João Pedro Lobato de Carvalho

Master of Science

A Systems Approach to Minimize Wasted Work in Blockchains

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: João Carlos Antunes Leitão, Assistant Professor,
NOVA University of Lisbon

Co-adviser: Bernardo Ferreira, Postdoctoral Researcher,
NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2019

A Systems Approach to Minimize Wasted Work in Blockchains

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To the ones who believed this work would be complete.

Acknowledgements

First of all, I would like to thank my advisors for all the work they put in this dissertation, especially to João Leitão for his help in achieving the solution for this work. Without them, this work would not be possible.

I would like to acknowledge the Department of Informatics of the Faculty of Sciences and Technology of NOVA University Lisbon and the NOVA LINCS research center, which provided the necessary resources to conduct the experiments of this work.

Finally, I would like to thank my friends and family that were always there to support me.

This work was partially supported by Fundação para a Ciência e Tecnologia under the grants UIDB/04516/2020 (NOVA LINCS) and PTDC/CCI-INF/32662/2017 (NG-STORAGE).

Abstract

Blockchain systems and distributed ledgers are getting increasing attention since the release of Bitcoin. Everyday they make headlines in the news involving economists, scientists, and technologists. The technology invented by Satoshi Nakamoto gave to the world a quantum leap in the fields of distributed systems and digital currencies. Even so, there are still some problems regarding the architecture in most existing blockchain systems.

One of the main challenges in these systems is the structure of the network topology and how peers disseminate messages between them, this leads to problems regarding the system scalability and the efficiency of the transaction and blocks propagation, wasting computational power, energy and network resources.

In this work we propose a novel solution to tackle these limitations. We propose the design of membership and message dissemination protocols, based on the state-of-art, that will boost the efficiency of the overlay network that support the interactions between miners, reducing the number of exchanged messages and the used bandwidth. This solution also reduces the computational power and energy consumed across all nodes in the network, since the nodes avoid to process redundant network messages, and, becoming aware of mined blocks faster, avoid to perform computations over an outdated chain configuration.

Keywords: Blockchain, Peer-to-Peer, Overlay Networks, Membership, Message Dissemination

Resumo

Os sistemas de cadeia de blocos e livros-razão distribuídos têm ganho cada vez mais atenção desde o lançamento da Bitcoin. Todos os dias aparecem nas manchetes dos jornais económicos, científicos e tecnológicos. A tecnologia inventada por Satoshi Nakamoto deu um salto quantíco nos campos da computação distribuída e das moedas digitais. Mesmo assim, ainda existem alguns problemas relacionados com a arquitetura da maioria dos sistemas de cadeia de blocos.

Uma das principais limitações nestes sistemas é a estrutura da topologia da rede e a forma como os pares disseminam mensagens entre si, levando a problemas relacionados com a escalabilidade do sistema e a eficiência da propagação de transações e blocos, gastando assim poder computacional, energia e recursos da rede.

Neste trabalho, propomos uma solução original para atacar estas limitações. Vamos propor protocolos de filiação (redes sobrepostas) e de disseminação de mensagens, baseados no estado da arte, para melhorar a eficiência da rede que suporta as interações entre mineiros, reduzindo o número de mensagens trocadas e a largura de banda utilizada. Esta solução reduz o poder computacional e energia consumidos entre os nós na rede, uma vez que os nós evitam processar mensagens redundantes, e, tornando-se conscientes sobre os blocos minados mais rapidamente, evitando computações sobre configurações desatualizadas da cadeia.

Palavras-chave: Cadeia de Blocos, Entre-Pares, Redes de Sobreposição, Filiação, Disseminação de Mensagens

Contents

Chapter 1

Introduction

Cryptocurrencies and Blockchain systems have recently become very active and relevant fields in Computer Science, that gained a lot of exposure and attention [1]. In 2009, Satoshi Nakamoto released the Bitcoin [2] cryptocurrency as open-source software, ten years after this event, many people believe that the technology that Satoshi presented with Bitcoin is revolutionary and can solve many real-world problems (e.g. particularly related with economy) [3].

The technology behind the Bitcoin protocol is called Blockchain. Blockchain emerged with the realization that the technology behind the bitcoin could be used not only as a particular (crypto) currency but also in the context of other applications and domains. This led to a huge interest between financial institutions [4] and entrepreneurs, raising investments to discover how blockchain could impact our daily life and be exploited in other contexts (e.g. healthcare, insurance, voting).

In 2013, Vitalik Buterin described Ethereum in a white paper [5], and launched it in 2015. Ethereum is currently the second biggest public blockchain released to date. The main difference between Bitcoin and Ethereum is related to the type of operations supported by these systems. In particular, Ethereum supports records for assets such as contracts and not just currency. It also can be used to build smart contracts, which have programming capabilities (discussed in Section 2.1.5).

Blockchain systems are not perfect and there are lots of issues regarding the costs and the energy consumption of the devices that participate in the network[6]. One of the main problems is the protocol used for consensus amongst the network (i.e. proof-of-work), which are currently being replaced in most of these systems[7]. If we talk about numbers, at the current date, each Bitcoin transaction consumes 392KWh of electricity, giving a total of 45.76TWh annually. If we rank all the world countries and Bitcoin, Bitcoin is the 53rd "country" that consumes more electricity (for reference, Portugal is the 51st). If

we add Ethereum to the calculations, the sum of both systems would occupy the 45th position[8].

Scalability is also a challenge that has to be tackled by such systems,as most of these distributed systems are aimed at a medium-scale network and fail by design to provide a fully functional and reliable operation at large-scale[9].

There are already many new blockchains proposals that aim at solving most of these problems (we provide and discuss several examples in Section 2.2), but, because of the complexity of these systems, there is still a long way until the emergence of a flawless blockchain. Even though, most of the systems currently available in the industry have active teams that improve those daily, improving many fundamental constructs of these solutions, more fundamental changes to the operation of these systems still have to be researched.

The work conducted in this dissertation is motivated by the following observations in (most) current Blockchain systems:

- The resources wasted due to inefficient message dissemination protocols. There are too many redundant messages being disseminated in the network, spending CPU cycles in participating nodes to process these messages while also impacting the network bandwidth consumption.
- Computational resources waste due to outdated state on nodes. When new blocks are added to the blockchain, if the dissemination mechanism does not allow all nodes to become aware of the new block, nodes will continue to perform computations over an outdated state of the blockchain. Those computations will typically not be useful for the progress of the system.

Therefore, the main goals of this dissertation are: *i)* study in detail how the current blockchain systems enable nodes to join the network and how messages are disseminated through it, *ii)* design an improved dissemination protocol which ensures reliability while reducing the number of messages in the network and lowering overall latency, and *iii)* implement a proof-of-concept to compare with current solutions employed by bitcoin and ethereum systems.

The remaining of this document has the following structure:

Chapter 2 presents an introduction to Blockchain systems and the current systems available, the underlying network protocols in Bitcoin and Ethereum, what is message dissemination and overlay networks, and the solutions found in the literature

Chapter 3 describes our proposal to improve the dissemination layer of the blockchain systems. We introduce some key features of the protocols we use and explain why they are beneficial for this class of systems. We also detail how our proposal operates and how it will solve the waste of resources.

Chapter 4 evaluates our solution against the currently popular schemes used in blockchain systems. We give some details on the simulator implementation and configurations, present the results for the selected metrics and briefly discuss the results.

Chapter 5 concludes the dissertation and suggest directions for further work in the future.

Chapter 2

Related Work

This chapter overviews the relevant state of the art related to the goals of the project (as discussed previously).

In Section 2.1 we start with fundamental concepts related to blockchains and the architecture behind it. Section 2.2 presents the most common blockchain systems in production now-a-days and discusses how they address the challenges related to privacy and scalability. We will also explain in depth the mechanisms used in Bitcoin and Ethereum to create and manage the overlay topology and disseminate messages throughout the network. In Section 2.3 we will explain what is message dissemination and present some relevant solutions found in the literature. In Section 3.1 we will explain what is a membership protocol, and present some fundamental concepts related to them and survey a few concrete solutions found in the literature. Finally, Section 2.5 concludes this chapter with a discussion on the key points of this state of the art survey.

2.1 Blockchain General Concepts

The idea of a distributed virtual currency is older than the Bitcoin system being constantly iconized in the Cyberpunk pop culture and the mind of every Cypherpunk activist expressed through their manifesto *"We must come together and create systems which allow anonymous transactions to take place."*[10].

The appearance of Bitcoin gave us a key advancement towards a distributed virtual currency, but also gave the society at large a better understanding on how we can use this technology, also known as Blockchain, to store permanent records in a decentralized and public network. Even though we now have the technology, it is still complex to take full advantage to build relevant applications on top of it. We now introduce the fundamental operation of the Blockchain technology.

2.1.1 Blockchain Structure

A blockchain system is nothing more than a distributed and public (any node in the system can access it) ledger. This ledger is leveraged to store every transaction (valid) that is executed between any user.

Apart from keeping track of all the transactions, there is additional key functionality in this ledger: any user in the system can add transactions to the ledger. With this functionality arises one big challenge, if we do not trust the remaining users of the system, how can we assure that they will not add transactions without the approval of the other parties involved? To tackle this challenge the blockchain solutions typically rely on digital signatures, being every transaction signed by the sending user. To guarantee that the signatures are not forged and that transactions cannot be replayed, every transaction is also numbered and use public-key cryptography to sign the transaction, this way, even if we have similar transactions, the signature on the transaction will be different.

To distribute the ledger, every node in the system keeps a full copy of it. When some node wants to execute a transaction, it needs to broadcast the same transaction throughout the entire network and every node will add it to their local ledgers after they verify the sender has enough currency to execute the transaction (as well as validating all relevant cryptographic signatures). This is a naive approach, in the real world we can not expect that the message will be delivered nor guarantee the order messages are received by different nodes in the network. Therefore, we can not assume that all the ledgers in the network are equal and can not take different decisions regarding approving or reject particular transactions. To solve this problem, the nodes need to achieve consensus (explained in Section 2.1.3), which is the process of agreeing on a single order for processing all submitted transactions. Originally in the Bitcoin system, a consensus mechanism presented in Hashcash[11] was employed to this end, later labeled as proof-of-work (we will detail other consensus mechanisms also in Section 2.1.3). The consensus mechanism has two fundamental functionalities, it provides an extra layer of security and provides a consensus regarding the evolution of the system state. Therefore, instead of having a ledger where nodes insert broadcasted transactions, transactions are grouped in blocks and those blocks are linked together, effectively building the ledger as a chain of different blocks: the blockchain.

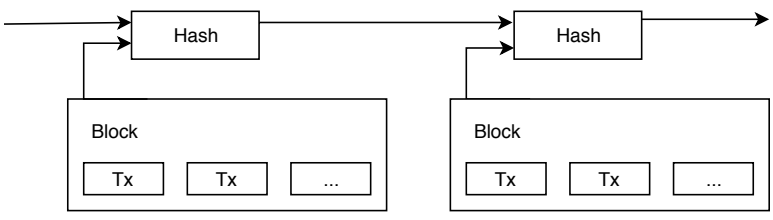

Figure 2.1: Blockchain schematic as in Bitcoin paper

All nodes in the system can create blocks with the transactions that are being broadcasted through the network (and not inserted in the blockchain yet). To create new blocks, the node has to give proof of his work, which involves a hard cryptographic puzzle that can be easily verifiable. In the case of Bitcoin, the crypto puzzle is based on calculating a number that can be added to the end of the block such that the hash of it will have a pre-configured number of zeros. To guarantee the order of the blocks, the block needs to have the hash of the previous block as parts of its contents (see Figure 2.1). In the end, a block becomes composed of the hash of the previous block, the transactions, and the proof of work. After a block is successfully generated it is broadcasted throughout the network, where it will be approved or rejected after other nodes confirm the transactions and the proof of work. If a node receives two different blocks the chain might diverge. Nodes will always select the sub-chain that is larger since that sub-chain has more effective work invested in it; if both sub-chains have a similar length, the node will wait to receive another block broadcast that will make one of these chains longer than the other, therefore, to approve a transaction, a node must wait until some blocks are chained on top of the one that contains the transaction, to make sure that the chain where the block with the transaction was inserted will prevail.

Every time a node computes a new block, there is a special transaction on top of it, which is, the attribution of a pre-defined amount of the currency for the block creator (also called the miner). This amount is issued by the system itself (called the block reward) and is used to encourage the nodes to actively participate in the network. This whole system model is fail-proof unless most of the nodes in the system are not trustworthy, as we can see in a recent attack against Ethereum classic[12], where the attacker could get more than 51% of the total network processing power, producing a "trustworthy"chain. This attack caused $1.1 million dollar losses and raised concerns about security on blockchain systems.

In the following sections, we will describe how blockchain systems can be classified concerning some design decisions.

2.1.2 Permissions and Access to Data

Regarding the permissioning, we can divide the blockchain systems into two main categories[13]:

Permissioned blockchains where only a restrict number of nodes can contribute to the progress of the system and the processing of transactions (including generating new blocks to the chain).

Permissionless blockchains where there are no restrictions regarding the identities of the nodes, thus everyone can mine and create blocks (including submitting and validating transactions).

Blockchain systems can also be public or private, usually, permissioned blockchains are also private while permissionless blockchains are public[14].

In a **public blockchain** everyone can download the blockchain ledger and read all the transactions and data.

In a **private blockchain**, only a set of predefined users can read the data and download the ledger.

Permissionless systems let any node that is willing to contribute with processing power to be part of the network, without requiring to prove its identity. This is a perfect scenario for large scale distributed systems removing the centralization of the power to control the network. However, this scenario comes also with a huge cost, regarding the consensus mechanism available, because many nodes are not trustable, which requires longer and more computationally expensive consensus schemes.

2.1.3 Consensus Mechanisms

The consensus mechanism allows nodes to agree on the outcome of all the transactions submitted to the system to ensure all nodes reach the same state. The mechanism has to ensure that the transactions only will be added if they are valid and that the transactions are not recorded more than once.

There are several mechanisms used by the community, amongst those, there are three main mechanisms, the proof of work, proof of stake, and the practical use of byzantine fault tolerance distributed protocols.

Proof-of-Work[15] uses a cryptographic puzzle that requires processing power to solve it and determine the winner, usually the complexity of this puzzle increases with the number of times it has been solved.

Proof-of-Stake[16] determine the winner using a combination of randomness and the stake weight (coins that a user has in their wallet), stake based on age can also be used. In non-cryptocurrency blockchains, the Proof-of-Stake is replaced by alternative approaches such as voting, since there is no stake by default.

Practical Byzantine Fault Tolerance mechanisms[17] are often used in permissioned blockchains (e.g. Hyperledger[18]), where new blocks are added if more than $\frac{2}{3}$ of all peers effectively validate them.

Proof-of-Work was the original consensus mechanism used in Bitcoin, but with the growth of the coin value, there was also a significant increase in the number of miners, which led to a huge need for computational power to mine new bitcoins. This need for computational power was proved a waste of resources and energy[19]. Practical Byzantine Fault Tolerance mechanisms are a very good consensus mechanism, but it is not usable in a network where more than one-third of the peers are malicious. Proof-of-Stake also have some limitations regarding the centralization of the mining due to the centralization of the stakes, although, Vitalik Buterin (co-founder of Ethereum and co-founder of Bitcoin Magazine) suggested a hybrid approach to this issue where blocks are produced via proof-of-work and signed by several stakeholders[20], solving the disadvantage of proof-of-stake (because of its high dependence on proof-of-work).

2.1.4 Currency

The blockchain system emerged with the Bitcoin[2], the first modern cryptocurrency designed to prevent the weaknesses in their predecessors, such as double-spending attacks and centralization of the ledgers, that were limiting aspects towards scalability and potentially to privacy[21]. Being a cryptocurrency, it provided a mathematical way to generate currency that is nearly impossible to falsify[22]. The system also specified how the generated currency should be distributed among the users of the network and the operations available to the users (send currency to each other).

Many blockchains came after Bitcoin (most of them not designed for trading, but applications) and most of them keep the idea of providing a currency available for the users of the system, mainly for the maintenance, giving it as a reward to the users (mainly the ones who process the transactions).

2.1.5 Smart Contracts

Smart contracts[23] are similar to contracts in the real world, except they are digital. They are computer protocols intended to facilitate, verify, and enforce a contract. To achieve these goals, smart contracts have two properties, they are immutable and distributed.

With smart contracts, blockchains gain a programming capability where they can build and deploy decentralized applications. This comes from the development of the blockchain system, where the developers provide a Turing-complete scripting language (e.g.: Ethereum[24]). Some blockchains have limited programming capability, making it harder to develop applications on top of them (e.g.: Bitcoin).

2.2 Blockchain Systems

After Bitcoin, there was a sudden emergence of alternative systems (also known as altcoins) that in one way or another improved the original features of Bitcoin or just tried to solve some of the limitations of the existing blockchain systems.

2.2.1 Systems Overview

Litecoin[25] and Dash[26] appeared as alternatives to Bitcoin to improve the high transaction latency, reducing the block mining time and allowing faster transactions. Peercoin[16] presented proof-of-stake as an alternative to the proof-of-work approach, changing how the consensus operates in the network. Ripple[27] proved the application of blockchain technology in the financial industry with its real-time gross settlement system. NEM[28] demonstrates that it is possible to have multiple ledgers in the same blockchain and provided a new version of proof-of-stake: the proof-of-importance.

Ethereum gave us a Turing-complete scripting language proving that it is possible to perform full computations across the blockchain, although it was possible to perform computations in prior blockchains, it was unnoticed until the appearance of Ethereum.

IOTA[29] appeared to try to mitigate the scalability problems with the Tangle concept. Instead of having the transactions grouped into blocks, IOTA allows transactions to be entangled together. This project was directed to the Internet of Things applications, but it can also be applied to other application domains.

Regarding privacy, Monero[30] was devised with privacy in mind. The main feature of this system is the usage of an obfuscated public ledger where anybody can broadcast or send transactions to, but no one (other than the sender and the receiver) can tell the source, amount, or destination. Enigma[31] provides privacy through secret contracts. Secret contracts are smart contracts that enforce the correctness of the system and privacy since they can hide the data completely from other nodes. Enigma uses secure multi-party computation[32], which lets the different peers perform computations over their data while maintaining their inputs secret.

At the end of 2015, the Linux Foundation announced the Hyperledger Project[18] which consists of a group of tools and frameworks for the development of blockchain-based systems. This project is also supported by the research community and industry (e.g. IBM and Intel), giving it high credibility with the big names involved in the project. The most famous framework of the project is Hyperledger Fabric[33], that provides blockchain infrastructures for developing applications on demand.

2.2.2 Bitcoin

We already discussed the design and core features of Bitcoin[2], this is because of the importance of this system as a pioneer in the domain of blockchain. It was the first blockchain system applied in the real world and it is still reliable after more than 10 years of operation.

We will now present and explain the underlying mechanisms of the overlay network/membership systems of the Bitcoin system.

Join Mechanism

Nodes join the network[2][34][35] by discovering a bootstrap node and sending it a version message (with the node version number, block, and current time). The bootstrap node will respond with his version message and then both nodes send an acknowledgment message indicating that the connection has been established.

After the initial connection is established, the new node sends a *getaddr* message to gather additional peers. When a node receives a *getaddr* request it sends the addresses that have a timestamp in the last 3 hours, to a maximum of 2500 addresses (selected at random). Nodes also attempt to keep a minimum number of connections p (this is a protocol parameter, in the case of Bitcoin is 8), if the number of open connections is

below p, the node will try to connect with known addresses, these peers do not close open connections even if they more than p open connections. Therefore, the total number of open connections is likely to be higher for nodes that accept incoming connections (nodes behind firewalls will have p open connections, since they are not directly reachable by other nodes in the network).

With the current join mechanism, the overlay network will be unstructured, forming a random graph, with potentially a very high degree (i.e. each node will have a large number of neighbors), and with some nodes becoming extremely popular in the network[36].

Node Discovery

In order to discover other nodes[2][34][35], there are many possibilities regarding which methodology to employ. When a client joins the Bitcoin network for the first time, it can try to discover neighbors through:

- **Hard coded DNS seeds** The client has a group of hardcoded DNS hostnames. After querying one or more of those DNS services, it expects a response with multiple node addresses that may accept new incoming connections. These addresses will initially have a timestamp equal to zero, therefore they will not be announced in response to *getaddr* messages.
- **Hard coded seed addresses** The client has a group of hardcoded seed addresses that may be used as a last resort. There are mechanisms to cancel the connections to these nodes as soon as possible to avoid overloading them. These addresses will initially have a timestamp equal to zero.
- **Command line addresses** The user can specify multiple IP addresses through the command line. These addresses will initially have a timestamp equal to zero.
- **Text file addresses** The client can have multiple IP addresses in the file "addr.txt"under the bitcoin data directory. These addresses will initially have a timestamp equal to zero.

If the client is currently in the network, there are additional mechanisms to discover additional overlay neighbors:

- **Local client external address** The client discovers is external address (using a third-party service) and announce its public address to the network.
- **Connect callback address** After making the initial connection, the client issues a *getaddr* message to gather more peers.
- **Ongoing addr advertisements** The client can listen to ongoing *addr* advertisements since nodes advertise addresses in different situations (when they relay them, when they advertise their own, and when a new connection is made).
- **Addresses stored in a database** The client also adds peer addresses to a local database, if it needs additional peers, it can try to connect to the peers in its local database.

Nodes Leave and Failure

Bitcoin does not have a way for nodes to leave the network[2][34][35]. Nodes leaving the network and failing will have their addresses lingering in the network for some time. Nodes by default send a heartbeat message to peers from time to time (before 30 minutes of inactivity). If a node does not receive a message from a peer for more than 90 minutes, it will assume the connection with the peer has failed and it will purge it from their known peer addresses set.

Message Dissemination

To disseminate transactions and blocks in the network, the bitcoin node announces their availability with an *inv* message to its neighbors (once verified)[2][34][35][37]. If a neighbor does not have the transaction or the block, it will send a *getdata* message to the original node with the hashes of the information being requested. After that, the information is sent to the neighbor. Therefore, this information is propagated using a lazy push gossip approach[38] discussed further ahead in the document.

The message dissemination efficiency in the current model will depend heavily on the topology of the network, that in this case, it is fundamentally random.

There was also recent scientific research to improve information dissemination in the Bitcoin network[39] based on neighbor rankings. It showed an improvement in the performance when compared with the techniques employed today for information dissemination, but the neighbor ranking did not provide a relevant improvement in some of the metrics (e.g. bandwidth consumption).

2.2.3 Ethereum

Ethereum is an open-source and permissionless blockchain system that enables users to build and run smart contracts and distributed applications (ĐApps). The Ethereum blockchain implements a Turing-complete language providing a good framework for users to build smart contracts and ĐApps. These applications are offered the possibility of creating tokens, therefore, most of the startups that launch tokens, are issuing a distributed application over the Ethereum network.

The principal characteristic of Ethereum is its active community and the continuous evolution of the Ethereum protocol that strives to optimize many of the different subprotocols in use on their blockchain.

Node Discovery and Join Mechanism

Ethereum implements its own peer discovery protocol[40], based on the Kademlia protocol[41]. Originally, Kademlia was designed for storing and locating resources in a peer-to-peer network and also as a mechanism to locate particular nodes. In the Ethereum network, it is only used for node discovery and topology maintenance. The only RPC

calls enabled in the discovery protocol are *Ping* and *FindNode*, where *FindNode* accepts the node identifier (not Kademlia ID) as an argument. The Kademlia ID is calculated with the hash function applied over that node identifier (generating a 32-byte value) which is used to calculate the "distance" between nodes (leveraging XOR-based computation).

The Kademlia's routing table consists of a set of lists (or *k-buckets*) where each *k-buckets* holds a maximum of *k* entries. The number of *k-buckets* is the same as the number of bits of the Kademlia ID. Each *k-bucket* corresponds to a distance to the other nodes, this method guarantees that the node will have different peers at different distances, improving the reliability of their neighboring connections and the routing on the Kademlia network.

For a node to discover a bootstrap node (when he wants to join the network) he has two alternatives. He can get one bootstrap node from a hardcoded list in the implementation or he can ask a well-known discovery endpoint set for that particular purpose (some of these endpoints are maintained by the Ethereum Foundation).

When the joining node finds a bootstrap node, it starts a join process where it will initialize its buckets and querying the bootstrap node for neighbors (eventually removing the bootstrap node to avoid overloading it).

With the current join mechanisms, the overlay of the Ethereum network is fundamentally structured. Its structure is similar to the topology of the Kademlia DHT-based network.

Nodes Leave and Failure

When a node wishes to leave the network[42], it will send a *Disconnect* message to its peers. This message receives a *reason* parameter where nodes can tell their peers if they are leaving the network or if there is another reason for the disconnecting (e.g. incompatible P2P protocol version). After sending the message, hosts wait a short time before disconnecting themselves, for peers to disconnect first.

In Ethereum there is no mechanism to handle node failures, but a node can easily detect if one of its peers has failed the next time it sends him a message (because of the use of TCP connections[43]). Even though, this mechanism can lead failed nodes to remain in the buckets of correct nodes for long periods, leading to a low accuracy of the views of the system maintained by each node in the logical network[44].

Message Dissemination

There are many different types of messages in the Ethereum protocol[42] but the three key messages are the *NewBlock*, the *NewBlockHashes*, and the *Transactions*.

If a node has a block that the peers should know about, it will send a *NewBlock* message with the block itself.

When a new block (it can be more than one) appears on the network, the node sends a *NewBlockHashes* message to his peers. This message can have a maximum of 256 block hashes. If the peer does not have at least one of the blocks corresponding to those hashes,

it sends a *GetBlocks* message requesting the blocks unknown to him, then it will receive a *Blocks* message with the contents of the requested block.

When a node has a transaction (it can be more than one) that the peers should include in their transaction queue, the node uses a *Transactions* message.

The efficiency of the message dissemination protocol, likewise to the bitcoin system, is highly dependent on the network topology. However the Ethereum network, due to its structured nature, appears to be more efficient than the Bitcoin network for message dissemination.

2.3 Message Dissemination

2.3.1 Dissemination Protocols

Dissemination protocols define how messages are disseminated throughout the network. One of the typical examples of a dissemination protocol is the flooding mechanism, which will flood all the network links with the message being broadcasted. Of course, this is not efficient and many protocols achieve better network usage compared with flooding. We will now present key data dissemination schemes and how they define the way messages travel over the overlay network topology.

Publish-Subscribe

Publish-Subscribe[45] is an architectural messaging design pattern. With this pattern, the peers who send messages to the network (*publishers*) and the peers who receive the messages in the network (*subscribers*) do not have to know each other. Publishers categorize the messages with two different subscription models: *topic-based* and *content-based*. Subscribers express their interest in a category of messages (based in the subscription models being employed) and only receive messages related to that (or those) category (or categories). The process of selecting the subset of messages for each subscriber is called filtering.

In *topic-based* solutions, messages are classified into different topics. Subscriptions include all the topics of interest to the subscriber. When a message has a topic that matches with a peer subscription, the message is delivered to the subscriber.

In *content-based* solutions, messages receive multiple attributes where subscriptions are allowed to define ranges over these attributes. When the attributes of a message match the specification of the subscription, the message is delivered to the subscriber.

The advantages of the publish-subscribe model are:

- There is no need for peers to know the existence of each other or the rest of the system topology. If the messages are broadcasted through the network correctly, it is guaranteed that the peers will receive the messages they subscribed to.

- It provides better scalability than centralized approaches (e.g. client-server model). This advantage comes from the distributed system point of view since there is no overload in a central peer and there is the possibility of doing parallel operations.

There is also a big weakness in this model related to the first advantage. As stated before, the messages need to be broadcasted through the network, correctly. Such a system needs to provide strong guarantees of message delivery, otherwise, peers may lose most of the messages disseminated in the system.

Gossip Protocols

Gossip protocols[46] (also known as anti-entropy protocols or epidemic protocols), are an efficient way to spread messages over the network. They were proposed to solve many problems in distributed systems (e.g. publish-subscribe problems[47]).

In a gossip protocol, all the peers collaborate to disseminate information through the network. When a node wants to broadcast a message, it sends the message to several random nodes (the number of nodes is parameterizable, and usually called *fanout*). When a message is received, there are two scenarios, the node receives the message for the first time or has already received it. In the first scenario, the node will forward the message to several random nodes (typically using the same *fanout*). In the second scenario, the node simply discards the message[36].

Likewise to Publish-Subscribe, gossip protocols are foundations for highly scalable systems. It also provides robustness to systems because of the intrinsic redundancy in gossip-based dissemination, being able to mask network omissions and node failures.

It is also important to note that there are different strategies to implement gossip protocols:

- **Pull:** Nodes query random peers for information, if the peers have new information (e.g. messages), they forward it to the node.
- **Eager push:** Nodes send information to random peers as soon as they receive new information.
- **Lazy push:** Nodes send a unique identifier to random peers when they receive new information. If the receiving peer does not have the corresponding information, it will make an explicit pull request.

These strategies are good for specific scenarios. There also exists the possibility of combining different strategies to build hybrid approaches.

Epidemic Broadcast Trees

The Epidemic Broadcast Trees[38] paper presented a gossip protocol, named push-lazy-push multicast tree (or *Plumtree*), that shows how to mitigate the message redundancy in gossip-based dissemination protocols without compromising their robustness. To do that, the authors rely on a multicast tree that covers all nodes in the network, effectively

achieving a structured overlay on top of a random overlay (we discuss overlay networks further ahead).

The *Plumtree* protocol has two main components:

- **Tree construction:** responsible for choosing the links in the random overlay network that will forward the messages using an eager push strategy.
- **Tree repair:** responsible for repairing the tree when node failures happen. The main objective is to guarantee that all nodes are covered by the spanning tree.

After constructing the spanning tree, each node maintains two sets of peers, one destined to use eager push gossip and the other to use lazy push gossip. The whole process of creating both of these sets guarantees that: *i)* when the first broadcast is terminated, a tree is created, *ii)* assuming a stable network, the generated spanning-tree minimizes the message latency (to the sender of the original message), and *iii)* there is a decentralized and timely mechanism to detect and recover from failures (thanks to lazy push messages).

This protocol offers high reliability even in the presence of a large number of faults. It also showed lower latencies and less message redundancy with a low-cost scheme to build and maintain the topology.

2.4 Overlay Networks

In the context of peer-to-peer systems, we can define an overlay network[48] as a virtual (or logic) network deployed on top of another (typically the physical) network to improve the utilization of the information and the resources in the system. This virtual network is composed of many logical links between the different peers that compose the system, independent from the physical network. Based on the organization scheme and the graph formed considering the nodes and the links between them, we can classify an overlay network as unstructured or structured[49].

Structured Overlays are created with a well-defined organization scheme as the basis, typically, by relying on unique identifiers for each node, having very predictable topologies. Because of the well-defined structure of these networks, we can easily find the location of a node even without a whole image of the system. These networks usually have a lower network diameter, enable more reliable links between nodes that can be organized considering a specific performance metric (e.g. latency or hop count) and also enable easier content finding through the whole network since such contents can be indexed based on node identifiers.

Unstructured Overlays usually have a random topology and are unpredictable, even with complete information about the system filiation, in the sense that even knowing all the details about one joining node, one can not predict where the node will stay in the network and what will be the nodes that will be connected to it. Because of this random nature (that imposes very few restrictions), these networks are often more reliable in scenarios where a large number of nodes may fail or where a large number of changes in the overlay topology can happen (e.g. in churn scenarios)[50].

2.4.1 Distributed Hash Tables (Structured Overlay Networks)

One of the best examples of structured overlays are Distributed Hash Tables (DHTs). They are a decentralized system that enables the same lookup features as a Hash Table, mapping keys to values.

At the end of the nineties and the beginning of the new millennium, many peer-to-peer systems appeared to take the full capabilities and resources of the Internet to build file-sharing services, some examples of these systems are Gnutella[51], Napster[52] and BitTorrent[53]. Some of these systems were not fully decentralized, relying on a centralized server to index the files in the network, this model would maintain the issues related to single points of failure[54]. After that, there was a motivation to research DHTs to provide the decentralized operation and efficiency features.

The appearance of protocols such as Chord[55] and Pastry[56] ignited the DHTs as a hot research topic, enabling the appearance of advanced protocols like Kademlia[41], the overlay protocol used by the Kad network[57].

The main reason for the boost of popularity of this type of overlay network is its high flexibility and scalability properties, providing an application-level routing infrastructure with minimal information about the membership of the network.

The Hash Table has two functions **get(key)** and **put(key, value)**. When we call the get function, the Hash Table will hash the key to find the bucket that has it, in a DHT a key is hashed to find the node responsible for that key (e.g. the name of the file we want lookup in file-sharing services). To ensure that every time that the number of nodes in the system changes, we do not have to redistribute all of the items again over objects, one usually resorts to consistent hashing[58].

2.4.2 Membership Protocols (Unstructured Overlay Networks)

Some membership protocols are applied to unstructured overlay networks to define a management strategy. We will now give examples of membership protocols and how they manage networks.

One key algorithm used by these protocols are random walks[59]. A random walk is a type of random process (or stochastic process), which involves taking a series of steps where each step is determined with some random probability. Random walks are used in many different fields apart of epidemiology (e.g. physics).

Cyclon

Cyclon[60] is a gossip-based membership management protocol designed to manage unstructured overlays, building graphs with low diameter, low clustering, similar node degrees and being resilient to node failures (in terms of overall connectivity). Its operation is based on a partial view of the whole network with a parameterizable fixed size

maintained individually by each node in the system, nodes in the partial view of a node are considered their neighbors.

The protocol uses a shuffle algorithm to continuously change the neighbors of a peer from time to time. The algorithm works as follows, periodically, a node increments the neighbors age and then chooses the oldest node in its neighbors and exchanges a subset of its neighbors with a subset of the other node neighbors. If the other node does not reply to the shuffle message, the node will assume that the node failed and it will remove it from the neighbors when a node executes one of there steps it also creates a new identifier with age sera that is sent alongside the scope of its partial view. There is no distinction between nodes leaving and nodes failing.

To join the network, a node has to know another node that belongs to the overlay. The join operation is done with random walks on the overlay. This process has an important observation: *"... due to the randomness of the connectivity graph, a random walk of length at least equal to the average path length is guaranteed to end at a random node of the overlay, irrespectively of the starting node."*, with this property, there is also a guarantee that the in-degree of all nodes will remain unchanged.

Scamp

Scamp[61] is a gossip-based membership protocol characterized for removing the need of complete knowledge of the global membership and having two partial views, one where the nodes select the target nodes for sending gossip messages (*outView* for reference) and other with the nodes from which they receive gossip messages (*inView*). Scamp has a scheme where the size of the views converges to an optimal value for gossip to succeed. This value is a function of the system size (we have to keep in mind that these nodes do not know the system size).

When a node wants to join the network, it sends a subscription request to a member of the overlay network and puts that node in its *outView*. When a node receives a subscription request, it forwards the node identifier to all the nodes in their *outView* and creates a fixed number (design parameter to determine the proportion of failures tolerated) of additional copies that are sent to randomly chosen nodes also in the *outView*. When a node receives one of the forwarded subscription requests, it adds the new subscriber in its views with a probability depending on the size of its local partial view, otherwise, it forwards the subscription to a random node in its *outView*. To prevent node isolation, there is a periodic check mechanism that nodes can use.

When a node wants to leave the network, it sends an unsubscription message, which is disseminated through the whole network as a gossip message. When a node receives an unsubscription message, it removes the node from its views (only if it is present in its views, of course). If the node fails ungracefully, the protocol guarantees that eventually, all other nodes in the system will remove the failed node from their partial views.

HyParView

HyParView[36] is also a gossip-based membership protocol that addresses the challenge of minimizing the restoration time in the presence of large numbers of (concurrent) node failures, ensuring high levels of reliability in such scenarios. HyParView uses two partial views, one with the peers with whom that node communicates (*active view*) and another with different peers from the whole overlay (*passive view*), used for fault recovery.

When a node wants to join the network, it sends a join request to a node already present in the system. A node that receives a join request adds the new node to its active view (if it has to drop a node from the view, it sends a disconnect message to the node being removed from its views) and it will forward the join request to all its active view nodes. This request will be propagated through the overlay using a random walk. The protocol has two configuration parameters which define when a node will be added to the active and passive views, these are the *Active Random Walk Length* and the *Passive Random Walk Length*, respectively.

There is no distinction between leaving or failed nodes. Nodes have a reactive mechanism to maintain their active view always updated. When a node suspects that one node in their active view has failed, it will remove it and substitute for one in the passive view, removing every node that it fails to connect to from both views. When sending a neighbor request, the node can set the priority as high (if the node does not have any elements in its active view) or as low otherwise. When receiving such requests, if the priority is set to high, the node will be added in the receiver active view (if it has to drop a node from the view, it sends a disconnect message), otherwise, it will accept the request only if it has a free slot in its active view.

To maintain the passive view, nodes perform a shuffle operation with random peers. The node will create a set with its identifier, a subset of the active view and a subset of the passive view (the numbers of elements for each subset are protocol parameters). After the creation of this set, the node sends a shuffle request to a random peer which will be propagated using a random walk, likewise to peer identifiers in the join request. The node at the end of this random walk will send a shuffle reply with a subset of its passive view, with the same number of peer identifiers as in the original shuffle request. In the end, both nodes will integrate the received shuffle messages in their passive views, ensuring that all the nodes in the overlay have a diversified passive view.

2.4.3 Topology Mismatch Problem

The topology mismatch between the overlay network and the underlying physical IP topology can impose huge stress on the physical network, leading to issues related to the efficiency of applications and network overhead. This problem is known as *Topology Mismatch Problem* and can be found on structured and unstructured overlay networks alike.

T-Man[62] and X-BOT[44] appeared to solve this problem. These protocols allow to bias the network overlay to optimize some criteria. X-BOT appeared after T-Man and proved itself to preserve some key properties (e.g. the overlay connectivity) in scenarios that even T-Man was not able to. For instance, when T-Man uses latency as a criteria to bias the topology of an unstructured overlay network, it might compromise the overlay connectivity and partition the network. Both of these works were further studied to create optimized DHT protocols[63][64].

2.5 Summary and Discussion

Blockchain systems are continuously evolving. In this chapter, we presented how general blockchain systems operate, some of those and how they are solving the blockchain challenges. For entrepreneurs, it is a good idea on having a different blockchain system solving each problem, because that way they keep differentiate from each other and continue the innovation, giving space for new businesses each with an advantage. But on a systems view, it is better to have a flawless blockchain. The blockchain system that is working best from a systems perspective is Ethereum[5], they keep working on the protocols under the hood, improving these core protocols to get better system performance, instead of being afraid of the change, as the Bitcoin community.

This is one more reason to study the Ethereum system in depth. We saw how Bitcoin and Ethereum networks are defined and disseminate information between peers.

After some understanding on both of these systems, we can relate the challenges existing in blockchain systems to the overlay topology of the network, that is not maintained correctly, and to the message dissemination protocols used, that do not deliver messages about transactions and blocks effectively.

We also presented the state of the art protocols related to membership and message dissemination. Those protocols present the base idea to solve the blockchain challenges.

In the next chapter, we will provide some insight into how to solve one of the main challenges regarding blockchain leveraging on the technologies presented so far.

Chapter 3

Biased Membership and Epidemic Tree Dissemination

The message dissemination protocols used in blockchain systems are based on flooding, across the entire network (i.e. across all nodes in the system), all messages concerning both transactions and blocks. The effectiveness of this message flooding mechanisms is highly dependent on the overlay network topology. Considering a topology supporting the operation of a blockchain system, we can improve the efficiency of the flooding mechanism, but it is not enough, the system network will still have many redundant messages and significant network bandwidth will be used in a unnecessary way, while also having significant overhead in terms of processing power (CPU cycles), and (ultimately) energy consumption.

There is also a waste of computational resources due to the unawareness of the new blocks in the blockchain (due to dissemination mechanisms that can exhibit high latency). Nodes may perform computations over an outdated state of the blockchain that will not be useful for the progression of the system.

These are some of the main problems with blockchain systems at the moment. Apart from the waste of resources mentioned earlier, these issues can also translate to scalability problems. Blockchain systems depend on the capacity to process blocks and transactions across the network, if we do not have mechanisms that allow doing that efficiently, as the network grows, the time to process blocks and transactions will also grow (unfortunately above linearly).

In the previous chapter, we presented state-of-the-art on message dissemination and membership protocols and described how Bitcoin and Ethereum systems handle their membership and how they disseminate new transactions and blocks.

In this chapter we will discuss in detail how can we apply state-of-the-art protocols for the peer-to-peer community in Blockchain systems to improve the message dissemination

in these systems. In Section 3.1 we will present our membership solution, the membership messages and the nodes join/leave mechanisms. In section 3.2 we will present how transactions and blocks are disseminated over the network and which messages are necessary to support the proposed approach.

3.1 Overlay Network and Membership

Our solution consists of creating an overlay network that combines two different forms of topology bias, leveraging on X-BOT[44] as a starting point to build such a solution. We have to keep in mind that this approach needs to be adapted to a blockchain system and it does not consist only in applying the X-BOT protocol to create an overlay network for blockchains. Applying X-BOT over the blockchain system might create new attack vectors where an attacker can exploit the neighbors of a node, blocking all the transactions or blocks mined by their target.

3.1.1 X-BOT Requirements and Architecture

In order to maintain a highly available overlay network, there are three main requirements that must be met: connectivity, balanced in-degree across all nodes in the system, and low clustering coefficient.

- Connectivity: The overlay is connected if there is at least one path that allows every node to reach another node.
- Degree Distribution: The degree of a node is the number of edges of the node. If asymmetrical partial views are employed, the in-degree is the number of nodes that have our node identifier in their views, and the out-degree is the number of node identifiers the node has in his view. If views are symmetric than we consider only the degree (since the in-degree and out-degree are similar).
- Clustering Coefficient: The clustering coefficient of a node is the number of edges between the node and his neighbors, divided by the maximum possible number of edges across those neighbors. The clustering coefficient of the overlay network is the average of the clustering coefficients of the nodes. The clustering coefficient is a value between 0 and 1, and should be the smallest possible.

The X-BOT protocol maintains a small active view and a larger passive view, relaxing stability in order to continuously improve the overlay according to some metric. Periodically, each node attempts to swap one member of its active view with one better node from its passive view.

These *optimization rounds* work as depicted in Diagram 3.1.

Node i represents the node that starts the optimization round (initiator), node o is an old node from the i's active view which is replaced during the optimization round, node

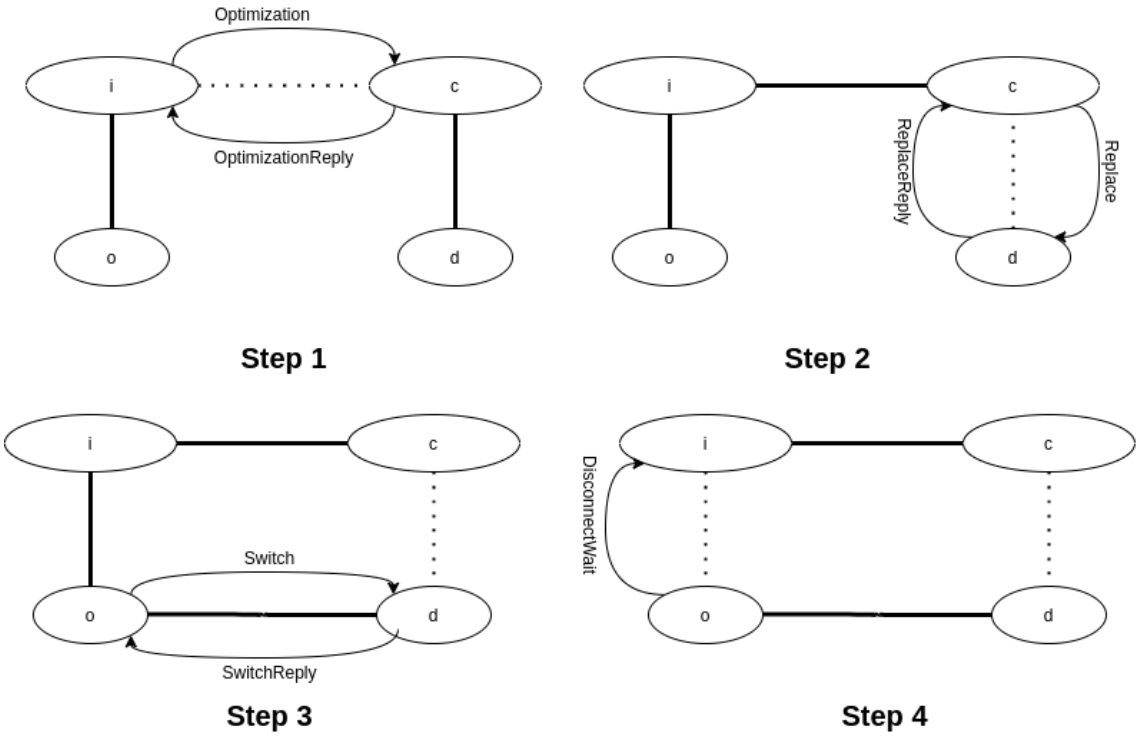

Figure 3.1: Optimization rounds adapted from [44]

c (candidate) is a node from *i*'s passive view that will be upgraded to the active view, and node *d* (disconnected) is the node that will be removed from the candidate's active view in order to accept the initiator.

3.1.2 Biased Membership

Blockchain networks usually do not require to lookup node information in the network, since every node is a replica of the full system state (i.e. the blockchain being that replicated state) including both blocks and transactions. Because of this feature, we do not have to guarantee the communication between nodes that are not in the neighbor list of a specific node.

Our proposed protocol has a relaxed biased membership. The topology bias criteria that we considered are the latency and the hamming distance[65] between the hash of node identifiers. With the latency bias we aim at acquiring lower latency between the nodes that communicate directly, this bias might form clusters that can make the entire network more susceptible to network partitions due to direct correlation with geolocation, a phenomenon studied in previous works[36][44]. To address those challenges we selected the hamming distance between the hash of node identifiers as the primary bias. Hamming distance will maintain the network connectivity since the hash function result does not have any correlation with nodes attributes (due to hash functions properties[66]).

The membership protocol is asymmetric, meaning if node *n2* is in the neighbors list

of node *n1*, it does not mean that *n1* is in the neighbors list of *n2*. This feature guarantees that each node will have the best neighbors possible and is not restricted by having non-optimal nodes in its neighborhood due to being the best neighbor on the non-optimal node.

3.1.2.1 Hashing Functions and Hamming Distance

Hashing functions are mathematical functions that map data of arbitrary size to fixed-size values (deterministically). They are used regularly in computer science to map data in tables (hash tables) and cryptography (among others).

We calculate the Hamming Distance[65] between the result of the SHA256 hash function on the node identifiers (Similarly to Kademlia[41]). Then we obtain the logarithm of this distance to the base of a x parameter and the result will be our final distance (as in equation 3.1).

$$log_x\ hamming\_distance(sha256(id1),\ sha256(id2)) \tag{3.1}$$

The x parameter can be modified to increase or decrease the collision rate, this way we can set a level of preference to this bias (as in example 3.2).

$$\begin{aligned} x = 2 &\rightarrow 256\ max.\ distance \\ x = 4 &\rightarrow 128\ max.\ distance \\ x = 16 &\rightarrow 64\ max.\ distance \\ x = 256 &\rightarrow 32\ max.\ distance \end{aligned} \tag{3.2}$$

3.1.2.2 Latency and Geolocation

Latency is defined as the time interval between a request and a response, geolocation is the estimate of a real-world geographic location of an object. Geolocation proximity of nodes usually is tied with lower latencies because the hop counts tend to be smaller from the source to the destination.

Since we use the latency as the secondary bias, there is no way to set a level of preference. When there is a collision in this mechanism, we can rely on the latency to untie the biasing process.

3.1.3 Membership Messages

To maintain the overlay network, we need four types of messages:

- FINDNODE: It requests information about nodes close to the destination node. When received, it should reply with a NEIGHBORS message with a sample of the neighbors list (selected at random with a maximum size of *sampleSize* parameter).

The sender node is added to the database and might be added to the list of neighbors of the receiver node.

- NEIGHBORS: Is the reply to FINDNODE message. Upon received, the node will add the nodes in the sample to their database and might add in the list of neighbors.
- PING: Should be sent from time to time to confirm that the endpoint is up. When received, it should reply with a PONG message.
- PONG: Is the reply to ping. Updates the database entry for the sender with the current timestamp.

How these messages are related with each other is depicted in Diagram 3.2.

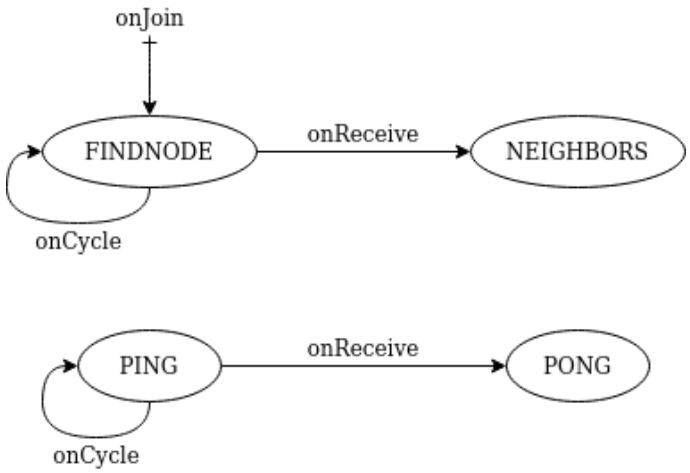

Figure 3.2: Membership as diagram

Algorithm 1 illustrates the overlay network membership management algorithm. The constants *tableSize* and *sampleSize* are configurable, we present the values that we used in Section 4.1.4. Each node also maintains a database of peers in the network, which might be used in case the minimum neighbors threshold is not met. The procedure *AddEntryNeighbs* also add nodes in the lazy push peers set (more details on Algorithm 2), but for simplicity, we hide that part of the pseudo-code.

3.1.4 Join and Leave Mechanisms

A node joins the network discovering a bootstrap node (the way nodes discover a bootstrap node will not be addressed in this work and can be the same as employed in the original Bitcoin system) and sending a *findnode* message. It will receive an answer containing a set of nodes that it will use to populate its neighbors set.

In our proposed protocol there are no messages to handle nodes leaving/failing. Nodes will remove neighbors and other known nodes addresses from the database when they do not send a heartbeat message for more than 24 hours. Neighbors are removed

Algorithm 1 Membership Management Algorithm

```
1:  procedure AddEntryNeighbs(sender)
2:      if sender ∈ neighbEntries then
3:          return
4:      if len(neighbEntries) < tableSize then
5:          neighbEntries ← neighbEntries ∪ {sender}
6:          return
7:      worst = sender
8:      for each n ∈ neighbEntries do
9:          if hammingDistance(self, worst) < hammingDistance(self, n) then
10:             worst = n
11:         if latency(self, worst) < latency(self, n) then
12:             worst = n
13:     if worst ≠ sender then
14:         neighbEntries ← neighbEntries ∪ {sender}
15:         neighbEntries ← neighbEntries \ {worst}
16:
17: procedure LookupNeighbs
18:     neighbs ← neighbsEntries
19:     while len(tmp) < sampleSize do
20:         if len(neighbs) == 0 then
21:             return tmp
22:         n = random(neighbs)
23:         tmp ← tmp ∪ n
24:         neighbs ← neighbs \ n
25:     return tmp
26:
27: upon event Receive(FINDNODE, sender) do
28:     call AddEntryNeighbs(sender)
29:     neighbs ← call LookupNeighbs()
30:     trigger Send(NEIGHBORS, self, neighbs)
31:
32: upon event Receive(NEIGHBORS, sender, nodes) do
33:     call AddEntryNeighbs(sender)
34:     for each n ∈ nodes do
35:         call AddEntryNeighbs(n)
```

when they fail to respond more than four times in a row (to any message that expects an answer).

3.2 Transactions and Blocks Dissemination

To tackle the existing limitations of blockchain systems, we have to change not only the structure of the overlay network but also the strategy that is employed by nodes to disseminate messages (namely blocks and transactions) over that network.

We want to minimize message redundancy, lower dissemination latency, and improve robustness, therefore we will use an approach similar to the one originally proposed in epidemic broadcast trees[38] but it will also be adapted for our use case, particularly in

what respects to the use of the topology biasing mechanism in the underlying unstructured overlay network.

3.2.1 Protocol Messages

Protocol messages are employed to disseminate blocks and transactions, correctly, to every node in the network.

- STATUS: Inform a peer of its current state. This message should be sent when the node joins/return the network. It sends the latest block number and the hash of the best-known block. The incorrect peer synchronizes with the up-to-date peer.
- ANNOUNCEMENT: Sent when a node broadcast blocks/transactions to its peers (through lazy push). When received, the node should ask the sender peer for any content being announced that is not known, after some delay (configurable).
- GETDATA: Receives the type of data (transaction or block), a list of hashes and a sync flag set to true when a incorrect peer synchronizes trough STATUS message (in this case, the list of hashes will contain only one hash with the last hash in common). Returns a list with the objects corresponding with the list of hashes, or, in the case of sync flag, a list with all the blocks since the hash received.
- BLOCK: Sends one block to a peer. When received it might graft or prune the peer (sending a message).
- TXS: Sends one or more transactions to a peer. When received it might graft or prune the peer (sending a message).
- GRAFT: Require the peer to put the sender node in the eager push set.
- PRUNE: Require the peer to remove the sender node from the eager push set and put it in the lazy push set.

Diagram 3.3 is a representation of the relation between the different protocol messages interact with each other.

3.2.2 Broadcast Tree Construction

As mentioned before, epidemic broadcast trees have two sets of peers, the eager push and the lazy push. This sets are maintained by grafting or pruning other peers to create a broadcast tree.

In the original algorithm, initially a subset of all peers is in the eager push set. When a node receives a gossip message for the first time, disseminate the same message to all peers in the eager push set and sends an announcement to peers in the lazy push set informing about the existence of the message. If a node receives the same message from

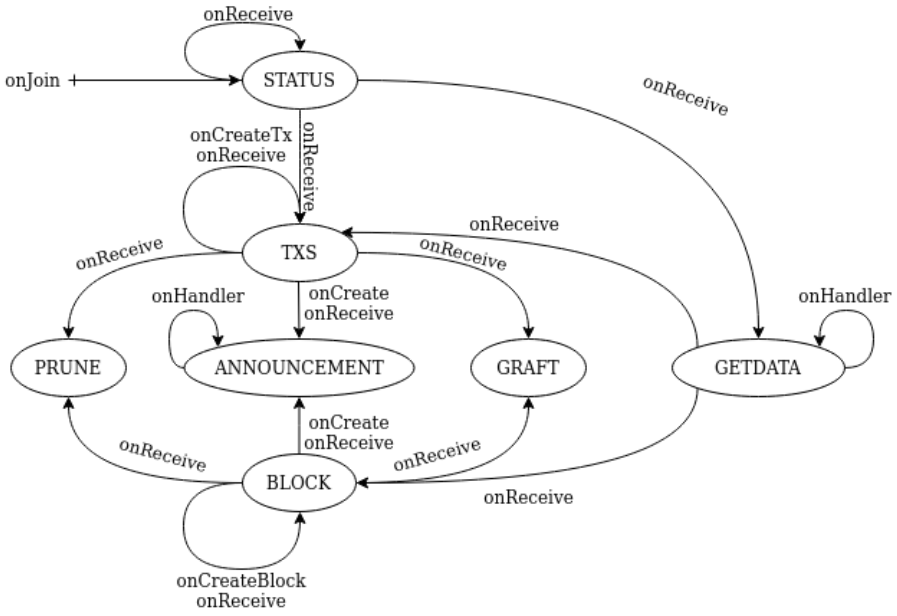

Figure 3.3: Dissemination Protocol messages diagram

different peers, it will send a prune message to peers that delivered the message latter, this message will remove the node from the peer eager push set and it will put it in the lazy push set.

Due to our overlay network being relaxed and having different types of messages we want to propagate, we need to adapt the original algorithm to our use-case. Initially, when a node joins the network it will put all the bootstrap peers in the eager push set, sending a GRAFT message to the peer. Peers in the eager push set and the lazy push set will not correspond to peers in the neighbors set (membership peer set), in other words:

$$\forall x \in neighborsSet, x \in eagerPushSet \vee x \in lazyPushSet \wedge$$
$$\forall x \in (eagerPushSet \cup lazyPushSet), x \in neighborsSet \vee x \notin neighborsSet$$

Algorithm 2 illustrates how we can build and maintain the broadcast tree in the overlay network.

To construct the spanning tree we move peers from eagerPushPeers set to lazyPushPeers set and vice versa, in such a way that the overlay defined by the first set becomes a tree.

When a node receives a GOSSIP message for the first time it sends the message through eager and lazy push (Algorithm 2, lines 15-16) and includes the sender in the eagerPushPeers set, sending a GRAFT message (Algorithm 2, lines 18-20), ensuring that the link to the peer is bidirectional and belongs to the broadcast tree. If the GOSSIP message is a duplicate, it will move the sender to the lazyPushPeers set, sending send a PRUNE message (Algorithm 2, lines 22-24).

Algorithm 2 Broadcast Tree Construction Algorithm

```
1:  procedure EagerPush(sender, m)
2:      for each p ∈ eagerPushPeers : p ≠ sender do
3:          trigger Send(GOSSIP, self, m)
4:
5:  procedure LazyPush(sender, m)
6:      for each p ∈ lazyPushPeers : p ≠ sender do
7:          lazyQueue ← (announcement(self, p, m))
8:      announcements ← policy(lazyQueue)
9:      trigger Send(announcements)
10:     lazyQueue ← lazyqueue \ announcements
11:
12: upon event Receive(GOSSIP, sender, m) do
13:     if m ∉ knownMessages then
14:         knownMessages ← knownMessages ∪ m
15:         call EagerPush(sender, m)
16:         call LazyPush(sender, hash(m))
17:         if sender ∉ eagerPushPeers then
18:             lazyPushPeers ← lazyPushPeers \ {sender}
19:             eagerPushPeers ← eagerPushPeers ∪ {sender}
20:             trigger Send(GRAFT, self)
21:     else if limiter(m) then
22:         eagerPushPeers ← eagerPushPeers \ {sender}
23:         lazyPushPeers ← lazyPushPeers ∪ {sender}
24:         trigger Send(PRUNE, self)
25:
26: upon event Receive(PRUNE, sender) do
27:     eagerPushPeers ← eagerPushPeers \ {sender}
28:     lazyPushPeers ← lazyPushPeers ∪ {sender}
29:
30: upon event Receive(GRAFT, sender) do
31:     lazyPushPeers ← lazyPushPeers \ {sender}
32:     eagerPushPeers ← eagerPushPeers ∪ {sender}
```

Due the high volume of gossip messages in the network (with different origins), there is a need for a limiter when pruning peers. This limiter can be set with modular arithmetic (e.g. when hash(m) mod x == 0, where x is configurable).

This whole process ensures that, when the first broadcast is terminated, a tree has been created[38].

The lazy push implementation consists in a set of announcement messages, that only contain the message hash. This messages do not need to be sent immediately, the only requirement of the schedule policy is that every announcement is eventually scheduled for transmission.

3.2.3 Fault Tolerance

In face of failures the eager push dissemination process (most likely) will not ensure that messages are delivered through the entire network, since at least one tree branch

is affected. Lazy push messages enable nodes to recover the missing messages and to quickly repair the tree.

When a node receives an announcement message corresponding to a missing message it will request the missing message to the first node that sent the announcement and starts a timer. If the message is received before the timer expires, it will process the GOSSIP message and graft the sender node (Algorithm 2, lines 13-20), if the timer expires, it will request to the next node that sent an announcement referring to the same message. The process repeats until the node receives the message or there are no more announcements referring to the message.

If several nodes disconnect the network due a single failure, the remaining nodes will try to repair the spanning tree degenerating into a structure that has cycles. This will eventually be fixed by sending PRUNE messages when a node receives a duplicated message (Algorithm 2, lines 22-24).

3.2.4 Dynamic Membership

Since we have a dynamic membership protocol managing the underlying unstructured overlay network, nodes are constantly joining and leaving the network. In our membership protocol we do not have a mechanism to handle nodes leaving, we only know if a node actually left the network if the TCP connection fails. In this case, we remove the node from eager and lazy push sets, and we delete all pending announcements that we would send to the leaving node. When a node joins the membership, we simply add them to our eager push set, eventually he will become part of the broadcast tree.

When sub-trees are generated, it is required that one of the disconnected nodes receive an announcement message in order to effectively repairing the whole tree (as long as only few nodes fail)[38]. Any redundancy generated in this process will be handled on the next reception of redundant messages.

3.3 Summary

In this chapter we presented the main contribution of this dissertation, our solution create a biased membership and a broadcast tree to disseminate blocks and transactions.

We started by motivating our work by remembering the main problems this thesis aim to solve. Then we presented our membership solution, the metrics we use to bias the same membership (hamming distance and latency) and the messages we need to maintain the membership (as well as join and leave mechanisms). Finally, we end the chapter presenting our gossip-based protocol showing the main algorithm to build the broadcast tree, some mechanisms for fault tolerance and tree repair, how the algorithm behaves with a dynamic membership, and some variations to approach the broadcast tree protocol.

Chapter 4

Evaluation

We will now evaluate our approach, taking advantage of our membership and dissemination protocols and compare the performance achieved with the original Bitcoin and Ethereum solutions. To evaluate the different protocols we relied on a simulator implemented by us on top of SimpleDA (Simple Message Passing Simulator for Distributed Algorithms)[1]. Other simulators that exist are either outdated or not adequate to blockchain systems, requiring an excessive effort in modifying and validating the simulation.

Initially we wanted to implement a client for our proposed solution and run tests in distributed servers to show the feasibility of the solution, this was not possible due to time constraints and will be addressed as future work.

For the evaluation of the different protocols we considered the following performance metrics:

- **Latency:** Measures the latency between nodes and their outbound connections. This metric is important because we want to significantly reduce the latency between (direct) connections established by nodes in the system, being a key objective for the proposed membership protocol.
- **Messages sent:** Measures the number of membership/dissemination messages sent in the network. The contributor for this metric is dissemination messages, which measure the number of messages related with synchronizing the blockchain state and the transmission of blocks/transactions. If we reduce the number of repeated messages, the number of dissemination messages sent will lower, imply a better usage of network resources (and less CPU power processing redundant messages).
- **Repeated blocks:** Measures the number of times a node received an already known block. This metric will evaluate our message dissemination protocol, since we want

[1] https://github.com/miguelammatos/SimpleDA

to minimize the number of repeated messages in the network. This metric is related with the previous one.

- **Created blocks:** Measures the number of created blocks during the whole simulation. This metric evaluates the efficiency of the work performed by the system with lower number of blocks indicating more useful work.

Regarding the X-BOT based mechanism to bias the network we validate the following relevant properties:

- The overlay network ensured the connectivity of the graph.
- The distribution of the nodes degree was highly dependent of the configuration file, filling the maximum number of connections (table size), since the overlay network had symmetric views, in other words, the in-degree was equal to the out-degree.
- The clustering coefficient was not calculated due to limited computational resources and time constraints.

This chapter will present the evaluation of the different blockchain protocols. In Section 4.1 we will present the simulator and give some details about the protocols implementations (in the simulator). In Section 4.2 we will present the key results yielded by our experimental work. In Section 4.3 we will discuss the results presented previously.

4.1 Simulator and Implementation

The simulator is implemented in Python 3 and runs in PyPy (in this case, it runs faster than using CPython). The code for the SimpleDA simulator is open source under GPL-3.0 license. The implementations for the different blockchain protocols and the proposed solution presented in this dissertation are also open source[2].

The simulations were conducted using 500 nodes which resulted in resource intensive simulations that lasted several days. We will present more details on the configurations of the simulator in Section 4.1.4.

4.1.1 Bitcoin

Our implementation of the Bitcoin protocol has only simple messages for membership creation/maintenance. We recreated the whole process for message dissemination that is optimized to reduce the messages size, increasing the number of messages (e.g. sending an INV message with the hashes instead of sending the blocks/transactions).

We did not implemented Compact Blocks (BIP 152[3]) which might have added value to the results, since this update introduced a method of reducing the amount of bandwidth used to propagate new blocks to nodes.

[2]https://github.com/jpldcarvalho/block-sim
[3]https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

4.1.2 Ethereum

Likewise to the Bitcoin protocol, we have the membership creation/maintenance messages. We implemented the Kademlia[41] protocol which is used as the base of the Ethereum membership protocol. For the dissemination protocol, we implemented the PV62 which does not include NodeData and Receipts. We do not believe that those features would be relevant for the experiments that we are conducting.

As of February 2020, Ethereum released the version 1.9.11 where they reduced the bandwidth related with transactions messages (EIP 2464[4]). The implementation of this feature in the simulator would also improve the results of the protocol, but we did not do it due to lack of time and computational resources.

4.1.3 Proposed Protocol

For the proposed protocol, we based our messages in both Bitcoin and Ethereum protocols, implementing the previous explained protocols, adding protocol specific messages, such as GRAFT and PRUNE messages.

4.1.4 Configurations

Each simulation used a system composed of 500 nodes and executed for a period of 35 hours, where each simulation cycle (the virtual time unit of the simulator) had 1250 milliseconds. Nodes stopped creating transactions and blocks in the last 10% cycles of the simulation in order to stabilize the simulation (the simulations reproduced the state of the network for 31.5 hours).

Each cycle generated 2 transactions (from random nodes), giving us a total of 180 000 transactions. There were 5 miners in the network (randomly selected), each miner had a minimum transaction number of 960 in order to mine a new block, giving a total of approximately 188 blocks (approximately 1 block per 10 minutes).

All configurations parameters are available in Table 4.1.

Configurations	
CYCLE_NUMBER	100 000
NODE_NUMBER	500
NODE_CYCLE	1 250
TX_PER_CYCLE	2
MINERS	5
MIN_TX_PER_BLOCK	960

Table 4.1: General configurations for the simulator

The sample of latencies used was obtained from PlanetLab, where nodes are uniformly distributed across the world.

[4]https://eips.ethereum.org/EIPS/eip-2464

Apart of the above simulator configurations, each protocol had their specific configurations, such as minimum and maximum values for connections (p constant in Bitcoin, bucket size in Ethereum, table size in the proposed protocol), maximum number of samples/headers sent in messages, lookup timeouts and advertising time intervals.

Tables 4.2, 4.3 and 4.4, present the protocol specific parameter configurations for Bitcoin, Ethereum and our proposal, respectively.

Bitcoin configurations	
P	8
SAMPLE_SIZE	2 500

Table 4.2: Bitcoin parameter configurations

Ethereum configurations	
BUCKET_SIZE	16
BUCKETS_NUMBER	256
ALPHA	3
LOOKUP_TIMEOUT	1 000
MAX_BLOCK_HEADERS	100

Table 4.3: Ethereum parameter configurations

Our Proposal configurations	
TABLE_SIZE	20
SAMPLE_SIZE	250
NEIGHB_THRESHOLD	5

Table 4.4: Our proposal parameter configurations

4.2 Results

The following section will present the results of the simulations. For each protocol we did a series of ten independent simulations and then we did the average of the results observed in each one.

The minimum and maximum values in the connections graphs represent the average of the minimum and maximum connections of all the simulations. The remaining minimums and maximums reflect only one simulation (the one which had that value).

All values presented in the graphs were rounded with a floor function to make them more perceptible.

4.2.1 Latency

The inbound and outbound latency represent the average of inbound and outbound connections latency for each node in the simulation. Less latency means faster communications, in other words, less latency on communication links should yield better performance.

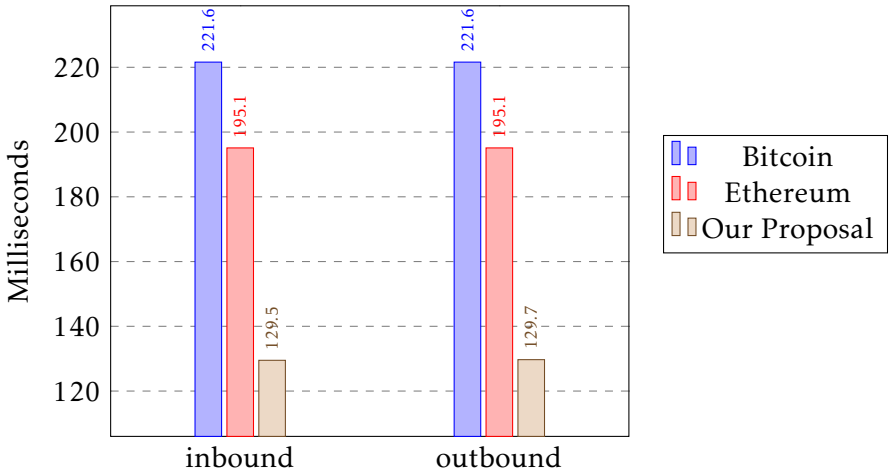

Figure 4.1: Inbound and outbound latency

Figure 4.1 shows the inbound and outbound latency for each protocol.

This metric was one of the main motivations for this work. Our protocol demonstrate that is possible to reduce the outbound and inbound latency while maintaining a stable membership, i.e., a membership managed in a decentralized way and that ensures global connectivity.

4.2.2 Number of Messages Sent

The number of messages sent depends on the degree distribution (the out-degree) and on the message dissemination algorithms. Less messages is better assuming that the network is connected (and does not have a significant difference in the diameter)

In the counting of the number of sent/received membership messages we did not considered PING and PONG messages (which are present in all protocols).

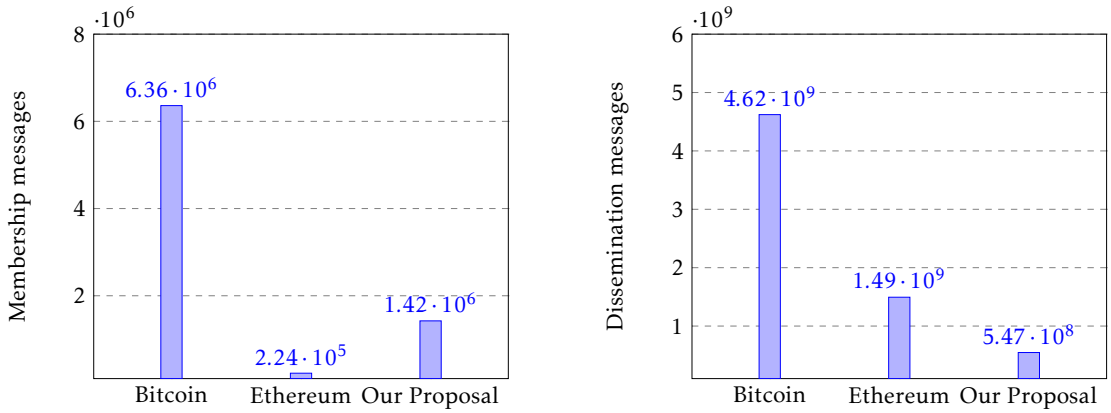

Figure 4.2: Number of messages sent

Figure 4.2 depicts the number of membership and dissemination messages sent in the network (left and right, respectively). The reader should note that the scale for both graphs is different. Membership messages are counted in millions, while dissemination messages are counted in billions.

The Ethereum protocol require a significantly lower number of messages in order to maintain their membership while our proposed solution require less messages to propagate blocks and transactions. Overall the proposed protocol sends less messages, but needs further work in the membership maintenance. The overhead of the number messages to maintain the membership protocol might be worth in order to minimize the number of dissemination messages.

4.2.3 Number of Repeated Blocks

The number of repeated blocks represent the number of times a node receives a block (in the case of the miner, it will have the counter for the mined block at one). The best case scenario is all nodes receiving each block exactly once.

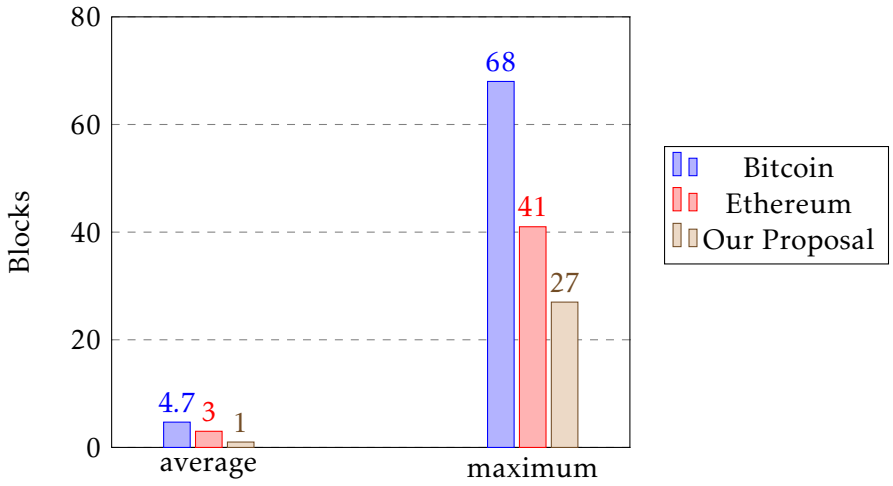

Figure 4.3: Number of repeated blocks

Figure 4.3 shows the average and maximum number of repeated blocks, there were no need to plot the minimum which is one for every protocol. Every protocol presented a 100% message delivery rate, there were also no need to plot the delivery rate.

Our protocol achieved an average of 1.0 of repeated blocks (which is the best case scenario) and outperformed the other protocols on the maximum value of repeated blocks, but, we think the maximum value is still high and there is room for improvement. The reader should note that the average is 1.0 (even with a 47 maximum) because there were a big number of generated blocks that received the generated blocks once. The real average is 1.00014, if we increase the decimal point.

4.2.4 Number of Created Blocks

The number of created blocks shows us the number of blocks each protocol created during the simulations. In our simulations, the reference number is the minimum of blocks created which has the value 188. In a perfect scenario, the number of blocks created by a protocol should be equal to the reference number, meaning that there were no computations on outdated/wrong blocks.

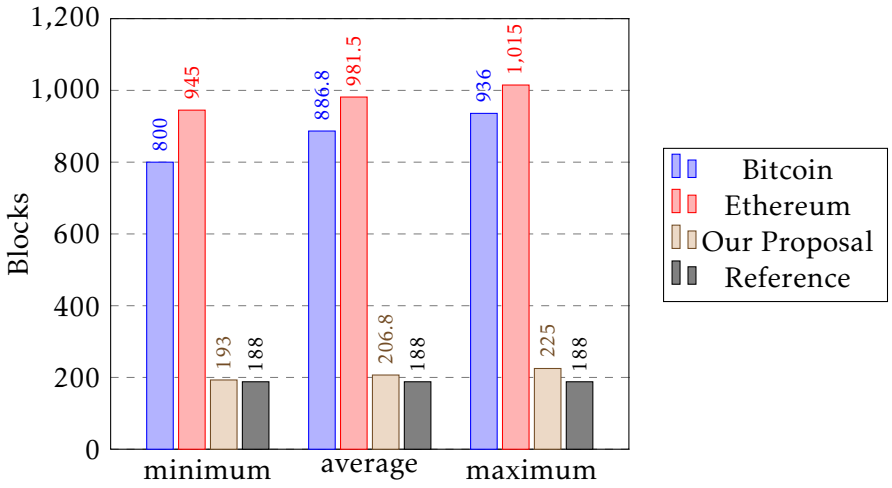

Figure 4.4: Number of created blocks

Graph 4.4 shows the minimum, average and maximum created blocks in the simulations. We greatly outperformed the remaining protocols in this metric, presenting values close to the reference value.

4.3 Discussion

Overall, the proposed changes to the membership and dissemination protocols proved better in most metrics. There are still some room for improvement, mainly in the standardization of the in-degree distribution and in the number of messages required to maintain the membership.

The results presented in Section 4.2 are the baseline for further work, there are some functionalities and features that were not implemented that might influence these results (as explained before). With the quick and continuous evolution in blockchain protocols (most of the times with very limited documentation), it is difficult to implement and maintain a trustworthy updated simulator.

For more reliable results we should implement a client with the right tools for monitoring the network and compare with modified Bitcoin and Ethereum clients (with the latest version) in a private network.

4.4 Summary

In this chapter we presented the results of the contribution produced in this dissertation comparing them with existing reference systems.

We started by giving some implementation details for each protocol and how we configured the simulator. Then we presented the results for each metric while discussing them. We close this chapter with a discussion on the implementation and the results.

Chapter 5

Conclusion

Blockchain systems promise to have a significant impact on the technological landscape, but they still have to tackle some limitations until they can be efficiently used in our daily lives.

In this document we considered the design of blockchain systems and the internal operations, identifying some key limitations in their designs and put forward a new proposal that aims at solving some issues on key components of these systems. There are still many topics that were not presented or not deepened in this document.

We believe that currently, blockchain systems might be an overhyped hot topic. The technology has the potential to be applied to many society problems, but existing solutions are not yet mature to efficiently address those problems (as discussed in Chapter 1). With our work, we proposed a solution to one of the main problems regarding the scalability of the system and the efficiency of the communication between the network nodes.

To achieve our objective, we worked on a novel solution that creates an overlay network with a topology that is biased towards supporting highly efficient and fast data dissemination, providing better node neighbors without sacrificing the overlay connectivity nor any key blockchain characteristic.

We also reduced the number of messages that are disseminated in the network using an adaption of epidemic broadcast trees as a gossip protocol in order to effectively broadcast transactions and blocks over the network with much reduced redundancy and without affecting the reliability.

5.1 Future Work

As future work there are two main options that might improve the current protocol and the ability to test all the future protocols.

Firstly, there is a need for a general simulator for blockchain networks that is able to run bigger simulations in a timely fashion. This simulator may also have the most popular blockchain network protocols implemented (e.g. Bitcoin, Ethereum, EOS). The main counter argument in pursuing this direction is the (human) resources needed to maintain such simulator.

Secondly, it would be important to validate these results with a real client implementation of the presented protocols, thus, applying the protocols in a real-world scenario and environment. This way, other blockchains could also apply such protocols once proved effective, in a controlled environment.

Bibliography

[1] T. Abate. *Stanford computer scientists launch the Center for Blockchain Research*. 2018. URL: https://engineering.stanford.edu/news/stanford-computer-scientists-launch-center-blockchain-research.

[2] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf*.

[3] D. Pollock. *The Fourth Industrial Revolution Built On Blockchain And Advanced With AI*. 2018. URL: https://www.forbes.com/sites/darrynpollock/2018/11/30/the-fourth-industrial-revolution-built-on-blockchain-and-advanced-with-ai/.

[4] L. Shen. *Blockchain Will Be Used By 15% of Big Banks By 2017*. 2016. URL: http://fortune.com/2016/09/28/blockchain-banks-2017/.

[5] V. Buterin. *Ethereum White Paper: A Next Generation Smart Contract And Decentralized Application Platform*. Tech. rep. URL: http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

[6] M. Orcutt. *Blockchains Use Massive Amounts of Energy - But There's a Plan to Fix That*. 2017. URL: https://www.technologyreview.com/s/609480/bitcoin-uses-massive-amounts-of-energybut-theres-a-plan-to-fix-it/.

[7] A. Hertig. *Ethereum's Big Switch: The New Roadmap to Proof-of-Stake*. 2017. URL: https://www.coindesk.com/ethereums-big-switch-the-new-roadmap-to-proof-of-stake.

[8] *Bitcoin Energy Consumption Index*. 2019. URL: https://digiconomist.net/bitcoin-energy-consumption.

[9] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. "On Scaling Decentralized Blockchains - (A Position Paper)." In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. 2016, pp. 106–125. DOI: 10.1007/978-3-662-53357-4_8. URL: https://doi.org/10.1007/978-3-662-53357-4_8.

[10] E. Hughes. *A Cypherpunk's Manifesto*. 1993. URL: https://www.activism.net/cypherpunk/manifesto.html.

[11] A. Back. *Hashcash - A Denial of Service Counter-Measure*. Tech. rep. 2002. URL: http://www.hashcash.org/papers/hashcash.pdf.

[12] J. A. Lanz. *Hacker Responsible for 51Returns Part of the Stolen Funds*. 2019. URL: https://ethereumworldnews.com/hacker-51-percent-attack-ethereum-classic-returns-funds/.

[13] A. Kadiyala. *Nuances Between Permissionless and Permissioned Blockchains*. 2018. URL: https://medium.com/@akadiyala/nuances-between-permissionless-and-permissioned-blockchains-f5b566f5d483.

[14] P. Jayachandran. *The difference between public and private blockchain*. 2017. URL: https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/.

[15] M. Jakobsson and A. Juels. "Proofs of Work and Bread Pudding Protocols." In: *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*. 1999, pp. 258–272.

[16] S. N. Sunny King. "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake." In: (2012). URL: https://peercoin.net/whitepapers/peercoin-paper.pdf.

[17] J. Connell. *On Byzantine Fault Tolerance in Blockchain Systems*. 2017. URL: https://cryptoinsider.com/byzantine-fault-tolerance-blockchain-systems/.

[18] *An Introduction to Hyperledger*. Tech. rep. The Linux Foundation, 2018. URL: https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf.

[19] A. Tayo. *Proof of work, or proof of waste?* 2017. URL: https://hackernoon.com/proof-of-work-or-proof-of-waste-9c1710b7f025.

[20] V. Buterin. *On Stake*. 2014. URL: https://blog.ethereum.org/2014/07/05/stake/.

[21] D. O. Michael Peirce. *Scaleable, Secure Cash Payment for WWW Resources with the PayMe Protocol Set*. 1997. URL: https://www.w3.org/Conferences/WWW4/Papers/228/.

[22] E. Rykwalder. *The Math Behind Bitcoin*. 2014. URL: https://www.coindesk.com/math-behind-bitcoin.

[23] *Smart Contracts: The Blockchain Technology That Will Replace Lawyers*. URL: https://blockgeeks.com/guides/smart-contracts/.

[24] B. Allen. *Turing-completeness: How Ethereum Does what it Does*. 2017. URL: https://thebitcoinmag.com/turing-completeness-ethereum/1712/.

[25] *Litecoin Homepage*. URL: https://litecoin.org/.

[26] D. D. Evan Duffield. *Dash: A Privacy-Centric Crypto-Currency*. Tech. rep. URL: https://whitepaperdatabase.com/wp-content/uploads/2017/09/Dash-Whitepaper.pdf.

[27] *Product Overview: A technical overview of xCurrent*. Tech. rep. Ripple, 2017. URL: https://ripple.com/files/ripple_product_overview.pdf.

[28] *NEM: Technical Reference*. Tech. rep. NEM Foundation, 2018. URL: https://www.nem.io/wp-content/themes/nem/files/NEM_techRef.pdf.

[29] S. Popov. *The Tangle*. Tech. rep. 2017. URL: http://untangled.world/wp-content/uploads/2017/09/iota1_3.pdf.

[30] S. N. Shen Noether. *Monero is Not That Mysterious*. Tech. rep. 2014.

[31] G. Zyskind, O. Nathan, and A. Pentland. "Enigma: Decentralized Computation Platform with Guaranteed Privacy." In: *CoRR* abs/1506.03471 (2015). arXiv: 1506.03471. URL: http://arxiv.org/abs/1506.03471.

[32] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. "Secure Multiparty Computations on Bitcoin." In: *Commun. ACM* 59.4 (Mar. 2016), pp. 76–84. ISSN: 0001-0782. DOI: 10.1145/2896386. URL: http://doi.acm.org/10.1145/2896386.

[33] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukoli, S. W. Cocco, and J. Yellick. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains." In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal: ACM, 2018, 30:1–30:15. ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190538. URL: http://doi.acm.org/10.1145/3190508.3190538.

[34] Bitcoin.org. *Developer Guide - Bitcoin*. URL: https://bitcoin.org/en/developer-guide.

[35] *Bitcoin Wiki*. URL: https://en.bitcoin.it/wiki/.

[36] J. Leitão, J. Pereira, and L. Rodrigues. "HyParView: A membership protocol for reliable gossip-based broadcast." In: *In IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE Computer Society, 2007, pp. 419–428.

[37] C. Decker and R. Wattenhofer. "Information propagation in the Bitcoin network." In: *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704. URL: https://doi.org/10.1109/P2P.2013.6688704.

[38] J. Leitão, J. Pereira, and L. E. T. Rodrigues. "Epidemic Broadcast Trees." In: *26th IEEE Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, October 10-12, 2007*. 2007, pp. 301–310. DOI: 10.1109/SRDS.2007.27. URL: https://doi.org/10.1109/SRDS.2007.27.

[39] M. M. João Marçal Luís Rodrigues. "Adaptive Information Dissemination in the Bitcoin Network." In: *34th ACM Symposium on Applied Computing (SAC 2019), Limassol, Cyprus, April 8-12, 2019*. 2019.

[40] *Ethereum devp2p Wiki*. 2015. URL: https://github.com/ethereum/devp2p/wiki.

[41] P. Maymounkov and D. Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric." In: *Peer-to-Peer Systems* (2002), pp. 53–65.

[42] *Ethereum Wiki*. 2019. URL: https://github.com/ethereum/wiki/wiki.

[43] J. Leitão, L. Rosa, and L. Rodrigues. *Large-Scale Peer-to-Peer Autonomic Monitoring*.

[44] J. Leitão, J. P. da Silva Ferreira Moura Marques, J. Pereira, and L. E. T. Rodrigues. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks." In: *IEEE Trans. Parallel Distrib. Syst.* 23.11 (2012), pp. 2175–2188. DOI: 10.1109/TPDS.2012.29. URL: https://doi.org/10.1109/TPDS.2012.29.

[45] S. Pallickara, H. Bulut, and G. C. Fox. "Fault-Tolerant Reliable Delivery of Messages in Distributed Publish/Subscribe Systems." In: *Fourth International Conference on Autonomic Computing (ICAC'07), Jacksonville, Florida, USA, June 11-15, 2007*. 2007, p. 19. DOI: 10.1109/ICAC.2007.18. URL: https://doi.org/10.1109/ICAC.2007.18.

[46] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas. "Efficient Reconciliation and Flow Control for Anti-entropy Protocols." In: *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*. LADIS '08. Yorktown Heights, New York, USA: ACM, 2008, 6:1–6:7. ISBN: 978-1-60558-296-2. DOI: 10.1145/1529974.1529983. URL: http://doi.acm.org/10.1145/1529974.1529983.

[47] P. Th, E. R. Guerraoui, S. B. Handurukande, A. m. Kermarrec, and P. Kouznetsov. "Lightweight probabilistic broadcast." In: *ACM Trans. Comput. Syst* 21 (2003).

[48] D. DD and D. O'Mahony. "Overlay networks: A scalable alternative for P2P." In: *Internet Computing, IEEE* 7 (Aug. 2003), pp. 79 –82. DOI: 10.1109/MIC.2003.1215663.

[49] M. Castro, M. Costa, and A. Rowstron. *Peer-to-peer overlays: structured, unstructured, or both*. Tech. rep. Microsoft, 2004. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2004/07/structella-tr.pdf.

[50] R. Baldoni, S. Bonomi, A. Rippa, L. Querzoni, S. Tucci Piergiovanni, and A. Virgillito. "Evaluation of Unstructured Overlay Maintenance Protocols under Churn." In: *26th International Conference on Distributed Computing Systems Workshops (ICDCS 2006 Workshops), 4-7 July 2006, Lisboa, Portugal*. 2006, p. 13. DOI: 10.1109/ICDCSW.2006.49. URL: https://doi.org/10.1109/ICDCSW.2006.49.

[51] M. Ripeanu. "Peer-to-Peer Architecture Case Study: Gnutella Network." In: *1st International Conference on Peer-to-Peer Computing (P2P 2001), 27-29 August 2001, Linköping, Sweden*. 2001, pp. 99–100. DOI: 10.1109/P2P.2001.990433. URL: https://doi.org/10.1109/P2P.2001.990433.

[52] *Napster Homepage*. URL: https://us.napster.com/.

[53] *BitTorrent Homepage*. URL: http://www.bittorrent.org/.

[54] K. Dooley. *Designing Large-Scale LANs*. O'Reilly, 2002. URL: https://userpages.umbc.edu/~dgorin1/451/lan_design/LANs_Design_Types.htm.

[55] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications." In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '01. San Diego, California, USA: ACM, 2001, pp. 149–160. ISBN: 1-58113-411-8. DOI: 10.1145/383059.383071. URL: http://doi.acm.org/10.1145/383059.383071.

[56] A. Rowstron and P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems." In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. Heidelberg, Germany, Nov. 2001, pp. 329–350.

[57] D. Wang, H. He, and J. Shi. "The Measurement and Analysis of KAD Network." In: *Trustworthy Computing and Services - International Conference, ISCTCS 2012, Beijing, China, May 28 - June 2, 2012, Revised Selected Papers*. 2012, pp. 117–123. DOI: 10.1007/978-3-642-35795-4_15. URL: https://doi.org/10.1007/978-3-642-35795-4_15.

[58] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web." In: *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: ACM, 1997, pp. 654–663. ISBN: 0-89791-888-6. DOI: 10.1145/258533.258660. URL: http://doi.acm.org/10.1145/258533.258660.

[59] C. Gkantsidis, M. Mihail, and A. Saberi. "Random walks in peer-to-peer networks: Algorithms and evaluation." In: *Perform. Eval.* 63.3 (2006), pp. 241–263. DOI: 10.1016/j.peva.2005.01.002. URL: https://doi.org/10.1016/j.peva.2005.01.002.

[60] S. Voulgaris, D. Gavidia, and M. V. Steen. "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays." In: *Journal of Network and Systems Management* 13 (2005), p. 2005.

[61] A. J. Ganesh, A. Kermarrec, and L. Massoulié. "SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication." In: *Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings*. 2001, pp. 44–55. DOI: 10.1007/3-540-45546-9_4. URL: https://doi.org/10.1007/3-540-45546-9_4.

[62] M. Jelasity and O. Babaoglu. "T-Man: Gossip-based overlay topology management." In: *In 3rd Int. Workshop on Engineering Self-Organising Applications (ESOA'05)*. Springer-Verlag, 2005, pp. 1–15.

[63] A. Montresor, M. Jelasity, and O. Babaoglu. "Chord on demand." In: *In Proceedings of the 5th International Conference on Peer-to-Peer Computing (P2P 2005)*. IEEE, 2005, pp. 87–94.

[64] J. L. João Carvalho Nuno Preguiça. "Ouroboros: Uma DHT Auto-organizável Tolerante a Churn." In: *Sétimo Simpósio de Informática* (2015).

[65] P. Wegner. "A Technique for Counting Ones in a Binary Computer." In: *Commun. ACM* 3.5 (May 1960), p. 322. ISSN: 0001-0782. DOI: 10.1145/367236.367286. URL: https://doi.org/10.1145/367236.367286.

[66] "Hashing in Computer Science: Fifty Years of Slicing and Dicing." In: *Hashing in Computer Science*. John Wiley & Sons, Ltd, 2010, pp. i–xvii. ISBN: 9780470630617. DOI: 10.1002/9780470630617.