Blockchain BAR Gossip

Accountable, deterministic and clustered gossip protocol

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Financial Technology and Computing

presented by
Gianmarco Fraccaroli

under the supervision of
Prof. Fernando Pedone

September 2019

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Gianmarco Fraccaroli
Lugano, Yesterday September 2019

To everyone who believed in me.

Arise, you have nothing to lose but your barbed wire fences!

Timothy C. May

Abstract

In the last 10 years, Blockchains have become a fundamental component of several applications and services. The most common use is cryptocurrencies system, such as Bitcoin and Ethereum but their use cases can space also in notary and juridical area. As these systems become more and more popular, they require to be more performant and stable, qualities which they are struggling to meet.
One of their most important and common building block is the gossip protocol used to exchange information between members of the system and the membership mechanism used to provide peers with a view of other participants.
This thesis present Blockchain BAR Gossip, a protocol inspired by BAR Gossip, to exchange information in a blockchain context. BBAR Gossip is characterised by the fact that peer are accountable for their choices and can be punished if misbehaving. Moreover, we evaluate the protocol under more practical assumptions derived from the Byzantine Altruistic Rational model.

Acknowledgements

First off all I want to thank Prof. Fernando Pedone, for his guidance but, most of all, for keeping up with me for two thesis considering my awful english.
I would also like to thank the experts who were involved in the development of the thesis, especially Zarko Milosevic.
Further, I want also to thank my parents for all the support during the last rollecostered five years. You are and always been there for me.
Last, but not least, I would like to thank all the people I have met in the last five years. You guys rock (sometime).

Contents

Contents xi

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Scope . . . 2
- 1.3 Structure of the thesis . . . 2

2 Background 3
- 2.1 Blockchain . . . 3
  - 2.1.1 Consensus . . . 3
- 2.2 Gossip . . . 4
  - 2.2.1 Types of gossip protocols . . . 5
  - 2.2.2 Gossip threats . . . 5
  - 2.2.3 Gossip in blockchains . . . 6
- 2.3 BAR model & BAR Gossip . . . 6
- 2.4 BAR Gossip in details . . . 7
- 2.5 Problem definition . . . 7

3 BBAR Gossip 9
- 3.1 Model assumptions . . . 9
- 3.2 Cryptographic primitives . . . 10
- 3.3 Type of peers . . . 10
- 3.4 Protocol description . . . 11
  - 3.4.1 Epoch concept . . . 11
  - 3.4.2 Registraton phase . . . 11
  - 3.4.3 Peer selection phase . . . 13
  - 3.4.4 Exchange phase . . . 13
- 3.5 Details . . . 16
  - 3.5.1 Proof-of-Misbehavior . . . 16
  - 3.5.2 Shuffle peer list each epoch . . . 16
  - 3.5.3 Multiple BN registration per peer . . . 17
  - 3.5.4 Fair exchange problem . . . 17

3.5.5 PoM rewards 17
3.6 Optimizations 17
3.6.1 History exchange 17
3.6.2 Message header 18

4 Evaluation **19**
4.1 Simulation vs Prototype 19

5 Experiments **23**
5.1 Setup 23
5.2 Convergences rate 26
5.3 Duplicates 27
5.4 Exchange types 28
5.4.1 Number of exchange per epoch 29
5.5 Insights 29
5.5.1 Connectivity 29

6 Conclusion **31**
6.1 Future works 31

Bibliography **33**

Figures

3.1 Register phase chart 11
3.2 Exchange phase flow 14

4.1 Simulation connectivity 20
4.2 Prototype connectivitys 20
4.3 Simulation convergence rate 20
4.4 Prototype convergence rate 20
4.5 Simulation exchanges per epoch 20
4.6 Prototype exchanges per epoch 20
4.7 Simulation exchange type per epoch 21
4.8 Prototype exchange type per epoch 21

5.1 1st simulation peer distribution and nature 24
5.2 2nd simulation peer distribution and nature 24
5.3 3rd simulation peer distribution and nature 24
5.4 Number of peers in common between bootstrap nodes 25
5.5 1st simulation convergence rate 26
5.6 2nd simulation convergence rate 26
5.7 3rd simulation converge rate 26
5.8 1st simulation duplicate updates 27
5.9 2nd simulation duplicate updates 27
5.10 3rd simulation duplicate updates 27
5.11 1st simulation exchange types per epoch 28
5.12 2nd simulation exchange types per epoch 28
5.13 3rd simulation exchange types per epoch 28
5.14 1st simulation number of trade per peer per epoch 29
5.15 2nd simulation number of trade per peer per epoch 29
5.16 3rd simulation number of trade per peer per epoch 29
5.17 4th Simulation heatmap 30
5.18 4th simulation exchange types 30
5.19 4th simulation convergence rate 30
5.20 4th simulation duplicates per epoch 30

Tables

3.1 Login message description 12
3.2 Register message description 12
3.3 Login message description 12
3.4 Token message description 13
3.5 View message description 13
3.6 Header description 15
3.7 ConnectionRequest message description 15
3.8 HistoryDivulge message description 15
3.9 Exchange message description 15
3.10 Promise message description 16
3.11 Briefcase message description 16
3.12 Key Request message description 16
3.13 Key message description 16

4.1 Simulation parameters description 19

5.1 Simulation parameters description 23
5.2 Simulation parameters 24

Chapter 1

Introduction

During the last ten years, after the birth of Bitcoin[Nakamoto et al., 2008], blockchain-based systems became an important building block in several applications, ranging from financial to legal services. As the need for this kind of distributed structure grew, more advanced and sophisticated versions have been built to meet real-world throughput demands.
In fact, as the current Bitcoin protocol can handle only seven transactions per second, other protocols like Tendermint[Kwon, 2014] or Hyperledger can handle volumes of operations in the order of thousands per second.
This is because the latter systems differ in the consensus mechanism, allowing for better performance and higher scalability at the cost of less decentralization.
One of the aspects that is typical in most of those systems is the fact that they exchange information through a gossip protocol. Most of the time, the network can be freely joined by anyone, meaning that everyone receives and shares updates on the state with other peers. As receiving updates reliably and quickly is a key aspect for these kind of protocols, the P2P network is a crucial aspect as it offers many advantages and disadvantages.
Building a faster, safer, and more reliable gossip protocol for blockchains would overall improve those systems, making them closer to real-world scenarios needs.

1.1 Motivation

Blockchain systems offer an alternative way to settle an operation between two or more parties without having to rely on a trusted third party. The most famous and common example is the use of Bitcoin as a method of payment without having to rely on a bank or any other financial service.
This is particularly useful because it is possible to decrease the costs associated with the service offered by the mediator and acquire a higher level of trust as the operations can be carried on just by the two (or more) persons involved in the exchange.
As we said before, due to the decentralized nature of these systems, gossip protocols, sometimes called P2P, are heavily employed since they offer a high degree of scalability, fault-tolerance, robustness, fast-spreading and resilience, proprieties that are a perfect match for a decentralized system.
Still, these protocols have their disadvantages. First of all, as anyone can join the network without any sort of registration, it is hard to account for their behaviour and punish misbehaviours.

Being not able to punish peers for their incorrect behaviour is even worse in cases where this is done on purpose leading to Sybil[Douceur, 2002] or Eclipse[Singh et al., 2006] attacks (which are going to be discussed later on).
So, due to the crucial role of the gossip protocol in a blockchain system, we need to be able to ensure its correctness even in the presence of malicious peers.

1.2 Scope

Gossip protocols used in a blockchain-context don't usually take into consideration malicious peer, or if they do, they generally use easy and simplistic countermeasures. Since most of the blockchain networks are freely joinable by anyone, malicious peers can't be punished by the system in case of bad behaviour as it is always possible for them to change identity and rejoin the system at zero cost. This thesis aims to find a way to make peers accountable for their behaviour, punishing those who either try to gain more benefits at the expense of other peers or try to slow or even disrupt the system. This goal has to be thought in a context where gossip is a crucial aspect of the system, meaning that a slow, inefficient, or expensive protocol would have a significant impact on the overall performance.

1.3 Structure of the thesis

The remaining chapters are structured in the following way.
The second chapter will cover some key concepts about blockchains, gossip protocols, and BAR model.
The third chapter will then discuss in detail the Blockchain BAR Gossip protocol in all his components, from the model assumption to the different phases.
The fourth and fifth chapters will discuss the validity of the implementation of our prototyped simulation and on the experiments derived.
Finally, the last chapter draws the conclusions of this work, discussing some possible future works.

Chapter 2

Background

This chapter outlines the concepts and structures related to this thesis in order to better understand the challenges and solutions of the BBAR protocol.

2.1 Blockchain

Blockchains are distributed systems composed of cryptographically linked blocks, where each block store information about the state up to that point in time. Participants in a blockchain network cooperate in order to agree and keep a copy of the common state. This state is often called ledger.
Members of the network decide which state is correct by processing information through a common set of rules called consensus. The goal of the consensus rules is to agree on a common progression of blocks and prevent bad actors from altering it.
Blockchain systems are usually divided into three main groups: public blockchains (permissionless), consortium blockchains, and private blockchains (permissioned).
The main difference between permissioned and permissionless blockchains is the fact that in the latter, everyone is welcome to be part of the network and take part in the core activities, while the former presents a central authority who is in charge of validating, writing information, and selecting who is able to read transactions.
Instead, consortium blockchains, also known as federated blockchain, are semi-public systems controlled by a group of members. They lie in the middle between public and private blockchains.
This difference is of key importance for our work as peer identities are unknown in public blockchains and known in private blockchains. Instead, peer identities in federated blockchain are project dependent.

2.1.1 Consensus

As most of the features are common for most blockchain systems, what makes the great majority of them unique is the way they reach consensus. Briefly, it is possible to describe consensus as the set of rules which make sure that everyone agrees upon which information and state are correct.
As we said before, there exist several types of consensus, with the most famous being: Proof

of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS), and Proof of Authority (PoA). The first one, PoW, is used by Bitcoin and consists of a competition among peers in the network to find a solution to a cryptographic puzzle. These peers are called miners; the "solution" is called hash-puzzle.
As more and more peers join the network to find a valid hash-puzzle, the difficulty of the puzzle increases.
The second one, PoS, doesn't exploit any computationally hard puzzle but instead gives the possibility to any member of the network to 'stake' an amount of currency to be probabilistically assigned a chance to be the one validating the block.
DPoS is similar to PoS but employs a more democratic system to choose the peer who will be validating the next block. The last one is PoA and works in a similar way to PoS. In fact, only a set of peers, called validators, are able, after reaching a two-third majority, to add the next block to the chain. Validators are required to stake an amount of currency that will be slashed in case of misbehaviour. Moreover, their identity is public.

2.2 Gossip

Gossip protocols work by periodically exchanging information between members of the network. The exchange works in the following way: first, a peer selects randomly another peer in the system. Later, it contacts the peer and starts to share information with it based on a specific strategy. As peers are chosen randomly, this ensures that there will be a time T in which every peer knowns about all information.
These kinds of protocols are usually compared to epidemic diseases as their mathematical proprieties are similar. One of the main proprieties is the so-called spreading rate, which describes how many hosts get infected as a function of the communication round and the number of infections per host:

$$Y_r = \frac{1}{1 + ne^{-f}r}$$

where r is the round, f is the number of host a peer can infect, Y_r is the number of infected hosts in round r. Basically, the convergence rate of a gossip protocol is based on how many exchanges it can execute in each round and the number of rounds with an increase factor of e^f per round.
The benefits of gossip protocols are:

- High scalability, as messages need on average $\log(N)$ rounds, where N is the number of peer in the network, to reach every node and nodes need to send a constant amount of messages independently from the size of the network.
- Fault tolerance, since nodes connectivity and irregularities in their behaviour can be tolerated up to a certain level. Moreover, the same information can be provided by different peers meaning that there is a certain level of redundancy.
- High convergence rate, as we have seen above, peers reach a global state exponentially quickly.
- Simplicity, as every node runs the same code.
- Resilience, as there are exponentially many routes by which information can flow from its source to its destinations.

Still, gossip protocols also have his disadvantages:

- Fragile with malicious peers, as many attacks can be carried on a gossip protocol that would break the whole system.
- Message size limits the scalability of the system, meaning that if information cannot be encoded into a single message, it will require another round of spreading, decreasing the overall scalability.

2.2.1 Types of gossip protocols

There exist several kinds of gossip protocol strategies, but most of them are based on the concepts of push and pull.
In push-based strategies, as soon as a node receives a new update, it randomly selects a peer and sends the full payload. This strategy can be further divided into "eager" or "lazy" pull. The lazy version of push-based gossip differs because a node, instead of directly sending the full payload, sends an identifier (such as a hash). The partner will then respond if it wants or not the full payload.
Instead, in pull-based strategies, a node randomly selects a peer in the network and asks for recently or available information. Upon receiving that list, they exchange missing information by directly asking the partner for them.
Selecting the best strategy is a matter of a trade-off: push-based strategies achive a better convergence rate (as they require one fewer message exchange w.r.t the other strategies) but also produce more redundant data, meaning a higher waste of bandwidth and resources. There is also the possibility to combine the strategies mentioned above to create more efficient but more complex protocols.
One example of such a strategy is the eager push and pull, which is characterized by the fact that gossiping is divided into 2 phases: a first one where peers exchange information through a push-based strategy and a second where a pull-based method is used in order to minimize redundant data.
Gossip protocols are also characterized by how they manage the membership of the network, as each peer can maintain a full-view or a partial-view of the network and structure of the network, as peers can be organized in tree or circle based shape.

2.2.2 Gossip threats

Peers in a blockchain network does not usually need to be pursue any registration as no PKI infrastructure exists. This fact is perfect to increase the level of decentralization, but it also introduces no way of punishing a misbehaving peer, as he can easily change his identity. The most dangerous attacks on this kind of p2p networks are rational attacks, Sybil attack, and Eclipse attacks.
The first one includes all those scenarios where a peer acts selfishly, increasing the load of other more altruistic peers. An example is a peer who always asks for updates without sharing anything, minimizing the use of his resources and maximizing his gain.
A Sybil attack is a kind of security threat where one person tries to take over the network by creating multiple identities. In a blockchain-context, a malicious actor could create a high number of nodes polluting most of the network's view of other peers. This scenario could lead to a bad situation where most of the peers would not be able to receive or transmit information.

Eclipse attacks are a more powerful Sybil attack where an attacker is able to pollute the whole network view of a peer, meaning that the victim cannot understand whether the information he receives is correct or not.

2.2.3 Gossip in blockchains

As of 2019, many blockchains were born, each one providing different services, ranging from financial to juridical and using different gossip protocol. Most of the time, the p2p network is used to propagate messaging containing blocks or transactions, but there are also examples of blockchains which use gossip to disseminate messages for consensus purposes.
The most famous and most used blockchain is Bitcoin. In the Bitcoin p2p network, peers maintain a routing table containing the connection information of all the peers they have known about until that time. Every Δ time T, a subset of peers is chosen to start exchange messages using a lazy push-based strategy. Another relevant blockchain is Ethereum[Wood et al., 2014], which gossip protocol is pretty similar to Bitcoin's. One relevant fact about Ethereum's p2p protocol is the fact that it has been found vulnerable to an eclipse attack[Wüst and Gervais, 2016] in 2016 and since then a new peer sampling mechanism has been added to remove the issue.
Looking at less decentralized blockchain, we have Hyperledger Fabric which divides gossip into push-based and pull-based times. Another one of this kind is Tendermint, which instead follows an eager push-based strategy. Tendermint exploits also gossip to disseminate consensus messages between validators.

2.3 BAR model & BAR Gossip

Traditionally, fault-tolerance theory admits three different models to describe actor's behaviours, determining how much the system can withstand part of it deviating from the protocol.
The first model is the simple fault-tolerance, where the idea is that each actor can either precisely follow the protocol or fail, meaning that the system has to be able to detect and recover from crashes and failures. This model is most useful in those scenarios where the system is architecturally decentralized but politically centralized (ex. Google cloud hosting).
However, what if the considered system is managed by several organizations or individuals? In this case, the model has to be exchanged for the Byzantine fault-tolerance, which states that most of the nodes will follow the protocol, but some of them will deviate either by acting maliciously or because they crash/fail. Byzantine fault-tolerance systems need to survive against a percentage of such byzantine members.
The last, more realistic and complex model extends the notion of Byzantine fault-tolerance; is called Byzantine Altruistic Rational model an differs from the former by adding a simple reasoning: considering a real-life scenario, every actor's behaviour in a system is determined by the incentives he gets when following the protocol. Most members of the system will always try to maximize their utility function even if this requires to deviate from the protocol. A simple example are push and pull forms of gossip: as peers are not obligated to exchange information in a Tic-For-Tac fashion, they can ask for updates and never share them, increasing the burden on a smaller subset of peers.
The BAR model assumes three types of actors:

- Altruistic: actors who always follow the protocol.

- Rationals: actors who follow the protocol if it suits them, and do not follow the protocol if it does not.
- Byzantine: actors who only want to disrupt the system.

If a system survives under the assumptions of the BAR model, it is said to be incentive-compatible. Looking at the literature, several gossip protocol, assuming the BAR model, have already been studied. One of the most famous is the work done in BAR Gossip[Li et al., 2006], which developed the first data streaming application with predictable throughput and low latency.
In BAR Gossip a central tracker distributes updates to a subset of peers, who will then disseminate them with other members of the network in such a way that every peer action is accountable and provable. As to our knowledge, there is no work on gossip protocols assuming a BAR model in a blockchain-context.

2.4 BAR Gossip in details

BAR Gossip is one of the first research on gossip protocols where peers follow the BAR model. It was mainly designed for streaming-context applications and provides a "fairly novel exchange" which avoids the needs of reputation systems and improves the proprieties of the network in presence of rational or byzantine actors.
BAR Gossip relies on a central tracker which distributes a some updates during an epoch and keep tracks of peers identity in the network.
The two main components are the membership mechanism used to choose which peer to contact and the exchange protocol.
The first one is based on a verifiable random function which output is used to deterministically and verifiably select the partner fo each epoch.
The exchange protocol define how peers interact in order to share informations. BAR Gossip defines two exchange type: BAL and OPT. The first one is a Tic-For-Tac exchange meanwhile the second one is more a fallback for peer who are having troubles catching up.
The main feature of the exchange protocol is that every message describe the action taken by the peer followed by a digital signature. This scheme allows the creation of Proof-of Misbehaviour, proofs that can be sent to the central tracker which will punish the Byzantine peer with the eviction.
We will exploit these features into our protocol.

2.5 Problem definition

As we want to include the same membership mechanism and exchange protocol, we are faced with some problems. Blockchain network are by nature decentralized, but BAR Gossip exploits a central tracker to distribute and keep tracks of peer. Moreover, peers in a blockchain network have different demands, with respect to streaming application, in term of data to exchange.
In the next chapter we look at how we managed to overcome these challenges.

Chapter 3

BBAR Gossip

After having discussed the needed background, we can now move onto describing our BBAR Gossip protocol.
It is important to remember that the primary goal of the protocol is to implement a more reliable p2p network able to withstand or minimize the attack surfaces introduced by p2p protocols. This goal must be persuaded, taking into account that decentralization and convergence rate are crucial factors in a blockchain network, meaning it could severely impact overall performances. This chapter starts describing the model assumptions. It then moves to describe the several kinds of peers in the network and their relationship. The last three sections outlines the actual protocol with a focus on implementation details and further optimizations.

3.1 Model assumptions

The model we are using makes the following assumptions:

- The p2p network is not built upon any structure.
- Every peer in the network adheres to the BAR model.
- There exist two kinds of peers: validators and full nodes.
- Validators are block producers and can get punished for misbehaving.
- Validators do not know each other. If they need to talk, they do it by gossiping.
- Validators are also full nodes and are not recognizable by other peers.
- Identities of Validators are known and bound to a Public Key, but their network address is hidden and unknown.
- Full nodes keep track of network operations and can generate Proof-of-Misbehavior against malicious peers.
- There exist seed nodes, which a newly joined peer can contact to receive an initial set of peer addresses.

Our model assumptions are compatible with most blockchain using PoS as consensus mechanism such as Tendermint.

3.2 Cryptographic primitives

In order to be able to make peers accountable for their actions, we need several kinds of cryptographic primitives: hash function, symmetric cryptography, asymmetric cryptography, and pseudo-random number generators.
Hash functions are any function which can be used to map arbitrary length data onto fixed size data. We need hashing function with three proprieties:

- Pre-image resistance: given a hash value h it should be difficult to find any message m such that $h = hash(m)$.
- Second-image resistance: given an input $m1$, it should be difficult to find a different input $m2$ such that $hash(m1) = hash(m2)$
- Collision resistance: it should be difficult to find two different messages $m1$ and $m2$ such that $hash(m1) = hash(m2)$.

In addition to their normal use, we will exploit hash function to build cryptographic hash-based proof-of-work, as Hashcash[Back et al., 2002]. Hashcash PoWs, are resource intensive to compute but easy to verify and provide a valid mechanism to limit spam attacks.
The hashing function chosen in our implementation is SHA256, but other good candidates are SHA3 or BLAKE2.
Symmetric encryption is an encryption mechanism that uses the same key to encrypt and decrypt some data. It is composed of an encryption algorithm, a decryption algorithm, the plain-text, the cipher-text, and a secret key. The encryption algorithm takes as parameters the plain-text and secret key and outputs the cipher-text. The decryption algorithm takes as parameters the cipher-text and secret key and outputs the plain-text. The symmetric encryption algorithm chosen in our implementation is AES256, but Salsa20 or ChaCha are also good candidates.
Asymmetric encryption uses two key, one called private and one public, to either encrypt/decrypt or perform digital signatures. Digital signatures are a mathematical technique to validate the authenticity and integrity of a message and are commonly used to acknowledge the consent of the signer.
Our implementation uses Elliptic Curve cryptography. Pseudo-random number generators are algorithms that deterministically generates a sequence of numbers whose properties approximate the properties of random number sequences. PRNGs cannot be considered really random as their output is totally determined by an initial value called seed. Our implementation follows the Mersenne Twister PRNG, but the XORShift PRNG family is also a good candidate.

3.3 Type of peers

BBAR Gossip introduces two new different node types in the network along with the standard validator, full and seed node. It also modifies the default behaviour of seed node. The new nodes are called bootstrap and entry nodes.
Each node has a particular role in the network:

- Bootstrap nodes are the main difference with traditional gossips. Their primary purpose is to register and keep track of new peers, remove malicious peer, and coordinate the exchange of messages. Each Bootstrap node identity is tied to a corresponding validator's

identity as they can be made accountable for their actions and, in case of misbehaviour, be slashed.

- Entry nodes are an additional component of every bootstrap node. They act as reverse proxies to keep the network identity (IP address) of the bootstrap node hidden to avoid DDoS related attacks.
- Full nodes are simple peers in the network who want to exchange messages with other peers to sync their state. They have different behaviour as they agree to the BAR model.
- Seeds node keep track of Bootstrap node's addresses as new peers need to be able to contact a bootstrap node to start interacting with the network.

3.4 Protocol description

We have divided the protocol into three phases, that a peer has to follow to start exchanging messages and sync his state.

3.4.1 Epoch concept

A cardinal concept and crucial building block of the protocol is the notion of Epoch.
An epoch is a $\Delta\ T$ time used to divide time and synchronize peers, and each Bootstrap node decides how long an epoch last. Epochs have to be a monotonically increasing sequence of numbers and must not overlap.

3.4.2 Registraton phase

The first thing a peer has to do to be included in a network is to contact a Seed node to receive some Bootstrap's node entry node addresses. From the list of addresses, the peer will randomly contact a subset of them and start the following message exchange.

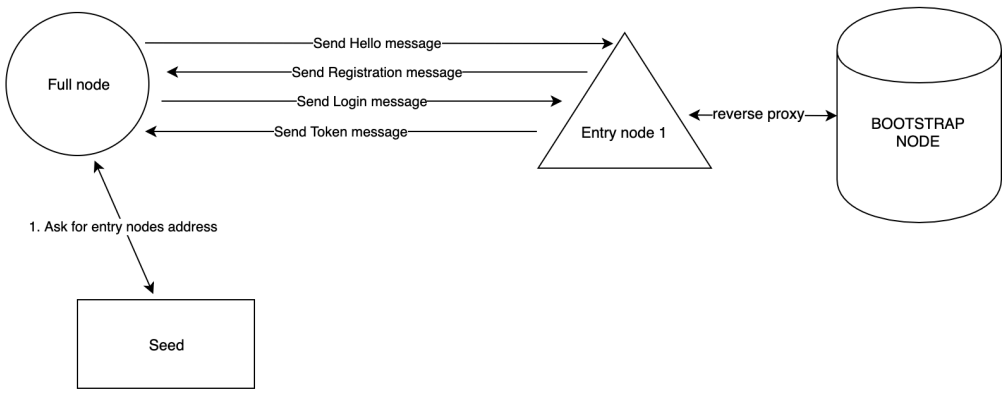

Figure 3.1. Register phase chart

- Peer sends a HELLO message[3.1] to the entry node/s, with his public-key and network address.
- Entry node/s contact the Bootstrap node and send back a REGISTRATION message[3.2], which includes a Hashcash-style puzzle and the corresponding difficulty.
- As soon as the peer finds a solution for the crypto-puzzle, it sends to the entry node/s a LOGIN message[3.3] with the solution.
- The entry node/s checks the validity of the solution. If the solution is valid, sends back a TOKEN message and includes the peer information in the next epoch. Together with a TOKEN message, a VIEW[3.5] message is also sent.

Each bootstrap node can choose the difficulty of the desired solution, meaning that they can control how many registrations they get on average in the next T epochs.
In the last step, after checking that the solution is valid, the entry nodes returns a TOKEN message. The TOKEN message contains a digital signature over the hash of the string generated as the concatenation of the crypto-puzzle, the corresponding solution, and epoch. It is possible for everyone, using this format, to check the validity of the Hashcash solution, (to avoid collusion between a Bootstrap node and a peer) and binds the token to a specific epoch. Moreover, the TOKEN message, includes a field called Key, which stores 16 random bytes which will be used later in the protocol.
Still in the last step, the entry nodes sends also a VIEW message which contains a list of all the registered peers with their informations and the next epoch start time.
When the next epoch start, every peer in the network has to send back their token to the corresponding Bootstrap node. If the Bootstrap node does not have any PoM against that peer, it will send back a new TOKEN and VIEW message.

	Size	**Description**
Host	variable length	Network address of the peer
Port	4 byte	Network port of the peer
Public Key	33 byte	Public key encoded as the x coordinate and oddness bit

Table 3.1. Login message description

	Size	**Description**
Difficulty	4 bytes	Integer describing the difficulty of the solution
Puzzle	49 bytes	Concatenation of the encoded public-key plus a 16 byte nonce

Table 3.2. Register message description

	Size	**Description**
Puzzle	49 bytes	Concatenation of the encoded public-key plus a 16 byte nonce
Proof	Variable length	The solution to the crypto-puzzle

Table 3.3. Login message description

	Size	Description
Puzzle	49 bytes	Concatenation of the encoded public-key plus a 16 byte nonce
Proof	Variable length	The solution to the crypto-puzzle
Token	71 bytes (on average)	The BN digital signature used as token
Epoch	4 bytes	The integer describing the current epoch
Key	16 bytes	Random data used as encryption key

Table 3.4. Token message description

	Size	Description
Peer List	Variable length	List of peer and their corresponding PK and network address
Epoch	4 byte	Integer describing the epoch
Next epoch	4 byte	Unix timestamp of the next epoch time

Table 3.5. View message description

3.4.3 Peer selection phase

As soon a the registration phase is over, a peer should have received a valid token from the TOKEN message and a list of peers he can contact from the VIEW message.
Instead of randomly selecting a peer in the peer list, the selection is computed using a PNRG seeded with the token received. The output of the PNRG modulus the size of the peer list will generate the index of the selected partner in the list.
If the index number is the peer itself, it generates the next random number using the same PRNG, until a valid one comes out.
Using this mechanism, we can make peers accountable for peer-selection and create PoMs in case of misbehaviour.

3.4.4 Exchange phase

The last phase, called Exchange phase, is heavily inspired by BAR Gossip and exploits a sequence of messages to maintain peers accountable for every decision they make.

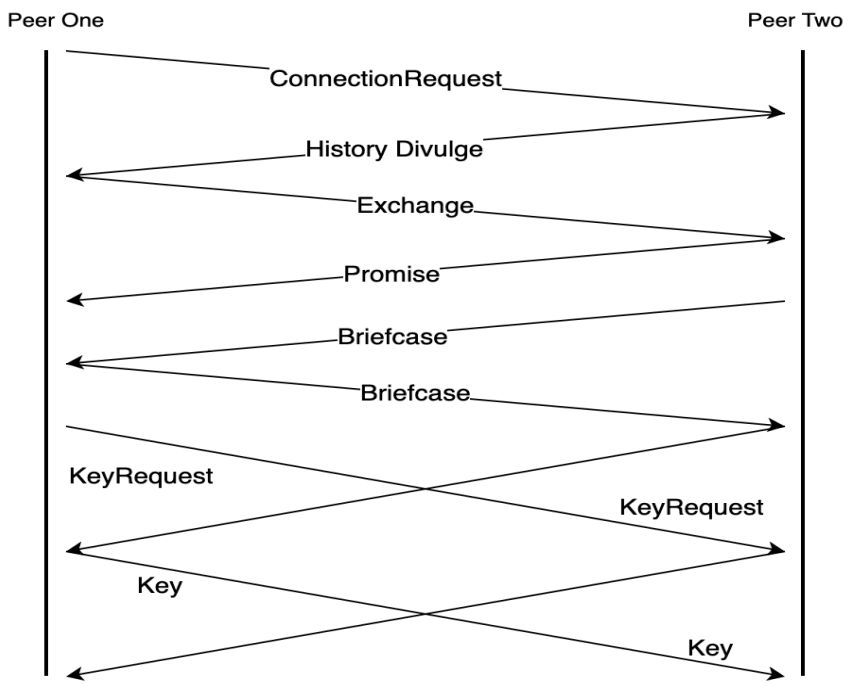

Figure 3.2. Exchange phase flow

As soon as a peer determined who has to contact, the protocol follows these steps:

- A CONNECTIONREQUEST[3.7] message is sent to the selected partner.
- After validating the connection, the partner sends back a HISTORYDIVULGE[3.8] message, informing the peer that he has accepted his request and giving him the details about his mempool/state as a list of hash identifiers.
- On receiving the HISTORYDIVULGE message, the peer computes the set difference with his mempool/state. Based on those results, it decided if his best choice is a BAR or OPT exchange and sends to his partner an EXCHANGE[3.9] message containing the exchange type to perform and a list of updates he wants and promise.
- The partner checks if the exchange is valid (it will be explained later what it means), and sends back a PROMISE[3.10] message, which state the fact that he accepted the exchange, and a BRIEFCASE message, which contains the updates encrypted.
- Upon receiving a valid PROMISED message, the peer also sends to his partner a BRIEFCASE[3.11] message.
- Upon receiving a BRIEFCASE message, both peers will start sending a KEYREQUEST[3.12] message to the partner, until either the epoch finish or until a KEY message arrives.
- The KEY[3.13] message includes the secret key to open the corresponding briefcase, containing the requested updates.

First of all, each message sent during this phase contains a header with the token plus the relative information to verify his validity and a hash of the previous message, so that it is possible to rebuild the message history.
A crucial aspect of the protocol, which is heavily inspired by BAR Gossip, are the two types of exchanges available: BAL or OPT. A BAL exchange is characterized by the fact each member of the trade promised to exchange the same number of updates, in a Tic-For-Tat fashion. As this is not always possible, OPT provided a safe-net mechanism for peers who do not have something to exchange, and they rely on the altruism of other peers. In fact, a peer A who starts an OPT exchange, promises to the other 0 or more updates and asks the partner to send back either the same number of updates as him or more. Mooreover, as we want to incentivize the more fair BAL exchange, partner B will add a percentage of random data to the BRIEFCASE message.
Another fact is that the maximum number of sharable updates in each trade is bounded both for BAL and OPT, as this increase the possibility for a peer to start a BAL exchange.
The Briefcase message is encrypted using a symmetric encryption algorithm where the secret key is received by the Bootstrap node in the TOKEN message.
Last, but not least, as in BAR Gossip, all messages are sent through TCP connection expect the KEYREQUEST and KEY message, which are sent via UDP.

	Size	**Description**
Puzzle	49 bytes	List of peer and their corresponding PK and network address
Solution	Variable length	Integer describing the epoch
Epoch	4 bytes	Unix timestamp of the next epoch time
Token	71 bytes (on average)	The BN digital signature token
Prev. Message Hash	32 bytes	The hash of the previous message

Table 3.6. Header description

	Size	**Description**
Header	Variable size	The header of the message used to verify the peer request validity

Table 3.7. ConnectionRequest message description

	Size	**Description**
Header	Variable size	The header of the message used to verify the peer request validity
Update ids	Number of updates * 20 bytes	The list containing the identifiers of the updates

Table 3.8. HistoryDivulge message description

	Size	**Description**
Header	Variable size	The header of the message used to verify the peer request validity
Needed	Variable size	Identifiers of updates the peer needs
Promised	Variable size	Identifiers of updates the peer will trade
Type	1 byte	The type of exchange: BAL, OPT or ABORT.

Table 3.9. Exchange message description

	Size	Description
Header	Variable size	The header of the message used to verify the peer request validity
Needed	Variable size	Identifiers of updates the peer needs
Promised	Variable size	Identifiers of updates the peer will trade
Type	1 byte	The type of exchange: BAL, OPT or ABORT.

Table 3.10. Promise message description

	Size	Description
Header	Variable size	The header of the message used to verify the peer request validity
Briefcase	Variable size	The encrypted updates

Table 3.11. Briefcase message description

	Size	Description
Header	Variable size	The header of the message used to verify the peer request validity

Table 3.12. Key Request message description

	Size	Description
Header	Variable size	The header of the message used to verify the peer request validity
Key	16 bytes	The secret key used to encrypt the briefcase

Table 3.13. Key message description

3.5 Details

3.5.1 Proof-of-Misbehavior

The idea behind PoMs is to have a mechanism to prove that a member of the network tried to deviate from the protocol. In our protocol (as well as in BAR gossip) this is accomplished through the use of cryptography: digital signatures and hash functions are combined to keep each peer accountable.
As soon as a peer misbehaves, the partner can generate a PoM and send it to the corresponding Bootstrap node, who will schedule an eviction for the next epoch.

3.5.2 Shuffle peer list each epoch

A Bootstrap node could collude with a peer to particularly build the order of the peer list in such a way that a peer always contacts the same subset of peers. Due to this issue, the peer list has to be shuffled in a random but verifiable way. An option, which we have used in our implementation to accomplish this task, is the following:

- Sort the peer list by X coordinate of the public-key.

- Seed a PNRG with a public seed (i.e., the current epoch) and generate a sequence of unique numbers from 0 to the size of the list.
- Shuffle the peer list according to the generated sequence (i.e., first peer in the sorted list goes to the position specified by the first number in the sequence).

3.5.3 Multiple BN registration per peer

An important factor for this protocol is that peers need to be registered with multiple Bootstrap nodes. This is because there has to be at least a path which links all the Bootstrap node subnetworks into a single one. Otherwise, peers would be able to exchange only the information inside a single bootstrap network, or inside a subset of the entire network.
Obviously, this comes with a trade-off: the more registration a peer has, the more trades he has to do meaning that he will use more bandwidth and resources.

3.5.4 Fair exchange problem

The reason why a Bootstrap node has to give the secret key for encrypting the briefcase together with the token is that a malicious peer could create some fake updates identifiers to perform a BAL exchange, wait for the partner KEY message and then fake a crash or timeout. Using this strategy, one of the peers receives the updates; meanwhile, the other would not be able to decrypt the BRIEFCASE, which contains garbage data. This problem can be reduced to the fair exchange problem, which is actually impossible to solve without a trusted third party[Pagnia and Gärtner, 1999]. In our case, the Bootstrap nodes act as a TTP. If a peer does not receive the KEY message, he can send an MPoM (Maybe-Proof-of-Misbehavior) with the encrypted briefcase. The bootstrap node will decrypt the BRIEFCASE message, checks the validity of the updates, and if the updates are legit, sends back the key to the peer. Otherwise, it will ban the peer's partner.

3.5.5 PoM rewards

As sending PoM is resource expensive for peers, we need to incentivize this behaviour. A possible solution is the following: since bootstrap nodes are linked to Validators which are block producers, they can give priority of receiving the new blocks to those peers that send valid PoMs or MPoMs.

3.6 Optimizations

There are several ways to improve the performance of the protocol, but the ones with the most gains are the following two.

3.6.1 History exchange

One of the most resource-consuming messages is the HISTORYDIVULGE, as the partner has to provide a list of hash identifiers, one for each information he knows. This scale linearly with the number of updates.

The more straightforward way to decrease the size of the list is to encoded the identifiers with a smaller format and to reduce the number of identifiers. This is archivable by:

- Using a hash function with a smaller output such as RIPEMD-160.
- Heuristically chose a subset of updates.

A more sophisticated way to accomplish this task is to encode the updates into a more efficient data structure such as an Invertible Bloom Lookup Table[Goodrich and Mitzenmacher, 2011] or a Set Sketch[Dodis et al., 2003].
Those kind of data structure are very efficient for set reconciliation tasks as they are able to encode data (in our case the identifiers) in a bloom filter fashion but with the added features that it is possible to recover the whole set of encoded data, and it is possible to perform the subtraction between two of them.

3.6.2 Message header

The message header takes a non-trivial amount of bytes in each message. It is possible for each peer to send just the first message with the full header, while the following messages could contain just a hash of it.

Chapter 4

Evaluation

4.1 Simulation vs Prototype

In order to evaluate our protocol we decided to implement both a prototype, with all the features described in the previous chapter, and a simulator, which is more scalable and easier to manage. We setup the prototype and simulator with the same parameters:

	Value	Description
Bootstrap nodes	3	The number of BNs in the simulation
Full nodes	25	Total number of FN in the simulation. They are ALTRUISTIC by default.
Rational nodes	10	Number of rational nodes
Byzantine nodes	5	Number of byzantine nodes
Epochs	40	Number of epochs
PoW Difficulty	3	Difficulty expressed as number of epochs
Byzantine level	0.3	Probability for a byzantine node to act byzantine during an exchange
Mempool size	20, 0	Mean and standard deviation of peer's mempool size
Number of subscriptions	2	Number of registration per peer
Max BAL exchange	8	Maximum number of updates per BAL exchange
Max OPT exchange	15	Maximum number of updates per OPT exchange

Table 4.1. Simulation parameters description

Due to some limitation in the prototype implementation and as most of the actions taken are based on a random coin flip, we are not able to run two identically models even if the parameters are the same.

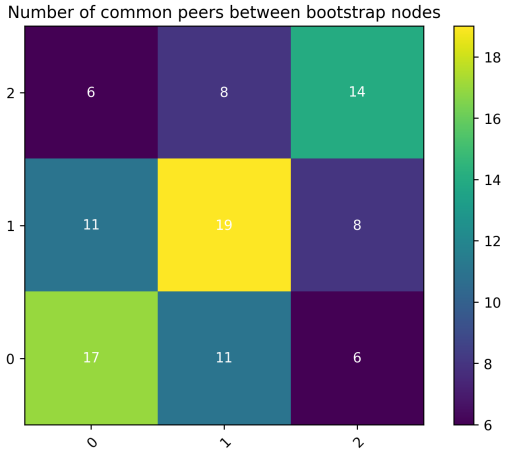

Figure 4.1. Simulation connectivity

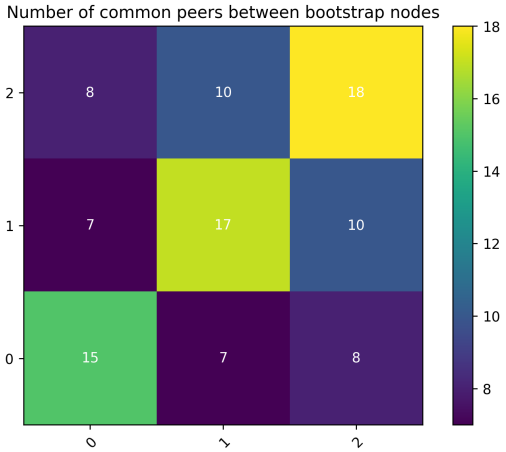

Figure 4.2. Prototype connectivitys

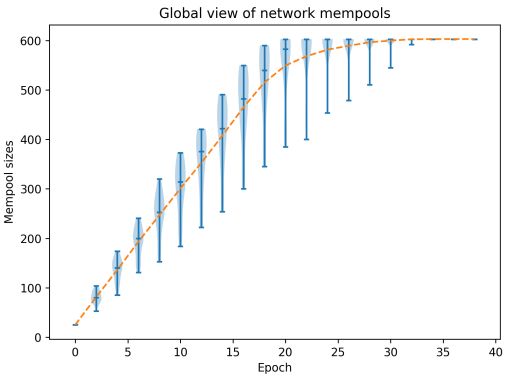

Figure 4.3. Simulation convergence rate

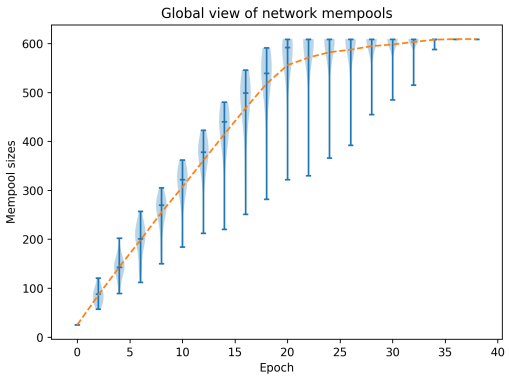

Figure 4.4. Prototype convergence rate

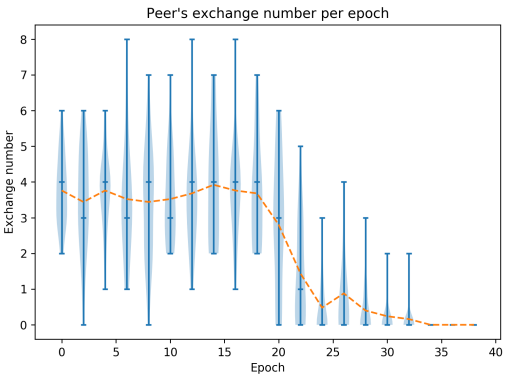

Figure 4.5. Simulation exchanges per epoch

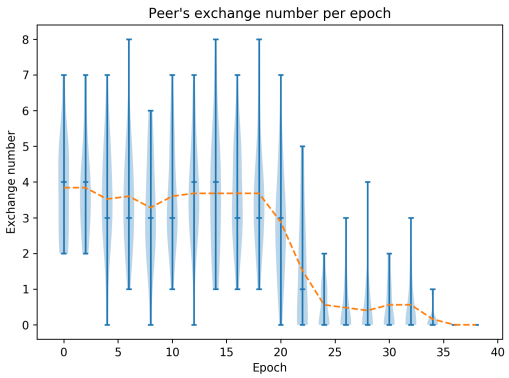

Figure 4.6. Prototype exchanges per epoch

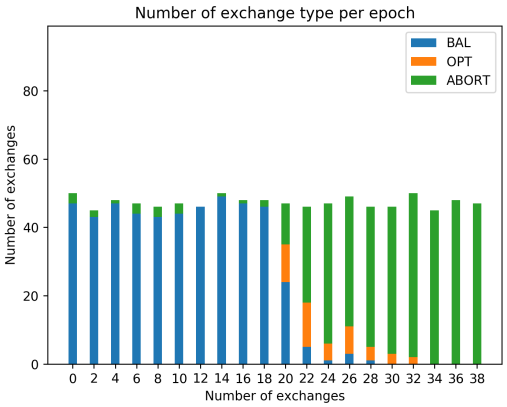

Figure 4.7. Simulation exchange type per epoch

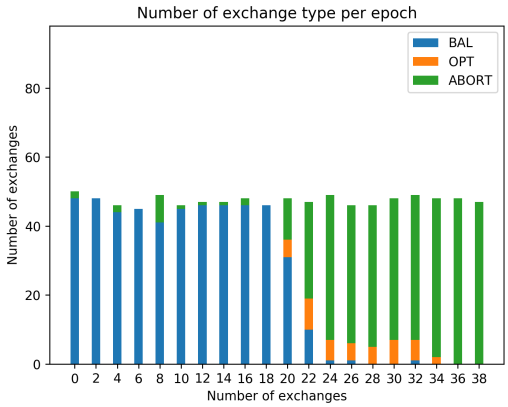

Figure 4.8. Prototype exchange type per epoch

Most of the metrics in the figures above, show very similar data. Our prototype has a little slower convergence rate and trades per epoch but this is probably due to small known implementation issues.
Still, looking at the plots gives us a high confidence that the simulation can faithfully reproduce a real environment.

Chapter 5

Experiments

In this chapter we are going to test the protocol under different assumptions. The parameters taken into account are the following:

	Value	Description
Bootstrap nodes	8	The number of BNs in the simulation
Full nodes	250	Total number of FN in the simulation. They are ALTRUISTIC by default.
Rational nodes	Variable	Number of rational nodes
Byzantine nodes	Variable	Number of byzantine nodes
Epochs	200	Number of epochs
PoW Difficulty	3	Difficulty expressed as number of epochs
Byzantine level	0.4	Probability for a byzantine node to act byzantine during an exchange
Mempool size	70, 25	Mean and standard deviation of peer's mempool size
Number of subscriptions	Variable	Number of registration per peer
Max BAL exchange	12	Maximum number of updates per BAL exchange
Max OPT exchange	20	Maximum number of updates per OPT exchange

Table 5.1. Simulation parameters description

The main goal of these experiments is to see how the protocol behaves under different assumptions such as presence of rational nodes or byzantine nodes, or both. We analyze the simulation looking at different factors such as convergence rate, number of duplicate packets, type of exchange, etc.

5.1 Setup

In the following sections, we will three general simulation in order to see how much the presence of rational and byzantine nodes afflict the system.
The parameters of the simulations are the following:

	Simulation One	Simulation Two	Simulation Three
Rational nodes	0	100	100
Byzantine nodes	0	0	50
Number of subscriptions	2	2	2

Table 5.2. Simulation parameters

The maximum mempoool size is 129 and the minimum is 6.
Other important factors in the simulations are the connectivity between Bootstrap nodes and the number/distribution of Full nodes by their nature.

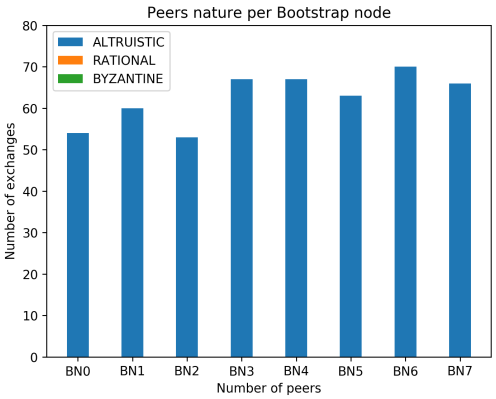

Figure 5.1. 1st simulation peer distribution and nature

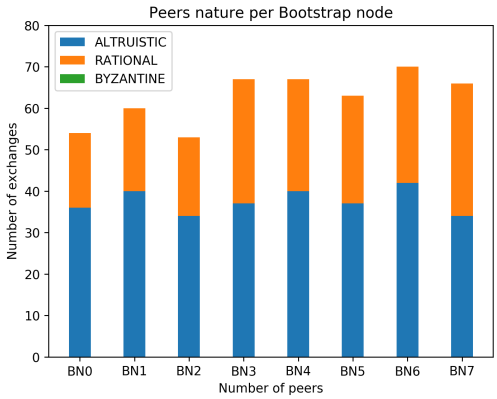

Figure 5.2. 2nd simulation peer distribution and nature

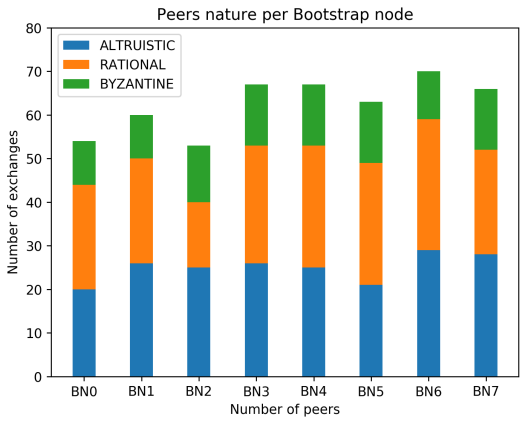

Figure 5.3. 3rd simulation peer distribution and nature

As we can see from figure 5.1, 5.2 and 5.3, each bootstrap node has more or less the same number of peer and, in the second and third simulation, also the distribution of rational and byzantine actors is even.

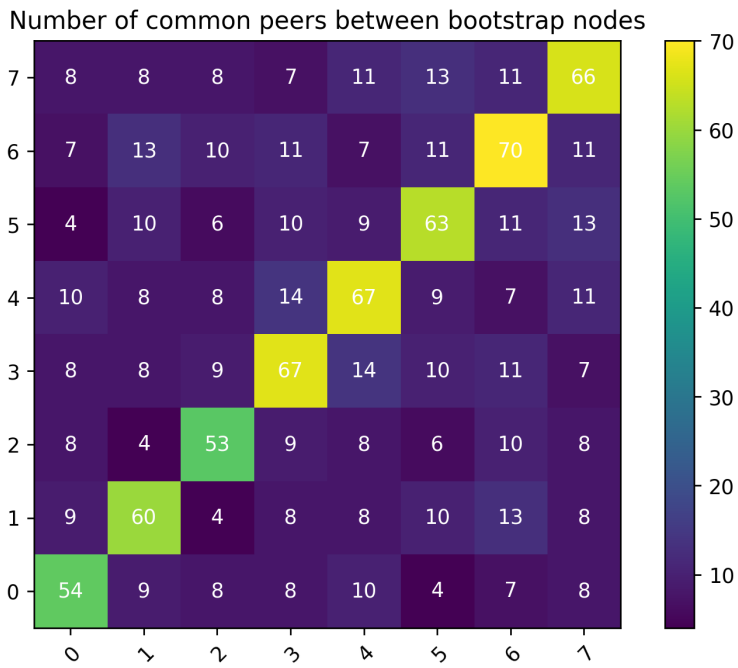

Figure 5.4. Number of peers in common between bootstrap nodes

Moreover, from figure 5.4, we can also see that each bootstrap node has a connection with all the other bootstrap node meaning that the network is fully connected. This is a needed condition in order for the protocol to work.

5.2 Convergences rate

The first propriety we want to analyze is the convergence rate.

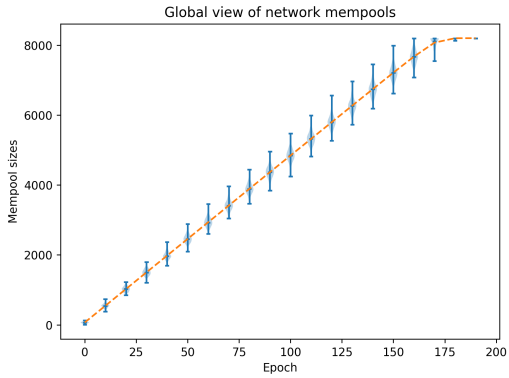

Figure 5.5. 1st simulation convergence rate

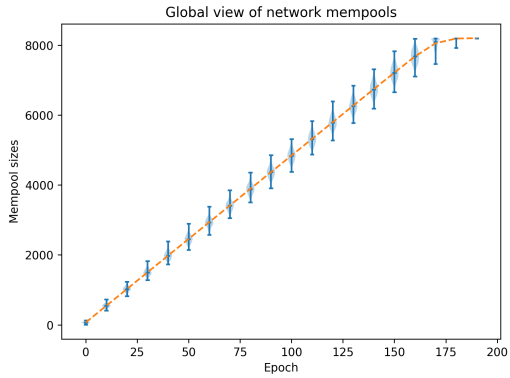

Figure 5.6. 2nd simulation convergence rate

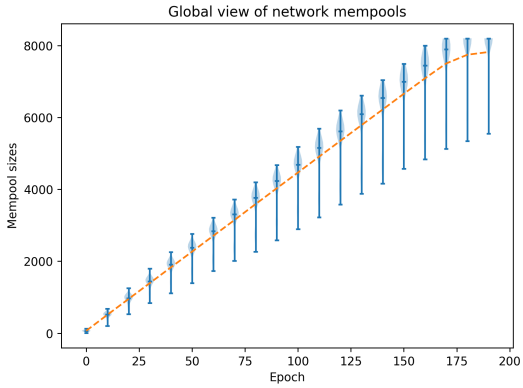

Figure 5.7. 3rd simulation converge rate

As we can see from figures 5.17 and 5.18, the presence of rational nodes does not affect the convergence rate at all until the last 20 epochs. This sounds reasonable as peer's mempool are going to be more full and OPT messages help peers get the last missing updates if they don't have anything to share. Since rational nodes don't accept OPT exchanges, full network convergence is more difficult to reach.
The situation change in figure 5.7, when byzantine actors are introduced. In fact, we can clearly see that more peers are having a hard time receiving updates and that 200 epochs are not enough to reach the same level of convergence of the previous situations. Still, even if more peers are having troubles, most of them have a mempool size above the mean.

5.3 Duplicates

As we want to avoid wasting resources, an important metrics is the number of duplicates updates received.

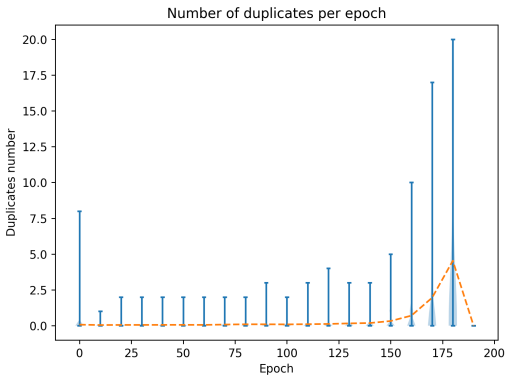

Figure 5.8. 1st simulation duplicate updates

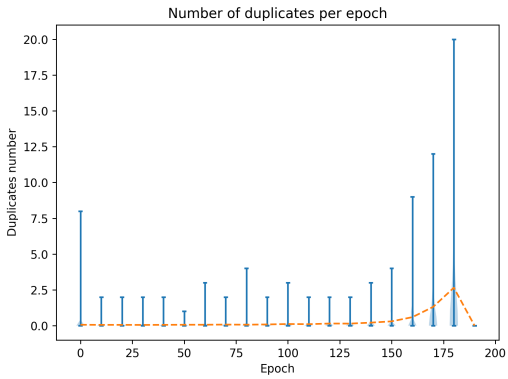

Figure 5.9. 2nd simulation duplicate updates

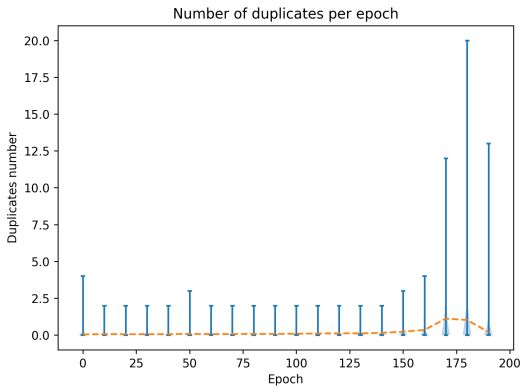

Figure 5.10. 3rd simulation duplicate updates

As we can see from figures 5.11, 5.12, and 5.13 the number of duplicates updates tends to be higher in the final steps of the simulations. This is because mempools are becoming fuller and so, since peers exchange updates concurrently, there is higher probability to receive a duplicate.
Moreover, as more byzantine and rational peer are introduces in the network, number of duplicates mean also decrease. This is because less exchange are made as rational peers don't accept OPT messages and byzantine have a probability of invalidating the exchange.
Still, even if the mean in the last figure is lower than in the other two, we have to remember that the third simulation isn't able to reach full network convergence.

5.4 Exchange types

Another crucial metrics is the percentage of BAL exchange versus OPT and ABORT. Since we want to incentivise peers to execute, as much as possible, the Tic-For-Tac trade we need to see if our simulations support this requirements.

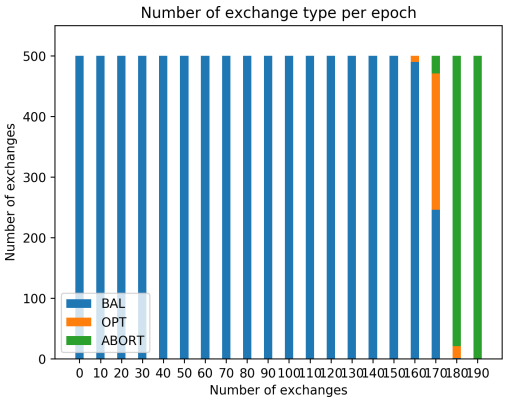

Figure 5.11. 1st simulation exchange types per epoch

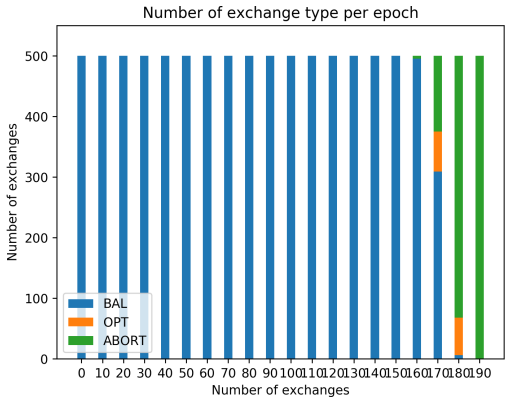

Figure 5.12. 2nd simulation exchange types per epoch

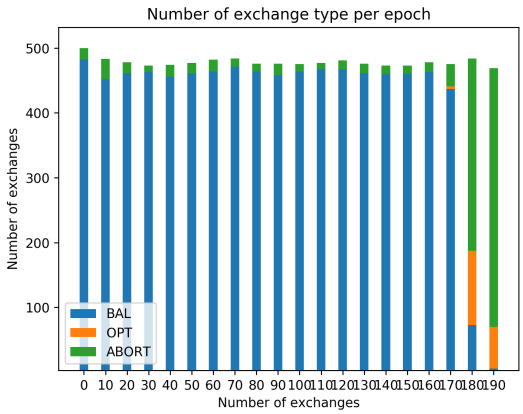

Figure 5.13. 3rd simulation exchange types per epoch

It is clearly visible that the most used exchange is BAL, followed by ABORT and then OPT. Interesting is the fact that OPT exchanges are needed in the last phases of the protocol, when peers are not interested anymore in updates (as they have all of them) and only trades if they are altruistic. In fact, the first simulation is the one containing more OPT exchanged, as everyone is altruistic.
Instead, the last simulation is the one with more ABORT exchange, as part of the peers act byzantine. Due to this high number of abort the network is not able to converge as fast as the two other scenarios.

5.4.1 Number of exchange per epoch

The last interesting metric is the number of non-aborted exchange done in each epoch.

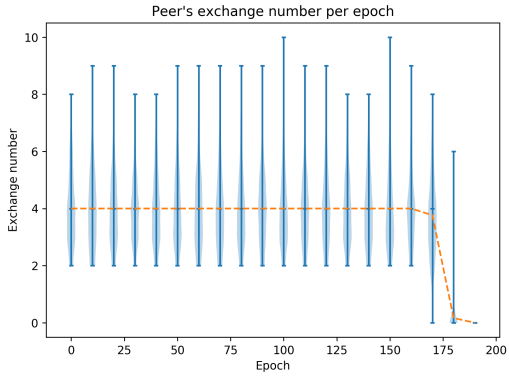

Figure 5.14. 1st simulation number of trade per peer per epoch

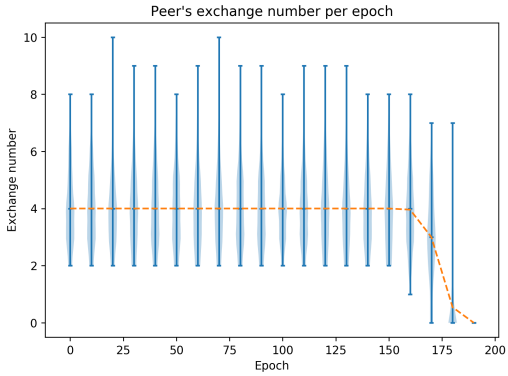

Figure 5.15. 2nd simulation number of trade per peer per epoch

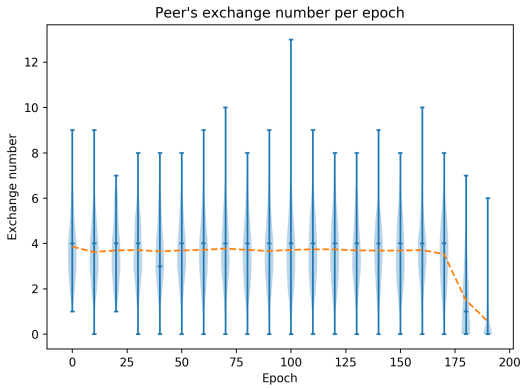

Figure 5.16. 3rd simulation number of trade per peer per epoch

We can instantly see that the presence of byzantine peers decrease exchange number mean in the last figure 5.16. Instead, in the the first two plots 5.15 5.14, the only difference is the steepness of the tail. This is due to the presence of rational nodes who don't want to accept OPT trades which are mostly present in the last epochs, meaning that more trades are needed to reach converence.

5.5 Insights

5.5.1 Connectivity

The number of subscription per peer is one of the most important parameters: too low and some bootstrap node would be low connected to the network or even disconnected meaning

that peers wouldn't be able to receive information from the whole network. Instead, a too high parameters would create too many exchange, meaning a higher number of duplicates and wasted bandwidth.

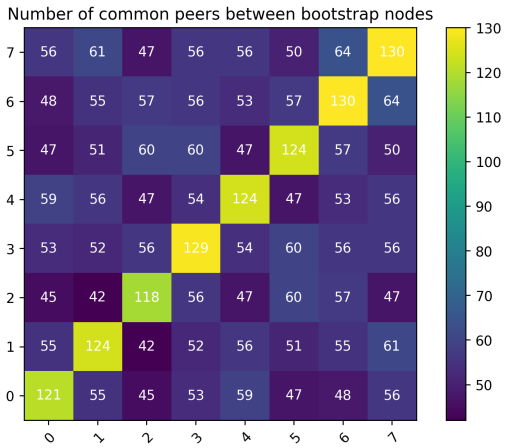

Figure 5.17. 4th Simulation heatmap

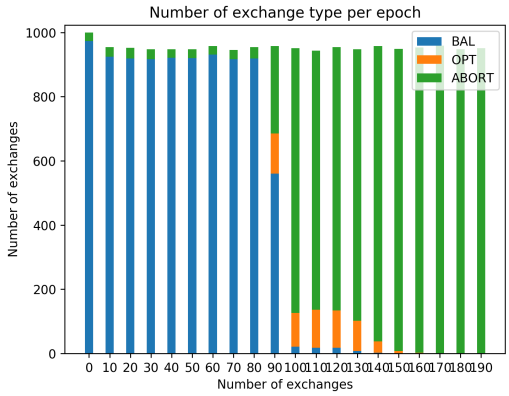

Figure 5.18. 4th simulation exchange types

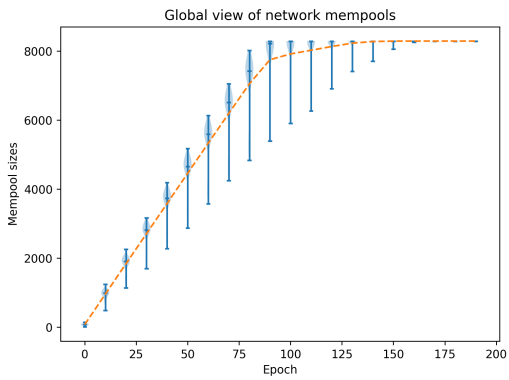

Figure 5.19. 4th simulation convergence rate

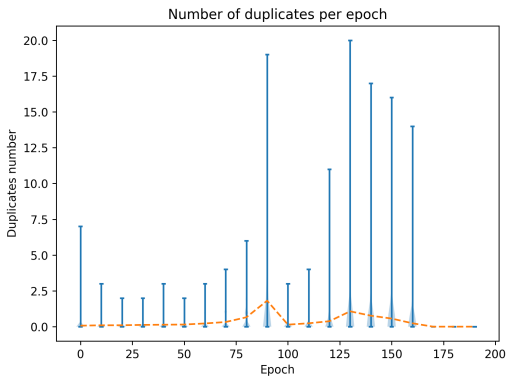

Figure 5.20. 4th simulation duplicates per epoch

In order to prove this arguments we have run a simulation with the same parameters as our third, but a number of subscription of 4.
Due to the fact that, on average, each peer completes 2 trades per epoch per bootstrap node, with 4 bootstrap node we have a total of 8 connection per peer. This drastically increase the number of received updates meaning a faster convergence rate as shown in 5.19. It also means a higher number of duplicates 5.20.
We also want to point out that the a higher convergence rate could be archived by incresing the number of max update sharable in a BAL or OPT trade. Even if this is true, it would limit the fact that we want maximize the total number of BAL updates. Moreover, a less clustered network is more robust and reliable to sybil and eclipse attacks.

Chapter 6

Conclusion

Let us now summarize the report and draw the final conclusion.
We started the first chapter by introducing what are gossip protocols, where are used and why they important, with a focus on blockchain systems.
In the second chapter, we started a discussion on the background needed to understand the major challenges of this work, introducing the concepts of blockchain, gossip protocols and BAR game theory model.
We then moved to the third chapter, where we described in details how BBAR Gossip work, from the description of the network and environment until the message exchanges.
In the next two chapters we first evaluated our simulator using a prototype and then we used the simulator to better understand how our protocol behaves. From the test we have plotted several metrics which gave us some very interesting insights.
In conclusion, we have been able to provide a gossip protocol for blockchains considering the BAR model assumption and adopting techniques derived from BAR Gossip.

6.1 Future works

Many improvements can be made to improve both the performance and reliability of BBAR Gossip. As we already described, sharing the entire history with the partner is a very resource wasting task and needs to be optimized by either share only a subset or by using some better, more suited data structures.
Moreover, other improvements discussed in subsequent researches of BAR Gossip, such as erasure code as done in Flighpath[Li et al., 2008], could be used to generally improve the system.

Bibliography

Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. Cryptology ePrint Archive, Report 2003/235, 2003. `https://eprint.iacr.org/2003/235`.

John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011.

Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1:11, 2014.

Harry C Li, Allen Clement, Edmund L Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 191–204. USENIX Association, 2006.

Harry C Li, Allen Clement, Mirco Marchetti, Manos Kapritsos, Luke Robison, Lorenzo Alvisi, and Mike Dahlin. Flightpath: Obedience vs. choice in cooperative services. In *OSDI*, volume 8, pages 355–368, 2008.

Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology âĂę, 1999.

Atul Singh et al. Eclipse attacks on overlay networks: Threats and defenses. In *In IEEE INFOCOM*. Citeseer, 2006.

Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

Karl Wüst and Arthur Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.