Adaptive Information Dissemination in the Bitcoin Network

(extended abstract of the MSc dissertation)

João Esteves Marçal
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisor: Professor Miguel Matos and Professor Luís Rodrigues

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

In this dissertation, we study the mechanisms of information dissemination used in Bitcoin and propose a set of adaptive mechanisms that lower network resource usage. Our experimental evaluation shows that is possible to lower the bandwidth consumed by 10.2% and the number of exchanged messages in 41.5%, without any negative impact in the number of transactions committed.

I. INTRODUCTION

All cryptocurrencies, and Bitcoin (the most used cryptocurrency at the time of this writing) in particular, maintain a decentralised record that keeps track of all transactions that have happened in a serial order [1]. The ability to maintain, in a decentralised manner, a shared log that can be updated by almost anyone has been considered useful for many other fields beyond the cryptocurrency market, where this concept was initially introduced. For instance, a shared log can be used to keep a record of contracts, avoiding the need for the physical presence of a notary.[1]

This distributed log is usually maintained as follows. First, for efficiency reasons, multiple transactions are grouped together in what is called a *block*. Then, blocks are linked together to form a list which is called *blockchain*. This linked list enforces a serial order over the blocks and, as a consequence, over all transactions listed in the blocks. The blockchain is maintained, in a decentralised manner, by a set of peers. An interesting aspect of cryptocurrencies, like Bitcoin, is that they use a decentralised open peer-to-peer membership system. This means that nodes do not have to know all the other nodes of the system, and any node can join or leave the network at any given time, and still, the protocol ensures the consistency of the blockchain. The distributed protocol is designed to work even if a fraction of the nodes exhibit a rational or byzantine behaviour.

The protocol initiates by letting the nodes in the system concurrently receive, validate, and relay transactions to other nodes. Additionally, in parallel, each node also attempts to generate the next block in the chain. To do this, nodes are required to solve a challenging cryptographic puzzle called the *proof of work*. When a node generates a block, it will broadcast it through the network. The reception of a new block makes all other nodes cancel the generation of concurrent blocks and start attempting to generate the subsequent block. Before accepting and relaying a block, each node validates the blocks it receives.

From the brief description above, it easy to realise that the task of broadcasting transactions is a fundamental procedure in any distributed ledger. First, the transactions need to reach the nodes that are generating blocks so that they can be added to blocks. Second, they also need to reach the remaining nodes, as knowledge about the existing transactions is required to validate new blocks. In Bitcoin, the broadcast of transactions works by letting nodes periodically advertise to their neighbours the transactions they currently have. Their neighbours, upon receiving advertisements for transactions that they miss, will reply with requests for those transactions. Therefore, a node can receive multiple advertisements for the same transaction. In fact, it is desirable that the protocol exhibits some redundancy, as this allows the propagation of transactions to be reliable, even in the presence of faults. Unfortunately, as we will discuss later in detail, the amount of redundancy induced by the current Bitcoin is excessive, causing a significant waste of network resources.

In this dissertation, we propose a number of techniques to improve the efficiency of the transaction broadcast protocol of *Bitcoin*. Our algorithms take advantage of already existing asymmetries in the network. In fact, in the *Bitcoin* network, only a fraction of the network (currently around 10%) spends resources generating new blocks (these nodes are called miners); the majority of nodes just relay information and maintain a copy of the blockchain. Based on this observation, our strategy consists of skewing the dissemination algorithm such that transactions reach miners faster, which will also result in lower amounts of duplicated advertisements. The propagation of transaction to nodes that are not miners may, in result, exhibit higher latency, but this is not an issue for

---

[1] For a list of examples, refer to https://blockgeeks.com/guides/blockchain-applications/

the execution of the Bitcoin protocol as the time window to generate a new block is roughly 10 minutes. Also, our algorithms leverage on the most recent mechanisms that have been added to the standard *Bitcoin* protocol, namely on new control messages that improve the dissemination of transactions once they are added to a block. Our algorithms are adaptive and adjust the dissemination bias as miners leave or new miners join the network. An experimental evaluation of the changes proposed shows a reduction in 10.2% of the bandwidth consumed and a reduction of 41.5% in the total number of messages exchanged, without any negative impact on the system resilience or transaction latency.

II. THE BITCOIN LEDGER

Bitcoin was created in 2009 with the objective of providing a system that allows two entities to exchange goods in a secure and anonymous way, without having to trust each other or any single third entity. This is achieved with a cryptographic coin, that can be exchanged between the parties involved in a transaction. Transactions are grouped in blocks and registered in a distributed ledger. Furthermore, the ledger keeps track of all the coins that have been spent, which forbids users from trying to spend the same coin twice (an attack known as *double spending*). The *Bitcoin* ledger is built by linking each block to its predecessor, hence, forming an infinite chain of blocks named *blockchain*.

The protocol to maintain the Bitcoin ledger is quite complex with many components and functionalities that complement each other. Also, the protocol is evolving, as the community finds new ways of improving its operation. One of the protocol components is a membership algorithm, that aims at ensuring that each node maintains connections to other nodes of the system, chosen at random. These connections among nodes form an overlay that is then used to disseminate information, including the transactions created by clients and mined blocks.

As noted previously, new blocks are generated concurrently by nodes named *miners*. Each miner picks a group of transactions to form a block. A valid block has to contain only valid transactions and also a proof that the node solved the cryptographic puzzle. This cryptographic puzzle is a function of the transactions included in the block and of the hash of the previous block. Note that transactions take different times to reach different nodes. Thus, it is likely that the set of transactions chosen by two nodes to include in a new block is going be different. The use of cryptographic puzzles in this context has two advantages. First, it discourages the creation of blocks with invalid transactions, since the only way a for a node be rewarded is if its block gets accepted into the blockchain. Second, the difficulty of the puzzle lowers the probability that two nodes generate a new block at the same time, an occurrence that may create a fork in the chain. To avoid corrupted or invalid blocks from being broadcast, each node has to validate a block before relaying it. For a node to be able to validate a block it needs the hash of the previous block and all the transactions inside the block (a transaction is valid if it does not uses an already spent coin). If a node does not have all these pieces of information, it cannot validate the block immediately, and it has to wait before relaying the block.

If a node receives a block at the same height as the one it is trying to mine, the process of mining is interrupted. This property also lowers the probability of two different blocks, at the same height, begin generated concurrently. However, if this happens and both blocks are broadcast through the network a fork will happen. Forks are solved when one of the branches grows longer than the other which will make the network adopt the longest branch.

The algorithms used by Bitcoin to broadcast transactions and blocks have been evolving over the years. Recently, recognising that the Bitcoin protocol may consume an excessive amount of network resources, a patch was introduced in the protocol aimed at saving network bandwidth [2]. We briefly describe the current version of the protocol, including the most recent patches. As referred previously, transactions are broadcast through advertisements sent in *Inv* messages. When a node receives an *Inv* message, it determines which transactions it does not have and sends a *GetData* message requesting those transactions. Finally, when a node receives a *GetData*, it will reply with a *TX* message for each transaction requested. Blocks are broadcast mainly in two ways. The first one, and older, is through advertisements similar to transactions. Once a block is found, a *Headers* message is sent advertising the block. When a node receives a *Headers* message referring to a new block, it requests such block with a *GetData* message. The node will then receive the block requested through a *Block* message. The second strategy for broadcasting blocks consists in sending a summary of the block through a *CmpctBlock* (compact block) message. When trying to validate a block received by a *CmpctBlock* message, if the node does not know all the transactions required to validate the block, it can send a *GetBlockTX* message requesting them. The first strategy ensures that a node can validate a block as soon as it receives it because all transactions are sent in the *Block* message, even if the recipient has already received that information via *TX* messages. The second approach aims at reducing this redundancy, at the cost of a potentially slower propagation of blocks in the network.

Additionally, each node maintains, for each neighbour, a queue containing messages scheduled to be sent in the future. When a new *TX* is received, after being validated, it is added to the queue associated with each neighbour. These queues are updated every time the node receives a *TX* or a *Block* message. In particular, if a transaction T is scheduled to be propagated to some neighbour n, but n sends a *TX* or a *Block* containing T, T is deleted from the corresponding send queue. This prevents nodes from sending to a neighbour information that it already owns. Periodically, the queues are flushed by sending *Inv* messages to the respective neighbours.

The introduction of *CmpctBlock* messages helped in reducing some amount of unnecessary redundancy in the Bitcoin protocol. However, we have found that there is still significant room for improvement and that the redundancy can be further reduced, as discussed in the next section.

III. IMPROVEMENT IN THE BROADCAST OF TRANSACTIONS

In this section, we propose a set of changes to the dissemination algorithm of transactions with the objective of making it more efficient, namely by lowering the number of redundant advertisements that each node receives. Our proposal is based on the following observations:

- Currently, each node receives on average 6.6 duplicate advertisements for each transaction (when would be enough to receive a single one to ensure the reception of a transaction).
- The network currently possesses two methods to disseminate transactions: exchange of advertisements (used when a transaction is not in a block) exchange of block (used when a transaction is already added to a block).
- For historical reasons, the second mechanism is more efficient than the first one, since all the missing transactions that a node might request are sent in a single message (meanwhile through the advertisement method a node has to send a message for each individual transaction).
- In Bitcoin, the requirements for broadcasting transactions are weak because the rate of generation of the blocks is much slower than the processes of dissemination of transactions (on average a block is generated once every 10 minutes).
- Miners are only a small fraction of the total number of nodes in the network. However, although it is essential that transactions reach miners, the protocol does not distinguish miners from the rest of the nodes.
- In the current protocol, nodes send their advertisements to all neighbours (125 in the worst case). This value is substantially higher than the theoretical value for epidemic broadcasting algorithms, which suggests that even in the presence of failures, it is enough to send information to a logarithmic number of neighbours with respect to the size of the network [3]. With the current size of $\approx 10\ 000$ nodes it would be enough to send to $ln(10\ 000) \approx 10$ neighbours.

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

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

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

Next, we discuss the ranking process.

A. *Ranking Neighbours*

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

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

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

- Disseminate blocks as fast as possible - because miners once they mine a block want to disseminate that block as fast as possible in order for the rest of the network to build on top of it;
- Disseminates all relevant blocks - because once a miner learns about a new block it starts immediately trying to mine on top of that block hence is in its best interest to also relay that block;

- Adds transactions as fast as possible to blocks - because apart from the block reward miners also profit from the taxes imposed on transactions when they are added to blocks.

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

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

where:

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

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

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

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

Algorithm 1 Top neighbours computation

```
1: function UPDATE_NODES_CLASS($node\_to\_update$)
2:     $scores \leftarrow$ [ ]
3:     for $node$ in $neighbourhood$ do
4:         $score \leftarrow get\_classification(node)$
5:         $scores.append([score, id])$
6:     end for
7:     $sort(scores)$ // sort by score from lower to higher
8:     $top\_nodes \leftarrow$ [ ]
9:     for $i$ in $range(0, max\_t\_nodes)$ do
10:        $top\_nodes.append(score[i][1])$
11:    end for
12: end function
```

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

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

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

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

Hence in our protocol, each time a node receives a block from a neighbour the classification of the neighbours will be updated using Algorithm 1. The node will iterate over his neighbours and for each one, it will calculate his classification using Equation 3.3 and then it will append his classification together with his ID to a vector named scores (lines 3 to 6). After the node sorts the scores vector from the lowest score to the highest score, this means the nodes with the lowest score will be closest to the index 0 of the vector (line 7). Finally, the node will iterate from 0 to the max number of top nodes and append the first max_t_nodes of the vector of *top nodes* (lines 9 to 11). The values for the variable max_t_nodes will be discussed in the next section.

In the end, the node will have in its list of *top_nodes* the set of nodes with the lowest classification. Note that a low classification means that a node has a good/fast connection

Algorithm 2 Nodes to send transactions advertisements computation

```
1:  function NODES_TO_SEND($tx$)
2:      if ($ip == True$ and $tx.source() == self$) then
3:          return $neighbours$
4:      end if
5:      $total \leftarrow max\_t\_nodes + max\_r\_nodes$
6:      if $size(neighbours) < total$ then
7:          $total \leftarrow size(neighbours) - max\_t\_nodes$
8:      else
9:          $total \leftarrow total - max\_t\_nodes$
10:     end if
11:     if $total > 0$ then
12:         $r\_nodes \leftarrow rand\_choice(neighbours, total)$
13:     end if
14:     return $t\_nodes + r\_nodes$
15: end function
```

to at least a miner or a neighbour of a miner.

B. Skewed Relay

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

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

We address this - and simultaneously node failures and Byzantine behaviour - by sending transactions not only to the *t* top nodes but also to *r* random nodes, as described in Algorithm 2. The variable *ip* (ip) indicates that if a transaction is generated by a node, the node has the option of either sending it to *t* plus *r* neighbours or to all of them. We implemented this feature to be able to understand if the first relay had much impact in the commit time of a transaction.

Hence, every time that a node has to relay a transaction it will start by verifying if the variable *ip* is enabled or not if it is it will return the full neighbourhood similar to Bitcoin (lines 2 to 4). If it is not then the node will start by adding the values of max_t_nodes and max_r_nodes to simply check if the size of the neighbourhood of the node is big enough to cope with the value of both variables added. In the end, the node will have in the variable $total$ the number of nodes not in top_nodes that can potentially be chosen as random nodes (lines 5 to 10). With this, the node will then randomly choose $total$ nodes from the set neighbourhood excluding the nodes in top_nodes (line 11 to 13), in the end, this algorithm will return both sets.

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

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

C. Adapting to Network Changes

A key aspect of p2p networks is that nodes can leave or join the network at any time. With this in mind, we designed an algorithm that adapts to the network in order to keep the commit time of the transactions while still trying to send as few messages as possible.

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

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

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

If the average of all unconfirmed transactions does not surpass the threshold of the 30 minutes, then the node will first check if the average time it took to commit its confirmed transactions in the last hour took less than *TM_TX_CONF* (lines 14 to 15). If so the node will check if it can lower the

Algorithm 3 Increase or decrease top and random lists computation

```
1:  function INCREASE_RELAY()
2:      $now \leftarrow get\_current\_time()$
3:      $avg\_time \leftarrow get\_avg\_time\_unconfirmed()$
4:      $timeout \leftarrow avg\_time > TM\_TX\_CONF$
5:      $space \leftarrow max\_t\_nodes + 1 \leq neighbourhood/2$
6:      $cooldown \leftarrow last\_inc + TM\_TO\_WAIT \leq now$
7:      if $timeout$ and $space$ and $cooldown$ then
8:          $increase(t, r, 1)$
9:          $had\_to\_inc \leftarrow True$
10:         $update\_nodes\_classification()$
11:         $last\_inc = now$
12:         $relay\_delayed\_TX()$
13:     end if
14:     $cooldown \leftarrow last\_dec + TM\_TO\_WAIT \leq now$
15:     if not $had\_to\_inc$ and $cooldown$ then
16:         $avg\_time \leftarrow get\_avg\_time\_confirmed()$
17:         $timeout \leftarrow avg\_time <= TM\_TX\_CONF$
18:         $space \leftarrow max\_t\_nodes - 1 \leq 0$
19:         if $timeout$ and $space$ then
20:             $decrease(t, r, 1)$
21:             $update\_nodes\_classification()$
22:             $last\_dec = now$
23:         end if
24:     end if
25: end function
```

values of *max_t_nodes* and *max_r_nodes* by one, if yes it will do it otherwise it will not do anything (lines 16 to 23).

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

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

IV. EVALUATION

To evaluate the proposed approach, we built an event driven simulator that models the broadcast of transactions and blocks in the Bitcoin network. We decided to implement our own simulator because all the simulators that we found were either outdated or were not working [2].

A. Simulator Tuning

To configure our simulator such that it reproduces faithfully the original protocol, we extended the *Bitcoin Core* client, the most used Bitcoin client to log metrics about the messages exchanged between clients. The metrics logged were the following: i) transactions advertisements; ii) received transactions; iii) transactions present in compact blocks that the node had to request to be able to rebuild the block. We deployed two instances of this client in two distinct physical locations for a whole month and used the metrics logged by these two clients to tune our simulator. Furthermore, we also used information publicly available on the website https://blockchain.info/ to determine the number of transaction generated, the distribution of blocks generated by miners and the average transaction size. With all these metrics we implemented the original protocol, and then we added our changes to the protocol. We experimentally tuned our simulator so that the results observed were the same as the ones in the real client. The network model that we used in the experiments of Section IV-B were composed solely of nodes that followed the protocol accordingly.

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

B. Skewed Relay Impact

We started by exploring the different possible solutions to reduce network usage without having a negative impact on the system. In all experiments below, we use the following notation: *Tn* where n specifies the value of the variable *max_t_nodes*; and *Rn* specifies the value of the variable *max_r_nodes* present in the previous algorithms. Note that for these experiments we did not used Algorithm 3 because we wanted to determine the best values for the aforementioned variables.

Initially we tested with multiple combinations of $n = 1, 2, 3, 4$ for both *T* and *R*. After these preliminary experiments, we observed that for values of $n = 3, 4$ the results were practically the same as the results without our approach. However, with $n = 1, 2$ we observed a considerable

---

[2] Some examples https://github.com/shadow/shadow-plugin-bitcoin and https://github.com/arthurgervais/Bitcoin-Simulator

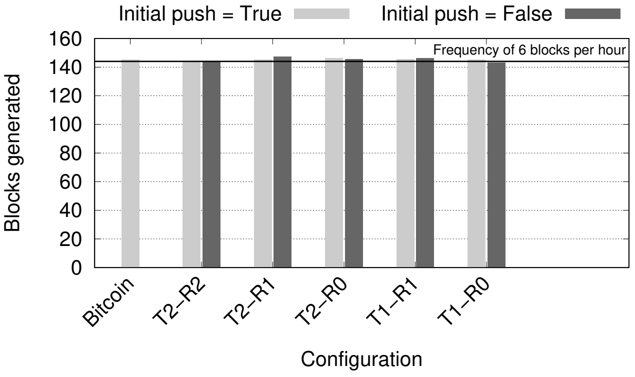

Figure 1. Blocks generated.

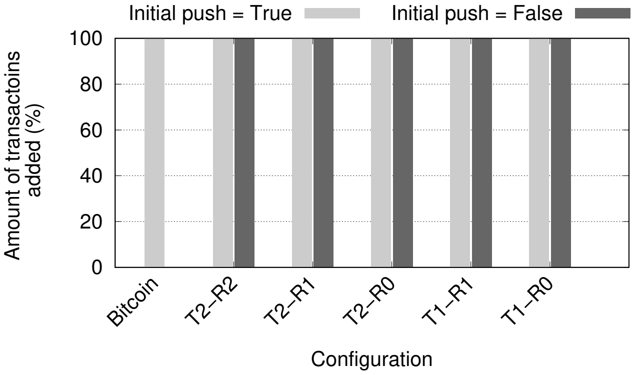

Figure 2. Percentage of transactions committed.

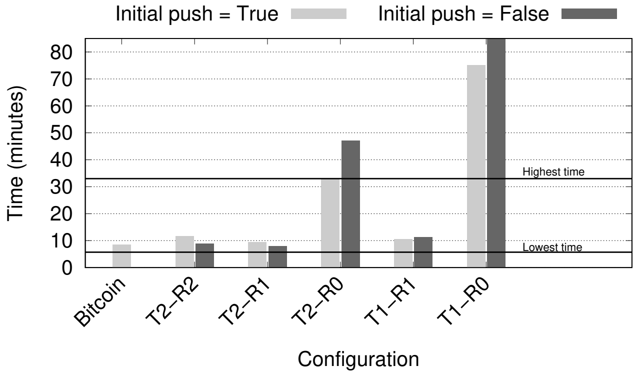

Figure 3. Average time it takes for a transaction to be committed.

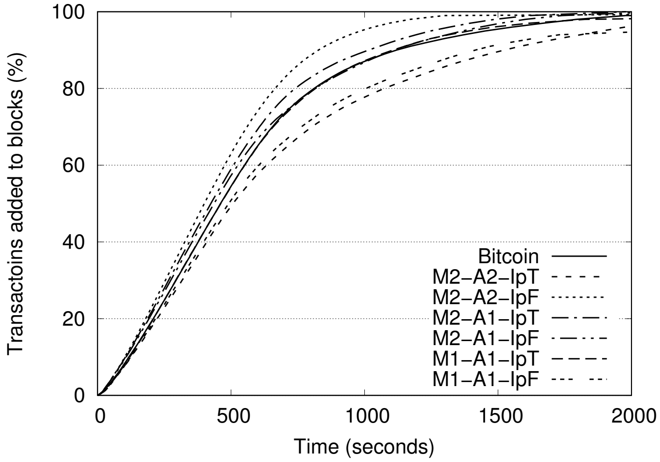

Figure 4. Cumulative distributed function of the time it takes for a transaction to be committed.

reduction in the number of duplicated advertisements. These results also support our logs in the real client, where the average number of duplicates was 6.6. With this in mind, for the rest of the experiments, we considered only the combinations of: T2_R2; T2_R1; T2_R0; T1_R1; T1_R0. Additionally, for each configuration, we also experimented with both values of the variable *ip*.

Figure 1 shows, for each configuration, the amount of blocks that were generated during each experiment, while Figure 2 shows the percentage of transactions added to blocks. As it is possible to observe, the simulation generated the expected amount of blocks for a day (≈ 144) and committed all the created transactions ($\approx 100\%$). This shows that, for every configuration, all transactions reached at least a miner that added them to a block.

We also measured the average transaction commit time for each configuration, depicted in Figure 3. The horizontal lines represent the highest and lowest average time it took for a transaction to be committed in Bitcoin. We can clearly see that both configurations *T2-R0* and *T1-R0* are not good enough to achieve a commit time comparable to Bitcoin. This shows that sending transactions for at least one random node alongside the top nodes has a great impact in the commit time as previously discussed in Section III-B.

Figure 4 shows the cumulative distributed function of the time took to commit all the transactions, hence, it is a different perspective of Figure 3. We can observe sending a new transaction it to all the neighbours (*ip=T*) has a very low impact in the time it takes to commit a transaction. We attribute this to the fact that the time it takes for a transaction to reach all the miners is orders of magnitude (few seconds versus dozen of minutes) lower than the rate at which blocks are being generated.

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

Figure 5 shows ratio between the total number of messages sent in to the same amount in the Bitcoin network. As expected, the configurations with a higher amount of savings were the configurations that did not relay to random nodes. This happens because when we send a transaction to a random node there is a higher chance that that node still does not have that transaction and will request it. Unfortunately, as we have seen previously, both these configurations are not viable because both take very long to commit transactions. Figure 6 shows a different perspective by depicting the savings in the total amount of information transmitted which,a s

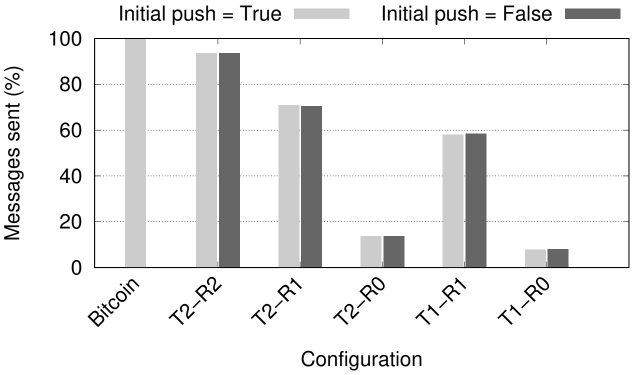

Figure 5. Total number of messages sent.

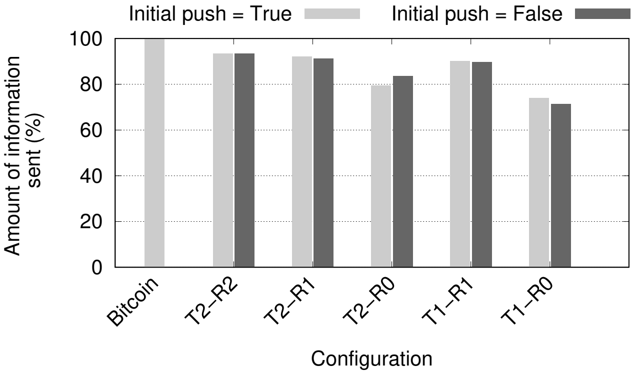

Figure 6. Amount of information sent.

expected, follows a similar pattern to Figure 5. We can also see that the savings from Figure 6 are not as big as the ones from Figure 5. This happens because the advertise messages that we avoid sending are not very big in size. However, processing those spurious incurs an additional cost to the nodes.

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

C. Effect of Adaptation

To determine the best possible configuration we used a stable network, where miners were always the same nodes. However, as in any large network, Bitcoin is prone to changes. We now study the adaption policy introduced in Algorithm 3. Initially, all nodes send advertisements to all their neighbours as in the regular Bitcoin. Then throughout the simulation, the algorithm will progressively determine the best Tn-Rn configuration for each node. We performed three experiments, one where we did not make any changes to the network, another where we change two miners in the network at 12 hours into the simulation and finally a third one where we changed all the miners at 12 hours into the simulation. With these experiments, we want to determine if our solution is able to adapt to the network changes and preserve the commit time while still sending as few messages as possible.

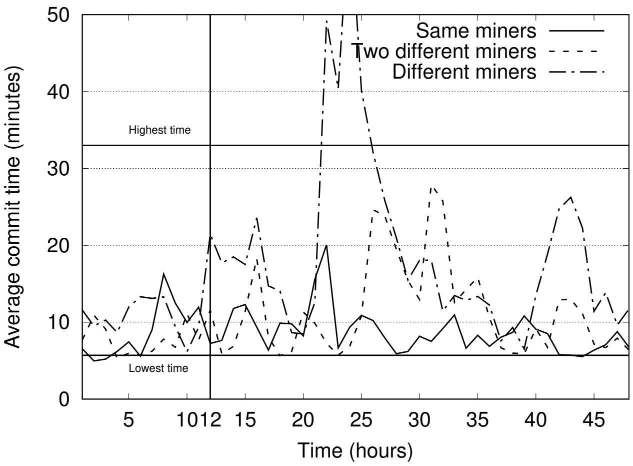

Figure 7. Average commit over time.

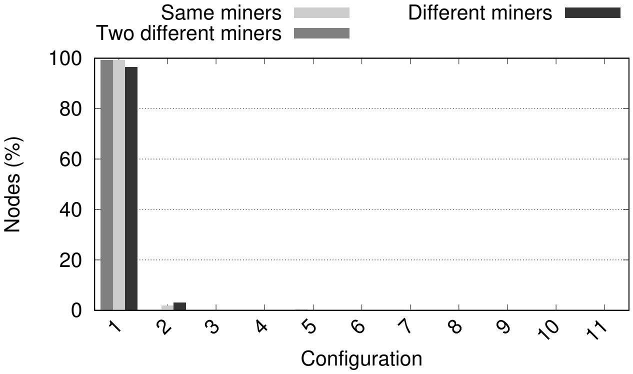

Figure 8. Distribution of the nodes by the different possible configurations.

Figure 7 shows the average commit time over the period of time simulated. It also shows the two horizontal lines that were in Figure 3, delimiting the minimum and maximum observed Bitcoin commit time. Figure 8 displays the percentage of nodes in each configuration at the end of the simulation for the three runs. We can drawn several interesting configurations. First, we can see that if we are in the presence of a stable network, then our solution is going to start adapting to the network and converge to the configuration that we previously deemed ideal (*T1-R1*), as seen in Figure 8. Secondly, we can conclude that if there are slight changes to the network, then the average commit time of our solution is going to deteriorate a little bit, but soon after the algorithm will increase the size of T and R to cope with the changes and the average commit time will once again converge to more regular times. Finally, if our solution is confronted with drastic changes to the network it will not be able to maintain the current commit time, given that at 12 hours into the simulation most nodes were configured to *T1-R1* which is not resilient enough for these cases. We note however that such sudden shift in mining power is unlikely to happen. Regardless, after some time our approach will start converging to the desirable stable configuration.

V. Related Work

In Bitcoin, the dissemination of information is one of the most important mechanisms for the network to function properly. Multiple studies have focused on analysing the protocol of information dissemination and the problems it currently has that may lead to forks in the network [4], [5], [6]. In [4] the authors shed some light to a problem that nowadays seems to not happen as frequently in Bitcoin. The problem was eclipsing of information this happened when two nodes were able to mine a block at the same height. Then both this blocks would be relayed through the network until they reached a node that already had the previous one. This would make the node not relay the other block hence, dividing into two parts. Currently, this seems to have been fixed with the introduction of compact blocks that accelerated the process of dissemination one of the suggestions in [4] to solve the problem previously described.

Other works have focused on how to explore vulnerabilities in the current dissemination mechanisms in order to benefit the attacker or put the victim in a disadvantageous situation [7], [8], [9], [10], [11]. For instance, delaying the dissemination of information could put miners at a disadvantage if the miner retaining the information already has a block mined. Other example is the attack described in [10] where a set of nodes could isolate a node by disseminating to him multiple fake addresses, that would lead that node do discard valid addresses then, once the node had to terminate its current connections it would be only left with addresses of nodes that either did not exited or were attackers.

Regarding previously developed work with similar objective as our, we only were able to find [12] where the authors propose a new protocol named Bitcoin Clustering Based Ping Time, BCBPT is a solution that aims to increase the proximity of connectivity among nodes in the Bitcoin network based on round-trip ping latencies. Since currently, in the Bitcoin network, a node connects with nodes regardless of any proximity criteria. The main objective of this article is to lower the overhead in transaction verification which makes some nodes of the systems vulnerable to double spend attacks

The dissemination mechanism already has gone through multiple changes since its introduction [1] in part to mitigate known attacks. Some of these changes can be found in multiple *Bitcoin Improvement Proposals* [2], [13], [14] but several significant changes required a detailed analysis of the source code as documentation is scarce of non-existent.

VI. Conclusions

Despite the multiple iterations and improvements that have been done to the Bitcoin dissemination protocol since its introduction, there are still some aspects that need to be improved. As Bitcoin becomes more popular and new clients join the system, it is fundamental to have an efficient and robust dissemination substrate for the network to function properly. In this dissertation, we took some steps in this direction by improving the existing algorithm to do a more selective dissemination. Our improvements allow to save, 10.2% of the current bandwidth used and 41.5% of the number of messages exchanged without compromising the robustness of the current approach.

As future work, we plan to leverage more detailed membership information to build more efficient dissemination paths.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] M. Corallo. (2016) Compact block relay. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[3] P. T. Eugster, R. Guerraoui, A. M. Kermarrec, and L. Massoulie, “Epidemic information dissemination in distributed systems,” *Computer*, vol. 37, no. 5, pp. 60–67, May 2004.

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

[5] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, “On scaling decentralized blockchains,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[6] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, “Discovering bitcoin?s public topology and influential nodes,” *et al.*, 2015.

[7] M. Apostolaki, A. Zohar, and L. Vanbever, “Hijacking bitcoin: Routing attacks on cryptocurrencies,” *arXiv preprint arXiv:1605.07524*, 2016.

[8] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, “Optimal selfish mining strategies in bitcoin,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 515–532.

[9] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in *International conference on financial cryptography and data security*. Springer, 2014, pp. 436–454.

[10] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse attacks on bitcoin’s peer-to-peer network.” in *USENIX Security Symposium*, 2015, pp. 129–144.

[11] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath, “Dandelion: Redesigning the bitcoin network for anonymity,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, p. 22, 2017.

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

[13] S. Daftuar. (2015) sendheaders message. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0130.mediawiki

[14] B. Developers. (2018) Bitcoin improvement proposal. [Online]. Available: https://github.com/bitcoin/bips