Bitshield

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. X
Supervisor:	Prof. Y
Member of the Committee:	Prof. Z

October 2018

Agradecimentos

Gostava de agradecer a

Acronyms

P2P Peer to Peer. 12, 13

PoW Proof of Work. 15, 16

Resumo

Gostava de agradecer a

Abstract

Gostava de agradecer a

Contents

1 Introduction

Cryptocurrencies like Bitcoin and others gained a lot of exposure in the previous year, which has led to their popularization with an increase in average daily users. As a consequence of this increase, there has also been an increase in the resources spent by cryptocurrencies.

This thesis addresses two problems, the immense amount of resources spent on disseminating information and BLANK (MEMBERSHIP). In particular, we propose a new algorithm for the dissemination of transactions as well as an improvement in the current BLANK (MEMBERSHIP)

1.1 Motivation

Cryptocurrencies and their associated mechanisms have gained an increasing relevance in recent years. 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]. Moreover, the main technology behind Bitcoin, the Blockchain, has emerged as useful for a myriad of other applications, from birth, wedding, and death certificates to the monetization of music[2].

For instance in Bitcoin, in an attack known as the partition attack, an Autonomous System (AS) can isolate a partition of the network to make it vulnerable to double spending (see (Apostolaki, Zohar, and Vanbever 2016)). Attacks such as this are increasingly more appealing due to the significant financial advantage that an attacker can gain, if successful. It is therefore of the utmost importance to the future of cryptocurrencies to design mechanisms that eliminate, or at least mitigate the possibility of such attacks without sacrificing other important aspects of cryptocurrencies.

We are particularly interested in studying the protocols that support information dissemination in blockchain networks, and their vulnerabilities, as these are the backbone that allows cryptocurrencies to function properly. Even though cryptocurrencies have attracted lots of attention from the community on higher level subjects such as the consensus protocol used, few works explore the subject of information dissemination (Miller, Litton, Pachulski, Gupta, Levin, Spring, and Bhattacharjee 2015; **?**; **?**; **?**).

{TODO: Check from here} In this work, we aim to address this problem by: i) doing a principled analysis of information dissemination vulnerabilities in the Bitcoin network and ii) proposing an extensible architecture that eliminates or mitigates these vulnerabilities. We consider attackers with varying resources, from a rogue Autonomous System to a set of colluding individuals that target a specific victim, and different motivations in the system. To address these, we look at techniques that have been

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

proposed in distributed systems literature such as overlay networks (Jesi and Montresor 2009) or node behaviour (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006).

Leveraging these techniques, we propose BitShield, an extensible architecture that aims to harden information propagation in cryptocurrency networks. More precisely, we focus on Bitcoin, but as we will see later the approach can be applied to other cryptocurrencies as well. The reason for choosing Bitcoin is that it is the most popular cryptocurrency and its core design is shared by many other cryptocurrencies, hence improvements over Bitcoin could be easily applied in other systems.

A maioria das criptomodedas existentes, e em particular a *Bitcoin* (a criptomoeda mais usada correntemente), mantêm um livro-razão descentralizado que regista a sequência ordenada de todas as transacções executadas no sistema (Nakamoto 2008). A manutenção de um livro-razão de forma descentralizada e aberta tem vindo a ser reconhecida como uma abstracção útil para várias aplicações, para além do uso original como meio de pagamento. Por exemplo, livros-razão distribuídos podem ser usados para registar todo o tipo de transacções, contratos, ou outros actos que tipicamente requerem o uso de notários.[3]

Um livro-razão distribuído é tipicamente mantido da seguinte forma. Em primeiro lugar, e por razões de eficiência, agrupam-se várias operações que são registadas em conjunto; estes grupos de transações são designadas por *blocos*. Posteriormente, os blocos são organizados numa lista ligada, também designada por *cadeia*. Na literatura anglo-saxónica, esta cadeira de blocos é simplesmente designada por *blockchain*. Um dos aspectos interessantes de criptomoedas como o Bitcoin consiste no facto dos nós participantes terem um mecanismo de filiação aberto e descentralizado. Isto é, nenhum nó no sistema necessita de conhecer todos os outros nós do sistema e qualquer nó pode juntar-se ou sair do sistema que o protocolo garante a coerência da cadeia gerada mesmo que alguns nós possam ter um comportamento racional ou bizantino.

Tipicamente, os sistemas de manutenção de cadeias de blocos funcionam da seguinte forma. Os nós do sistema, concorrentemente, recebem transacções que posteriormente distribuem pelos restantes nós. De forma também concorrente, os nós tentam criar o próximo bloco da cadeia, o que envolve a resolução de um puzzle criptográfico computacionalmente exigente. Assim que um nó gera um bloco, distribuí-o pela rede, o que leva ao cancelamento da geração de blocos concorrentes e dá inicio à criação de um novo bloco. Antes de aceitarem (e re-distribuirem) um novo bloco, os nós da rede validam que o bloco é bem formado e constituído por transacções válidas.

Pela breve descrição acima, é fácil depreender que o processo de propagação de transacções é central na operação dos livro-razão distribuídos. Em primeiro lugar, as transacções necessitam de chegar aos nós que estão a criar blocos, de modo a poderem ser inseridas na cadeia. Em segundo lugar, necessitam também de ser conhecidas pelo restantes nós, pois são necessárias para validar a correcção dos blocos gerados. Na *Bitcoin*, a propagação de transacções baseia-se numa estratégia em que os nós anunciam periodicamente aos vizinhos quais as transacções que possuem. Estes, após receberem anúncios de transações em falta solicitam-nas aos primeiros. Este processo gera uma redundância de mensagens de anúncios. Apesar desta redundância ser necessária para tolerância a faltas, experimentalmente observámos que cada nó recebe um número excessivo de anúncios duplicados para cada transação no sistema.

[3] Para uma lista de exemplos, consultar `https://blockgeeks.com/guides/blockchain-applications/`

1.2 Contributions

This thesis contributes with an updated description of how the Bitcoin protocol disseminates information and builds its network. It also introduces two improvements over the current version of Bitcoin and alike Blockchain technologies:

- An amendment to the dissemination mechanism able to reduce the amount of bandwidth spent and the number of messages sent without compromising the resilience of the current system;
- An improvement to the membership mechanism:

In addition, it provides an experimental evaluation of the performance of both improvements based on simulations. Finally, it also details the implementation of a simulator where we evaluated our solutions.

1.3 Results

The results produced by this thesis can be enumerated as follows:

- A specification for a new dissemination protocol
- A specification for a new membership system
- Implementation of the proposed improvements
- Experimental evaluation of the proposed systems

1.4 Structure of the Document

The remaining of this document is organized as follows. Chapter 2 provides an introduction to Bitcoin and its systems for information dissemination and membership protocol, it also covers some attacks to the cryptocurrency and problems related with Peer to Peer systems. Chapter 3 describes the assumed network structure and characteristics and then describes the proposed improvements implemented. Chapter 4 presents the results of the experimental evaluation of BitShield's improvements. Lastly, Chapter 5 concludes the document by summarizing its main points and discussing future work planned.

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. It starts by covering the common basics of Bitcoin and Blochain technologies, along with Bitcoins mecanisms for information dissemination. Afterwards it surveys previous work devolped on Bitcoin vulnerabilities. Finaly it covers some work on peer-to peer systems, like secure overlays and node behaviour.

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 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 a 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 decentralized, and the blockchain is maintained cooperatively by multiple servers.

In the rest of this section, we provide a more detailed description of several components 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. Finally, in Section 2.1.4, we describe the operation of the network of nodes that maintain the blockchain.

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 report. 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 serialized 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 of an account are transactions where the account appears in the array of outputs of those transactions, meaning that the owner of that account was paid the amount of Bitcoins that he now owns.

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 Fig. 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.

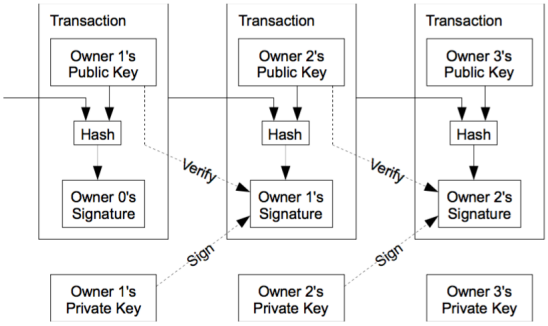


Figure 2.1: Signing mechanism. Original from (Nakamoto 2008)

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.

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 because transactions reach different nodes at different times:

- A node could receive a transaction that transfers Bitcoins from owner *B* to owner *C* but that node has yet to receive the transaction that transferred the Bitcoins from owner *A* to owner *B*. Which might result in either the transaction not being accepted or 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 sent his transaction to node *A*, *A* would accept it because in its record the user had those Bitcoins to spend. But if the user had sent his transaction to node *B*, *B* might not have accepted it because in its record the user 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 commit transactions in a different order, as seen previously, 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 summarizes 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. 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. Due to the very low probability of successful generation, it's unpredictable which miner nodes in the network will be able to generate the next block.

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 2.2.2.

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.

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 those that were not, they will be re-applied only if they are valid. 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 over the network 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 Fig. 2.2 this chain of blocks is called *blockchain* (Decker and Wattenhofer 2013). The first block in the chain is called the genesis block.

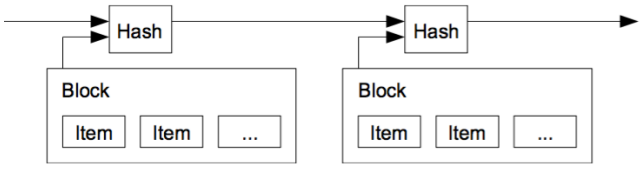


Figure 2.2: Blockchain representation. Original from (Nakamoto 2008)

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). Which is very hard to do because they would have to re-find all the previous PoW and a new PoW while competing against the rest of computational power present in the network.

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 discuss this in section 2.2.2.

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. This will go on until one of the branches overtakes the other (Decker and Wattenhofer 2013).

When a node has as blockchain head the 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'* then the node only has to apply all transactions in the intermediate blocks incrementally, if any, and then apply the transactions in *b'*,
- If *b* is on another branch 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 sequentially the end of the blockchain 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 else they will not be affected.

2.1.4 Network

Bitcoin is supported by a network of peer-to-peer nodes that mine new Bitcoins, create and disseminate transactions and keep a record of what transactions happened. This network has two components that should be looked into to understand some of the vulnerabilities that we will study in the next section. Those two components are:

1. Network overlay - how the Bitcoin peer-to-peer network is built;
2. Information propagation - how does the Bitcoin protocol forward information.

Network overlay

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 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.

By default a node connects to a minimum of 8 outbound peers (connections initiated by this node) and allows up to 125 inbound peers (connections initiated by other nodes). This limitations exist to prevent nodes from being isolated and to prevent nodes from becoming essential to the system. For instance, if a node was responsible for connecting all the nodes of Europe with all ones from Asia if that node failed the system would collapse.

{TODO: This part has to be reworked} Once connected to the network nodes exchange among them the addresses of other nodes, using the *getaddr* and *addr* messages, to maintain the network overlay up to date. Usually, an *addr* message is sent in response to a *getaddr*. However, the *addr* message may also arrive unsolicited, because nodes advertise addresses gratuitously when they (Bitcoin.org 2008):

- Relay addresses - once a node adds to its list of neighbours a newly received IP addresses it may relay it to other nodes if certain conditions are met;
- Advertise their own address - every 24 hours, the node advertises its own address to all connected nodes;
- Establish a connection;

In order to maintain a connection with a peer, nodes by default will send a message to peers before 30 minutes of inactivity. If 90 minutes pass without a message being received by a peer, the node will assume that connection has closed (Bitcoin.org 2009).

There are also organizations called *mining pools*, composed by multiple nodes that work together to find the *PoW* more efficiently. In mining pools each node test different nonces to find the *PoW* which is faster than a single node testing all the possible nonces. *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.

Information propagation

In order to exchange messages a Bitcoin node has to each of its neighbours, a message queue with messages that will be sent once a timer associated with each queue ends.

Transactions The usual way transactions are relayed to other nodes works as follows. Node *A* creates a new transaction, *A* will add the advertisement for that transaction to the message queue of all his neighbours. If *A* learns that one of it's neighbours already has that transaction, it will remove the advertisement for that transaction from the queue of that neighbour. Once a neighbour *B* receives the advertisement it will check if it already has that transaction and if it does not it will sent to *A* a *GetData* requesting the new data. Node *A* will then send the requested data individually meaning that if it receives a request for ten transactions it will send ten separate messages. 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.

Transactions can also be sent unsolicited.

Blocks Until Bitcoin version 0.13.0 (23-08-2016) the usual way to relay blocks was similar to the way transactions are relayed. However with the addition of new message compact blocks this changed. Although blocks can still be relayed through advertisements the main method for relaying a block between uptodate nodes is through compact blocks. The difference between the two methods of relaying is that, through advertisements once a node *A* sends a *Block* message replying to a *GetData* message, the message *Block* has all the contents of the block which gives to the receiving node the ability of fully validating the block. In the case of a *CMPCTBLOCK* message, *A* will only send crucial information to *B* so that it is able to reconstruct the block, this means that some information like transaction are summarise and only sent their id's. This process of relaying blocks is an improvement over the previous method because it reduces the amount of bandwidth consumed when sending blocks. If for instance a node does not have all the transactions necessary to rebuild a block it will send a *GetBlockTX* message to the node that sent him the compact block. Once a node receives a *GetBlockTX* it will respond with a *BlockTX* message containing all the missing transactions.

This way there are in total tree ways a block can be relayed as we can see in Fig.

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 because it speeds 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, they can use these messages to ask their neighbours for the missing block (Bitcoin.org 2009).

2.1.5 Summary

In conclusion, Bitcoin is made up of multiple modules as seen in Fig. 2.3. We will briefly describe them (Bitcoin.org 2008).

Txs and *Blocks* represent transactions and blocks respectively.

Mempool is the place where the node stores the transactions that are going to be included in next blocks. The *Validation Engine* is in charge of validating the transactions/blocks that are received. The *Miner* is the module responsible for mining blocks. Finally, the *Storage Engine* is the module that manages all the databases where a node stores all the relevant data like *Blocks*, *Blocks Headers* and *Coins*.

The *Wallet* is the module where the Bitcoins of the user are stored and the *RPC* module is used so that applications can interact with Bitcoin, hence providing an API to the outside.

The modules that we are more interested in are the *Peer Discovery* and the *Connection Manager* because of their connection to the network aspect of Bitcoin. The *Peer Discovery* is responsible for building the Network and maintaining it. The *Connection Manager* is accountable for managing the way data and control messages are broadcasted.

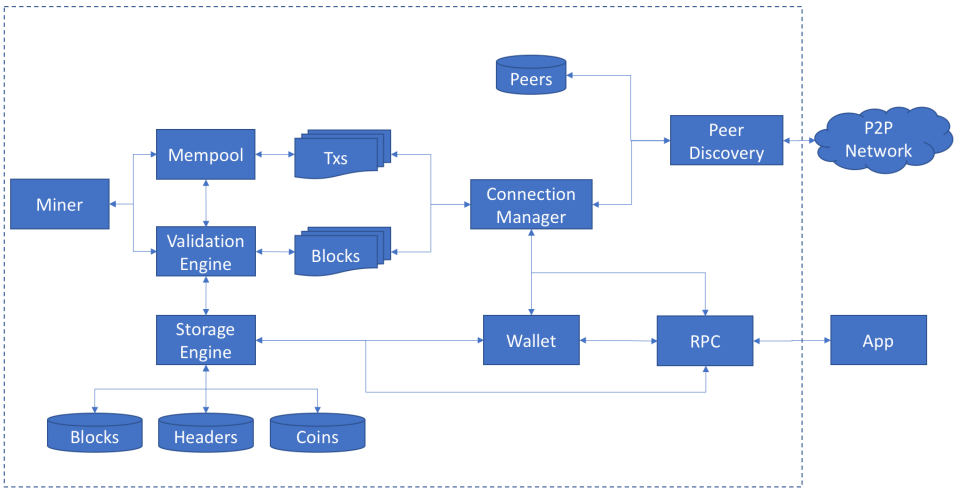


Figure 2.3: Bitcoin architecture (Source Eric Lombrozo)

Next, we will take a look at some of the vulnerabilities of Bitcoin and compare it with other cryptocurrencies 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 look at known vulnerabilities of Bitcoin, as well as, look into the solutions proposed to those vulnerabilities. We will also compare Bitcoin with other cryptocurrencies to see where Bitcoin stands in comparison to the state of the art.

This section is divided into four sections, one for each of the main vulnerabilities we have identified.

2.2.1 Information Eclipsing

This vulnerability was found in *Information Propagation in the Bitcoin Network* (Decker and Wattenhofer 2013) and it happens because of following behavior. Lets consider that the whole network recognizes 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"*.

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 to reach 50% of nodes, 40 seconds for it to reach 95% of nodes and that the mean delay for a block to be 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 to reach a node, so in mean 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 time would have to go down. But if we maintain the 12.6 propagation delay and decrease the block 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.

Regarding solutions, the authors suggested multiple options to optimize the propagation of a block, which would lower the 12.6 seconds delay and the lower probability of stale blocks being created. One of the solutions consisted in changing the block verification process. In this solution, the verification of a block would be split into two phases an initial difficulty check and a transaction validation. Since the transaction validation is the phase which takes longer, a node would perform the initial difficulty check where it checks the validity of the *PoW* and once this was done the node could broadcast the *inv* messages right away while performing the transaction validations asynchronously.

Ethereum solved part of this problem through the implementation of uncle blocks. Rather than discarding stale blocks as in Bitcoin, Ethereum includes them in new blocks, hence the name uncle blocks. Uncle blocks are included in new blocks up to a certain height difference between the uncle block and the new block being generated. This happens because nodes that generated uncle blocks are rewarded with a fraction of the full reward, so the limit imposed by the height difference avoids having a group of nodes mining on lower heights.

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

With this solution Ethereum solved this problem because now blocks to be built also include other previously built blocks in addition to the already included parent block. This means that no computational power is lost. Because for an attacker to forge a block he would need to remake all previous blocks of the main chain and all the uncle blocks of that block, since uncle blocks also had the transaction that the attacker is trying to eliminate. This attack is further explained in the next section.

The disadvantage of this solution is the extra computational power and memory required to include the uncle blocks.

2.2.2 51% Attack

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 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.

Other coins affected by this problem tried to solve the problem in different ways. Ethereum for instance, started using uncle blocks as we have seen in the previous section. This solution does not solve the problem because an attacker could still gather enough computational power to remake all the uncle blocks and blocks in the main branch. But it makes it harder for the attacker to perform the attack as it described in Section 2.2.1.

Ripple is also not affected by this attack because in ripple consensus is not connected with computational power. In ripple, the consensus is achieved by a set of trusted nodes called validators. Validators are chosen based on the expectation they will not collude in a coordinated effort to falsify data relayed to the network. A disadvantage of this approach is that it removes decentralization characteristics of the

cryptocurrency. In Bitcoin, everyone “votes” when they start trying to mine on a block in Ripple power of “voting” is removed from the users.

2.2.3 Partition Attack

This attack allows an adversary 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 2016).

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 isolated. 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. This aspect will be discussed in more detail in Section 2.3.3.

Regarding solutions for this attack in (Apostolaki, Zohar, and Vanbever 2016) 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. Still, a module that can be changed in Bitcoin to address this problem is the *Peer Discovery*. Because this module should establish more resilient connections.

In Ethereum and Ripple, this attack is not possible as the connections between nodes are encrypted which would make impossible the analysis of traffic that the AS would need perform. But they sacrifice performance.

2.2.4 Delay Attack

The objective of this attack is to slow down the propagation of new blocks sent to a set of nodes without disrupting their connections (Apostolaki, Zohar, and Vanbever 2016).

The mentioned attack 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 simple node then this attack will be a Denial of Service and that node will not have guarantees for double spending if that attack is towards a gateway of a pool it could be used to engineer block races.

This attack is successful because Bitcoin connections are not tamper proof and Bitcoin does not requests the block again when presented with a corrupted block, this is a problem related to the *Connection Manager* module.

A solution for this attack is monitoring the RTT, as the RTT increases considerably when the node is being attacked. Other solution proposed in (Apostolaki, Zohar, and Vanbever 2016) is encrypting Bitcoin communications which would not avoid the packets from being dropped by an attacker but it would prevent them from eavesdroppping and tampering with connections. The disadvantage with this approach is that it would require additional computations making the system slower.

Ethereum and Ripple are not affected by this attack as all the connections are encrypted but the attacker can still drop packets. But they sacrifice performance.

2.2.5 Summary

We have analyzed several attacks that can be performed against Bitcoin. Table 2.1 summarizes these attacks for Bitcoin but also for other popular cryptocurrencies. For some attacks, we were unable to assess if the attack was really possible due to some undocumented parts of the cryptocurrencies. These are noted in the table as "Might be affected by the attack".

We can see in Table 2.1 that all the cryptocurrencies related with Bitcoin like *Bitcoin Cash* and *Bitcoin Gold* suffer from the same vulnerabilities as Bitcoin. Whereas all cryptocurrencies not relatable with Bitcoin have for the majority of the attacks found a solution for them or at least a minor fix. This happens because unlike those cryptocurrencies Bitcoin does not have an organization that can make decisions on its own regarding the direction of the cryptocurrencies, as seen recently by the Segweit agreement. So when decisions have to made it takes a lot of time for the community to reach a consensus and sometimes these decisions even lead to forks of the cryptocurrencies which decreases the value of the cryptocurrencies.

It is also worth noticing that although cryptocurrencies not relatable with Bitcoin have solutions for the vulnerabilities described they had to sacrifice other characteristics. Ethereum for instance sacrifices performance because all its connections are encrypted and uncle blocks are used. Ripple has pour privacy features and also sacrifices performance because its connections are also encrypted.

2.3 Related Problems

In this section we will take a look at some problems related with network membership and node behaviour. These problems have been the focus of many studies because they are important for a good and sustainable peer-to-peer network.

Bitcoin is supported by a peer-to-peer network where is expected that all nodes have a random sample of neighbours and disseminate information as soon as they receive it. But as we have seen in the previous sections this does not always happen with attacks being performed at the network level like the partition attack. Furthermore, as we will see in Section 2.3.3, even some nodes inside the network do not behave properly and try to take advantage of high number of connections or selfish behaviour. However, all these problems are not new in the distributed systems field and have been the aim of a lot of research. Hence, is in our interest to look into how these problems affected other peer-to-peer technologies and study the solutions proposed to them.

We will start by looking into secure overlays in Section 2.3.1, where we will explain what secure overlays are, present some secure overlays vulnerabilities and present some approaches to building secure overlays.

In Section 2.3.2, will take a look at the different possible behaviours a node can have in the network, as well as, look into approaches that punish undesirable behaviours.

Finally, in Section 2.3.3, we will discuss how these problems affect Bitcoin.

2.3.1 Secure Overlays

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 is infeasible for a peer to know the overlay of the whole network, not only because of the size these networks achieve 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 systems (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 peer sampling 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 the complete defeat of the system, if the malicious nodes simply disappear after having gained such leading position.

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 that some peer-to-peer overlays like Bitcoin might have is **Supernodes**. Despite **Supernodes** not being desirable in Bitcoin in some peer-to-peer architectures like Skype they are actually desirable and contribute to the performance of the system. Supernodes are nodes that have a number of connections higher than the average number of connections. This is a problem because these nodes start having an important role in the network and if for instance one of these nodes leaves the network the system might crash.

We will now take a look at some overlays technologies.

Secure Peer Sampling Service: The Mosquito Attack (Jesi and Montresor 2009) This system was designed to extend the SPS functionality protecting it against the mosquito attack. 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 peer. 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.

Brahms (Bortnikov, Gurevich, Keidar, Kliot, and Shraer 2009) This system was designed to provide 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.

Identifying Malicious Peers Before It's Too Late: A Decentralized Secure Peer Sampling Service (Jesi, Hales, and Van Steen 2007) This system was built to cope with malicious nodes executing "hub attacks". 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 snapshot 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.

MACHETE (Raposo, Pardal, Rodrigues, and Correia 2016) This system was designed to establish an overlay network and scatter data over the available paths, thus reducing the effectiveness of snooping attacks. 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. This system is interesting to us because of its feature of sending data through multiple paths, something that if applied correctly to Bitcoin could protect against certain attacks as we will see in Section 3.1. This tool is not like the other PSS presented because it provides every client with a full view of the network instead of a partial view.

2.3.2 Node's Behaviour

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; **?**).

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. This will be further discussed in Section 2.3.3.

Next, we discuss how some approaches deal with some of these possible behaviours.

BitTorrent (Cohen 2003) The approach described in this paper was conceived to cope with free riders in a file sharing network. 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. This system 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.

BAR gossip (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006) This tool was designed with the objective of being the first streaming application that guaranteeing 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. This system tries to prevent free riders by having two protocols to disseminate information: Balanced Exchange and Optimistic Push.

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.

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

LiFTinG (Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010) This protocol was the first 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*.

2.3.3 Discussion

As we have covered in the introduction of this section, the problems described previously also affect Bitcoin. But given the difference between Bitcoin and the usual peer-to-peer technologies the solutions that we just covered cannot be implemented directly.

So in this section, our objective is to cover each problem presented previously and explain how they transfer to Bitcoin and the implications they have.

Mosquito attack While analyzing the Bitcoin protocol to disseminate addresses, we found that we might be able to reproduce this attack in Bitcoin.

The way this attack would be reproduced in Bitcoin is through the DoS prevention system that Bitcoin has. In this DoS prevention system, once node *A* receives from node *B* more than 1000 IPs (max number of address entries 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.

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.

The module that allows this attack to be possible is not only the *Connection Manager* because it is what enforces the DoS prevention system but also the lack of authentication in the messages sent through the network.

Supernodes 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 on the network and have an 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 centralizing 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.

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 supernode will be disseminated much faster through the whole network hence, decreasing the probability of forks happening. So once we implement a solution for supernodes we will have to evaluate if the advantages of not having supernodes compensate the disadvantages.

Rational Node's In Bitcoin, *rational* or *selfish* nodes are nodes that do not broadcast a block right after its discovery. 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.

	Bitcoin bitcoin.org	Ethereum ethereum.org	Bitcoin Cash bitcoincash.org	Ripple ripple.com	Dash dash.org	Bitcoin Gold bitcoingold.org
51% Attack				✓		
Information Eclipsing		✓		✓	◯	
Partition Attack		✓		✓	✓	
Delay Attack		✓		✓	◯	

Table 2.1: Cryptocurrencies and their respective problems.
✓ - Not affected by the attack
◯ - Might be affected by the attack

3 Bitshield

3.1 Architecture

Given the problems presented beforehand in this section, our goal is to build a set of extensions to Bitcoin and blockchain technologies to overcome these threats. We have chosen Bitcoin to implement these extensions because it not only has a big user base but it also well documented and has been the focus of many research articles. Whereas other cryptocurrencies do not usually have proper documentation of the protocols they use and their code. With this lack of information, it would be difficult to identify the key modules that we would need to modify or improve to create these extensions for those cryptocurrencies.

As we have seen Bitcoin still has a lot of problems linked to the network protocol and the overlay used, that the other cryptocurrencies do not have. It is worth noticing that those cryptocurrencies like Ripple, for instance, sacrifice other properties in exchange for being protected against those vulnerabilities, for instance, connections in Ripple are encrypted which imposes an overhead on the nodes. Therefore, in this work we propose to extend Bitcoin architecture with a module, which we call BitShield, to address these problems while also having a low impact on performance. The architecture is presented in Fig. 3.1.

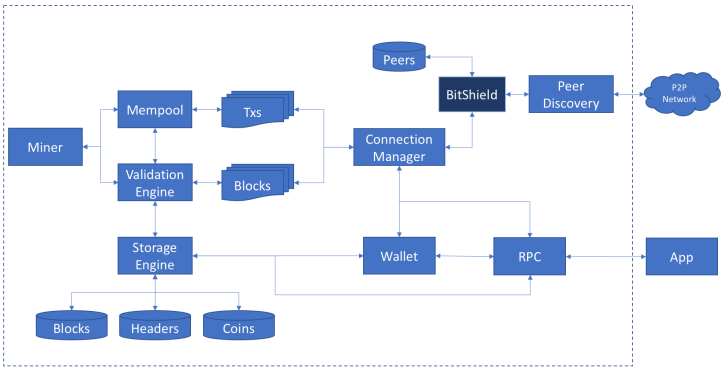


Figure 3.1: BitShield architecture

BitShield is positioned between the *Connection Manager* and the *Peer Discovery* because those are the modules we have identified as the cause of the problems Bitcoin has.

In the rest of this section, we will cover each vulnerability individually and propose possible approaches that will help Bitcoin overcome these vulnerabilities. In Section 3.1.1, we will propose possible

approaches to vulnerabilities that attackers can use to exploit Bitcoin. Whereas in Section 3.1.2, we will propose possible strategies to cope with behaviours that nodes inside the network might have to exploit Bitcoin and increase profit.

3.1.1 External vulnerabilities

Information Eclipsing Our approach to this problem is to implement the optimization in the propagation discussed in Section 2.2.1. To achieve this, when a node receives a block the verification of a block is split into two phases an initial difficulty check and a transaction validation. After the node validating that the block is valid through the first validation phase, the node would asynchronously broadcast *inv* messages to his neighbours while performing the transaction validation. This will decrease the mean time it takes for a block to reach a node hence reducing the probability of forks happening. The drawback of this solution is that it only lowers the probability this problem happening.

It would also be interesting implementing the solution Ethereum used to solve this problem, *Uncle Blocks* mentioned in Section 2.2.1. Where nodes include uncle blocks in future blocks strengthening the chain. They also reward nodes that found uncle blocks and nodes that add uncle blocks to their blocks which would compensate nodes that had their computational power wasted in a block that was not accepted. The problem with this approach is that it would have an increase in the block size and it would require more computational power from the nodes.

Partition attack This attack is hard to protect against because the problem is not only related with Bitcoin but also with the protocol used by the ASes.

Hence, our aim is to use MACHETE and also make nodes establish some extra connections. In this approach, a node would have to establish extra connections while having into account the route that the packets were going to have then, nodes would also send certain key messages (like the *inv* for instance) through multiple paths using the multipath approach of MACHETE. Nodes will use a tool like *traceroute*[1] to identify the best routes to avoid packets from passing always through the same AS. With this approach, a node is connected to the Bitcoin network through multiple AS and is able to communicate with different points of the network preventing the attack. A drawback of this approach would be the extra overweight imposed on the nodes to create the extra connections and send the extra messages.

Delay attack This attack can be done in two different directions. Our approach is to address the problem in both directions as follows. Since this attack is targeted at a single connection that a node has with a neighbour of his. We will use a strategy like MACHETE and send multiple multiple block requests to different neighbours. With this strategy the attacker has to intercept more connections, this strategy also prevents the attack in both directions.

This solution might add overweight to the network because more request messages will be traversing the network. But if we compare the computational power wasted by the victim when being attacked

[1] Diagnostic tool for displaying the route and measuring transit delays of packets across an Internet Protocol (IP) network.

with our solution, we believe that our solution will be better. Because not only are the messages small in size but also the victim will not be wasting computational power that could be used for strengthening the blockchain.

We also want to modify the behaviour of nodes in the $N \rightarrow V$ direction. Where once a node receives a corrupted block instead of staying idle it requests the block again. If the node fails 3 times to receive the block form that neighbour it drops the connection and asks for the block to another neighbour.

Mosquito attack Our procedure to cope with this attack is to implement a system where every node analyzes the *addr* messages that their neighbours share with them to identify if the neighbours are trying to attack them. The idea is if all neighbours are sharing the same set of addresses with a node through a period of time then the probability of them being malicious is high. If a node identifies a set of neighbours as malicious, he would ban those nodes and re-start its connection to the network. The disadvantages of this strategy are the extra computational power and memory wasted on the analysis of the *addr* messages.

If we were to implement an approach where it costs a bit of the currency to send messages like Ripple or Ethereum. We would cope with this attack but it would require relevant changes to the Bitcoin protocol something that is not desirable by users in Bitcoin.

Implement authenticated messages for sharing IP addresses would also protect against the attack but it would require much more computational power and a system to distribute asymmetric key pairs.

51% attack For this attack, we are not going implement any solution in particular since this attack is always possible unless we control who joins the peer-to-peer network.

Still performing this attack in Bitcoin is very hard. Given the current competition in the mining community and the current size of the network, it would be very costly to gather 51% of computational power.

3.1.2 Internal vulnerabilities

We will now address some of the solutions to the problems Bitcoin has with the behaviour of peers in the network and the way they connect between themselves.

Supernodes To prevent this behaviour we are going to take advantage of the analysis that nodes are going to perform on *addr* messages to prevent the mosquito attack. During the analysis, we will collect the number of times a node appears in those *addr* messages and if a node reaches a certain number of appearances it is safe to presume that is a supernode. In this case, the node that identified the supernode will drop the connection with him and connect to another node. The disadvantages of this approach are once again the extra computational power required and also the possible increase of the mean time it takes for a block to propagate through all the network.

We could also introduce incentives towards a more random topology. An example of those incentives is the approach that Ripple and Ethereum used, where to send certain messages a node has to

pay a fee. In our case to prevent supernodes, if a node exceeded a pre-established number of connections then for every extra connection the node would have to pay a fee in order to be able to sustain that connection. The problem with this approach is that we would have to modify the Bitcoin protocol something that is highly vetted against by the Bitcoin community.

Rational nodes In Section 2.3.3, we saw that in Bitcoin *rational nodes* or *selfish miners* do not behave like *free riders* so we cannot implement directly approaches like the ones presented in before. In the context of file sharing, *free riders* are nodes that would set their upload rate to low while having a high download rate. This is not applicable to Bitcoin because *rational nodes* still work but they do not share the results of their work so that others waste their time and resources.

To prevent this we are going to implement a solution like LiFTinG (Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010) where nodes verify each other. In this solution, nodes analyze the messages their neighbours are broadcasting. The idea is for a node to analyze if a neighbour is broadcasting a new block right after receiving another block at the same height. If the neighbour did this a certain number of times then it would be safe to assume that that neighbour was a *selfish node*. Nodes punish *selfish nodes* by dropping the connection with them. The disadvantage of this solutions is the extra computational power required to perform these verifications and also the possibility of false positives.

Another possible approach that would minimize the impact of this behaviour is the implementations of uncle blocks. Because in this system node still gets rewarded for finding a block at the same height as the new blockchain head, up to a certain height difference.

3.1.3 Summary

Table 3.1 summarizes the approaches we discussed above. As is possible to observe, there is no solution that fixes all the problems.

From Table 3.1 there are some approaches that are not feasible to implement given the Bitcoin requirements for performance, those approaches are the *encrypted messages* and *authenticated messages*. Both approaches would require nodes to perform considerable more computations which would lower the profit of miners. Hence, these solutions would never be accepted by the Bitcoin community.

Other approaches like *uncle blocks* and the *implementation of fees* would be equally hard to implement given the relevant impact they would have on the protocol. Since *uncle blocks* requires multiple nodes to be rewarded by the discovery of a new block and the *implementation of fees* would require fees to be imposed on messages. Hence, both solutions would lower the profit of not only miners but also users and would be highly vetted against by the Bitcoin community.

However, there are still some approaches in Table 3.1 that combined are able to cope with the multiple vulnerabilities while having a low impact on performance. Those approaches are the *optimization in block propagation*, establishment of *extra connections between nodes*, the use of *MACHETE* to send messages through multiple paths, *analysis of addr messagues* and the use of a *LiFTinG approach* to cope with rational nodes. Although all these solutions combined will probably require more computational power from the nodes if we compare them with the approaches described before their impact is much lower.

3.2 Melhorando a Propagação das Transacções

Nesta secção propomos um conjunto de alterações ao mecanismo de propagação de transacções na Bitcoin com o objectivo o tornar mais eficiente, nomeadamente, de reduzir o número de anúncios redundantes que cada nó da rede recebe. A nossa proposta é fundamentada nas seguintes observações:

- Actualmente, os nós recebem em média 3.33 anúncios para cada transacção (quando bastaria receberem no máximo um único para garantir a recepção da transacção).
- A rede possui dois mecanismos complementares para propagar transacções: a troca de anúncios (usada antes da transacção estar inserida num bloco) e a troca de blocos (que implicitamente anunciam as transacções que lhe estão associadas).
- Por razões históricas, o segundo mecanismo é mais eficiente, pois todas as transações em falta num nó são enviadas numa única mensagem (enquanto que o mecanismo de propagação epidémica inicial, obriga a troca de uma mensagem individual para cada transacção).
- A rede Bitcoin não tem requisitos estritos de latência para a propagação das transações, uma vez que a taxa de geração de blocos é significativamente mais lenta que o processo de propagação epidémico (em média, é gerado um novo bloco a cada 10 minutos).
- Os mineiros são uma fracção reduzida do número total de nós. Apesar de ser fundamental que as transacções sejam conhecidas pelo mineiros, o protocolo de disseminação não distingue os mineiros dos restantes nós.
- No protocolo epidémico actualmente usado na Bitcoin, os anúncios são escalonados para serem propagados para todos os vizinhos que o nó conhece (cerca de 125). Este valor é substancialmente maior do que o valor teórico dos protocolos de disseminação epidêmicos que sugere que, mesmo na presença de faltas, basta propagar informação para um número de vizinhos que é aproximadamente logarítmico com o tamanho da rede (**?**). Para o tamanho atual da rede, que possuí cerca de $10\ 000$ nós bastaria portanto propagar para $ln(10\ 000) \approx 10$ vizinhos.

O principal desafio é reduzir a quantidade de anúncios redundantes trocada na rede e, ao mesmo tempo, assegurar que as transações chegam aos nós mineiros. A intuição da abordagem proposta é enviesar o processo de propagação na direcção dos mineiros mais produtivos na rede sem, no entanto, deixar de assegurar que a informação é também propagada por caminhos alternativos para os restantes nós e mineiros. Isto é concretizado através de duas modificações ao protocolo. A primeira consiste em adicionar um novo campo às mensagens que propagam os blocos que indica o número de saltos que o bloco dá na rede. Isto permite que cada nó fique a saber a que distância está o mineiro que gerou esse bloco. Esta informação é codificada num inteiro e portanto tem um impacto negligenciável no tamanho total do bloco que ronda 1 MB. A segunda modificação consiste em identificar localmente, com o passar do tempo, qual o caminho para os mineiros mais próximos e enviar a propagação de anúncios preferencialmente por estes caminhos. Note-se que a partir do momento em que uma transacção é incluída num bloco, o processo de propagação por troca epidémica de anúncios deixa de ser relevante.

3.2.1 Seriação dos Vizinhos

Como foi referido acima, a estratégia proposta para poupar largura de banda pressupõe que cada nó descobre quais os melhores caminhos para chegar aos mineiros mais próximos. Isto é conseguido seriando o vizinhos por um critério de proximidade aos mineiros. Os vizinhos mais perto dos mineiros ficam listados no topo da tabela e os mais afastados são seriados no fundo da tabela.

Para obter esta classificação, cada nó mantém, para cada vizinho, três variáveis: *n*, o número total de blocos que recebeu desse vizinho; *k* a distância acumulada desses blocos até aos mineiros que o produziram; e finalmente *a*, o número total de blocos recebidos. Para permitir actualizar *k*, associamos ao processo de propagação de um novo bloco, um contador que regista quantos saltos o bloco deu na rede Bitcoin (este campo é incrementado sempre que um nó reencaminha o bloco). Com base nestas variáveis, a classificação dos vizinhos num determinado intervalo de tempo *T* é feita de acordo com a seguinte fórmula (em que os valores de k, a e n são os acumulados no período T):

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

Esta caracterização evita que nós com baixa capacidade de mineração que geram blocos muito esporadicamente, fiquem indefinidamente com uma boa classificação. Desta forma, balanceamos não só os vizinhos que têm melhor rácio distancia aos mineiros/número de blocos providenciados, mas também conseguimos ter em conta o número total de blocos que esse vizinho nos forneceu relativamente ao número total de blocos recebidos.

Uma vez que a classificação de cada um dos vizinhos se vai alterando ao longo do tempo, o valor usado para seriar os vizinhos é uma média deslizante da classificação instantânea acima referida, dada por:

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

O factor α existe para evitar que vizinhos de mineiros que já tenham sido extremamente activos no passado mas que no tempo actual já não o são, permanecem para sempre no topo da tabela. Nas nossas experiências usamos $\alpha = 0.3$ e *T* é configurado como sendo um intervalo de 4 horas.

Cada vez que um nó recebe um bloco de um vizinho, a classificação é atualizada conforme descrito no Algoritmo 1.

3.2.2 Propagação Enviesada

Dado que o nosso objectivo é fazer com que as transações que conhecemos cheguem rapidamente aos mineiros podemos utilizar o mecanismo descrito na secção anterior para atingir este objectivo. Assim se estivéssemos perante uma rede onde todos os nós cumpriam com o protocolo e os caminhos fossem resilientes o suficiente poderíamos apenas enviar as transações que recebíamos para o nosso vizinho que tivesse menor valor de classificação e inevitavelmente essas transações seriam adicionadas a um bloco.

Algorithm 1 Computação dos *m* melhores vizinhos

```
1: function ACTUALIZAR_ESTATISTICAS_NO(no_a_actualizar, saltos_bloco)
2:     nova_class ← recalcular_classificacao(no_a_actualizar, saltos_bloco)
3:     pior_class ← −1
4:     pior_no ← None
5:     for no in m do
6:         if nova_class =< no.class() and pior_class < no.class() then
7:             pior_class ← no.class()
8:             pior_no ← no
9:         end if
10:    end for
11:    if pior_class! = −1 then
12:        m.remove(pior_no)
13:        m.append(no_a_actualizar)
14:    end if
15: end function
```

Algorithm 2 Computação dos nós para enviar anúncios de transações

```
1: function NOS_PARA_ENVIAR(transacao)
2:     if m is not empty or (pi == True and transacao.criador() == mim) then
3:         return todos_vizinhos
4:     end if
5:     total ← tamanho_m + tamanho_a
6:     melhores_nos ← m
7:     if tamanho(todos_vizinhos) < total then
8:         total ← tamanho(todos_vizinhos) − tamanho_m
9:     else
10:        total ← total − tamanho_m
11:    end if
12:    if total > 0 then
13:        nos_aleatorios ← escolha_aleatoria(todos_vizinhos, tamanho_a)
14:    end if
15:    return melhores_nos + nos_aleatorios
16: end function
```

No entanto, para além das falhas por paragem, temos de ter em conta que os nós podem manipular o k enviado aos seus vizinhos, colocando-o a zero. Deste modo seriam erradamente priorizados e poderiam ficar em vantagem de duas maneiras: conhecer as transações mais cedo, e não propagar as transações recebidas. Para evitar este problema, além de disseminarmos as transações para os vizinhos com mais prioridade, enviamos também para um conjunto aleatório de vizinhos, conforme descrito no Algoritmo 2. Desta forma asseguramos que as transações continuam a ser disseminadas pelo resto da rede mesmo que tenhamos como vizinho um nó malicioso.

O valor de *pi* (Push inicial) indica se quando uma transação é gerada deve ser enviada apenas para *m* e *a* ou para todos os vizinhos. Desta forma os valores de *tamanho_m*, *tamanho_a* e *pi* podem depois ser alterados para obtermos diferentes padrões de propagação. Na Secção 4.1.1 discutimos as diferentes configurações testadas.

	Uncle Blocks	Optimization in block validation	Extra connections between nodes	MACHETE	Encrypted messages	Monitor RTTs	Analyze addr messages	Fees	Authenticated messages	LiFTinG approach
Information Eclipsing	◯	◯								
Partition Attack			◯	✓	✓	◯				
Delay Attack			◯	✓	✓	✓			◯	
Mosquito Attack							✓	✓	✓	
51% Attack	◯	◯	◯					◯		
Supernodes							✓	✓		
Rational nodes	◯		◯					◯		✓

Table 3.1: Solutions to the problems.
✓ - Protects against the problem
◯ - Helps protect against the problem

4 Evaluation

4.1 Evaluation

Given how adverse the Bitcoin community is to change once we evaluate our system we will have to take into account the following factors that are a priority for both Bitcoin users and miners.

- **Transaction and block latency** As it was described in the previous section, some of the approaches have an impact on the time it takes to disseminate transactions or blocks. So once we evaluate our system we will have to study if our system does not increase this latency time by a relevant amount. Otherwise, our system might induce more forks on the network which is not desirable.
- **Bandwith consumption** This factor is also important for two reasons: i) we want to preserve computational power and ii) we want our system to be scalable.

  We want to minimize the number of messages on the network to avoid nodes from wasting computational power processing irrelevant messages. Furthermore, with the growing number of Bitcoin users if our system was to put a big overweight on the network it would not scale well.

  Some of our approaches will impose an extra overweight on the network so once we evaluate our system we will have to take into consideration that extra bandwidth and try to minimize it.
- **Resources consumption** This is one of the most important factors given the performance requirements imposed by the Bitcoin community. If our system was to downgrade the performance of Bitcoin without a worthy tradeoff our system would never be adopted. It is certain that with the implementation of our approaches the performance aspect of Bitcoin will be affected. So once we evaluate our system we will need to take this into consideration and balance out the performance degradation with the advantages our system brings.

Given the complexity of Bitcoin and its dimension we will start by using the Shadow simulator (Jansen and Hooper 2011) to evaluate our system. We will use an incremental approach to test our system given the complex system that Bitcoin is. We will start by examining if each our approaches protect Bitcoin against their respective vulnerabilities. Then once we are sure that our strategies work as expected we will combine them together and once again evaluate their effectiveness and also measure the factors presented previously.

Our system should be tested with a workload that contains the same amount of transactions happening these days but the amount of messages should be proportional to the size of the emulated network, for instance, we are not going emulate a system with 10 nodes and 100K transactions.

Finally, once we have a good understanding of the impact Bitshield has in Bitcoin we will generate, if there is still enough time until the deadline of the thesis, a working Bitcoin client ready to be deployed in the network.

Para avaliar a abordagem proposta, construímos um simulador de eventos que modela a propagação de transações na rede Bitcoin (este simulador foi desenvolvido em Python). Decidimos implementar o nosso próprio simulador pois todos os outros que encontrámos não implementavam a versão mais actual do protocolo Bitcoin[1].

4.1.1 Calibração do Simulador

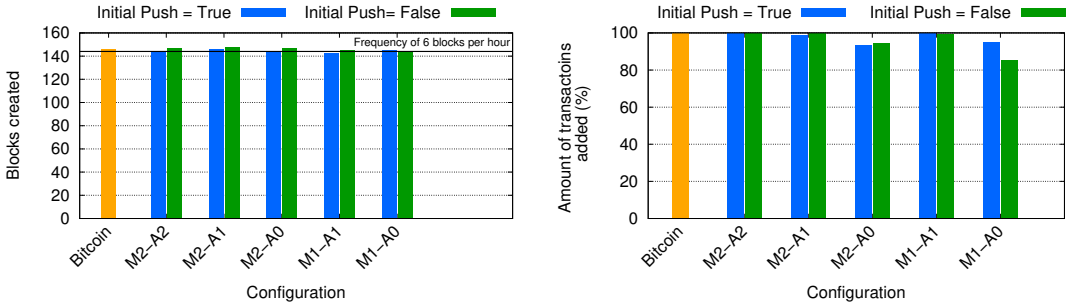
De modo a calibrar o simulador para ser o mais fiel possível ao protocolo original, extendemos o *Bitcoin Core*, o cliente Bitcoin mais usado, para recolher métricas relativamente às mensagens trocadas pelos clientes. As métricas recolhidas são as seguintes: i) anúncios de transações; ii) transações recebidas; iii) transações presentes em blocos compactos que o nó não possuía e tinha de pedir de forma a poder reconstruir o bloco. Corremos duas instâncias deste cliente, em localizações físicas distintas durante um mês e usámos as métricas recolhidas para calibrar e validar o simulador. Além disso usámos a informação publicamente disponível em `https://blockchain.info/` para determinar o número de transações criadas por dia, a distribuição de blocos gerados por mineiro, e o tamanho médio de uma transação. Munidos destas métricas, implementámos o protocolo original no simulador, bem como as nossas modificações. Calibrámos experimentalmente o simulador de modo a que os resultados observados no Bitcoin fossem equivalentes às observações efetuadas no cliente real. O modelo de rede que usámos para as simulações foi composto apenas por nós que seguem o protocolo devidamente.

Devido às simulações serem computacionalmente dispendiosas e demorarem muito tempo a correr (na ordem dos dias) para redes grandes, fizemos testes preliminares para verificarmos se podíamos diminuir proporcionalmente o tamanho da rede e respectivas propriedades sem comprometer os resultados obtidos. Para este efeito, corremos o protocolo original com 6000 nós e com 625 nós e comparámos várias métricas, que discutiremos mais adiante. Os resultados obtidos foram equivalentes para ambos os tamanhos de rede pelo que, no resto desta secção consideramos uma rede de 625 nós. Este escalonamento proporcional entre 6000 nós (tamanho de rede registado quando começamos as experiências) e 625 nós permitiu-nos explorar mais rapidamente o espaço de soluções e correr várias instâncias de cada teste. Os resultados apresentados são a média de 3 testes independentes e correspondem a 24 horas de tempo real.

4.1.2 Impacto da Propagação Enviesada

Começámos por explorar o espaço de soluções à procura de compromissos que nos permitam reduzir o consumo de recursos de rede sem ter um impacto negativo nas propriedades dos sistema. Para ser mais fácil denominar as diferentes experiências realizadas usamos a seguinte notação: *Mn*,

[1] Entre outros `https://github.com/shadow/shadow-plugin-bitcoin` e `https://github.com/arthurgervais/Bitcoin-Simulator`



(a) Quantidade de blocos criados.

(b) Percentagem de transações adicionadas aos blocos.

Figure 4.1: Blocos criados e percentagem de transações adicionadas aos blocos nos diferentes cenários considerados.

em que n especifica o valor da variável *tamanho_m* ; e *An*, em que n especifica o tamanho da variável *tamanho_a* conforme especificado no Algoritmo 2.

Inicialmente, testámos as combinações de $n = 1, 2, 3, 4$ quer para *M* quer para *A*. Após este conjunto preliminar de experiências, observámos que os resultados para $n = 3, 4$ eram praticamente indistinguíveis de $n = 1, 2$, com a excepção do número de duplicados que permaneceu perto do obtido na Bitcoin original. Estes resultados suportam também as medições no cliente real, onde observámos um número médio de duplicados de 3.33. Deste modo, para o resto das experiências efectuadas, consideramos somente as combinações: M2_A2; M2_A1; M2_A0; M1_A1; M1_A0. Adicionalmente, para cada configuração variámos o valor da variável *pi*. Nos dados apresentados abaixo descartámos também a primeira e a última hora de simulação de forma a podermos estudar o sistema em estado estável.

A Figura 4.1a apresenta, para cada configurações a quantidade de blocos que foi gerada ao longo da experiência, enquanto que a Figura 4.1b mostra a percentagem de transações que foram adicionadas aos blocos. Para o cenário Bitcoin obtemos o número esperado de blocos num dia (≈ 144) bem como o número de transações incluídas nos blocos ($\approx 100\%$). Todas as configurações produzem aproximadamente o mesmo número de blocos, contudo para as configurações M2_A0 e M1_A0 nem todas as transações são incluídas nos blocos. Na prática isto quer dizer que nem todas as transações chegam a todos os mineiros, e ilustra a importância de manter a aleatoriedade da disseminação.

Na Figura 4.2 estudamos o tempo médio desde que uma transação é criada até ser incluída num bloco. As linhas horizontais denotam o tempo médio para uma transação ser incluída num bloco no Bitcoin. Mais uma vez, a aleatoriedade ajuda não só a a que as transações cheguem a todos os nós como também diminui consideravelmente o tempo necessário para as transações serem incluídas nos blocos.

A Figura 4.3 apresenta a função de distribuição acumulada do tempo necessário para incluir transações num bloco, sendo portanto uma perspectiva diferente da Figura 4.2. Curiosamente, enviar a transação a primeira vez para todos os vizinhos (*pi=T*) tem um impacto residual no tempo que as transações demoram a ser incluídas no bloco. Isto deve-se ao facto de a taxa de criação de blocos ser bastante reduzida face ao tempo de propagação fim-a-fim da rede.

Analisado o impacto nos efeitos observáveis da rede, nomeadamente o tempo que uma transação demora a ser incluída num bloco e a quantidade de transações incluídas, focamo-nos agora no ganhos, em termos de mensagens poupadas e redução da quantidade de informação transmitida.

A Figura 4.4 mostra o número total de mensagens que foram enviadas em percentagem das enviadas na rede Bitcoin. Como esperado, as configurações com mais poupanças são as que não enviam

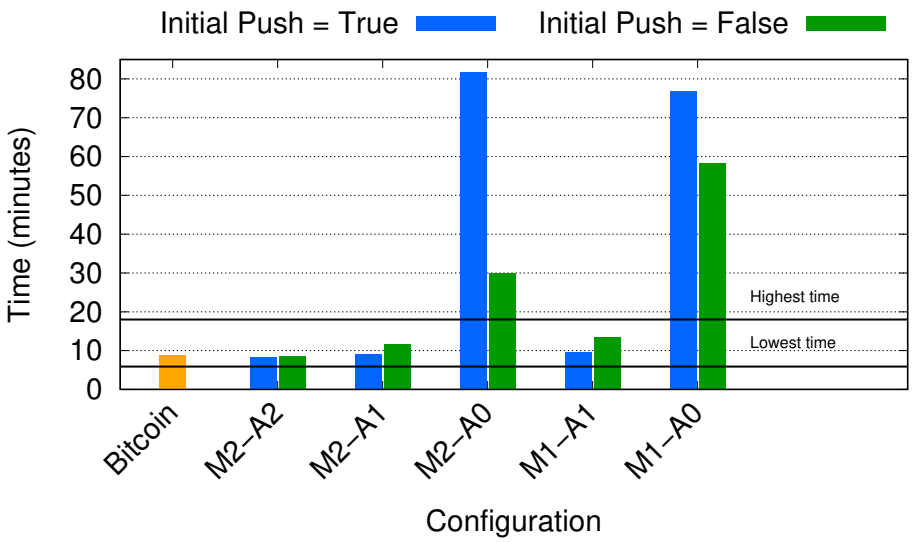


Figure 4.2: Tempo médio que demora desde que uma transação é criada até ser incluída num bloco.

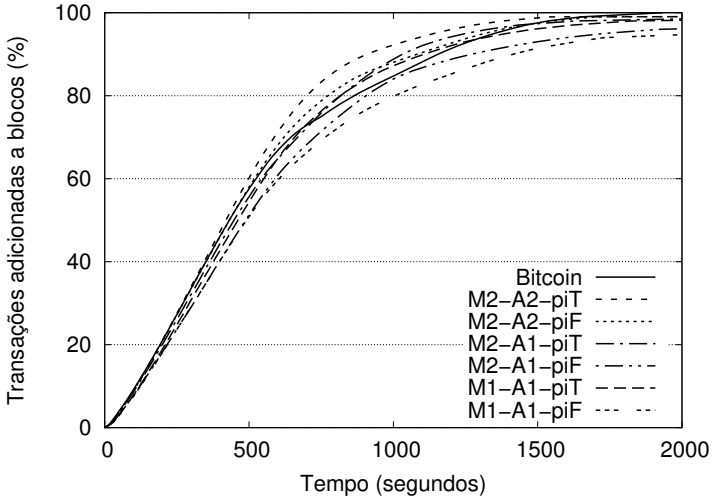


Figure 4.3: Função de distribuição acumulada do tempo necessário para uma transação ser incluída num bloco.

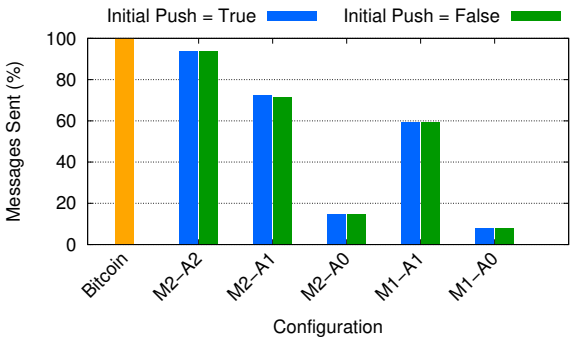


Figure 4.4: Numero total de mensagens enviadas.

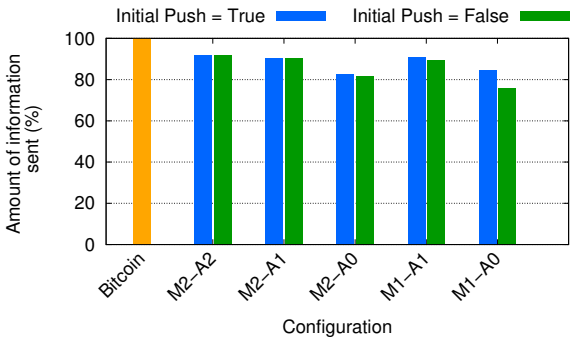


Figure 4.5: Quantidade de informação enviada.

transações para nós aleatórios, pois é provável que esses nós não tenham a transação e logo irão pedi-la ao emissor. No entanto, como vimos anteriormente, estas configurações não são viáveis pois nem todas as transações são incluídas e, as que são, tendem a demorar bastante tempo. A Figura 4.5 mostra a quantidade de informação total enviada em percentagem que, como esperado, segue uma tendência semelhante à Figura 4.4. Podemos observar que a poupança em quantidade de informação enviada não é tão significativa como a das mensagens, pois as mensagens de anúncio são pequenas. No entanto, na prática, processar essas mensagens desnecessariamente impõe um custo nos nós.

Analisando estes resultados, é possível concluir que a configuração mais atrativa é M1_A1 com *pi=F* pois obtém bons ganhos (redução da quantidade de mensagens em 40.5% e redução total da quantidade de informação enviada em 10.7%) ao mesmo tempo que preserva as propriedades da Bitcoin original.

5 Conclusions

Bitcoin and blockchain today still have a lot of vulnerabilities regarding network protocols. In spite of all the research being made in this field, few people have looked at the networks used by these technologies to disseminate information. This is an important aspect to look at because as we show in this report there are multiple vulnerabilities in these networks that allow attackers to exploit systems like Bitcoin that use them.

In this report, we surveyed other cryptocurrencies that use these networks to compare them with Bitcoin. We discovered that although some cryptocurrencies have implemented solutions to these vulnerabilities is at the cost of other factors like performance and privacy.

Some of the vulnerabilities that we studied vary from attacks done by AS to delay attacks performed by a singular attacker. We also cover some undesirable behaviours that nodes inside the network might have like *selfish mining*.

After, we compare multiple approaches to protect Bitcoin against the identified vulnerabilities by identifying the advantages and disadvantages of each one. Finally, we propose a set of extensions that have the purpose of helping Bitcoin overcome these threats.

In the end, we identify relevant factors that we must take into account when evaluating our solution in order for it to be accepted by the Bitcoin community.

Apesar do protocolo de disseminação da Bitcoin ter sofrido várias iterações e melhorias desde que o sistema foi originalmente introduzido, ainda existem hoje em dia várias ineficiências. Dado o tamanho da rede, e a cada vez maior adoção deste tipo de sistemas, torna-se imperioso construir sistemas cada vez mais eficientes, sem comprometer a sua correção e robustez. Neste artigo propusemos várias melhorias ao algoritmo de disseminação que obtêm reduções da largura de banda na ordem dos 10.7% e uma redução no número de mensagens trocadas na ordem dos 40.5%.

Como trabalho futuro, tencionamos estudar mais aprofundadamente o espaço de soluções à procura de combinações que permitam melhorar os resultados obtidos, bem como enriquecer o modelo de faltas dos nós maliciosos. Adicionalmente, pretendemos estudar em que medida o sistema de filiação pode ser enriquecido com a informação recolhida de modo a permitir construir soluções mais eficientes.

References

Apostolaki, M., A. Zohar, and L. Vanbever (2016). Hijacking bitcoin: Routing attacks on cryptocurrencies. *arXiv preprint arXiv:1605.07524*.

Bitcoin.org (2008). Bitcoin wiki.

Bitcoin.org (2009). Bitcoin developer documentation.

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.

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.

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

Jansen, R. and N. Hooper (2011). Shadow: Running tor in a box for accurate and efficient experimentation. Technical report, Minnesota Univ Minneapoiis Dept of Computer Science and Engineering.

Jelasity, M., R. Guerraoui, A.-M. Kermarrec, and M. Van Steen (2004). The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pp. 79–98. Springer-Verlag New York, Inc.

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 *Proceedings of the 7th symposium on Operating systems design and implementation*, 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.

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.