Blochchain BAR Gossip

Reliable and clustered gossiping 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
Finantial 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 my beloved

Someone said . . .

Someone

Abstract

This is a very abstract abstract.

Acknowledgements

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 Type 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 . . . 15
3.5.1 Proof-of-Misbehavior . . . 15
3.5.2 Shuffle peer list each epoch . . . 16
3.5.3 Multiple BN registration per peer . . . 16
3.5.4 Fair exchange problem . . . 16

3.5.5 PoM rewards . . . 16
3.6 Optimizations . . . 17
3.6.1 History exchange . . . 17
3.6.2 Message header . . . 17

4 Evaluation **19**
4.1 BBAR vs BAR . . . 19
4.2 Simulation vs Prototype . . . 19

5 Experiments **21**
5.1 Setup . . . 21
5.2 Convergences rate . . . 21
5.3 Duplicates . . . 21
5.4 Exchange types . . . 21
5.5 Edge cases . . . 21

6 Conclusions **23**
6.1 Discussion . . . 23
6.2 Future works . . . 23

7 Glossary **25**

Bibliography **27**

Figures

3.1 Register phase chart 11
3.2 Exchange phase flow 14

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

Chapter 1

Introduction

During the last ten years, after the born of Bitcoin, 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 structured 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 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 which is typical for most of those systems is the fact that they exchange information through a gossip protocol. Most of the time, this network can be freely joined by anyone, meaning that everyone receives and share updates on the state with other peers. As receiving updates reliably and quickly is a key aspect for these kinds 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 improve the overall of 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 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 payments 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, also called P2P, are heavily employed since they offer a high degree of scalability, fault-tolerance, robustness, fast-spreading and resilience, proprieties which 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 it. Being not able to punish peers for their incorrect behaviour is even worse in cases where this is done on

purpose leading to Sybil or Eclipse 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 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 behavior 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 account peers for their behavior, 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 though 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 going to be structured in the following way.
First of all, the next chapter will cover some key concept 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 chapter and fifth chapter will be used to discussed the validity of the implementation of out prototyped simulation and on the experiments derived.
Finally, in the last chapter draws the conclusions of this work, discussing some possible future works.

Chapter 2

Background

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

2.1 Blockchain

Blockchain is 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. Consensus rules goal 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 meanwhile the former present 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 mechanism 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, Proof of Stake, Delegated Proof of Stake, and Proof of Authority. 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 increase.
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 supermajority, 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 him 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 every information.
These kinds of protocols are usually compared to epidemic diseases as their mathematical proprieties are really similar. One of the main proprieties is the so-called spreading rate, which describes how many hosts get infected as a function of 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 a host will infect each round, 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 he can execute each round and the number of rounds with an increase factor of e^f per round.
Benefits of gossip protocols are:

- High scalability, as messages need on average $\log(N)$ round to reach every node and node 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 have also 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 Type of gossip protocols

There exist several kinds of gossip protocol strategies, but most of them are based on the concepts of pull 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 he wants or not the full payload.
Instead, in pull-based strategies, a node randomly selects a peer in the network and ask 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 trade-off: push-based strategies archive a better convergence rate (as they require one less 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 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

Peer in a blockchain network does not usually need to be registrered 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 the 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 fo peer 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 gossip protocol. Most of the time, the p2p network is used to propagate messaging containing blocks or transactions, but there are also examples of blockchain which use gossip to disseminate message 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, 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 in 2015 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 behaviors, 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 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 in exchange 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 deviating 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 previous one by adding a simple reasoning: considering a real-life scenario, every actor's behavior 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 obligated to exchange information in a Tic-For-Tac fashion, they can ask for updates and never sharing them, increasing the load burden on a smaller subset of peers.
This is way 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 { cite here BAR Gossip}, 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's 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

Is it worth to explain how BAR Gossip works? Maybe it can lead to easier explanation later.

2.5 Problem definition

Is it worth adding this section to explain which are the problem and challenges of implementing BAR Gossip in a blockchain context?

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 is a crucial factor in a blockchain network, meaning it could severely impact overall performances. This chapter will start describing the model assumption. It will then move to describe the several kinds of peers included in the network and their relationship. The last three sections will outline the actual protocol with a focus on implementation details and further optimizations.

3.1 Model assumptions

We are making the following assumption about our model:

- 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: validator and full node.
- 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. Moreover, we need hashing function with two more additional proprieties:

- Pre-image resistance
- Second-image resistance
- Collision resistance

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 usually 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 sequences ofÂărandom numbers. PRNGs cannot be considered really random as their output is totally determined by an initial value called seed. Our implementation follows the Mersenne TwisterÂăPRNG, but the XORShift PNRG 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. Its 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 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, which 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 Δ 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

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

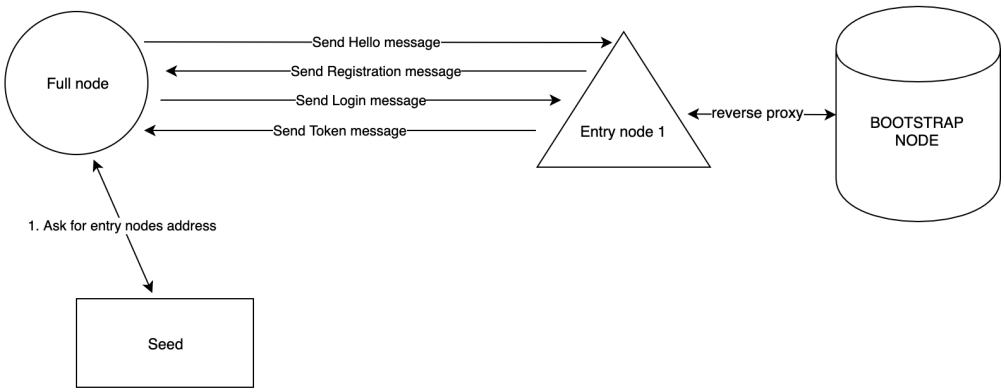

Figure 3.1. Register phase chart

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 entry points addresses. From the list of addresses, the peer will randomly contact a subset of them and start the following message exchange.

- 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], including 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 the bootstrap node and a peer) and bind 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 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 another 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.

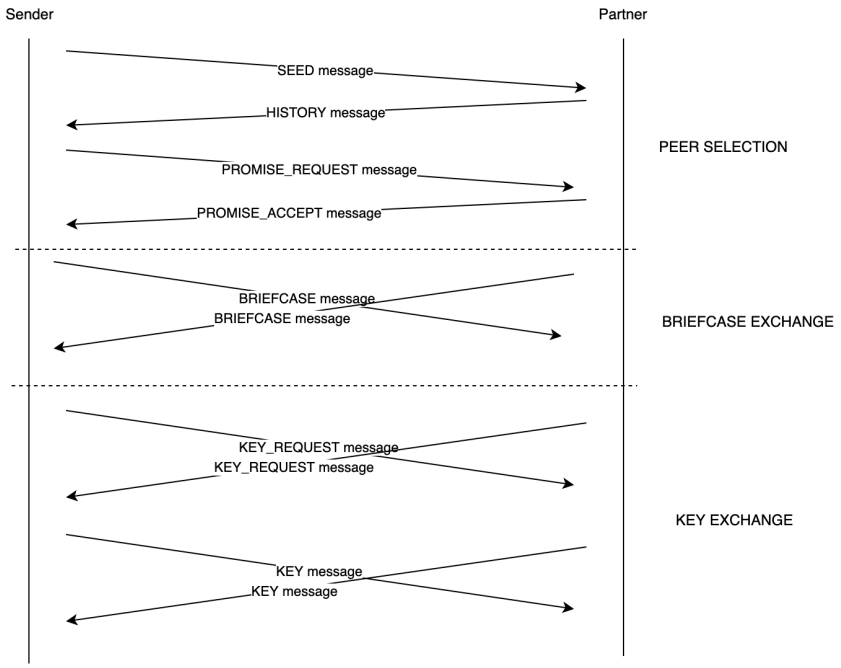

Figure 3.2. Exchange phase flow

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

- A CONNECTIONREQUEST message is sent to the selected partner.
- After validating the connection, the partner sends back a HISTORYDIVULGE 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 message containing a list of updates he wants, he will give back and the exchange type to perform.
- The partner checks if the exchange is valid (it will be explained later what it means), and sends back a PROMISE message, which affirms 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 message.
- Upon receiving a BRIEFCASE message, both peers will start sending a KEYREQUEST message to the partner, until either the epoch finish or until a KEY message arrives.
- The KEY message includes the secret to open the corresponding briefcase, containing the requested updates.

First of all, each message sent during this phase contains a header containing the token plus 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 BARGOSSIP 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 TicForTat 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.
As we want to incentivize the more fair BAL exchange, partner B will add a percentage of random data to the BRIEFCASE message.
A fundamental difference with BAR Gossip is the fact that the peer sending the CONNECTION-REQUEST message, is the one choosing the type of 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.
As in BAR Gossip, all messages are sent through TCP connection expect the KEYREQUEST and KEY message.

	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

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 use generate a PoM and send it to the corresponding Bootstrap node, who will schedule an eviction in the next epoch.

3.5.2 Shuffle peer list each epoch

A Bootstrap node could collude with a peer to particularly craft the order 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 networks into a single one. Otherwise, peers would be able to exchange only the information inside a single bootstrap network which is only a subset of the whole 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. In our case, the BN node acts 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, check the validity of the updates, and if the updates are legit, send 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 behavior. A possible solution is the following: since bootstrap nodes are linked to Validators which are block producers, we can incentivize this behavior by giving 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 containing a list of hash identifiers. This scale linearly with the number of updates. The more straightforward way to decrease the size of the list would be 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 Inverse Bloom Lookup Table or a Set Sketch.
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.
Meanwhile, the following messages could contain just a hash of it.

Chapter 4

Evaluation

4.1 BBAR vs BAR

4.2 Simulation vs Prototype

Chapter 5

Experiments

5.1 Setup

5.2 Convergences rate

5.3 Duplicates

5.4 Exchange types

5.5 Edge cases

Chapter 6

Conclusions

6.1 Discussion

6.2 Future works

Chapter 7

Glossary

Bibliography

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras

ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.