Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

A Faster Bitcoin Network

Semester Thesis

Chrysoula Stathakopoulou

csathak@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Christian Decker
Prof. Dr. Roger Wattenhofer

January 21, 2015

Abstract

Bitcoin is an electronic currency based on a peer-to-peer network for the propagation and verification of the transactions. Nowadays, cashless transactions are becoming increasingly popular and bitcoin could be an established currency for such transactions. Its distributed nature, though, and more specifically the delay overhead in transaction verification, not only makes the use of bitcoin inefficient for instant transactions, but also makes it vulnerable to double spend attacks. In this work we introduce a modified version of the bitcoin protocol and argue its impact on the delay of information dissemination in the bitcoin network. More specifically we examine how pipelining the message exchange between nodes in the bitcoin network as well as encouraging the connection of the geographically closest nodes affect the delay in propagation.

Contents

Chapter 1

Introduction

Bitcoin is a decentralized cryptocurrecny that allows users to perform transactions without a need of third trusted authority. Bitcoin transactions are anonymous, in the sense that users can hold multiple addresses that are not linked to any personal identifying information, yet transparent, since every single transaction ever made is recorded in a public ledger. The authenticity of the transactions can be easily verified since the owner of the coins, in order to perform a transaction, creates a message signed by her private key. Moreover, users contribute to the transactions verification, transaction fees are negligible. To those characteristics mainly Bitcoin owes its emerging popularity as an electronic cash system.

1.1 Motivation

Transaction validation is not trivial. The bitcoin network nodes must agree to a common public ledger. Due to Bitcoin's decentralized nature, inconsistencies to the replicas of the ledger that each node keeps are unavoidable, thereby introducing uncertainty about transactions until the nodes are synchronized to reflect a common transaction history. Moreover, a desynchronized network is prone to attackers that will attempt to impose their own transaction history, possibly trying to reverse transactions that they sent so as to use the same Bitcoins more than once (*double-spending*). Bitcoin network nodes can with high probability reach an agreement about transaction history, even in the presence of an adversary, but the latency in communication between nodes is critical [1][2].

The bitcoin network might require tens of minutes to reach a consensus [3]. Therefore merchants may choose not to wait so long and release their product as soon as they notice the transaction in the network, a method known as *fast payment*. However, Karame et al. argue that the probability of a double-spend is not negligible in such a case. The probability of success of the payment though, increases as message propagation decreases, which indicates the importance of fast information dissemination. Bamert et al. in [4] on the other hand, managed to perform fast payments within seconds with a probability of double-spends only 0.088%.

In this work, we try to tackle the problem of the agreement on a common transaction history among the nodes of the bitcoin network by speeding up the information propagation. Pipelining the message exchange between nodes and forcing connectivity between nodes that are geographically close we observe acceleration of message exchange.

1.2 Bitcoin Protocol Overview

Bitcoin was first introduced by Nakamoto in [5]. He proposes a distributed system for transactions directly between peers based on digital signatures so as to overcome the need of a financial institution. Furthermore, Nakamoto explicitly addresses the double spending problem in a peer-to-peer network and he suggests a timestamped chain of transactions that will serve as a verification network.

Two core entities of the bitcoin protocol are transactions and blocks. *Transactions* describe the transfer of Bitcoins between two accounts. As specified in [6], transactions have inputs and outputs. Inputs are records which reference the funds from other previous transactions and outputs are records which determine the new owner of the transferred Bitcoins, and which will be referenced as inputs in future transactions as those funds are present. For a transaction to be valid it must be digitally signed with the private key of the spending account. Moreover, the sum of all inputs must be equal to or greater than the sum of all outputs. Sole exception are the base transactions which have no inputs and correspond to the newly created Bitcoins, as a reward for the bitcoin miners. Miners collect and verify newly broadcast transactions into a new group of transactions called a *Block*.

A transaction must be included in a block, in order to be considered legitimate. The block as specified in [7], apart from the verified transactions, contains an answer to a hard-to-solve mathematical problem, unique for each block, that the miner must have found before the broadcast. The answer to the problem can be easily verified by the rest of the network. The difficulty of the mathematical problem is automatically adjusted by the network, such that it targets a goal of solving an average of 6 blocks per hour.

Every block that is created contains a hash of the previous block. This has the effect of creating a *Chain of Blocks* from the genesis block to the current block that defines a chronological order. For a block to be added in the chain it must have been found after its father, since its parents hash has to be included. Therefore a transaction in a lower block must have been verified before a transaction in a higher block. Editing data in a block would therefore require to change all the previous blocks in the blockchain, making it thus computational impractical. These properties make double-spending of Bitcoins very difficult.

The Block chain forms a public *ledger* that must be kept in each node of the network and serves the verification of the bitcoin transactions. Miners, are nodes that can add a new block to the block chain. A transaction can only be found once in a blockchain, or it would have already been verified in the past. However, more than one miners may have received the same transactions so they have to compete on adding the block to the blockchain by solving the afore mentioned hard-to-solve problem. The miner that gets to solve the problem first is the one that can announce the new block. The network must then verify the block and add it to the block-chain as the block-chain head, i.e. the block that is the furthest away from the genesis block.

Due to the distributed nature of the Bitcoin System, though, inconsistencies are unavoidable. It can happen that not all the nodes agree on the same blockchain header, a situation known as *blockchain fork*. As specified in [8] forks are created when the two blocks are created only few seconds apart, so that the information of the first block has not yet arrived to the miner that created the second. When that happens, generating nodes build onto whichever one of the blocks they received first. Whichever of the blocks is included in the next block then becomes part of the main block chain. A transaction can happen to exist in two different branches of the chain. When the bitcoin client switches to another, longer chain, because it has discovered block whose ancestor is head of a longer chain, all valid transactions of the blocks inside the shorter chain are re-added to the pool of queued transactions and will be included in another block. These blocks on the shorter chains are often called *orphan* blocks.

Decker and Wattenhofer [9] provide a model for the blockchain forks and estimate the probability of a blockchain fork at 1.78% in the current bitcoin network. As they pointed out, a transaction is never actually verified. The longer the chain in which the transaction is included, the more sure we can be about its validity. However, if a longer chain starts below the block containing the transaction, the transaction may be invalidated. Even after 60 seconds there still exist some probability that a block message is not known to the whole network. Thus, forks can lead in invalidating transactions due to the inconsistency in the network. Therefore, it is of paramount importance for the nodes to keep a consistent with the rest of the network replica of the ledger.

In order to synchronize their replica of the public ledger, nodes exchange *tx* and *block* messages. A *tx* message describes a bitcoin transaction and also contains the block number or timestamp at which this transaction is locked. A *block* message includes, apart a set of transactions, the reference to the previous block, the calculated difficulty target being used for this block, a timestamp recording when this block was created, version information, a nonce used to generate this block and the reference to a Merkle tree collection which is a hash of all transactions related to this block [6].

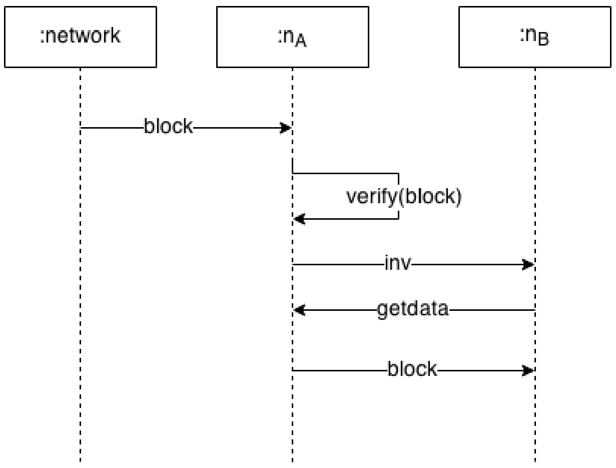


Figure 1.1: Message exchange with in order to forward a block message from node n_A to node n_B

When a node wants to send a set of *tx* messages it first sends an *inv* message, i.e. a list of the hashes of the available transactions. It then waits to receive a *getdata* message that includes the hashes of the transactions which the source of the message actually needs, in the sense that it could already know for some transactions. It then forwards the requested transactions. The receiver must then validate the transactions it receives and then is ready to announce the transactions to its neighbors. A node who is generating blocks will collect valid received transactions and work on including them in a block. When someone does find a block, they send an *inv* containing it to all of their peers, as above [10]. The message exchange is presented in Fig. 1.1.

1.3 Related Work

The problem of reaching a consensus in the bitcoin network falls into the *Byzantine Fault Tolerance* field. The objective of Byzantine fault tolerance is for a system to to be able to work despite Byzantine failures, in which the participants of the system fail in arbitrary ways. Reaching a Byzantine Agreement in a distributed system was first studied by Pease et al. [11] proving that in a synchronous system, it is possible to reach a Byzantine consensus regardless the number of faulty participants, provided that faulty participants are fewer than one third of the participants.

Regarding the Bitcoin network, Miller et al. [1] proved that it can reach a Byzantine Agreement with negligible disagreement probability in bounded running time, since the computational puzzles that nodes have to solve before broadcasting a new block prevent a computationally bounded adversary from claiming too many identities. The time interval required, though, depends on communication latency. Similarly, Garay et al. [2] prove that as long as the network is synchronized, in the sense that only few new blocks are announced within a round, where a round is defined as the time interval requested for the messages to be delivered to the whole network, nodes can establish a long common prefix in their block chains, even for an adversary with computational power close to 50% of the computational power of the network. However, when a network is desynchronized, it can tolerate an adversary with strict bounds on its computational power for the nodes to derive a long common prefix.

On information propagation, Decker and Wattenhofer [9] observed that the size of the message is strongly correlated to the propagation delay in the network. More specifically, for block messages delay is caused mainly due to verification time and therefore be considered a linear function of size. On the other hand for small messages delay is mainly due to the round-trip time for *inv* and *getdata* messages. Hence, they propose to pipeline the block propagation by forwarding invitation messages as soon as they arrive, an idea adopted in this work. Additionally, showing that the block propagation delay is responsible for blockchain forks, and in order to accelerate information dissemination, they propose to minimize the verification process by sending an invitation message for a block as soon as the proof-of-work solution is checked. They finally suggest that maintaining a large pool of connections decreases the distance between the nodes in the network and therefore contributes to the faster propagation of the messages.

Summing up, the bitcoin network must be well synchronized so as to be secure. Low latency in the message exchange contributes to this direction and thus fast information propagation is vital. Our contribution is to experiment with nodes that implement a simplified diversified version of the bitcoin protocol, as described in the next chapter, so as to accelerate the information dissemination.

Chapter 2

A Faster Bitcoin Network

As explained above, delay in information propagation can be responsible for inconsistencies in the bitcoin network. As a result, transaction verification is slower. Furthermore, attackers can take advantage of the inconsistencies to perform double spending attacks that in a slow network are more difficult to discover. Taking that into account, in this work we experiment with the bitcoin protocol to examine how we can accelerate the information propagation. More specifically, we introduce two changes:

- First, as suggested in [9] we pipeline the information propagation. When nodes receive an invitation message for new transaction, they immediately forward the invitation, instead of waiting for receiving the transaction. By doing this we aim to reduce to delay due to the waiting time for the transaction to arrive. While waiting for the transaction message, a time interval that would otherwise be idle, the invitation is forwarded to the network and the network replies with the getdata message. Ideally, when the transaction actually arrives, it can be immediately forwarded, assuming that the getdata message has already been received.

- We then examine how increasing the locality of connectivity speeds up the information propagation. A node suggests the other nodes in the network proximate nodes to connect to, in terms of geographic coordinates. We hypothesize that this approach will speed up the information dissemination by minimizing the delay due to propagation of the messages exchange on unnecessarily long links.

2.1 Implementation Details

2.1.1 A Content Distribution Network

The core idea of our approach is the implementation of Content Distribution Network. We deploy five points of presents in a number of geographical locations as uniformly distributed as possible. The nodes of the CDN are fully connected and known to each other. Each CDN node learns about available bitcoin nodes from a list of DNS services. It can then either connect to the bitcoin network nodes it discovers or wait for the other CDN nodes to suggest a set of the geographically closest bitcoin network nodes. Each CDN node is allowed to connect to up to 100 nodes of the bitcoin network. Since connection to a bitcoin network node can be lost, or new nodes can be discovered, node discovery, suggestion and connection are repeated periodically.

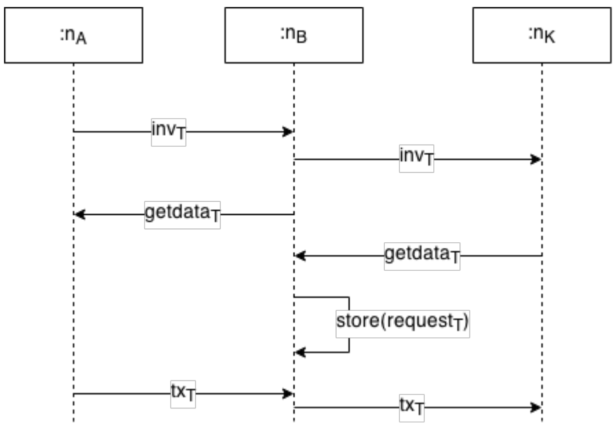
Among the nodes of the CDN we are able to apply our own optimized protocol, as described below. Our purpose is to examine if the information propagation among the CDN nodes is accelerated. Moreover, we want to test how faster we will be able to announce transactions to the nodes of the bitcoin network.

2.1.2 The CDN Bitcoin Client

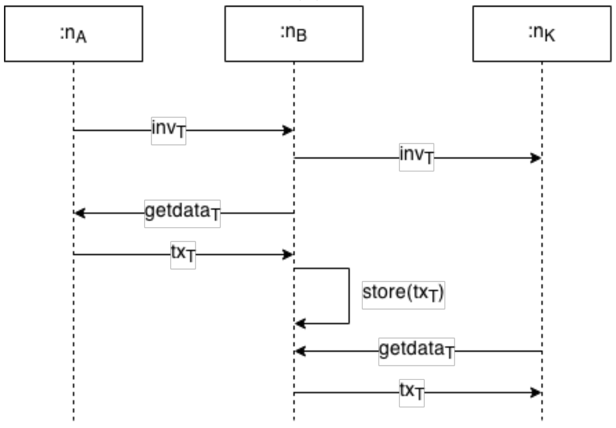
In order to examine how the alternatives to the bitcoin protocol suggested above affect the information propagation, we first created a simple bitcoin client. The client neither keeps a replica of the ledger, nor verifies the transaction messages it receives, since our purpose is to focus on the message exchange between the nodes. It therefore, simply accepts invitation messages for transactions, it replies with getdata messages for transactions unknown so far, accepts the transaction messages for the requested transactions and stores the transactions for a predefined interval and finally sends invitation messages for the transactions it received.

So far the client is well behaved and complies with the bitcoin protocol specification. We now add the following functionality. The CDN client can forward invitation messages for new transactions to the rest of the CDN nodes, as soon as they are received and before the actual transaction arrives Fig. 2.1. According to when the transaction arrives we encounter the two following scenarios. Suppose that CDN node n_A receives invitation inv_T message for a transaction T from node b_B. It forwards the transaction to the set of known CDN nodes N_k.

- Transaction arrives before any $getgata_T$ message arrives from a node $n_k \in N_k$. When later a $getgata_T$ arrives the client forwards the stored transaction.
- A $getgata_T$ arrives form $n_k \in N_k$ before the transaction. The client has to store the request and when the transaction arrives it then forwards the transaction to all the nodes that have requested it.



(a)



(b)

Figure 2.1: Message exchange with early *inv* forwarding when (a) *getdata* arrives before *tx*, (b) *tx* arrives before *getdata*

The CDN client does not store the transactions it receives indefinitely. Transactions expire after a chosen time interval. We apply this policy since memory in the CDN nodes is limited. This can have as an impact that a node will ask in the future for a transaction that is not available. According to [9], however, after 60 seconds information will have been disseminated with high probability over the whole network. We choose, therefore, to remove transactions 60 seconds after we receive them.

Additionally, as mentioned above, we can enable the client to make suggestions for connections to a set of known peers. Having a cluster of CDN nodes that know each other's IP, each node in the cluster periodically calculates the distance between the bitcoin nodes that she has discovered and the other nodes in the cluster. The CDN node then suggests to the other CDN nodes to connect to those of the discovered nodes that are proximate to the latter. We define proximity in terms of actual geographical location. Using MaxMind GeoLite City database [12] we retrieve from the IP adress of the node the Latitude and Longitude. We then calculate the great circle distance, i.e. the shortest distance between two points on the surface of Earth, in meters between the two nodes using the haversine formula. Harvesine a is defined as:

$$\alpha = \sin^2 \frac{\Delta\phi}{2} + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2 \frac{\Delta\lambda}{2} \tag{2.1}$$

where ϕ is the latitude, λ the longitude and R is the radius of Earth. The distance d in meters is then calculated as:

$$d = R \cdot 2 \cdot atan2(\sqrt{\alpha}, \sqrt{1-a}) \tag{2.2}$$

Two nodes are considered to be close when their distance is lower than a chosen threshold d_t

In Fig 2.2 we can see an example of the CDN connected to suggested nodes for a threshold $d_t = 2000km$

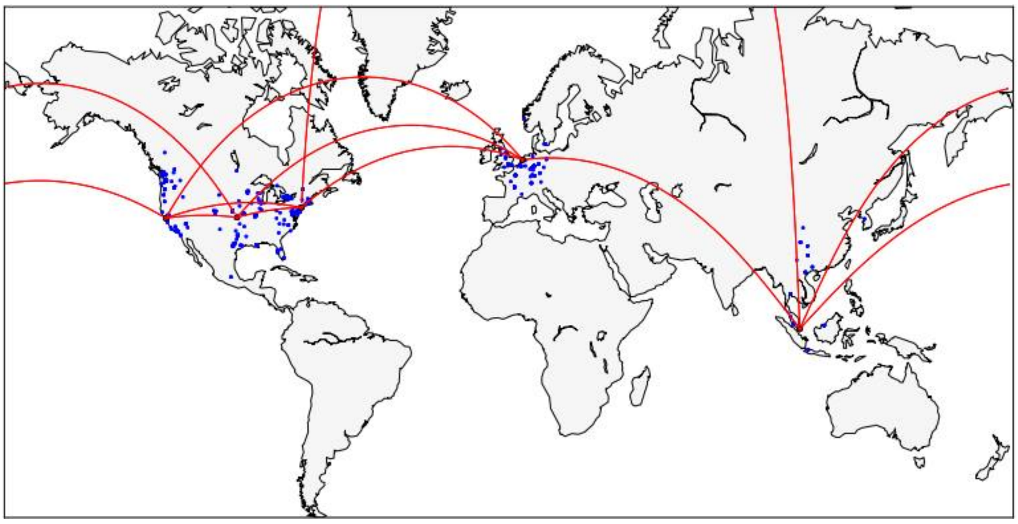


Figure 2.2: CDN nodes (red), connected with great circle arches and suggested bitcoin network nodes (blue)

2.1.3 Gathering Metadata

In order to evaluate the proposed protocol implementations a special node is set up as coordinator. The bitcoin clients send a message to the coordinator in the following cases:

- the client receives an *inv* message

- the client receives a *getdata* message
- the client receives a *tx* message
- the client connects to a proximate node after it receives a suggestion from another client

An important issue is how to synchronize the clocks of the clients so that the information gathered can be compared in order to reach to conclusions. The issue rises from the fact that bitcoin nodes do not synchronize clocks, but rather sample the current time of their neighbors [9]. To tackle this issue we calculate the difference between each client's clock and the clock of the coordinator. Periodically, the coordinator sends a message that includes a timestamp T_1 to the clients. The clients then subtract this timestamp from their clock's indication the time the message arrives T_2 and calculate an offset $T_o = T_2 - T_1$ that they send back to the pool. As it follows the offset can be estimated as the time shift T_s that the client's clock may have plus half the round-trip time RTT.

$$T_o \simeq T_s + \frac{1}{2}RTT \tag{2.3}$$

The equation above provides only a best effort estimation as RTT is not balanced. Finally we estimate RTT by subtracting from T_1 the pool's clock indication T_3 when the response arrives.

Chapter 3

Performance Evaluation

In order to evaluate the proposed modifications to the bitcoin protocol we compare the following scenarios:

1. The CDN nodes comply with the bitcoin protocol and connect to the random bitcoin nodes they discover
2. The CDN nodes comply with the bitcoin protocol and connect to the geographically closest nodes as suggest by the other CDN nodes
3. The CDN nodes pipeline the message exchange to each other and connect to the random bitcoin nodes they discover
4. The CDN nodes pipeline the message exchange to each other and connect to the geographically closest nodes as suggest by the other CDN nodes

For each scenario five experiments where conducted where the CDN was connected to the bitcoin network and real traffic was exchanged. For the experiments each CDN node was allowed up to 100 connections with bitcoin peers. Moreover, new peers where being discovered every 30 seconds. Transactions at CDN nodes would expire, as discussed above, after 60 seconds of their arrival. Messages for the synchronization of the CDN nodes with the coordinator were exchanged every 10 seconds. Finally, the threshold for a bitcoin peer to be considered close to a CDN node and thus suggested was $d_t = 2000$km.

We first examine whether connecting CDN nodes to the closest bitcoin peers available speeds up the information propagation. More specifically, we calculate how many *getdata* messages a CDN node receives from the bitcoin nodes per new *inv* message over the number of connected peers. Receiving more *getdata* per *inv* per connection, indicates that the CDN node communicates with the bitcoin nodes faster and gets to announce more transactions first. Comparing scenarios 1 and 2 we present the average values derived from the five experiments in Table 3.1

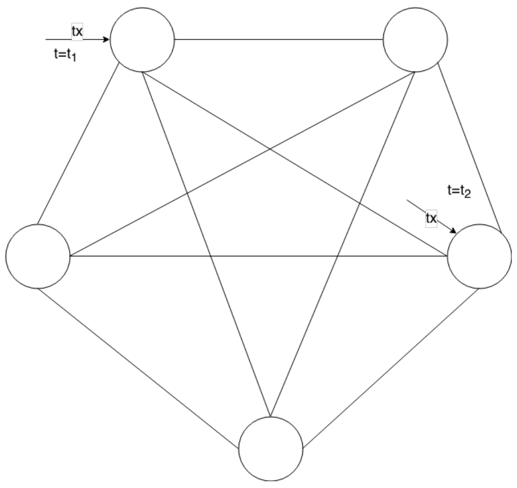


Figure 3.1: Transaction propagation within the CDN

connecting to random peers	connecting to the closest peers
0.86	1.14

Table 3.1: Average *getdata* messages per new *inv* message per connection

We indeed observe that *getdata* messages per new *inv* message per connection increase 32.6%. This result could be explained as follows. Connecting to peers that are geographically close we reduce the overhead due to message propagation as we expect less hops in the network and shorter likns; we don't send information over the Atlantic ocean for instance. Hence, message exchange among the nodes is faster.

Additionally, we want to examine how pipelining message exchange affects information propagation. To this purpose we calculate how fast a transaction is exchanged between the CDN nodes. More specifically, suppose *tx* message arrives at the CDN node n_A at t_1 and reaches the CDN node n_B at t_2, as illustrated in Fig. 3.1. We want to calculate $\Delta t = t_1 - t_2$

For scenarios 1 and 3 the arithmetic mean for Δt is presented in Table 3.2

without pipelining	with pipelining
0.7474 sec	0.2943 sec

Table 3.2: Average time a *tx* message needs to be propagates within the CDN

We therefore observe that *tx* messages exchange needs 60.6% less time when we forward *inv* messages, as soon as they arrive. Such an improvement was expected since while we wait for the *getdata* message to reach the initial source of the *inv* message and then for a *tx* message to arrive, we can already receive the *getdata* message from the new destination node and therefore forward the *tx* message as soon as possible.

We finally calculate the number of new transactions are announced to a CDN node from the other CDN nodes as a percentage of the total new transactions announced. We now compare scenarios 1,3,4. The average values derived from the experiments is presented in Table 3.3.

without pipelining	with pipelining	with pipelining and suggestions
48%	64%	71%

Table 3.3: Average percentage of *tx* messages announced by the CDN nodes

We observe that both adding pipelining and connecting to closest bitcoin peers decrease the time interval that the CDN node needs so as to announce the transaction and, therefore, less transactions get to be announced from the rest of the bitcoin network. When pipelining the messages, *inv* messages are forwarded immediately and therefore much faster than the rest of the network. When connecting to proximate bitcoin peers, the delay due to message propagation in the links between the nodes is reduced and therefore *tx* messages can arrive faster. The combination, as expected leads to the optimal result.

Chapter 4

Conclusion

The purpose of this work was to speed up information propagation in the bitcoin network so as to tackle the problem of inconsistencies in the bitcoin network that rises due to its distributed nature and results in slower transactions verification as well as jeopardizes its safety against an adversary that wants to perform a double-spent attack. To this direction, we implemented a Content Distribution Network that can apply an optimized version of the bitcoin protocol among its nodes. As an optimization, we pipelined the message exchange between the CND nodes and observed 60.6% average acceleration or transaction messages propagation between the CDN nodes. We also experimented with the bitcoin network topology by forcing the CDN nodes to connect to a set of geographical proximate nodes. As a result the CDN nodes where able to announce on average 32.6% more transactions than when connected to random bitcoin network nodes.

Numerous modifications of the bitcoin protocol can be, however, explored. The impact of a different message pipelining can be examined; e.g. forwarding *tx* messages as soon as they arrive, regardless if a *getdata* message has arrived. Different ways to connect the bitcoin nodes can also be explored, such as the number of hopes between the nodes and the link bandwidth. Finally, bitcoin nodes could be prompted to actively connect to such a CDN according to their geographic location, instead of the CDN nodes searching and suggesting bitcoin network nodes to each other. We hope that this work will be a motivation for such experimentations in the future.

Bibliography

[1] Miller, A., LaViola Jr, J.J.: Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin (2014)

[2] Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. Technical report, Technical report (2014)

[3] Karame, G., Androulaki, E., Capkun, S.: Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. IACR Cryptology ePrint Archive **2012** (2012) 248

[4] Bamert, T., Decker, C., Elsen, L., Wattenhofer, R., Welten, S.: Have a snack, pay with bitcoins. In: Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, IEEE (2013) 1–5

[5] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Consulted **1**(2012) (2008) 28

[6] : Protocol specification - bitcoin wiki. https://en.bitcoin.it/wiki/Protocol_specification (December 2014)

[7] : Block - bitcoin wiki. https://en.bitcoin.it/w/index.php?title=Block&action=info (December 2014)

[8] : Block chain - bitcoin wiki. https://en.bitcoin.it/wiki/Block_chain (October 2014)

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

[10] : Network - bitcoin wiki. https://en.bitcoin.it/wiki/Block_chain (June 2014)

[11] Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Volume 27., ACM (1980) 228–234

[12] : Maxmind - geolite legacy downloadable databases. http://dev.maxmind.com/geoip/legacy/geolite/