     Src_Group = /16 containing IP of peer sending IP.
5:
6:     i = Hash( SK, Src_Group, Group ) % 64
7:     Bucket = Hash( SK, Src_Group, i) % 1024
8:     return Bucket
```

Each (group, source group) pair hashes to a single *new* bucket, while each group selects up to 64 buckets in *new*. Each bucket holds up to 64 addresses like the *tried* table. If a position in a bucket is already occupied by other address then a function *isTerrible* is invoked. If the other address is terrible, in that it is (a) more than 30 days old, or (b) has had too many failed connection attempts, then the terrible address is evicted in favour of the new address; otherwise, the new address is discarded. A single address can map to multiple buckets if it is advertised by multiple peers; so if the other address is already in multiple buckets we prioritise the new address if the new address is not already in a bucket.

2.1.4.2 Propagating Network Information

Network information propagates through the Bitcoin network via DNS seeders and *Addr* messages.

DNS seeders A DNS seeder is a server that responds to DNS queries from Bitcoin nodes with a (not cryptographically-authenticated) list of IP addresses for Bitcoin nodes. The maximum possible number of IP addresses that can be returned by a single DNS query is 256. The seeder obtains these addresses by periodically crawling the Bitcoin network. The Bitcoin network has six seeders which are queried in two cases only (Heilman, Kendler, Zohar, and Goldberg 2015).

- When a new node joins the network for the first time; it tries to connect to the seeders to get a list of active IPs, and otherwise fails over to a hardcoded list of 1525 IP addresses.
- When an existing node restarts and reconnects to new peers, the seeder is queried only if 11 seconds have elapsed since the node began attempting to establish connections and the node has less than two outgoing connections.

Addr Messages *Addr* messages, containing up to 1000 IP addresses and their timestamps, are used to obtain network information from peers. If more than 1000 addresses are sent in a *Addr* message, the peer who sent the message is blacklisted. Nodes accept unsolicited *Addr* messages. An *Addr* message is only solicited by a node upon establishing an outgoing connection with a peer, the peer responds with the *Addr* containing up to 1000 addresses randomly selected from its tables. The peer sends a total of *n* randomly selected addresses from the peer's *tried* and *new* tables, where *n* is a random number between *x* and 2500, where *x* is 23% of the addresses the peer has stored (Heilman, Kendler, Zohar, and Goldberg 2015).

Nodes push *Addr* messages to peers in two cases. Once a day, a node sends its own IP address in a *Addr* message to each peer. Also, when a node receives an *Addr* message with no more than 10 addresses, it forwards the *Addr* message to two randomly-selected connected peers. Actually, if the *Addr* message contains addresses that are unroutable for the peer (e.g., a peer with IPv4 address gets an IPv6 address), it will forward the *Addr* message to one peer only. To choose these peers, the node takes the hash of each connected peer's IP address and a secret nonce associated with the day, selects the peers with the lexicographically first and second hash values.

Finally, to prevent stale *Addr* messages from being endlessly propagated, each node keeps a known list of the addresses it has sent to or learned from each of its connected peers, and never sends address on the known list to its peer. The known lists are flushed daily.

2.1.4.3 Selecting Peers

New outgoing connections are selected if a node restarts or if an outgoing connection is dropped. A Bitcoin node never deliberately drops a connection, except when a blacklisting condition is met (e.g., the peer sends *Addr* messages that are too large). A node with $\omega \in [0, 7]$ outgoing connections selects the $\omega + 1th$ connection as follows:

1. Decide whether to select from *tried* or *new* tables with a 50% probability.
2. Select a random address from the table, with a bias towards addresses with fresher timestamps:
   - (i) Choose a random non-empty bucket in the table.
   - (ii) Choose a random position in that bucket.
   - (iii) If there is an address at that position, return the address with probability:

$$p(r, \tau) = min(1, \frac{1.2^r}{1+\tau}) \tag{2.1}$$

   else, reject the address and return to step (i). The acceptance probability $p(r, \tau)$ is a function of r, the number of addresses that have been rejected so far, and τ, the difference between the address's timestamp and the current time in measured in ten minute increments.
3. Connect to the address. If connection fails, go to (1).

2.1.4.4 Feeler Connections

An outgoing connection that establish short-lived test connections to randomly selected addresses in *new*. If connection succeeds, the address is evicted from *new* and inserted into *tried*; otherwise, the address is evicted from *new*. Feeler connections clean trash out of *new* while increasing the number of fresh address in *tried* that are likely to be online when a node restarts.

2.1.4.5 Mining Pools

There are also organisations called *mining pools*, composed by multiple nodes that work together to find the PoW more efficiently. In mining pools, each node tests different nonces to find the PoW which is faster than a single node testing all the possible nonce's. *Mining pools* are composed by multiple nodes and gateways that connect the pool to the Bitcoin network. Once they find a PoW and generate a block the reward is then split among the members of the pool that worked on that PoW proportionately to the contribution that each member made. Accordinging to `blockchain.info` currently around 89.2% of the blocks created come from mining pools.

2.1.5 Information propagation

In order to exchange messages, Bitcoin nodes maintain for each of its neighbours, a message queue that is used to push the different types of messages. Then all the messages in the queue will be sent once a timer associated with the queue timeout. The timeout is calculated using a Poisson distribution that has in consideration if the neighbour is outbound or inbound and attributes higher timeouts to inbound nodes.

2.1.5.1 Transactions

Transactions are usually relayed to other nodes through advertisements as follows.

Node *A* creates a new transaction, and add the advertisement for that transaction to the message queue of all its neighbours. If *A* learns, from advertisements that its neighbours send, that one of its neighbours already has that transaction, it will remove the advertisement for that transaction from the queue of that neighbour. When the timer associated with the queue of, for instance, the neighbour *B* times-out, *A* will send all the advertisements in the queue to *B* in the format of an *Inv* message. Once *B* receives the *Inv* message, it will check if it already has all the transactions in the advertisement, after going through all advertisements, *B* will then send to *A* a *GetData* message requesting all the transactions advertised that it currently does not has in his *mempool*, structure where nodes save all the transactions that have not yet been added to a block. After receiving the *GetData* message, *A* will then add the requested transaction individually in a *TX* message to the queues of the neighbours that requested it. Finally, once *B* receives the new transaction it will verify if it is valid and if it is, it will also relay it through his neighbours.

However, transactions can also be relayed to other nodes in two other ways, direct push or *BlockTX* messages. In direct push the transaction is sent directly to the other nodes. However *BlockTX* messages are only sent to reply to *GetBlockTX* messages that nodes send when they received a compact block and do not have all the transactions necessary to rebuild the block as explained in the next paragraph.

2.1.5.2 Blocks

Until Bitcoin version 0.13.0 (23-08-2016) the usual way to relay blocks was also through advertisements, similar to the way transactions are relayed. However, with the addition of a new message type, compact block *CMPCTBlock*, the mechanism for relaying blocks changed a bit.

The difference between a block message and a compact block message is that the block message contains not only the header of the block but also all the transactions present in that block, whereas, the compact block message only contains the header of the block and the ids and the index of the transactions present in that block. Another difference is that once a node receives a compact block it has to rebuild the block and validate it before relaying it. Hence the main advantage of the compact block over the block message is the lower amount of bandwidth consumed to send a block. Because of this, nowadays the main method for relaying a block between up to date nodes is through compact blocks. However if, a node receives a block with ids that it does not know, it has to send to the neighbour that sent him the block a *GetBlockTX* message requesting the transactions it is missing. Finally, once a node receives a *GetBlockTX* it will respond with a *BlockTX* message containing all the missing transactions (Corallo 2016).

This means there are in total tree ways a block can be relayed as we can see in Figure. 2.3. In this figure we can see how the protocol behaves in the legacy relaying in A we can also see the two new ways to disseminate information. The ones described previously was A and B. In (Corallo 2016) the author claims that a Bitcoin node would use the protocol described in C to disseminate blocks if it had low bandwidth but during the tests we did and the inspections we did to the source code we never saw this protocol being used.

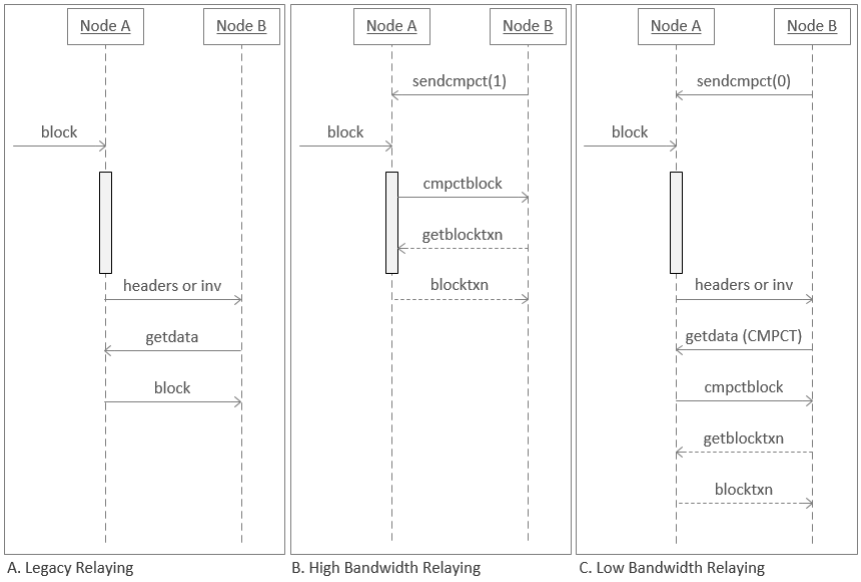

Figure 2.3: Protocol Flow (Source Matt Corallo)

There are also messages that allow, nodes that were disconnected from the network, to quickly get the data that they have missed. These messages are especially useful to speed up the process of gathering data for the creation of blocks. They are also useful when nodes do not have the parent block of a block they just received. In this case, they can use these messages to ask their neighbours for the missing block (Bitcoin.org 2009).

2.1.6 Summary

In conclusion, Bitcoin is a very complex system with many modules and libraries that allow it to work properly. However, in this thesis, we were mainly focused on improving its dissemination system. From the point of view of our work, the most important modules are depicted in Figure 2.4. We will now briefly describe each component.

Starting with the Blockchain module, this module is responsible for every operation related to the blockchain, from receiving transactions and validating them to broadcasting blocks. This module is composed mainly of three sub-modules, the broadcasting module, the processing module and storage module. We will now go over each individual module:

- **Broadcasting module** is responsible for broadcasting/receiving blockchain information to/from the current neighbours of a node;
- **Processing module** is in charge of validating every transaction and block a node receives;
- **Storage module** stores all information necessary to maintain the blockchain.

Regarding the membership, this module it is composed of four modules the connected peers and the feeler connections, a peer management module and a storage module.

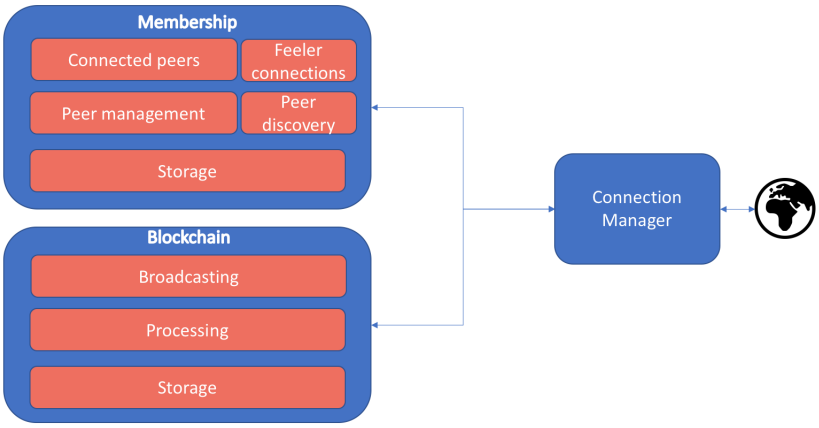

Figure 2.4: Bitcoin architecture

- **Connected peers module** as the name suggests is the set of peers currently connected to a node;
- **Feeler connections module** are the connections described at the end of Section 2.1.4;
- **Peer management module** is responsible for managing each peer and check if they should be banned or not.
- **Peer discovery module** is in charge of finding new peers to connect to;
- **Storage module** is responsible for storing information about each connected peer and as well as the *tried* and *new* tables.

Finally, the connection manager is in charge of establishing the low level connections between the peers.

Next, we will take a look at some of the vulnerabilities of some of the modules of Bitcoin to understand which modules have to be modified in order to fix those problems.

2.2 Bitcoin and Blockchain Vulnerabilities

In this section, our objective is to analyse known vulnerabilities in some of the aforementioned modules. Hence, this section is divided into two sub-sections, in Section 2.2.1 we will present vulnerabilities related with the blockchain module and Section 2.2.2 we will discuss vulnerabilities related with the membership module.

2.2.1 Blockchain Module

In this subsection, we will cover known vulnerabilities of the blockchain module in Bitcoin.

2.2.1.1 Information Broadcast Module

We will now describe three vulnerabilities in this module: information eclipsing, block delay and double-spending.

Starting with information eclipsing, this vulnerability happens because of the following behaviour. Lets consider that the whole network recognises a block *b* at height H_b as the blockchain head. Then at two different locations of the network two new blocks are discovered. These blocks are guaranteed to be different as seen in Section 2.1.2, lets call each one *b'* and *b"* (Decker and Wattenhofer 2013).

Now both blocks are going to be broadcast through the network because both are at height H_{b+1}. Hence, when a node receives either *b'* or *b"* it will consider it as the new blockchain head and will broadcast it to its neighbours. The problem is, when a node that already received *b'* receives *b"* or vice-versa that node will not broadcast *b"* through the network as it did with *b'*, because it already moved to a new blockchain head at the same height. As a consequence, only nodes that received both *b'* and *b"* know about the existence of a fork. This diminishes the effective computational power in the network because the network will be working on different blockchain heads. Hence, empowering some of the attacks in the following sections.

The authors of the paper discovered that it takes 6.5 seconds for the block reach 50% of nodes, 40 seconds for it to reach 95% of nodes and that the mean delay for a block to reach a node in the network is 12.6 seconds. Since Bitcoin has a block creation time of 10 minutes the authors concluded that the effective computational power in the Bitcoin network is only 98.20%. This happens because once a block is found it takes in mean 12.6 seconds for it to reach a node, so during those 12.6 seconds the rest of the network is wasting resources (resources that do not contribute to extend and strengthen the blockchain).

If we wanted to use Bitcoin for faster transactions, then the block dissemination time would have to go down. But if we maintain the 12.6 propagation delay and decrease the block dissemination time the effective computational power would also decrease even more as showcased in (Buterin 2014). The intuition is that with a lower block time the probability of stale blocks[1] being created increases. This will result in more forks being created and more resources more wasted resources.

It is worth noting that the problem identified in this work is worth revisiting as the protocol for block dissemination has evolved considerably over the years, for instance, with the introduction of compact blocks which implements some of the ideas proposed in (Decker and Wattenhofer 2013). Furthermore, at the time of this writing, according to `blockchain.info` there have only been five instances of soft forks in 2018, meaning that two blocks were created at the same height.

The block delay vulnerability is based on slowing down the propagation of new blocks sent to a set of nodes without disrupting their connections (Apostolaki, Zohar, and Vanbever 2017). This attack takes

[1] Blocks that were candidates to being the next blockchain head but were not chosen.

advantage of a specific condition where nodes wait 20 minutes before requesting a block to another node if the node how advertised the block did not reply with the block requested.

This vulnerability, called block delay, has two different ways of being performed depending on the direction the attacker is able to intercept the traffic: in the victim network $V \rightarrow N$ direction or network victim $N \rightarrow V$ direction.

If the attacker is able to intercept the connection in the $V \rightarrow N$ direction then the attack works as follows. When the victim requests a block to the network the attacker will change the block request to another block which will lead the victim waiting for a response. After almost 20 minutes, time limit where the victim requests the block to another neighbour, the attacker will modify another message sent by the victim to that neighbour, probably a transaction request since are the most common message type, to request the initial block that the victim wanted. This way the attacker avoids the victim from dropping the connection with that neighbour, which allows the attacker to perform the attack multiple times.

If the attacker is able to intercept the communication in the other direction $N \rightarrow V$. In this direction, once the victim requests a block to a neighbour the attacker will intercept and tamper with the response (the block itself) corrupting it, which will lead to it being discarded by the victim. But the victim will not request the corrupted block again. Then, after 20 minutes the victim will drop that connection because the block never arrived and will request the block to another neighbour. Hence, the attack performed this way can only be done once.

The impact of this attack depends on the node that is being attacked. If it's a common node, like the one a normal person would run, then this attack will be a Denial of Service and that node will not have guarantees for double spending. However if that attack is towards a gateway of a pool it could be used to engineer block races, block races will be further discussed in Section 2.2.1.2.

A solution for this attack is monitoring the RTT of the block requests, as the RTT increases considerably when the node is being attacked. Other solution proposed in (Apostolaki, Zohar, and Vanbever 2017) is encrypting Bitcoin communications which would not avoid the packets from being dropped by an attacker but it would prevent them from eavesdropping and tampering with connections. The disadvantage with this approach is that it would require additional computations making the system slower.

Finally, the double spending vulnerability, is possible because either a node is not well connected to the rest of the network or because the information is not being broadcast properly. As previously mentioned in Section 2.1.1 a double-spending attack occurs when a user tries to spend an already spent Bitcoin. Multiple mechanism have been implemented to lower the probability of this attack happening. We now highlight a proposed approach to this problem.

Bitcoin Clustering Based Ping Time protocol (BCBPT), is a solution that aims to increase the proximity of connectivity among nodes in the Bitcoin network based on round-trip ping latencies. Since currently, in the Bitcoin network, a node connects with nodes regardless of any proximity criteria. The main objective of this article is to lower the overhead in transaction verification which makes some nodes of the systems vulnerable to double spend attacks (Owenson, Adda, et al. 2017).

This protocol is implemented in two phases, distance calculation and cluster creation and maintenance.

A. Distance calculation In this phase each node is responsible for gathering proximity knowledge regarding the discovered nodes. Specifically, when a node discovers new Bitcoin nodes, it calculates the distance between itself and the Bitcoin nodes that it has discovered using the round-trip latency between itself and the other nodes.

B. Cluster creation and maintenance In this phase the DNS service nodes should recommend available nodes to, for instance, a node *N* based on the proximity in the physical geographical location as the geographic distance on the internet is many times a good indication of the topologic distance. So when *N* joins the network for the first time it learns about the available Bitcoin close to it. However, *N* should also make a ranking on which node to select and which not as the initial DNS seed service might return sub optimal peers. Therefore, *N* calculates the distance to each discovered node in order to get its proximity ordering based on a latency threshold. This ordering would help the node *N* to be directed to a specific cluster. Once the node *N* connects to the node *K*, it receives a list of IPs' of nodes that belong to the same cluster of the node *K* in order to allow the node *N* connects to the nodes that belong to *K*'s cluster only.

2.2.1.2 Validation Module

Now, we will take a look at two different vulnerabilities in the validation module. One is the 51% attack a well-known vulnerability common to many other blockchain systems and the other is the selfish mining vulnerability where a miner is able to generate an advantage to himself in the mining process.

Starting with the first one, the 51% attack was first introduced in the original Bitcoin paper (Nakamoto 2008). This attack requires the attacker to control 51% of the computational power available in the network, hence the name. Given the inefficiencies in the propagation of messages through the network seen previously the required computational power to perform the attack is actually 49.10% (Decker and Wattenhofer 2013). Once the attacker is able to obtain this computational power it will proceed to rebuild blocks erasing transactions where he was the payer. Note that this is the only thing the attacker can do because nodes would never accept a block with forged transactions. This is because valid transactions require the signature of the payer.

Although this attack is possible, it is very hard to perform because the attacker would need to rebuild not only the block where his transactions were present, but he would also need to rebuild all the blocks built on top of that block until he reached the blockchain head and overtook it, making his branch longer. As a consequence, this would result in the rest of the network considering that branch as the main branch, leading to the success of the attack. But as Satoshi Nakamoto explained in (Nakamoto 2008), if the attacker does not catches up to the blockchain head early on and overtakes it the difficulty of the attack will increase. Because more blocks will be built on top of the main chain hence he will have more blocks to generate.

This attack is possible because of the way consensus is reached in Bitcoin. Bitcoin reaches consensus by considering the main branch of the blockchain the taller branch on the blockchain tree. So if someone was able to control 51% of the computational power he could eventually generate a taller branch which would make the rest of the network consider that the main branch. Hence this problem cannot be solved unless we were able to control who joins the network.

Regarding the second vulnerability selfish mining is used to generate block races previously referenced in this Chapter. This vulnerability can be exploited in the following manner, let us assume that miners are divided into two groups, a colluding minority pool that follows the selfish mining strategy, and a majority that follows the honest mining strategy. The objective of the selfish miners is to reveal their mined blocks selectively to invalidate the honest miners' work. Hence, the selfish mining pool keeps its mined blocks private, secretly bifurcating the blockchain and creating a private branch. Meanwhile, the honest miners continue mining on the shorter, public branch. Because the selfish miners command a relatively small portion of the total mining power, their private branch will not remain ahead of the public branch indefinitely. Consequently, selfish mining judiciously reveals blocks from the private branch to the public, such that the honest miners will switch to the recently revealed blocks, abandoning the shorter public branch. This renders their previous effort spent on the shorter public branch wasted, and enables the selfish pool to collect higher revenues by incorporating a higher fraction of its blocks into the blockchain (Eyal and Sirer 2018).

2.2.2 Membership Module

In this subsection, we will cover known vulnerabilities of the membership module in Bitcoin in particular of two sub modules. The peer discovery module and the peer management module.

2.2.2.1 Peer Discovery Module

For this module we will start by introducing the concept of secure overlays and present some overlay technologies then we discuss vulnerability of this module.

An overlay is a network built on top of another network where peers are connected through virtual or logical links. For instance in Bitcoin, the overlay of a node is the connections that node keeps with other peers that are its neighbours.

In large systems like Bitcoin or BitTorrent it is infeasible for a peer to know the overlay of the whole network, not only because of the size of the networks but also because peers are constantly joining and leaving. So to solve this problem peers keep only a partial list of the peers ”closest” to them. Each peer is also responsible for updating and expanding his own list. To build these partial lists peers use Peer Sampling System (PSS), these systems are a scalable and robust approach to building these lists. They provide every peer with a random sample of peers to exchange information with (Jelasity, Guerraoui, Kermarrec, and Van Steen 2004).

An important issue of modern PSS is their potential exploitation by malicious peers. Many of the technologies of PSS did not take into account attacks like the hub attack. The goal of this attack is to subvert the network in order to achieve a leading structural position hence becoming a hub. This is problematic because it can evolve in severe problems to the system, if the malicious nodes simply disappear after having gained such leading position. For instance, if we consider a streaming service, a problem could be the system being temporally unavailable due to all the files being hosted by only that central server that disappeared.

Attacks to the PSS like the hub attack led to the creation of Secure Peer Sampling (SPS) services. SPS use heuristics based on social network analysis to allow the system to detect and react to the structural changes in the network in a timely manner. Hence, nodes which have gained a central role in the network are identified and banned.

But even attacks to SPS have been found (Jesi and Montresor 2009). The objective of these attacks is to put discredit on a subset of nodes in order to disconnect or isolate them. This is achieved by a set of malicious nodes broadcasting bogus messages to discredit the victims, which will eventually lead the SPS to react by suspecting and banning those nodes, which are non-malicious. This attack is called the Mosquito attack.

Another problem is nodes with a high number of connections. However in Bitcoin this is a problem as these nodes start having an important role in the network and if for instance one of these nodes leaves the network the system might become unavailable for a long period of time for some nodes as that was their main point from where they received updates.

We will now take a look at some overlays systems. Starting from the oldest system, this system that was built to cope with the *hub attacks* described previously. This system uses a set of multiple overlays, this means that each node belongs to different overlays, and the neighbourhood at every instance will be distinct with very high probability because the overlays have independently random-like topologies. In this system, the concept of extra caches is introduced as being the set of caches belonging to each peer; every cache in the set is a random snapshots of a distinct PSS overlay. Essentially, the multiple caches are useful in order to perceive how malicious nodes are spreading the infection from distinct directions over distinct overlays. Due to the spreading infection, is expected that common node ID patterns will emerge in all (or in the majority) of the caches. Then based on this patterns, each peer can build a set of statistics in order to guess or detect who are the malicious nodes. If a node is detected as malicious the node will decline gossip from that node (Jesi, Hales, and Van Steen 2007).

Followingly Bramhms appeared, a system with the objective of providing to nodes a random sample of nodes in a large dynamic system subject to Byzantine attacks that poison the views of correct nodes. Brahms is a membership service that stores a sublinear number of ids at each node and provides each node with independent random node samples that converge to uniform ones over time. This is achieved by Brahms because in its sampler every node has the same probability of being sampled and the gossip algorithm uses two means for propagation: (1) push – sending the node's id to some other node, and (2) pull – retrieving the view from another node. Pushes are needed to reinforce knowledge about nodes that are under-represented and pulls are needed to spread existing knowledge within the network. Brahms protects itself against poisoning attacks by limiting the number of pushes received by nodes (Bortnikov, Gurevich, Keidar, Kliot, and Shraer 2009).

The same author that proposed the first system also introduced a new system that was designed to extend the SPS functionality protecting it against the mosquito attack mentioned previously. In this system, the attacker is considered to be a group of peers. This system introduces the concept of a knowledge base, this knowledge base is used by a peer to possibly recover its partial view in case of corruption and to detect with a good accuracy malicious peers. Peers build their knowledge base by making a stochastic proportion of their gossip exchanges as ”explorative”. The intuition behind this system is that since the network should be random, detecting a peer showing a popularity value too

distant from the average means that it could represent a network hub. So each peer will record in its knowledge base the number of times that the address of each peer was shared with them, this value is called *hits*. Once a peer identifies another peer with a high number of *hits* it marks it as malicious. To protect against false negatives induced by attacks like the mosquito attack, each well-behaved peer will choose one malicious peer from their list of malicious peers and will make an explorative PS. If the results received contain more than 25% of the already known malicious peers the suspicion is confirmed. Otherwise, the peer is removed from the list of malicious peers (Jesi and Montresor 2009).

However other systems attempt to prevent other kinds of attacks, for instance MANCHETE. Note that this system is not like the other PSS presented because it provides every client with a full view of the network instead of a partial view. MANCHETE attempts to establish an overlay network and scatter data over the available paths, thus reducing the effectiveness of snooping attacks. A snooping attacks is unauthorised access to another person's or company's data. The practice is similar to eavesdropping but is not necessarily limited to gaining access to data during its transmission. Thus, it protects against passive attackers that eavesdrop on communications at certain physical locations. This system sends data through multiple paths using multiple interfaces that the computer has available, hence protecting against snooping attacks. In contrast to the other systems presented previously in MACHETE the client is provided with the full overlay of the system when he wants to send a message. Then the client will choose the best path according to its RTT value (Raposo, Pardal, Rodrigues, and Correia 2016).

We will now take a look at some vulnerabilities found in the peer discovery module, the common idea of these vulnerabilities is that they are possible because of how Bitcoin builds its overlay. Starting with the partition attack this vulnerability allows for an adversary with an Autonomous System (AS) level to isolate a set of nodes from the rest of the network using the Border Gateway Protocol (BGP) hijack.

BGP hijack consists in injecting forged information in the network on how to reach one or more IP prefixes, leading other ASes to send traffic to the wrong location.

The attack starts by the given AS diverting the traffic destined to a certain set of nodes (which we will call *P*) through BGP hijack. This means that all traffic destined to *P* goes to the AS instead. Then, the AS will examine the traffic being sent to *P* and it will drop every message that has a Bitcoin header in the TCP payload. The rest of the traffic is considered irrelevant and reaches *P*.

During the interception of traffic, there can be two types of relevant traffic: i) traffic crossing the partition from the outside to the inside; ii) traffic that appeared from inside the partition. In the first case, the AS only has to drop Bitcoin messages but in the second case, the AS has to analyse the exchanged Bitcoin messages to detect the "leakage points", which are nodes that have connections to the outside of the partition that the AS cannot control.

To accomplish this, the attacker checks, for every packet, if the sender belongs to *P*. If the sender belongs to *P* the attacker will check whether the sender is advertising information from outside *P*. Particularly, the attacker checks whether the packet contains an *Inv* message with the hash of a block mined outside of *P*. If yes then the sender was a "leakage points" (Apostolaki, Zohar, and Vanbever 2017).

Once the AS finds out the "leakage points" it will exclude them from *P*, successfully isolating the nodes in *P* from the rest of the network.

This attack leads to different consequences depending on the number of nodes successfully iso-

lated. If the number is low the impact of the attack is basically a Denial of Service and the nodes within *P* have zero confirmation for double spending. If the number of nodes is high it might result in revenue loss for miners and the side with higher computational power will decide which transactions are committed because they will generate blocks faster.

Regarding solutions for this attack in (Apostolaki, Zohar, and Vanbever 2017) the authors suggest, increasing the diversity of the node connections while taking routing into account, monitoring the RTT because the RTT value would increase if a node was being attacked and a few others.

The success of this attack depends on the protocol used by the ASes to advertise addresses. For instance, if BGP was not used by the ASes this attack would not be possible.

Another vulnerability of the overlay built by Bitcoin is that users are vulnerable to deanonymization. These attacks typically use a "supernode" that connects to active Bitcoin nodes and listens to the transaction traffic relayed by honest nodes. Beacause nodes diffuse transactions symmetrically over the network, researchers were able to link Bitcoin users' public keys to their IP addresses with an accuracy of up to 30%. In this works the authors present Dandelion, a new spreading protocol that consists in two phases; an anonymity phase and a spreading phase.

- **Anonymity Phase** In this phase each transaction is propagated on a random line; that is, each relay passes the message to exactly one (random) node for a random number of hops;
- **Spreading Phase** In this phase each message is broadcast using diffusion until the whole network receives the message.

Dandelion has two key constraints: (a) in the first phase, all transactions from all sources should propagate over the same line, and (b) the adversary should not be able to learn the structure of the line beyond the adversarial nodes' immediate neighbours. However, as stated by the authors this separated architecture is not necessarily optimal in terms of a latency-anonymity tradeoff (Bojja Venkatakrishnan, Fanti, and Viswanath 2017). Although we categorise this vulnerability as being a vulnerability of the peer discovery module it could also be considered a vulnerability of the broadcasting module.

Finally another vulnerability we found while analysing the Bitcoin protocol is the Mosquito attack. This vulnerability is exploited in the following manner.

This attack is reproduced in Bitcoin through the Denial of Service (DoS) prevention that system has. The Bitcoin DoS prevention system in the following manner, once node *A* receives from node *B* more than 1000 IPs (max number of address entries allowed on an *Addr* message) on an *Addr* message it punishes *B* (Bitcoin.org 2008). The punishment can vary from *A* simply dropping the connection to *B* to *A* banning the IP of *B* so he cannot immediately re-connect to *A* for a couple of hours.

Following the strategy described in (Jesi and Montresor 2009) a set of attackers would send to a node multiple *Addr* messages with more than 1000 IPs with the IP of the neighbours of the victim, which would lead the victim to punish her neighbours isolating herself. However, for this attack to be successful the attacker would have to know the IP of the current neighbours of the victim and prevent the victim from establishing connections to other peers once the attack starts.

The impact of this attack depends on the importance of the victim. If the victim was just a simple miner or node then this would function like a DoS attack and it might result in revenue loss for the miner.

If the victim was a gateway to a pool then this could result in much bigger revenue loss and it could be used to engineer block races.

2.2.2.2 Peer Management Module

We will start by giving an introduction to node behaviour then we will take a look at some work developed in this area and finally we will list some vulnerabilities found in this module.

In general, nodes can adopt different behaviours with respect to the specification of the protocol they are supposed to follow, namely, there are three types of nodes *altruistic nodes*, *byzantine nodes* and *rational nodes*. *Altruistic nodes* are nodes that follow the specified algorithm and are willing to disseminate information. *Byzantine nodes* are nodes that generate arbitrary data, and can behave in an arbitrary way, including pretending to be a correct one. *Rational nodes* are nodes that instead of strictly following the algorithm do what is best for them. For example, in the case of file sharing, a *rational node* is a node that only provides small rates of upload while having a high rate of download. These nodes are harmful to the system because they do not contribute to it, for instance, if all nodes were to follow the same logic there would not be enough seeders to support the network and the system would collapse (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006) (Cohen 2003).

It is important to look at these behaviours and the approaches to punish them because although it would be desirable that in the Bitcoin network all nodes behaved well that is not the case.

Next, we discuss how some approaches deal with some of these possible behaviours. In the context of a file sharing network node can chose to behave like *free riders*. A free rider in this systems is a peer that is trying to have the highest possible download rate while having a low upload rate. There have been some systems that try to prevent this behaviour, one of these systems tries to prevent *free riders* with a policy of tit-for-tat. This is achieved by peers uploading only to peers which upload to them. This way the network will have at any given time connections which are actively transferring in both directions (Cohen 2003).

Other system that tries to prevent free riders is BAR Gossip. This system tries to guarantee a predictable throughput and low latency in the BAR model, in which non-altruistic nodes can behave in a self-serving or even arbitrarily malicious way. BAR Gossip attempts to prevent free riders by having two protocols to disseminate information: Balanced Exchange and Optimistic Push (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006).

Balanced Exchange In this protocol peers exchange information while keeping the trade equal. This means that the amount of information that a peer uploads is the amount that it is able to download. This exchange is ciphered and signed so that both parties act faithfully (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006);

Optimistic Push This protocol exists to compensate peers that have fallen behind and are not able to perform a Balanced Exchange. In this protocol, the peers that have fallen behind when they are unable to provide useful updates they are allowed to send junk. To avoid peers from abusing this protocol the amount of junk sent has to be equal the amount of actual information. In this paper the authors also state that a rational node would not choose a strategy of just sending junk because i) it would not have a

discernible impact on benefit and ii) junk is more expensive to send than legitimate updates (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006).

Another system that tries to solve the problem of *free riders* is LiFTinG this systems is the first one to detect *free riders* in a gossip-based content dissemination system with asymmetric data exchanges. In this protocol, each peer is monitored by a set of peers chosen randomly. Each peer has a score that if it drops below a certain threshold, it is assumed that that peer is *free riding*. The score drops if a node that was exchanging information with the *free rider* suspects that he is not being faithful and broadcasts a blame message against him. Once a node is considered guilty by the managers they spread a message to inform the other peers hence, punishing the *free rider* (Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010).

We will now discuss some behaviours that nodes chose to adopt in the Bitcoin network. Starting with a behaviours previously mentioned is *Supernodes*, these are nodes that establish more connections than the desired amount. As we have seen Bitcoin was designed to have a random overlay topology. This protects Bitcoin because it makes it harder for nodes to achieve a central position and have excessive control over the network. But despite this, recent findings like the ones in (Miller, Litton, Pachulski, Gupta, Levin, Spring, and Bhattacharjee 2015) revealed that the topology of Bitcoin is not purely random with some nodes having more than 125 active connections (restriction of the mainline Satoshi client) sometimes by a factor of nearly 80 (Miller, Litton, Pachulski, Gupta, Levin, Spring, and Bhattacharjee 2015).

This is a problem for the stability of the Bitcoin because it is centralising resources. These nodes are usually gateways of mining pools. So if an attacker were to identify these nodes with a tool like AddressProbe (Miller, Litton, Pachulski, Gupta, Levin, Spring, and Bhattacharjee 2015) it could launch an attack on these nodes and have a big impact on the network. Because all the miners in the mining pool would be disconnected from the network causing revenue loss for both the miner and the mining pool and decreasing the computational power of the network.

It is also worth noticing that although *Supernodes* open some vulnerabilities in the network, they are also good for the performance of the system. For instance, when a block is found if it reaches a *Supernodes* will be disseminated much faster through the whole network hence, decreasing the probability of forks happening.

Another behaviour that nodes can adopt is being *selfish*. These nodes are nodes that do not broadcast a block right after its discovery, as previously mentioned in selfish mining. They keep it until a new block is announced by another node, only then they broadcast their own block with the intent that it being the one to get accepted as the new blockchain head. This will result in revenue loss for the node that discovered the other block as he will not be rewarded by the block that he found. Furthermore, the *selfish node* will benefit even more from this behaviour because it will have a lead in finding the next block if the one accepted was his own.

Single nodes usually do not have a very good connection to the rest of the network which translates to them having a low probability of their blocks being accepted versus already broadcasted blocks. So although *selfish mining* is not very effective if performed by a single node, if a mining pool decides to implement such behaviour it will probably succeed. Because usually, mining pools gateways have a lot of connections (Miller, Litton, Pachulski, Gupta, Levin, Spring, and Bhattacharjee 2015), the probability

of their blocks being accepted is very high as they can broadcast them through the network very quickly, resulting in revenue loss for other mining pools or other nodes.

Finally, there used to be a vulnerability in the peer management module that allowed an attacker to perform an eclipse attack to a victim. Since then multiple patches have been done to address the problem (Bojja Venkatakrishnan, Fanti, and Viswanath 2017).

The vulnerability was exploitable because, each time a node received a new address of a node through *Addr* messages it would add it to a bucket in the *new* table presented in Section 2.1.4 and if for instance, the bucket was full it would replace a previous address in there. As a result, the attack was performed in the following manner (1) the attacker would populate the *tried* table of the node with addresses for its attack nodes by connecting with the node, and (2) it would as well overwrite the addresses in the new table with "trash" IP addresses that were not part of the Bitcoin network. Eventually, the node would disconnect itself from its current neighbours and it would connect to the nodes of the attacker, once all connection outgoing connections were from nodes of the attacker the node would be isolated (Bojja Venkatakrishnan, Fanti, and Viswanath 2017).

The first patch to address the vulnerability was mapping each address to a specific position in a bucket. After the developers also introduced feeler connections that test addresses in the new table. An at the begging of the year they introduced test before evicting which happens if two addresses happen to map to the same position in the same bucket.

2.2.3 Discussion

We have seen that vulnerabilities have been identified in every module of the Bitcoin architecture. Some vulnerabilities are more severe than others and several vulnerabilities have already been patched. We will now go over each module discussed and assess the current state of the art:

- **Information Broadcast Module** This module has as vulnerabilities the information eclipsing, the block delay attack and the problem with the time transaction were taking to be committed. Nowadays, after studying the system we were able to assess that both the problems of information eclipsing and latency between transactions being created and being committed should be mostly solved. Because as said previously, the current soft fork rate is very low as well as the current amount of missing transactions when nodes try to rebuild compact blocks (5%). The block delay attack is still an open problem to the best of our knowledge.
- **Validation Module** In this module, we presented two vulnerabilities, the 51% attack and selfish mining. Starting with the 51% attack this problem is still not solved and probably will never be since it is extremely tied with the methodology for Bitcoin to reach consensus. Regarding the selfish mining problem it is also still not solved but like the 51% the probability of happening is very low. Because both these vulnerabilities have as a requirement that the attacker is able to gather big amounts of computational power and given the current size of the system, one possible attacker would be the current miners/mining pools. However, these entities would never attack Bitcoin as they care about the cryptocurrency which is probably the main source of revenue for them.

- **Peer Discovery Module** In this module, we discussed three vulnerabilities: the partition attack, the deanonymization of users and the mosquito attack. To the best of our knowledge the partition attack is still an open problem. Regarding the deanonymization of users although there is still no patch to fix this problem it seems that there are tendencies to adopt the proposed solution of Dandelion. Finally, even though the mosquito attack is possible to be performed reliably the requirements are very demanding.
- **Peer Management Module** In this module, we presented only a single a vulnerability the eclipse attack. As said previously this vulnerability has already been patched.

3 Adaptive Biased Dissemination

To better understand these attacks, we modified the Bitcoin client to gather measures about the state of the network. This helped us assess the feasibility and impact of each of the attacks and their respective vulnerabilities as well as to look into uncovered opportunities for improvement. Briefly, the introduction of compact blocks in 2017 helped mitigate the Information Eclipsing vulnerability resulting in a low fork rate. Also, we found out that only 5% of the transactions in a compact block had not been received previously, leaving room for marginal improvements. Conversely, we observed that nodes receive a substantial number of transaction duplicates which can clearly be improved. In the rest of this dissertation, we present our mechanisms to improve the efficiency of the information dissemination protocol without compromising the performance and reliability guarantees of Bitcoin.

This chapter describes an improved and adaptive dissemination protocol that not only solves current problems of resource utilisation but also maintains key aspects of the dissemination in Bitcoin.

Our proposal is based on the following observations:

- Currently, each node receives on average 6.6 duplicate advertisements for each transaction (when would be enough to receive a single one to ensure the reception of a transaction).
- The network currently possesses two methods to disseminate transactions: exchange of advertisements (used when a transaction is not in a block) exchange of block (used when a transaction is already added to a block).
- For historical reasons, the second mechanism is more efficient than the first one, since all the missing transactions that a node might request are sent in a single message (while in the advertisement method a node has to send a message for each individual transaction).
- In Bitcoin, the requirements for broadcasting transactions are weak because the rate of generation of the blocks is much slower than the processes of dissemination of transactions (on average a block is generated once every 10 minutes while transactions reach a the majority of the network in a manner of seconds).
- Miners are only a small fraction of the total number of nodes in the network. However, although it is more important that the transactions reach miners than non-miners, the protocol does not distinguish miners from the rest of the nodes.
- In the current protocol, nodes send their advertisements to all neighbours (125 in the worst case). This value is substantially higher than the theoretical value for epidemic broadcasting algorithms, which suggests that even in the presence of failures, it is enough to send information to a logarithmic number of neighbours with respect to the size of the network (Eugster, Guerraoui, Kermarrec,

and Massoulié 2004). With the current size of $\approx 10\ 000$ nodes it would be enough to send to $ln(10\ 000) \approx 10$ neighbours.

Our main objective is to lower the amount of duplicated advertisements in the network while ensuring that the transactions reach the miners. The intuition for the proposed approach is to skew the process of dissemination towards the most productive miners. However, this could put the resilience of the system at stake. To prevent this, we also broadcast transactions to the rest of the system through alternative paths.

To achieve these results we have to first solve some challenges. First, we have to be able to identify which nodes are going to mine blocks, or which neighbours are connected to miners. This is difficult because as we explained in the previous chapter the process of mining is random and any node can mine a block. Second, we have to give priority to these nodes without compromising the resilience of the system. As we have seen previously the process of dissemination is crucial for Bitcoin to work properly, since problems in the dissemination could open up Bitcoin to multiple attacks, from selfish mining to double-spending. Third, the paths that our protocol will establish cannot be definitive as the Bitcoin network is prone to changes given that nodes join and leave the network over time. As a matter of fact, the Bitcoin network is very prone to change as can be observed in sites such as `bitnodes.earn.com` that show a fluctuation in the number of nodes from 9500 nodes to 12500 nodes in the last year.

Our approach encompasses three changes to the protocol. First, nodes maintain, for each neighbour, a list of the transactions sent by that neighbour and how long it took for these transactions to be included in a block. Second, we also maintain for each neighbour the time, it took to disseminate a new block to the node. Finally, nodes use these metrics to rank their neighbours and prioritise the dissemination of transaction accordingly.

The rest of this chapter is organised as follows, Section 3.1 presents the system model assumed by our protocol, Section 3.2 describes the process used to rank the neighbours of a node, Section 3.3 details a new algorithm of dissemination informed by the ranked neighbours and finally Section 3.4 presents the algorithm used by our protocol to adapt to changes in the network. Finally, Section 3.6 summarises the work.

3.1 System Model

We assume the following. The network is asynchronous and the channels are fair-lossy. Nodes are also asynchronous and might fail by crashing or by showing arbitrary behaviour, i.e. nodes can be Byzantine. Since in Bitcoin a block is generated every 10 minutes and every node has an equal chance to find a block, regardless of when the previous block was mined. Then the time between blocks follows the exponential distribution, which is indeed the only continuous memoryless distribution. This means that the probability of a block not being found for x minutes or longer is equal to:

$$F(x) = e^{-x/10} \tag{3.1}$$

Regarding transactions, these can be generated by any node. Every second two random nodes generate a transaction in order to obtain the average of 172 800 transactions a day observed in Bitcoin.

Our protocol affects the broadcasting module inside the Blockchain module and the storage module inside the membership module as it is depicted in Figure 3.1 (green boxes). The broadcast module was changed because, as previously, said this module is the one responsible for broadcasting protocol of transactions and blocks. The storage module was tweaked to be able to maintain the metrics that are need to be collected by each node in order for our protocol to work properly.

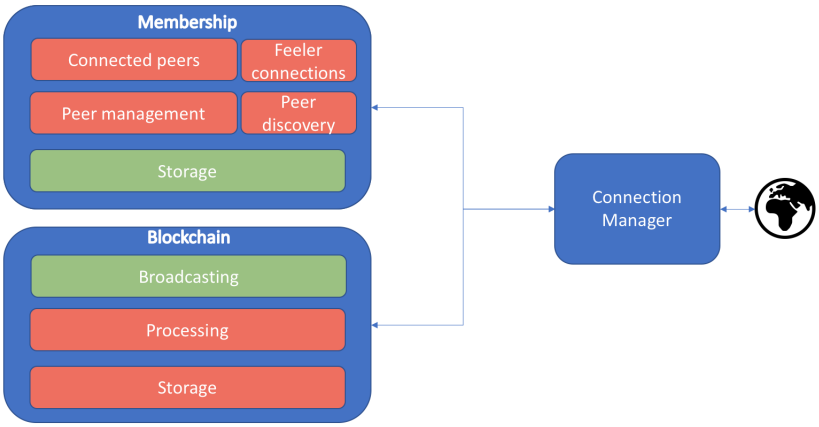

Figure 3.1: Bitcoin architecture with the modules that we modify highlighted in green

3.2 Ranking Neighbours

As miners are only a small fraction of the network, not all nodes will be directly connected to miners. As previously mentioned, our protocol has the requirement that nodes have to determine from their neighbours which ones are mining blocks.

A simple approach is to simply count the number of hops a block takes since it is created until it reaches a node. This means that if a node had a miner *A* and a miner *B* at distances 1 and 2 respectively. If for instance, the node received a block created by *A* then that block would have a field with the distance with value 1. With this information the node would give priority to *A* when disseminating transactions. However, this has two problems. The first is that this is prone to manipulation by Byzantine nodes. The second is that the smallest number of hops does not necessarily mean the fastest path from a node to a miner.

It is important to note that we are not trying to exactly determine which node is going to mine the next block, we just want to determine which nodes have a higher probability of mining a block or from the neighbours of a node which nodes are connected to these nodes. Hence, the protocol will attribute a higher rank to neighbours that:

- Disseminate blocks as fast as possible - because miners once they mine a block want to disseminate that block as fast as possible in order for the rest of the network to build on top of it;

- Disseminates all relevant blocks - because once a miner learns about a new block it starts immediately trying to mine on top of that block hence is in its best interest to also relay that block;
- Adds transactions as fast as possible to blocks - because apart from the block reward miners also profit from the taxes imposed on transactions when they are added to blocks.

With this intuition, each node is going to prioritise the neighbours that have the fastest paths to miners, or are miners themselves. This is done in a decentralised fashion but results in fast paths to miners emerging over other paths. Locally, each node, classifies neighbours as follows:

$$\text{class}^T = (\frac{k^T}{n^T} + a^T - n^T + \frac{y^T}{z^T}) \tag{3.2}$$

where:

- *k* it the accumulated time it took to a neighbour to disseminate each block to the node;
- *n* is the total number of blocks received by a neighbour;
- *a* is the total number of blocks received;
- *y* is the accumulated time it took for transactions, sent to a neighbour, to be accepted in a block;
- *z* is the total number of transactions sent to a neighbour.
- *T* is time frame that we use to know what transactions to take into account when determining the *class* of a neighbour

Going over each part of the equation, let us start with $\frac{k^T}{n^T}$. This fraction represents the pace at which a neighbour relays blocks to a node, for instance *P*. Here the time it takes for a neighbour to relay a block to *P* is given by the difference between the current time and the last time *P* received a new block from that neighbour. We add the time it took for *P* to receive all the blocks mined in T from a neighbour and then divide this value by the amount of blocks received, which gives us an average of time it takes for a neighbour to relay blocks to *P*.

The second part of this formula is $a^T - n^T$. This allows the classification to automatically adapt to situations where nodes that generate a block sparingly do not get a good classification indefinitely. Because we are subtracting the total number of blocks received (a^T) by the total number of blocks received from a neighbour (n^T). If for instance, a neighbour *J* does not have a fast connection to a miner or if *J* was a miner that stopped mining blocks then *J* will have lower priority versus other neighbours, since it is not able to reliably relay every block to us.

Finally, the thirth part of this equation $\frac{y^T}{z^T}$ is used to cope with nodes that might not relay transactions. In this fraction, we divide the sum of the time it took to commit transactions that *P* sent to a neighbour (*J*) by the number of transactions that *P* sent to *J*. With this, *P* will get an approximation of the average time that *J* takes to commit a transaction. However, given the large number of transactions that flow through the network, instead of maintaining timers for all of them, we only maintain a timer

every one hundred transactions. This prevents overloading nodes with metadata while still giving a good sample of the general network behaviour.

With this in mind, a neighbour has a good classification if: i) it has a good ratio of *time it takes to disseminate blocks/number of blocks we received from him*, ii) a good ratio of *blocks received from him/blocks received* and finally iii) a good ratio of *time it took for a transaction to be added to blocks if we sent it to him*.

Given that the classification of neighbours is prone to change over time, the actual value used to order neighbours is given by the following sliding average of the classification presented previously:

$$\text{class}^{t} = (1 - \alpha) \cdot \text{class}^{t-1} + \alpha \cdot \text{class}^{T} \tag{3.3}$$

The α factor exists to avoid nodes that generated a lot of blocks in the past but no longer do, from having a good classification forever and to prevent very dramatic fluctuations in the classifications of neighbours. In our experiments, we used an $\alpha = 0.3$ and a *T* configured to be an interval of four hours. We used a $\alpha = 0.3$ because we wanted to give more importance to that the past of a node than to the present as sometimes nodes might disconnect from that network or might not have the luck to mine a block in a longer time period. We also tried with other values of α, but we found 0.3 to give us the best results. Regarding the four hours of interval, we also tried multiple values and we obtained better results with the interval being four hours.

Algorithm 4 Top neighbours computation

```
1:  function UPDATE_NODES_CLASSIFICATION(node_to_update)
2:      scores ← [ ]
3:      for node in neighbourhood do
4:          score ← get_classification(node)
5:          scores.append([score, id])
6:      end for
7:      sort(scores) // sort by score from lower to higher
8:      top_nodes ← [ ]
9:      for i in range(0, max_t_nodes) do
10:         top_nodes.append(score[i][1])
11:     end for
12: end function=0
```

Hence in our protocol, each time a node receives a block from a neighbour the classification of the neighbours will be updated using Algorithm 4. The node will iterate over his neighbours and for each one, it will calculate his classification using Equation 3.3 and then it will append his classification together with his ID to a vector named scores (lines 3 to 6). After that, the node sorts the scores vector from the lowest score to the highest score, this means the nodes with the lowest score will be closest to the index 0 of the vector (line 7). Finally, the node will iterate from 0 to the max number of top nodes and append the first max_t_nodes of the vector of *top nodes* (lines 9 to 11). The values for the variable max_t_nodes will be discussed in the next section.

In the end, the node will have in its list of *top_nodes* the set of nodes with the lowest classification. Note that a low classification means that a node has a good/fast connection to at least a miner or a neighbour of a miner meaning the lower classification the better is that connection.

3.3 Skewed Relay

If all nodes followed the protocol and did not crashed or leaved the network, it would suffice to use the mechanism described above with the variable $max\_top\_nodes = 1$ to send transactions to only one node, as we would be sending the transactions to the best neighbour of each node which would eventually make the transactions appear in a block. With this we would also be lowering the amount of duplicated advertisements from 6.6 to 1.

However, even if we do not consider the problem of node failure and Byzantine behaviour, there is the problem of commit time. Commit time for cryptocurrencies is very important as not only a low commit protects the system against some attacks but also makes the cryptocurrency more appealing as transactions become confirmed faster. As mentioned previously, the variance of the mining process could result in a prolific miner not being able to successfully mine a block for an extended period of time, precluding transactions sent exclusively to it from being included in the blockchain which could make multiple nodes vulnerable to the double-spending attack. Furthermore it is not guaranteed which node is going to mine the next block hence, it is unadvised to send all the transactions of a node to only one neighbour.

Algorithm 5 Nodes to send transactions advertisements computation

```
1:  function NODES_TO_SEND(tx)
2:      if (ip == True and tx.source() == self) then
3:          return neighbours
4:      end if
5:      total ← max_t_nodes + max_r_nodes
6:      if size(neighbours) < total then
7:          total ← size(neighbours) − max_t_nodes
8:      else
9:          total ← total − max_t_nodes
10:     end if
11:     if total > 0 then
12:         r_nodes ← rand_choice(neighbours, total)
13:     end if
14:     return t_nodes + r_nodes
15: end function
```

We address this - and simultaneously node failures and Byzantine behaviour - by sending transactions not only to the *t* top nodes but also to *r* random nodes, as described in Algorithm 5. The variable *ip* (Initial push) indicates that if a transaction is generated by a node, the node has the option of either sending it to *t* plus *r* neighbours or to all of them. We implemented this feature to be able to understand if the first relay had much impact in the commit time of a transaction.

Hence, every time that a node has to relay a transaction it will start by verifying if the variable *ip* is enabled or not if it is it will return the full neighbourhood similar to Bitcoin (lines 2 to 4). If it is not then the node will start by adding the values of max_t_nodes and max_r_nodes to simply check if the size of the neighbourhood of the node is big enough to cope with the value of both variables added. In the end, the node will have in the variable $total$ the number of nodes not in top_nodes that can potentially be chosen as random nodes (lines 5 to 10). With this, the node will then randomly choose $total$ nodes from the

set neighbourhood excluding the nodes in top_nodes (line 11 to 13), in the end, this algorithm will return both sets.

This way we ensure that transactions reach the nodes with the higher probability of mining a block but also reach the rest of the network. Hence, ensuring that not only the transactions are committed in a timely manner but also reach other nodes lowering the probability of those nodes being attacked.

This dissemination process can then be configured with the following variables: *max_t_nodes*, *max_r_nodes* and ip to obtain different results in the information dissemination. We study the impact of these parameters more in depth in Section 4.2.1.

3.4 Adapting to Network Changes

A key aspect of Peer-to-Peer networks is that nodes can leave or join the network at any time. With this in mind, we designed an algorithm that adapts to the network in order to keep the commit time of the transactions while still trying to send as few messages as possible.

Hence, we designed an algorithm that starts by attributing a value of $size(neighbourhood)/2$ to both *max_t_nodes* and *max_r_nodes* simulating the Bitcoin dissemination process. Then each node monitors the commit time of its transactions, if this commit goes over or below a specified threshold the node is going to either increase or decrease the values of both *max_t_nodes* and *max_r_nodes* by one. With this end up with an algorithm that can behave in the worst case like Bitcoin and in the best case can bring improvements to the current protocol.

With this, each node is going to invoke Algorithm 6 every ten minutes as that is the average rate which blocks are mined. The algorithm starts by assigning to avg_time the average time its unconfirmed transactions are taking to be accepted (line 3). The time that is taking for an unconfirmed transaction to be confirmed is calculated by subtracting the current time with the time of creation of said transaction. Then the node will check if avg_time is bigger than the constant *TIME_TX_CONFIRM* (30 minutes).

If it is, this means the node will check if it can increase the values of both *max_t_nodes* and *max_r_nodes*, if it can then is going to increase both and relay the transactions that took more than 30 minutes to commit (lines 4 to 13).

If the average of all unconfirmed transactions does not surpass the threshold of the 30 minutes, then the node will first check if the average time it took to commit its confirmed transactions in the last hour took less than *TIME_TX_CONFIRM* (lines 14 to 15). If so the node will check if it can lower the values of *max_t_nodes* and *max_r_nodes* by one, if yes it will do it otherwise it will not do anything (lines 16 to 23).

We have chosen the threshold to be 30 minutes as that is the highest time registered in `blockchain.info` for transactions to be committed. We also have chosen to increase/decrease always both variables because as we are going to see in the next chapter when we run our protocol with $max\_r\_nodes = 0$ we did not obtain the best results, this way we make sure that both values will never be lower than 1. We also have chosen to only increase/decrease both values by one because we have also noticed that little changes like sending transactions to only one fewer neighbour already had a great impact.

Algorithm 6 Adaptive dissemination

1: **global variables**
2: $TIME\_TX\_CONFIRM \leftarrow 30 * 60$ // Maximum time allowed for a transaction to be committed
3: $TIME\_TO\_WAIT \leftarrow 4 * 60 * 60$ // Time to wait before the node is able to increase or decrease max_t_nodes and max_r_nodes
4: **end global variables**
5: **function** INCREASE_RELAY()
6: $now \leftarrow get\_current\_time()$
7: $avg\_time \leftarrow get\_avg\_time\_unconfirmed()$
8: $timeout \leftarrow avg\_time > TIME\_TX\_CONFIRM$
9: $space \leftarrow max\_t\_nodes + 1 \leq neighbourhood/2$
10: $cooldown \leftarrow last\_inc + TIME\_TO\_WAIT \leq now$
11: **if** $timeout$ **and** $space$ **and** $cooldown$ **then**
12: $increase(t, r, 1)$ // increase by one both max_t_nodes and max_r_nodes
13: $had\_to\_inc \leftarrow True$
14: $UPDATE\_NODES\_CLASSIFICATION()$
15: $last\_inc = now$
16: $relay\_delayed\_TX()$
17: **end if**
18: $cooldown \leftarrow last\_dec + TIME\_TO\_WAIT \leq now$
19: **if not** had_to_inc **and** $cooldown$ **then**
20: $avg\_time \leftarrow get\_avg\_time\_confirmed()$
21: $timeout \leftarrow avg\_time <= TIME\_TX\_CONFIRM$
22: $space \leftarrow max\_t\_nodes - 1 \leq 0$
23: **if** $timeout$ **and** $space$ **then**
24: $decrease(t, r, 1)$ // decrease by one both max_t_nodes and max_r_nodes
25: $UPDATE\_NODES\_CLASSIFICATION()$
26: $last\_dec = now$
27: **end if**
28: **end if**
29: **end function**

Regarding the variables $cooldown$ (lines 6 and 14) in the algorithm they are used because each time *max_t_nodes* and *max_r_nodes* are changed the node will not be able to change these values in the next 2 hours to prevent fluctuations in these values. Furthermore, every time a node increases *max_t_nodes* and *max_r_nodes* it will not be able to decrease these values in the next 4 hours in order to prioritise resilience over performance.

3.5 Implementation

To evaluate our proposal we have resorted to simulations, given that it would have been impossible to setup a realistic experiment with large number of users in the timeframe available to complete this work. Unfortunately, although a number of Bitcoin simulators have been proposed in the past, they are no longer supported and, furthermore, they implement algorithms that do not match the current version of Bitcoin. Therefore, we have opted to implement our own simulator. In order to collect data useful to calibrate the simulator, we have also created a modified version of the standard Bitcoin client that has been enriched with a number of probes to collect statistics about the Bitcoin operation.

In this section, we start by describing the modification we did to the Bitcoin client to collect statistics and then we describe our simulator.

3.5.1 Modifying the Bitcoin Client

As we will describe in the next section, we opted to build a new simulator, that runs the most recent version of the Bitcoin protocols. In order to ensure that the simulator accurately captures the behaviour of the real system, we need to collect a number of statistics regarding the operation of the Bitcoin network such as:

- Percentage of duplicated advertisements;
- Percentage of transactions that needed to be requested;
- Percentage of duplicated blocks received;
- Percentage of blocks received by neighbour;
- Percentage of blocks where a node needed to request transactions

Although there are a few websites, such as `blochain.info` and `bitnodes.earn.com`, that publish some information regarding the operation of the Bitcoin network, these sites do not offer the level of detail required to calibrate the simulator. Therefore, we opted to create a modified version of the Bitcoin client, that was augmented with a number of sensors to collect the following information:

- Blocks received - hash, source, message type
- Advertisements for transactions - hash, source

- Transactions received - hash, source
- Transactions requested with *GetBlockTX* - hash, block hash

After studying the sources of the Bitcoin distribution, we did realise that it was possible to implement most of these sensor by introducing small changes to the *net_processing.cpp* file of the Bitcoin distribution (this file includes most of the logic associated the the Bitcoin broadcast protocol).

In order to collect the desired statistics, we have deployed to instances of the modified client connect to two different services providers (one using the academic network and another using one of the main Portuguese internet providers). We have ensured that the two clients did not establish a connection between them as that would skew the results. We have collected the statistics by running the two modified clients for more than one month, after discarding the results collected during an initial warm-up period of 7 days.

3.5.2 Implementing the Simulator

To the best of our knowledge, when we started this work, two Bitcoin simulators were available, namely `https://github.com/shadow/shadow-plugin-bitcoin` and `https://github.com/arthurgervais/Bitcoin-Simulator`. Unfortunately, these simulators did not implement the latest versions of Bitcoin, namely they did not simulate the use of compacts blocks. Furthermore, the simulators were no longer supported. Finally, these simulators include many mechanisms that are irrelevant to evaluate our contributions. Instead of attempting to update these simulators, we opted to implement our own simulator, that would be focused on simulation the aspects of Bitcoin that we aim at improving with our algorithm.

Our simulator is an event driven simulator, where the behaviour of each node is described by a deterministic state machine, that consumes events and, in response, produces events. Each event is tagged with a *timestamp*, that captures at which instant the event should occur, and a *target*, that specifies the node that should process the event. For instance, if a node n_1 receives an event e_1 at time t, and this event represents an message that needs to be relayed immediately to node n_2, the processing of event e_1 would generate another event e_2 targeted at node n_2 and scheduled to occur at time $t + \delta$, where δ is a function of the network latency between n_1 and n_2. The simulator was coded in Python2.7.

The simulator allows the user to configure a number of Bitcoin network parameters such as the number of nodes, the initial neighbourhood size of nodes and the number of miners. Based on these configuration parameters, the simulator automatically assigns an initial view of 8 neighbours to each node. The duration of a simulation is measured by the number of cycles that each node will perform; in or implementation one cycle is equal to one second. In each cycle, each node checks the following i) if it needs to run Algorithm 6, ii) if it should generate a new transaction, iii) if it should generate a block, and iv) finally if it should send any messages to any of its neighbours. As noted in Section 3.1, the simulation is configured to generate blocks at a rate of 1 block per 10 minutes and 2 transactions per second, in order to obtain 172 800 transactions per day (which is the average number of transactions per day registered in Bitcoin at the time of this writing).

It is worth noting that the current version of the simulation was the result of multiple iterations, namely because earlier version did not optimise the memory usage and, therefore, could not run simulation with very large numbers of nodes. We omit the description of these code optimisation's as they are orthogonal to the main contribution of the thesis.

3.6 Summary

This chapter presented a new transaction dissemination protocol that aims at reducing the amount of resources used by the Bitcoin network while still maintaining the resilience of the system. In order to do so, a new algorithm to rank neighbours was developed as well as an algorithm to bias the dissemination of transactions in the Bitcoin network. Furthermore, we also developed an algorithm that allows the dissemination protocol to adapt itself, in response to changes in the network.

4 Evaluation

In this chapter, we evaluate our approach. The evaluation was done using the simulator described in the previous chapter.

When evaluating our approach we took into consideration multiple metrics that we considered the most relevant for the proposed approach, namely:

- **Percentage of committed transactions** This metric measures the percentage of transactions committed during the simulation. This metric is important because since we are effectively lowering the number of nodes that know of a transaction we wanted to make sure that in our approach transactions would not get lost in the dissemination process. Hence this metric measures the resilience of the system, meaning that if our protocol is not able to commit the same amount of transactions as the base protocol then it is not resilient enough;
- **Commit time** This metric measures the commit time of transactions. The commit time is the time it takes for a transaction to appear in a block since it was created. This metric is also very susceptible to changes to the protocol especially changes like the ones we are proposing since they are extremely connected with the dissemination of information. Furthermore, making sure this metric does not cross critical values (30 minutes) is also essential for our approach to be even considered one day to be integrated into the Bitcoin client. Hence, this metric is one of the indicators of the performance of our protocol, our objective when measuring this metric is to make sure that at least we maintain the same commit time;
- **Amount of messages sent** This metric measures the amount of messages exchanged in the network. This metric is very important because it will determine if our approach brings anything of value to Bitcoin as this is the main metric we want to have an impact on by lowering it. This metric is the one where we want to observe the biggest improvement if our protocol works properly;
- **Amount of information sent** This metric measures the amount of information exchanged in the network. We did not expect an improvement in this metric as significative as in the metrics ”Amount of messages sent” because the size of the messages exchanged is small.

This chapter is divided into two sections, Section 4.1 that details how we tuned our simulator in order for it to be as faithful as possible to Bitcoin. Then in Section 4.2 we will go over the results we obtained and discuss them. Finally, Section 4.3 summarises the results obtained.

4.1 Simulator Tuning

To tune the simulator we used the metrics that we had gathered from websites like `blochain.info` and `bitnodes.earn.com` and from our modified client, but we also tuned the simulator experimentally by comparing results we were obtaining by running the base protocol with the real Bitcoin client. We considered that all nodes were running a version of the Bitcoin core client that implements compact blocks. Blocks are always relayed through compact block unless the neighbour does not possess the previous block. When a node receives a transaction it will always relay it unless that node is Byzantine. Nodes can not have more than 125 connections. When a node tries to establish a connection with another node, the other node will always accept it. The network model that we used in the experiments of Section 4.2 were composed solely of nodes that followed the protocol accordingly.

Due to the complexity of the protocol, simulating the full network resulted in resource intensive simulations that lasted for days. To overcome this, we scaled down the size of the simulated network as follows. First, we ran the original protocol with 6000 nodes and with 625 nodes and compared the metrics discussed below. The results we obtained were equivalent for both network sizes hence, for the rest of this section we consider a network size of 625 nodes. This proportional scaling between 6000 nodes (size registered when we started experimenting) and 625 nodes, allowed us to quickly explore different possible solutions and run multiple instances of each test. The results presented are an average of 3 independent runs that correspond to 34 hours in real time. We discarded the first and last 5 hours of each run in order to study the system in a stable state.

4.2 Results

We started by exploring the different possible solutions to reduce network usage without having a negative impact on the system. In all experiments below, we use the following notation: *Tn* where n specifies the value of the variable *max_t_nodes*; and *Rn* specifies the value of the variable *max_r_nodes* present in the previous algorithms. Note that for these experiments we did not used Algorithm 6 because we wanted to manually determine the best values for the aforementioned variables.

Hence, this section is divided into two subsections in the Section 4.2.1 we will present the results of the experiments and how we determined the best possible configuration for the variables *max_t_nodes*, *max_r_nodes* and ip. After in Section 4.2.2 we will present the results of running Agorithm 6 together with the rest of our solution and study its abilities to converge to the optimal configuration found manually.

4.2.1 Skewed Relay Impact

Initially we tested with multiple combinations of $n = 1, 2, 3, 4$ for both *T* and *R*. After these preliminary experiments, we observed that for values of $n = 3, 4$ the results were practically the same as the results without our approach. However, with $n = 1, 2$ we observed a considerable reduction in the number of duplicated advertisements. These results also support our logs in the real client, where the average number of duplicates was 6.6. With this in mind, for the rest of the experiments, we considered only

the combinations of: *T2_R2; T2_R1; T2_R0; T1_R1; T1_R0*. Additionally, for each configuration, we also experimented with both values of the variable *ip*.

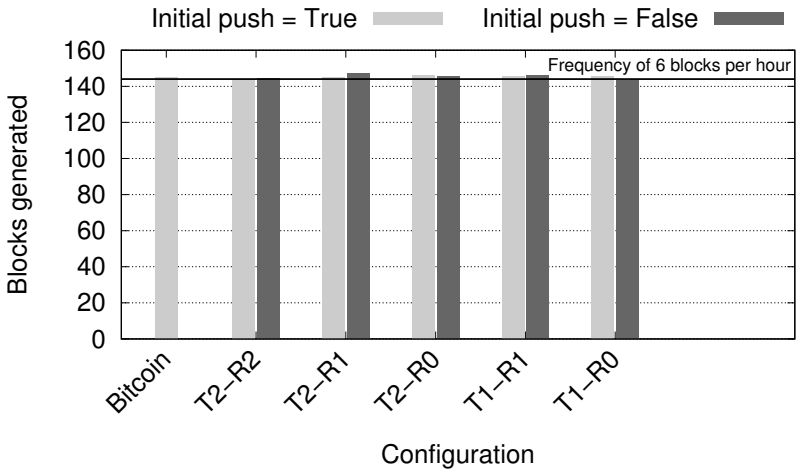

Figure 4.1: Blocks generated.

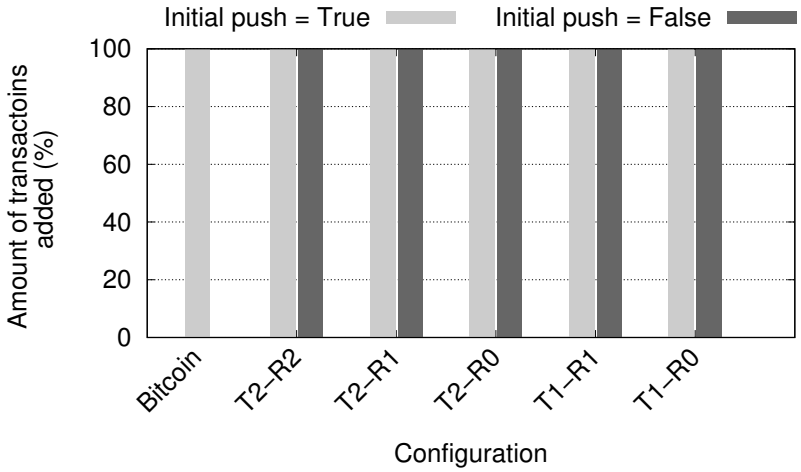

Figure 4.2: Percentage of transactions committed.

Figure 4.1 shows, for each configuration, the amount of blocks that were generated during each experiment, while Figure 4.2 shows the percentage of transactions added to blocks throughout the multiple configurations. As it is possible to observe, the simulation generated for all configurations the expected amount of blocks for a day (≈ 144) and committed all the created transactions (100%). From these results we can conclude two things. The first is that we configured our simulator properly as all simulations generated the expected amount of blocks per day, the second conclusion is that all the configurations tested were resilient enough to commit all the transactions generated. This means that both Algorithm 4 and Algorithm 5 are working properly as nodes are ranking neighbours properly and their transactions are reaching at least a miner which adds them to a block. However, we still need to check the performance of the metric commit time to draw conclusions on which is the best configuration.

Regarding the commit time Figure 4.3 shows the average transaction commit time for each configuration. The horizontal lines represent the highest and lowest average time it took for a transaction to be

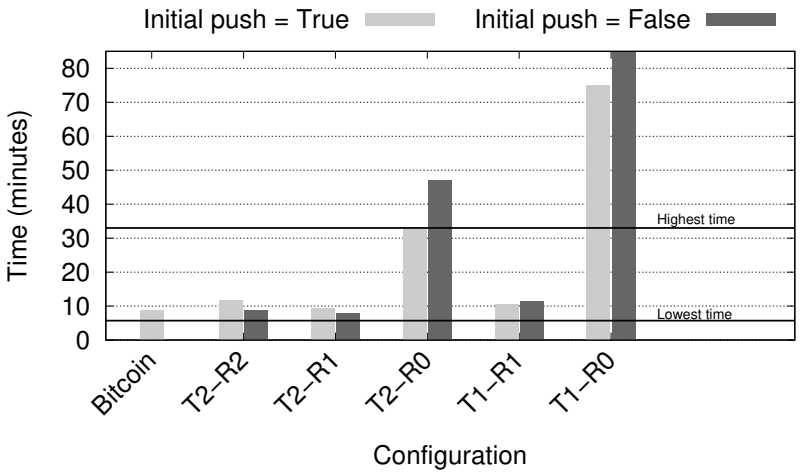

Figure 4.3: Average time it takes for a transaction to be committed.

committed in Bitcoin. In this image, we can clearly see that both configurations *T2-R0* and *T1-R0* are not good enough to achieve a commit time comparable to Bitcoin. This increase in the average commit time happens because although transactions are reaching miners they are not reaching the miner that is going to mine the next block. Another factor that contributes to this is that some cluster of nodes will all send their transactions to the same node. This shows that sending transactions for at least one random node alongside the top nodes has a great impact in the commit time as previously discussed in Section 3.3.

With this, both configurations are not eligible anymore for best configurations as a low commit time is one of the most important aspects of a cryptocurrency.

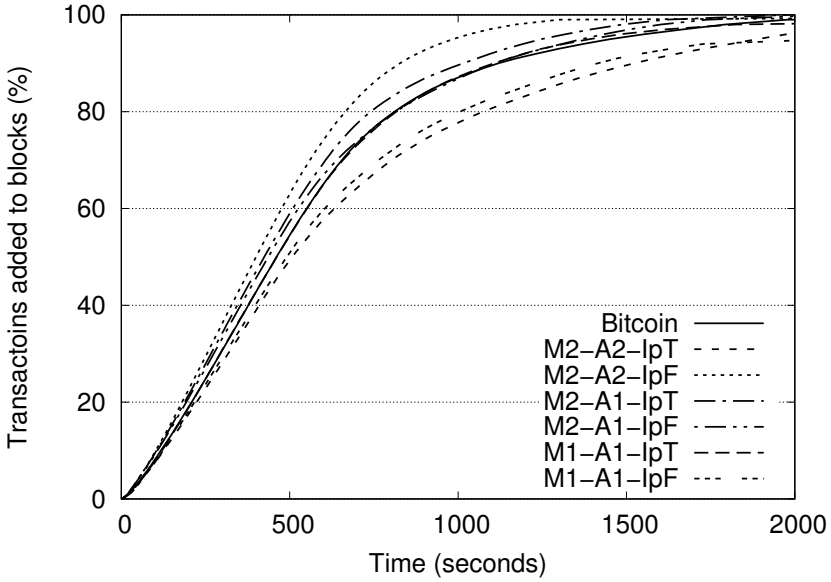

Figure 4.4: Cumulative distributed function of the time it takes for a transaction to be committed.

Figure 4.4 shows the cumulative distributed function of the time it took to commit all the transactions, hence, it is a different perspective of Figure 4.3. From both figuress, we can also conclude that sending

a new transaction to all the neighbours (*ip=T*) has a very low impact in the time it takes to commit a transaction. We attribute this to the fact that the time it takes for a transaction to reach all the miners is orders of magnitude (few seconds versus dozen of minutes) lower than the rate at which blocks are being generated. Therefore we can already exclude the configuration with (*ip=T*) as those configurations will always generate more messages than the other configurations without providing any major improvement in the other metrics.

With the impact of each configuration in the transaction commit time and number of transactions analysed, we now focus on the impact on reducing network usage.

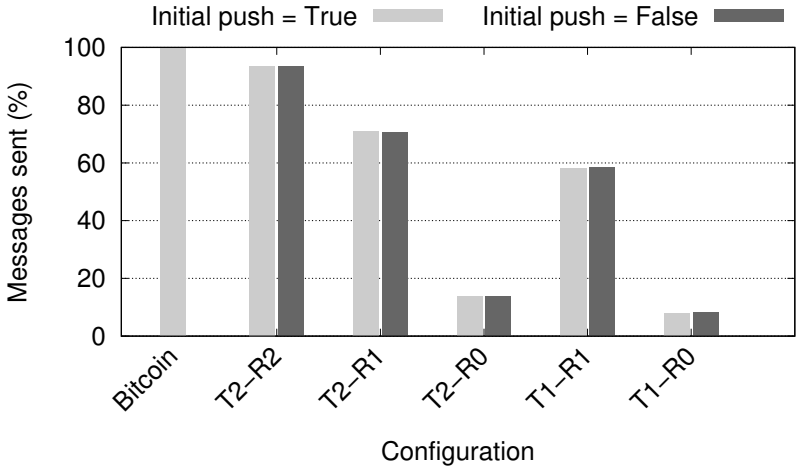

Figure 4.5: Total number of messages sent.

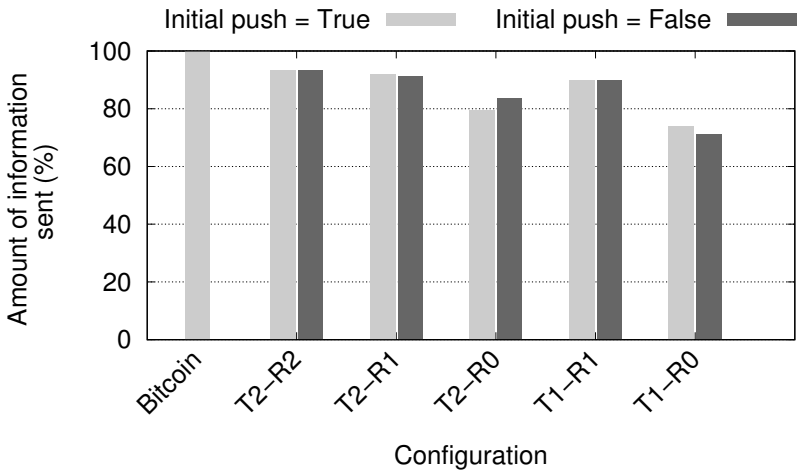

Figure 4.6: Amount of information sent.

Figure 4.5 shows ratio between the total number of messages sent in each configuration to the same amount in the Bitcoin network. As expected, the configurations with a higher amount of savings were the configurations that did not relay to random nodes. This happens because when we send a transaction to a random node there is a higher chance that that node still does not have that transaction and will request it. Unfortunately, as we have seen previously, both these configurations are not viable because both take very long to commit transactions. Figure 4.6 shows a different perspective by depicting the

savings in the total amount of information transmitted which, as expected, follows a similar pattern to Figure 4.5. We can also see that the savings from Figure 4.6 are not as big as the ones from Figure 4.5. This happens because the advertise messages that we avoid sending are not very big in size. However, processing those spurious incurs an additional cost to the nodes.

By analysing these results, we can conclude that the most promising configuration is *T1-R1* with *ip=False* because not only it achieves relevant savings (reduction in the number of messages sent in 41.5% and reduction of the amount of information sent in 10.2%) but also it preserves the properties of the original Bitcoin.

4.2.2 Adaptation

In the previous experiments, and to determine the best possible configuration we used a stable network, where miners were always the same nodes. However, as any large network, Bitcoin is prone to changes. We now study the adaption policy introduced in Algorithm 6. Initially, all nodes send advertisements to *T = neighbourhood / 2* and *R = neighbourhood / 2* which is the same as sending advertisements to all their neighbours as in the regular Bitcoin. Then throughout the simulation, the algorithm will progressively determine the best *Tn-Rn* configuration for each node by either increasing or decreasing both variables. We performed three experiments, one where we did not make any changes to the network, another where we change two miners in the network at 12 hours into the simulation and finally a third one where we changed all the miners at 12 hours into the simulation. With these experiments, we want to determine if our solution is able to adapt to the network changes and preserve the commit time while still sending as few messages as possible. These experiments were run for a simulation time of 58 hours in order so see if the system would stabilise after the miners being changed.

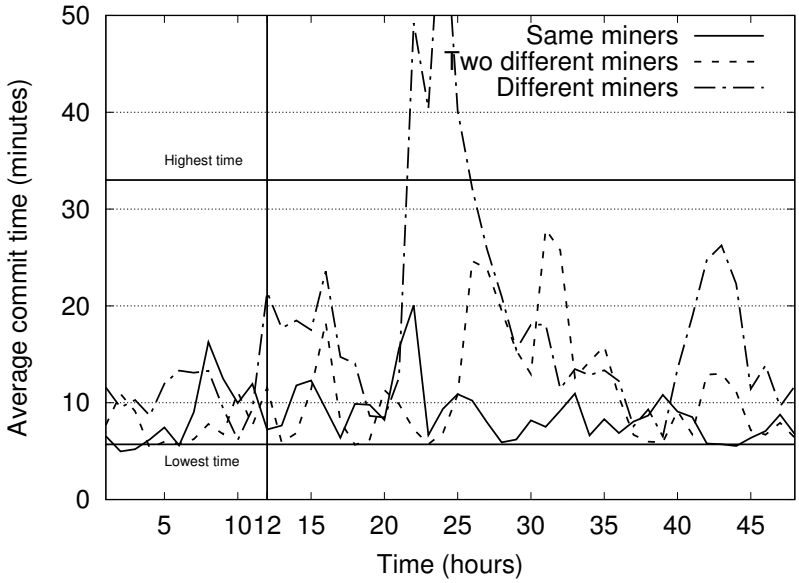

Figure 4.7: Average commit over time.

Figure 4.7 shows the average commit time over the period of time simulated. It also shows the two horizontal lines that were in Figure 4.3, delimiting the minimum and maximum observed Bitcoin commit

times.

This figure has three lines one for each of the configurations described previously. Starting with the line that represents the simulation where no miners were changed, we can see that for the majority of the time the commit time was pretty constants and that transactions were always committed in time. There is a moment where it peaked around 22 hours but that might have occurred due to some nodes lowering too much the size of *T and R*.

Regarding the line that represents the simulation where we changed two miners, it also behaves similarly to the previous line until around the 12th hour were then two miners were changed this produced a delayed peak between hours 14 and 17 and then at hours 24 to 35. The first peak was due to the change of miners and the second probably happened due to some nodes lowering too much the size of *T and R* which was then also aggravated by the change in miners. However is worth noting that both these peaks did not surpass the line of the highest time observed in Bitcoin.

Finally, for the line that represents the simulation where all the miners changed we can see that this line had a huge peak after the change of miners. This is because this scenario corresponds all current miners to stop mining and a new set of miners completely replacing them, which is very unlikely in practice. However, it is worth noting that in the end, all simulations started converging to the same commit that that we observed for the configuration *T1-R1* in Figure 4.3.

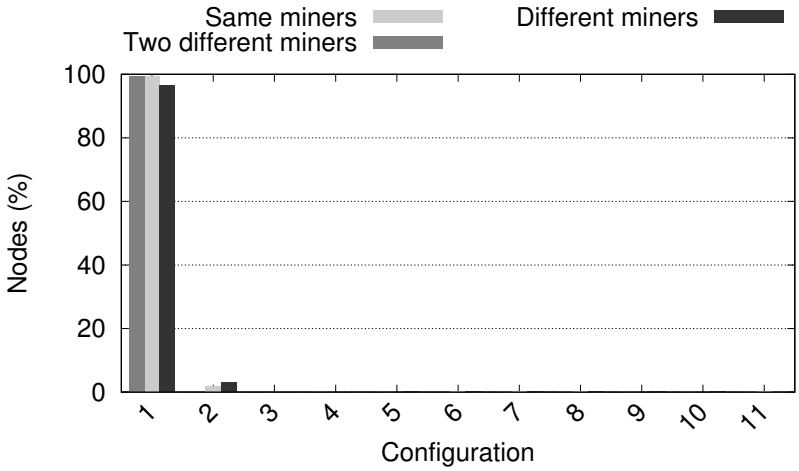

Figure 4.8: Distribution of the nodes by the different possible configurations.

Figure 4.8 displays the percentage of nodes in each configuration at the end of the simulation for the three runs. In the *y* axis is the percentage of nodes and in the *x* axis are the multiple configurations observed in the simulation, for instance we can see that for the simulation were the miners were always the same that at the end of the simulation almost 100% of the nodes were with a configuration *T1-R1* configuration 1 and that no nodes had the configuration *T11-R11* configuration 11. This image shows that in all three scenarios at the end almost all the nodes converged to the configuration that we had previously deemed ideal (*T1-R1*).

4.3 Summary

We can drawn several interesting conclusions from these experiments. First, we can see that if we are in the presence of a stable network, then our solution is going to start adapting to the network and converge to the configuration that we previously deemed ideal (*T1-R1*), as seen in Figure 4.8. Secondly, we can conclude that if there are slight changes to the network, then the average commit time of our solution is going to slightly deteriorate, but eventually the algorithm will increase the size of T and R to cope with the changes and the average commit time will once again converge the values before the changes. Finally, if our solution is confronted with drastic changes to the network it will not be able to maintain the current commit time, given that at 12 hours into the simulation most nodes were configured to *T1-R1* which is not resilient enough for these cases. We note however that such sudden shift in mining power is unlikely to happen. Regardless, after some time our approach will start converging to the desirable stable configuration.

This chapter presented the experimental evaluation of our protocol, detailing experiments conducted in order to assess which was the best configuration of the variables *max_t_nodes* and *max_r_nodes*. In the best case, our protocol is able to save 41.5% of the messages sent and 10.2% of the information sent by the traditional Bitcoin protocol. We have also shown that our protocol is able to sustain changes in the network while still maintaining the performance of the current Bitcoin protocol.

5 Conclusions

Despite the multiple iterations and improvements that have been done to the Bitcoin dissemination protocol since its introduction, there are still some aspects that need to be improved. As Bitcoin becomes more popular and new clients join the system, it is fundamental to have an efficient and robust dissemination substrate for the network to function properly. In this thesis, we addressed the problem of resources usage in the dissemination process.

We proposed a novel protocol that improves the existing transaction dissemination algorithm, that bias the dissemination to the neighbours that are more likely to reach miners quickly, since these nodes are the only ones that need to receive all the transactions before they are added to blocks. As we have shown in our experiments our protocol saves, in the best case scenario, 10.2% of the current bandwidth used and 41.5% of the number of messages exchanged without compromising robustness. Given the current size of the network, our changes results in savings, at the end of a day, of 1 640 000 messages and 610 GB. We have experimentally observed that these savings to not affect the performance of the Bitcoin network in other dimensions (for instance, it remains robust to node failures).

With these results, we think we were able to achieve our objectives as we were able to lower the resources consumed by the network without affecting resilience and commit time.

5.1 Future work

In this section, we will cover the different options we have available as future work.

Simulator performance We would like to improve the performance of our simulator in order to be able to run bigger simulations in a more timely fashion. One possible approach would be writing the simulator in a different language, such as C++, that can be compile into very efficient binaries.

Improve the current protocol Another interesting avenue for future work would be leveraging more detailed membership information to build even more efficient dissemination paths. Since we still have some fluctuations in the commit time that we would like to solve. An initial approach that we could try is to nodes together with *Addr* messages send also the information that our algorithm uses to choose which nodes to relay transactions to. With this information then when nodes were choosing an outgoing connection they could chose the nodes with better classification. However we would have to then evaluate how resilient would be the network that would emerge from using this approach and if we would have any improvements with this approach.

Bitcoin improvement proposal We would also like to integrate our algorithm into the default Bitcoin client and propose it as a Bitcoin improvement proposal. With this our protocol would be submitted to under review by the community and if approved it would be integrated in the client of Bitcoin core the most widely used Bitcoin client.

References

Apostolaki, M., A. Zohar, and L. Vanbever (2017). Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 375–392. IEEE.

Bitcoin.org (2008). Bitcoin wiki.

Bitcoin.org (2009). Bitcoin developer documentation.

Bojja Venkatakrishnan, S., G. Fanti, and P. Viswanath (2017). Dandelion: Redesigning the bitcoin network for anonymity. *Measurement and Analysis of Computing Systems, Proceedings of the ACM on 1*(1), 22.

Bortnikov, E., M. Gurevich, I. Keidar, G. Kliot, and A. Shraer (2009). Brahms: Byzantine resilient random membership sampling. *Computer Networks 53*(13), 2340–2359.

Buterin, V. (2014). Toward a 12-second block time.

Cohen, B. (2003). Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, Volume 6, pp. 68–72.

Corallo, M. (2016). Compact block relay.

Decker, C. and R. Wattenhofer (2013). Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pp. 1–10. IEEE.

Eugster, P. T., R. Guerraoui, A.-M. Kermarrec, and L. Massoulié (2004). Epidemic information dissemination in distributed systems. *Computer 37*(5), 60–67.

Eyal, I. and E. G. Sirer (2018). Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM 61*(7), 95–102.

Guerraoui, R., K. Huguenin, A.-M. Kermarrec, M. Monod, and S. Prusty (2010). Lifting: lightweight freerider-tracking in gossip. In *Middleware, Proceedings of the ACM/IFIP/USENIX 11th International Conference on*, pp. 313–333. Springer.

Heilman, E., A. Kendler, A. Zohar, and S. Goldberg (2015). Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pp. 129–144.

Jelasity, M., R. Guerraoui, A.-M. Kermarrec, and M. Van Steen (2004). The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Distributed Systems Platforms and Open Distributed Processing, ACM/IFIP/USENIX International Conference on*, pp. 79–98. Springer.

Jesi, G. P., D. Hales, and M. Van Steen (2007). Identifying malicious peers before it's too late: a decentralized secure peer sampling service. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO'07. First International Conference on*, pp. 237–246. IEEE.

Jesi, G. P. and A. Montresor (2009). Secure peer sampling service: the mosquito attack. In *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09. 18th IEEE International Workshops on*, pp. 134–139. IEEE.

Li, H. C., A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin (2006). Bar gossip. In *Operating systems design and implementation, Proceedings of the 7th symposium on*, pp. 191–204. USENIX Association.

Miller, A., J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee (2015). Discovering bitcoin's public topology and influential nodes. *et al.*.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Owenson, G., M. Adda, et al. (2017). Proximity awareness approach to enhance propagation delay on the bitcoin peer-to-peer network. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pp. 2411–2416. IEEE.

Raposo, D., M. L. Pardal, L. Rodrigues, and M. Correia (2016). Machete: multi-path communication for security. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pp. 60–67. IEEE.