UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

PBFT-BASED CONSENSUS ALGORITHMS FOR BLOCKCHAIN: A CASE STUDY

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

David Hernandez Lopez

In partial fulfilment

of the requirements for the degree of

MASTER IN ADVANCED TELECOMMUNICATION TECHNOLOGIES

Advisor: Olga Leon Abarca

Barcelona, July 2020

Title of the thesis: PBFT-based consensus algorithms for blockchain: a case study.

Author: David Hernández López

Advisor: Olga León Abarca

Abstract

Blockchain technology is an emerging and innovative technology that came out more than 10 years ago with the appearance of Bitcoin. Blockchain allows for a distributed monetary in which no central entity exists and where the members of the system make use of consensus protocols to validate transactions. The most known consensus method is Proof of Work (PoW), which is based on solving a hard mathematical problem and it is used in Bitcoin, among others blockchains. However, there are many other approaches such as Proof of Stake (PoS) or Byzantine Fault Tolerance (BFT) based ones.

Blockchains can be public, meaning that any user can freely join them, as it is the case of Bitcoin, or can be permissioned when users need access to join the system. In this work, we will study and analyse two permissioned blockchains: Exonum and Neo, which are based on the PBFT consensus algorithm and provide security against Byzantine attacks. To this end, test blockchains have been used and the performance of both blockchains is compared in terms of throughput (transactions per second) and commit time (time needed for adding a new block to the blockchain with new transactions).

To my parents, my sister, my girlfriend and my friends,
thanks to whom I am who I am
and to whom I can only express my sincere gratitude
for supporting me during these final step that today ends.

Acknowledgements

Thanks to Olga for advising me and giving me solutions to problems I was having with the thesis results, which were not being what we expected. Thanks to her ideas we have managed to redirect the thesis over time and reach the results. Thanks to the GitHub user $@slowli$ for answering multiple technical questions about how Rust-based Exonum code works. Thanks to user $@hal0x2328$ from the Neo-Smart Economy Discord group for advising and guiding me on setting up the Neo blockchain.

Revision history and approval record

Revision	Date	Purpose
0	27/03/2020	Structure of the master thesis document
1	27/05/2020	Results chapter revision
2	10/06/2020	State of the art chapter revision
3	24/06/2020	First overall document revision
3	12/07/2020	Final revision

Written by:		Reviewed and approved by:	
Date	11/07/2020	Date	12/07/2020
Name	David Hernandez Lopez	Name	Olga Leon Abarca
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

So far it is been more than 10 years since Nakamoto showed to the world a new revolutionary concept called peer-to-peer (P2P) electronic cash systems [1], which not only led to the invention of Bitcoin digital currency but also introduced the concept of blockchain. A blockchain is essentially a distributed ledger that can record transactions in a secure way and without the control of a central entity. Users can add new entries to the chain, which are validated by means of consensus protocols, but not delete existing ones, thus creating a transparent and auditable system. These features associated with blockchain are ideal for monetary applications such as Bitcoin, but they can also be very useful for many other applications beyond cryptocurrencies. Thus, after Bitcoin [1], other platforms have appeared where decentralized applications can be created and programmed by users, such as smart contracts. A smart contract is a piece of code which is added to the blockchain and, at some point in time, validates a condition or a set of conditions, and determines whether an asset should go to one person or another.

Despite the advantageous properties of Bitcoin and other pubic blockchains like Ethereum [2], they exhibit drawbacks that make them not appropriate systems for certain use cases (public blockchains cannot deal with high demanding transaction per second scenarios or with less computational environments). One of this drawback is their open, public and permissionless nature. By this we mean that anyone can connect to the blockchain platform without having been granted any special permission, and can contribute to the consensus process. Furthermore, the security of these blockchains is based on the computing strength of their members, where there is a need for giving incentives to users to contribute with their computing strength. In particular, nodes that contribute by registering transactions are rewarded with cryptocurrencies.

The initial idea of Nakamoto [1] has been deeply reviewed and studied from the security and security points of view. Since then, some other types of blockchain have appeared, such as private blockchains or permissioned blockchains, in which users need to be granted permission to access to it. These new approaches require further study in order to ensure safety and security.

In the following, we describe the goals of this master thesis and the structure of the document.

1.1. Statement of purpose

The main objective of this work is to test some permissioned blockchains in order to evaluate its performance under different scenarios and evaluate them in terms of speed (transactions per second and commit time). With this purpose, we will choose some blockchains based on the Practical Byzantine Failure Tolerance (PBFT) algorithm to compare them: Exonum and Neo. To do this, we will use real implementations of these blockchains, which are open-source and intended for testing purposes. These test blockchains allow simulating a blockchain in a local environment where we can control the network and test different configurations.

1.2. Project outline

The structure of the project is as follows:

> An explanation of what a blockchain is and the multiple possibilities it brings us can be found on **Chapter 2.** In addition, we describe some of the most famous consensus algorithms.

> On **Chapter 3**, we can find a detailed description of Exonum and how it works.

> **Chapter 4** describes another blockchain, Neo. We will review what is it and how it works.

> **Chapter 5** provides an explanation of the methodology followed to run all tests for evaluating the performance of both blockchains. After that, the results are shown and there is a final discussion on which blockchain performs better and why. Finally**,** there is a study of how to make an improvement and the conclusions of this.

> **Chapter 6** makes a final review of the work and details the conclusions obtained. It also adds improvement points for this work in case someone wants to do a study on this in the future.

2. Blockchain

A blockchain is a distributed ledger composed of blocks, each one containing a given number of transactions, typically monetary transactions in which a given amount of cryptocurrency is transferred from one user to another. Each block contains information related to the previous block in the chain in such a way that guarantees data integrity. By means of cryptographic techniques, the information stored in each block can only be repudiated or edited by modifying all subsequent blocks, which is considered impossible due to the security of the chain. The blockchain is implemented in a real environment, where thousands of nodes are connected to it. These nodes can get a copy of the ledger, and thus it is said to be decentralized and synchronous. Each user owns a public key pair, which is used to identify the former (public key) and to sign transactions (private key). Because users do not reveal their real identity but use their public key instead, user's privacy is guaranteed somehow.

Since the emergence of the first public blockchain, Bitcoin, other types of blockchain have been developed to overcome its limitations. Among them we find the lack of scalability, the relatively low number of transactions per second supported by the system, the low speed to register data (in Bitcoin blocks with new information are published every 10 minutes on average), or the fact that cryptocurrencies are needed in order to carry out operations in the blockchain. The latter can be a drawback because of the hassle of having to link payments to most operations, as well as having the uncertainty of the price that each transaction will have.

2.1. Technology behind blockchain

The blockchain itself is not a revolutionary invention. Blockchain technology is nothing more than the wise combination of already existing technologies and used in many areas of computer security such as, cryptographic keys, digital signatures, hashes, peer to peer networks ... The smart way to combine all of these, gave birth to this technology.

Blockchain can be considered a very secure technology, because the information is shared, produced and maintained in a completely decentralized way, due to being a distributed ledger. Everyone who joins the system can get a copy of the entire blockchain and according to the rules of the blockchain, they can accept or deny blocks by participating in the consensus algorithm. When the information is accepted, a block is created and distributed thanks to the peer-to-peer network.

A blockchain is just a growing list of records (basic data structure) that are called blocks and are linked using cryptography. Each block contains the hash of the previous block, a timestamp and the transaction data to be transferred between the two parties. The transactions data are normally represented using a Merkle tree [3].

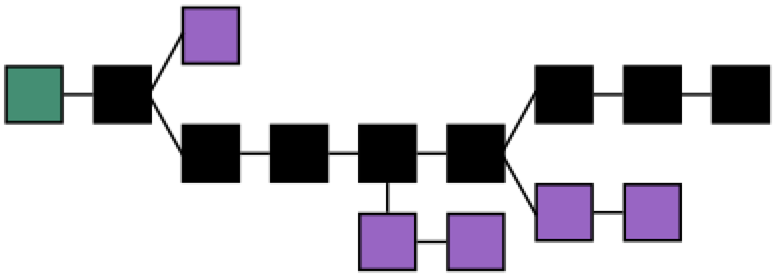


Figure 2.1. Simple blockchain structure (Extracted from [4])

As shown in **Figure 2.1** we can see the consecution of the linked blocks. It always start from a special block called Genesis Block, which is the only one that has no previous block. Then we can see two types of blocks. The black ones are the blockchain itself, one block after another linked to the previous one. On the other hand, we have the purple blocks, which are forks of the main branch. This happens when at a certain moment, two blocks are generated at the same moment and both are valid. To discard one of them, wait for other blocks to be added behind. Thus, the chain that is longer is the one that will finally last and the other branch that had been created is cancelled, also cancelling the possible transactions related.

The idea is simply what is shown in **Figure 2.2**. Ideally, someone wants to send money (it does not always have to be money [5]) to someone else, which generates a transaction indicating the issuer, the amount and the recipient. When all the nodes approve this transaction, it can be included in a valid block that the other nodes have to validate again. When this happens, this block is added to the blockchain and finally the transaction will be executed, leaving a record on the blockchain that no one will be able to delete or modify it.

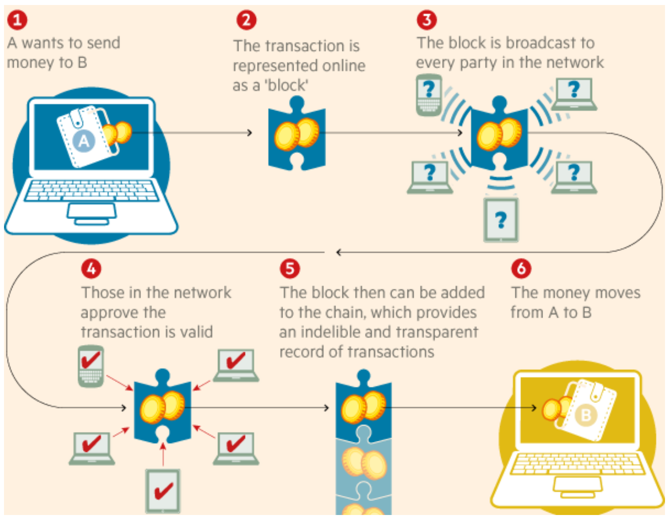

1 A wants to send money to B
2 The transaction is represented online as a 'block'
3 The block is broadcast to every party in the network
4 Those in the network approve the transaction is valid
5 The block then can be added to the chain, which provides an indelible and transparent record of transactions
6 The money moves from A to B
A
B

Figure 2.2 How a blockchain works (Extracted from [6])

In general, terms, we can define two types of distributed ledgers:

> **Public blockchains**, which are the best known due to Bitcoin. They are typically based on the Proof of Work (PoW) consensus algorithm. Everyone can download the blockchain, the software, and participate in the consensus algorithm.

> **Private or permissioned blockchains**, where only users that participate are known and trustworthy. The blockchain can be owned by a specific company or by consortium, or set of companies. Part of the blockchain can be shown to non-members and made public so that third parties trust your way of working. They use a voting / multi-party system as a consensus algorithm, they are much less expensive in terms of computing and their speed in terms of transactions per second is much greater than public blockchains.

In this thesis, we are going to focus on the second group, the permissioned ones.

2.1.1. Permissioned blockchains

Permissioned blockchains are usually closed systems where the information stored in the blockchain is kept private among its member nodes. In this sense, a blockchain with permissions is much less revolutionary than public ones. The blockchain can be owned by one single company or by a consortium composed of several companies. The members usually have certain conflicts of interest and the information that stored in the blockchain is usually of high value for several of the consortium members. For these reasons, the members of the consortium need a platform where the management and storage of the data is shared among them. In a permissioned blockchain, the members of the consortium may have special permissions depending on their responsibility within the platform, and, for example, they may have permissions to write new blocks, to validate them or just to use the system and send transactions.

2.1.2. Properties of a blockchain

The technology blockchain is considered revolutionary in many aspects. Let us look at its properties:

> **Safety**: As long as we have the majority of honest nodes, we are convinced that all the information added to the blockchain is correct and agreed by all the nodes.

> **Liveness**: As long as we have the majority of honest nodes, we know that we would always be able to add messages to the blockchain, as we will always have an agreement.

> **Peer-to-peer replication:** The entire ledger is replicated and shared with the peers. Any new node can request a copy to a peer and check every block.

> **Irreversibility and immutability**: Once something has added to the blockchain can never be changed or removed, as there is P2P replication and all nodes have already a copy of the ledger.

> **Decentralization**: There is no central node, which has the control of what is going on the blockchain. All nodes can send transactions to the blockchain and new blocks are agreed by consensus among nodes.

2.1.3. Consensus algorithm

The consensus algorithm is one of the key elements of a blockchain. Consensus algorithms are decision-making processes for a group, where each individual within the group builds and supports the decision that works best for them. It is a form of resolution where members must support the majority decision, whether they like it or not. In simple terms, it is just a method of making decisions within a group. Imagine a group of ten people who want to make a decision about a project that benefits them all. Everyone can suggest an idea, but most will be in favour of the one that benefits them the most. Others will have to deal with that decision, whether they like it or not. Consensus algorithms not only agree with the majority of the votes, but also agree with the one that benefits them all. Therefore, it is always a victory for the network. These blockchain consensus models consist of some particular objectives, such as:

> **Reaching an agreement**: The mechanism gathers all the group's agreements as much as it can.

> **Collaboration**: Everyone in the group points to a better agreement that results in the collective interests of the group.

> **Cooperation**: Each member will work as a team and set aside their own interests.

> **Participation**: Everyone in the network must participate in the voting. No one is left out or no one can be left out of the vote.

> **Activity**: Each member of the group is equally active. There are no members with more responsibilities than others in the group.

In the market, there are currently many types and depending on what the main objective of your blockchain is, it will be more convenient for you one or the other.

2.1.3.1. Practical Byzantine Fault Tolerance (PBFT)

The PBFT algorithm is based on the Byzantine fault tolerance (BFT) [7], which is the characteristic of a distributed network of reaching consensus (agreement on the same value) even when some of the nodes of the network do not respond or respond with incorrect information. The goal of a BFT mechanism is to protect against system failure by making collective decisions to reduce the influence of faulty nodes.

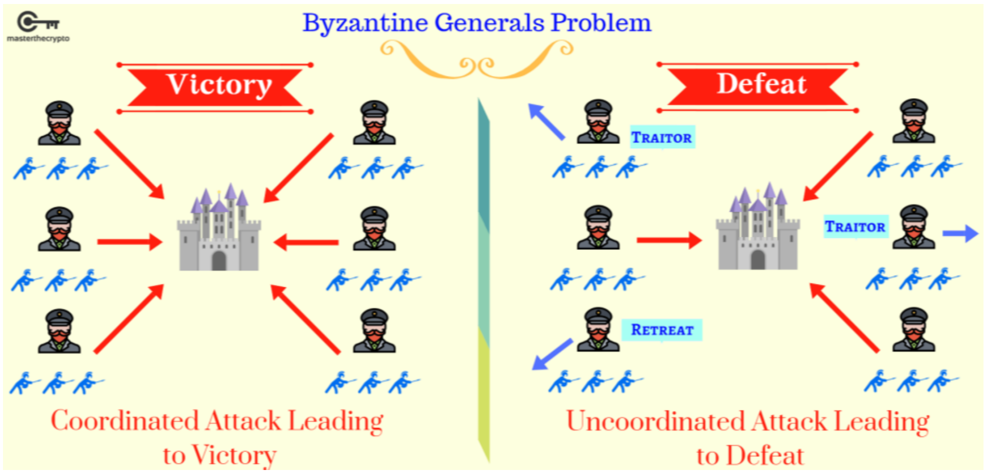

masterthecrypto
Byzantine Generals Problem
Victory
Defeat
Traitor
Traitor
Retreat
Coordinated Attack Leading to Victory
Uncoordinated Attack Leading to Defeat

Figure 2.3 Byzantine Generals Problem (Extracted from [8])

Figure 2.2. depicts the Byzantine Generals Problem, in which generals have to decide whether to attack or retreat. Some generals prefer to attack while others prefer to retreat. The important thing is that each general agrees on a common decision, since a half-hearted attack by a few generals would turn into defeat and would be worse than a coordinated attack or a coordinated withdrawal.

The idea then of the PBFT algorithm is that we need a minimum of $3f + 1$ messages where f is understood as the maximum number of faulty/byzantine messages. Then knowing the number of nodes that our network has, we can know how much byzantine nodes we can have. Thus, if we have 3 faulty nodes, we would at least 10 nodes in order to have the supermajority of 7 honest nodes.

The PBFT algorithm is designed for asynchronous consensus systems and optimized in a more efficient way to deal with all problems. The level of communication is quite high because nodes want to verify all the information that is found on the network. This eliminates the problem of unreliable information but introduces a new one: lack of scalability. Large numbers of nodes lead to many communications among them, so the system may have difficulty keeping track of all nodes and will not be able to communicate with each one. Because of this, it is suited for permissioned blockchains where the number of users is limited.

In this thesis, we will focus on studying and evaluating this algorithm in different blockchains, mainly Exonum and Neo.

2.1.3.2. Proof of Work (PoW)

Proof of work (PoW) was the first consensus algorithm introduced in a blockchain, in particular in Bitcoin. Many other blockchain technologies, such as Litecoin [9], use these consensus models to confirm transactions and produce blocks. This responsibility falls on all individual nodes called miners and the process they use to maintain it is called mining. The central principle behind this technology is to solve a complex mathematical problem that require a lot of computational power. The complexity of this problem can be adjusted to reach a given speed in terms of blocks/transactions per second. In Bitcoin, for example, a new block is created every 10 minutes.

PoW has many advantages, such as its high level of security, but it also comes with many flaws. Energy consumption is so great that it is becoming a problem for the world where we are running out of energy. Miners in the system have to face a large sum of money due to electricity consumption, which makes the process of mining not worth it if they do not reach to mine the new block. The advantage of this is that the attack of the 51%, that is, managing to control the blockchain by generating erroneous blocks but that the majority (51%) of them accept it since they are under your control, it is very difficult to manage due to the high cost that you must assume that in no case would it be profitable.

2.1.3.3. Proof of Stake (PoS)

Proof of Stake (PoS) is a consensus algorithm that deals with the main problems of the PoW algorithm. Miners can join the mining process using their coins to participate. Each individual can mine or even validate new blocks only depending on the amount of coins (stake) they have. Therefore, in this scenario, the more coins you have the more possibilities you will have to mine a block. Rather than investing in expensive hardware to solve a complex puzzle, validators invest in system currencies by blocking some of their currencies as a stake. A validator is chosen to generate a new block based on its economic participation in the network. Therefore, PoS encourages validators through an incentive mechanism to reach an agreement.

2.1.3.4. Delegated Proof-of-Stake (DPOS)

The delegated Proof of Stake (DPOS) is a variation of the PoS method described in the previous section. It is a consensus protocol designed for highly scalable blockchains. The implementation of this protocol offers Byzantine Fault Tolerance (BFT). This means that it provides high levels of security for use on public blockchains. All the participants in the network choose by voting a set of "delegates". The voting power of participants in the network is proportional to the amount of cryptocurrency it has. The more you have, the more important is your vote. Once chosen, they form a group that allows the BFT protocol to be implemented. This is because their quantity is defined and limited, and there is partial trust in them. The delegates define a rotation of leaders, which are in charge of creating the blocks. This means that each delegate has a turn within the rotation to produce a block. Thanks to this action, said delegate can generate a block and collect a reward for it.

The most widely used consensus algorithms nowadays are summarized in the following **Table 2.1**.

Table 2.1 Consensus algorithm comparison

	Byzantine Fault Tolerance	**Proof-Of-Work**	**Proof-Of-Stake**	**Delegated Proof-of-Stake**
Energy Consumption	Very Low	High	Low	Very Low
Transaction Per Second	100 – 2500	7 - 30	30 – 173	2,5 – 2500
Transaction Fees	Very Low	High	Low	Low
Structure	Decentralized	Decentralized	Decentralized	Centralized
Example	**Exonum, Neo**	Bitcoin	Ethereum, Dash	Bitshares

As we can see in **Table 2.1**, there is no single solution and this will depend on what we are looking for in our system. For example, Bitcoin, despite being one of the best known, which uses the POW algorithm, presents us with high-energy consumption and very low transactions per second. On the other hand, the DPOS offers us a high rate of transactions per second and a very low energy consumption. However, it is a system that, due to the voting system, tends to be centralized.

3. Exonum

The Exonum [10] project is an open-source framework for creating blockchain applications over the Exonum technology, implemented by the company Bitfury. It can be used to create cryptographically powered distributed ledgers. It is a permissioned blockchain, where all the set of nodes of the blockchain are known and controlled by a company or a consortium. It is based on Rust programming language [11], which is consider one of the most secure programming languages.

Exonum is a framework; it is not a ready-made blockchain (like Bitcoin). Instead, Exonum can be used to create blockchains, just as if some frameworks can be used to create web applications. Then there is no cryptocurrency relate to that blockchain, so there is no fee for adding a block to the blockchain. Then with Exonum, they bring several different services, endpoints, light clients and even a bitcoin anchoring for saving the actual state of the blockchain in a public environment.

In the following sections, the most relevant details of the consensus algorithm used in Exonum, which is based on the PBFT algorithm. For a more detailed and extensive description we refer the reader to the Exonum documentation [12].

3.1. Algorithm Overview

In this section, we first explain the main concepts needed to understand the algorithm, and then a description of the former.

There are three type of nodes:

> Full node: Which is a node that stores the full state on the blockchain. It can also receive transactions or help other nodes to reach the actual $height$ of the blockchain.
> Validator node: Which are the nodes in charge of proposing new blocks participating in the consensus algorithm.
> Leader node: Which is the validator node that is in charge of proposing a new block sending a new $proposal$ message.

The blockchain is a set of linked blocks. Each block that is generated equals a $height$ of the blockchain. Therefore, if we say that we are now at $height = 10$, we mean that the blockchain already has nine committed blocks. $Rounds$ are steps within one $height$ with the purpose of accepting a block. In each $round$, we have a leader who generates a block proposal and this must be accepted following the consensus algorithm by the other validating nodes. If this is not achieved, we move forward to a new round, which generates a change of leader and consequently a change of block proposal.

3.1.1. Consensus messages and their fields

The consensus algorithm uses the following types of messages:

> $Propose$, $Prevote$ and $Precommit$ messages which are the core of the algorithm and are used in the different phases of the consensus algorithm.
> $Request$ Messages, which are needed to request missing data from peers.

> $Block$ Messages, used to transmit an entire block of transactions to another lagging node.
> Auxiliary messages, such as $Status$ and $Connect$. If the $height$ of the node has not changed since some interval time, the node broadcast a $Status$ message in order to tell the other validator nodes. Moreover, the connect message is used to inform the other blockchains that you are still available on the network saying that you are alive.

The most important fields that all messages have are as follows:

> $Validator\_id$, which is the index related to one specific validator listed in the configuration file.
> $Height$**,** which indicates the height to which the message is related.
> $Round$**,** which indicates the round number to which the message is related. Then all messages can only be processed if we are in the round that it said. If it indicates a past round, it is discarded, but if it is from a future round it is keep in a queue.
> $Hash$**,** which is the hash of the message. It is used to verify that the message is the same as the sender broadcast.

There are also two parameters that depending on the message have which are: $Propose.prev\_hash$ which is the hash of the previous block and $Prevote.propose\_hash$ which is the hash of the $Propose$ message to which $Prevote$ belongs.

3.1.2. Parameters

The consensus algorithm has a set of global configuration parameters:

> $Propose\_timeout$, which is the proposal timeout after the beginning of a new height.
> $First\_round\_timeout$, which is the time before the second round starts.
> $Status\_timeout$, which is the period between the $Status$ message with a current $height$ information

Moreover, set of node parameters, if we focus on the important ones we have:

> $Current\_height$, which is the current blockchain height.
> $Queued$, queue of consensus messages ($Propose$,$PrevotePrecommit$) from the future height or round.
> $Proposes$, which is a set of know block proposals
> $Locked\_round$, which is the round in which the node has lock on a proposal.
> $Transaction\_pool$, which is the set of transactions that have not yet been added to the blockchain.

3.1.3. Proof-of-Lock

The proof-of-lock (PoL) state it is reached when a node receives more than 2/3 of prevote messages, regarding the number of total validators, for the same proposal in the current round and at the current $height$ of the blockchain. A node cannot have more than one PoL state. Therefore, if the node receives a message where another node indicates a PoL with a lower round, the node must replace this state with the new PoL.

3.2. Algorithm Properties

Any PBFT-based algorithm relies on the following properties:

> Each validator node is either written to the genesis block or chosen through the consensus.
> The validators and the network are partially synchronous. Then we can assume that mostly messages are processed instantly.
> $N \geq 3f + 1$, where *N* is the total number of validator nodes and *f* the number of byzantine nodes. Then, if we set the total number of validator nodes, we can compute the maximum number of Byzantine nodes that the blockchain can deal with.
> Absence of Forks. If some non-Byzantine node commits a block to the blockchain at certain $height$, no other node can add another block at that $height$.

3.3. Algorithm Description

In this section, we are going to explain how the algorithm PBFT works in the case of Exonum. We can have a deeply view with the following block diagram in **Figure 3.1**.

The scheme describes the process of adding a new block to the blockchain in 3 stages, which are explained in the following.

3.3.1. Leader Election and Broadcast Propose

Once a new blockchain $height$ or round inside it starts, the leader is changed. The leader is chosen following the following formula:

$$leader\_id = (current\_height + current\_round) \% \, validator\_count$$

where the validator with $leader\_id$ will be the one proposing a new block. This is in order to achieve a degree of censorship resistance (any correct transaction broadcasted to every validator will be committed eventually) and weak form of chain quality (any sequentially committed blocks is guaranteed to be proposed by non-Byzantine validators) [13].Thus if we are in the height 10, in the round 5, and we know that we have four validator nodes, we can easily know the leader validator:

$$leader_{id} = (10 + 5)\% \, 4 = 15 \, \% \, 4 = 3$$

Once we have selected the leader for the round, it will be in charge of broadcasting a $Propose$ message. The $Propose$ message contains a set of transactions to be included in the next block. The message includes only transactions hashes, to make the verification of transactions more efficient and faster since it avoids unnecessary calculations. Any validator node that receives this $Propose$ message, can request any missing transaction.

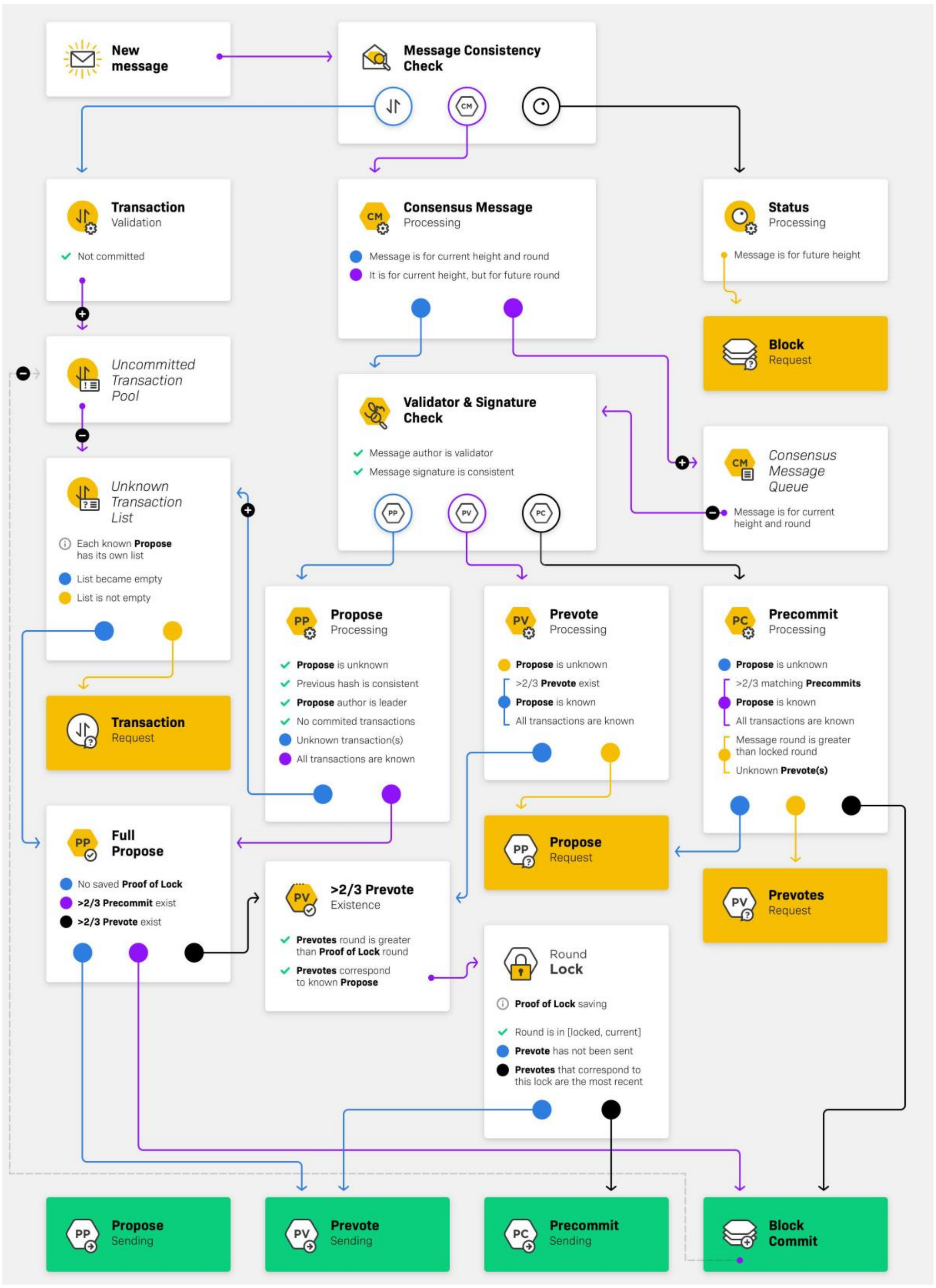

New message
Message Consistency Check
Transaction Validation
Not committed
Consensus Message Processing
Message is for current height and round
It is for current height, but for future round
Status Processing
Message is for future height
Block Request
Uncommitted Transaction Pool
Validator & Signature Check
Message author is validator
Message signature is consistent
Consensus Message Queue
Message is for current height and round
Unknown Transaction List
Each known Propose has its own list
List became empty
List is not empty
Transaction Request
Propose Processing
Propose is unknown
Previous hash is consistent
Propose author is leader
No commited transactions
Unknown transaction(s)
All transactions are known
Prevote Processing
Propose is unknown
>2/3 Prevote exist
Propose is known
All transactions are known
Precommit Processing
Propose is unknown
>2/3 matching Precommits
Propose is known
All transactions are known
Message round is greater than locked round
Unknown Prevote(s)
Full Propose
No saved Proof of Lock
>2/3 Precommit exist
>2/3 Prevote exist
Propose Request
Prevotes Request
>2/3 Prevote Existence
Prevotes round is greater than Proof of Lock round
Prevotes correspond to known Propose
Round Lock
Proof of Lock saving
Round is in [locked, current]
Prevote has not been sent
Prevotes that correspond to this lock are the most recent
Propose Sending
Prevote Sending
Precommit Sending
Block Commit

Figure 3.1 Exonum consensus algorithm schema (Extracted from [13])

3.3.2. Prevote Stage

Once a validator node receives the $Propose$ message from the leader node, it can broadcast a $Prevote$ message. A prevote means that the validator node successfully verify that the proposal is consistent (every field is correct and previously known by the node) and has all transactions specified in it. Once it has received at least $> 2/3$ $Prevote$ messages with respect to the total number of validators, it indicates a $lock\_round$ (**3.1.3 Proof-of-Lock**) and moves to the next stage and broadcasts a $Precommit$ message.

3.3.3. Precommit Stage

Once a validator node receives a $Precommit$ message, it knows that there is another node in the network that has received $> 2/3$ $Prevote$ messages. Then, it can send a $PrevotesRequest$ to this node to move to $lock\_round$ (it will check that all the $Prevotes$ messages are correct). Once it moves to the $lock\_round$, before broadcasting the message, it executes all the transactions specified in the voted proposal. Then the $Precommit$ message includes the result of the proposal execution in the form of a new state hash [14]. The $Precommit$ message expresses that the node has executed the transaction and it is ready to add the block to the blockchain. However, before that, it needs the supermajority votes of $> 2/3$ $Precommit$ messages to see what other validators have to say on the matter just to be sure. If it receives the supermajority votes, it can commit the block and move to the next $height$. The other nodes may also reach this state, but they may also not have received the messages for multiple reasons. If this is not the case, when they receive a message from the node that if they have committed the block they will ask them to send the block so they can reach the height of the other node. Upon receipt, they will verify the new block and update the copy of the blockchain they have stored.

3.4. <u>Distinguishing features</u>

Exonum is based on the PBFT algorithm but has some features that make it different from other PBFT-based approaches.

3.4.1. Unbounded Rounds

In Exonum, rounds have a fixed start time but do not have a defined end time (a round only ends when the next block is received). This helps decreasing the delay when network connection between validators is unstable. Thanks to the way the timeout is increased, which will described in the section **5.5.Results**, it makes the rounds get longer and longer. Thus, in cases where the network is very unstable, the nodes will wait long enough and will be able to receive the necessary messages for the majority of 2/3 and that the blockchain continues to commit blocks. In a scenario where this timeout does not increase, if the network remains unstable it would never be able to commit a block until it becomes stable.

3.4.2. Work Split

In Exonum, the Propose message only include transactions hashes. Transactions are only applied when a node locks on a Propose. Delaying the moment of transactions processing reduces the negative impact of malicious nodes. If the transactions are executed once the node receives a $Proposal$ and finally the proposal failed, it would have been a waste of time. Moreover, it splits the transaction process among the following steps:

At the first step, the prevote stage, validators only check if all the transactions in the Propose are correct and already stored in the pool of unconfirmed transactions by this node. Nodes verified the transaction when they receive it, as they do not store any incorrect transaction.

At the second step, the Precommit stage, validators apply the transactions to the current blockchain state. If they commit it, validators ensure that they achieved the same results (hash of the block) after applying the transactions in the proposal.

The added value of Exonum with respect to other approaches is that if a Byzantine validator submits a proposal with different transactions or in a different order, they do not need to check the order of the transactions. When receiving a prevote message from another node, the $propose\_hash$ will be different, so they will discard the prevote messages and wait to advance the round. With this, the negative impact of a Byzantine node is reduced, since the nodes do not need to do any kind of checking.

3.4.3. Requests Algorithm

In this case, Exonum is committed to a much more efficient algorithm when it comes to retrieving information from a node that is not up to date. Exonum creates an algorithm different from the typical one, known as gossip algorithm [15], which ask other peer nodes what information do they have and then ask the one with more information available, that many other blockchain uses. In this case, the advantage is that the information is learned and extracted from the consensus messages that the node receives, in order to try to request information from the best of the validators peers. The algorithm is detailed in the Exonum whitepaper ([13]).

4. Neo

Neo [16] is a distributed network that uses blockchain technology and digital identity [17] to digitize assets and automate the management of digital assets through smart contracts. The Neo network has two tokens, NEO, with a maximum of 100 million tokens, representing the right to manage Neo blockchain and GAS representing the right to use Neo Blockchain with a total limit of 100 million. The Neo network charges a fee for any type of operations and storages of tokens. Neo is focused on the implementation of NeoContracts, which are a kind of contract over the internet. NeoContracts refer to any computer program, which can automatically execute the terms of its preprogramed contract. They have also developed NeoQS, which is a lattice-based cryptographic mechanism in order to deal with the future quantum computers.

Neo proposes the dBFT (delegated Byzantine Fault Tolerance) consensus algorithm based on the Practical Byzantine Fault Tolerance (PBFT) algorithm. The dBFT algorithm determines the set of validators according to real-time blockchain voting, which effectively improves the efficiency of the algorithm, saving time on the block generation and the transaction confirmation. This is thanks that the validator nodes can be changed easily if there is a problem with them, such as disconnections or Byzantine nodes.

In the following sections, we are going to discuss how it is implemented in the Neo blockchain network. For a more detailed explanation, we refer the reader to the Neo 2.x documentation [18].

4.1. Algorithm Overview

The algorithm ensures security as well as usability. All consensus nodes are required to maintain a state table to record current consensus status. The data used for consensus from its beginning to its end is called a $View$. If consensus cannot be reached within the current $View$, a $View\ Change$ will be required. We identify each $View$ with a number v, starting from 0 and it may increase until achieving the consensus without no limit.

In this case, we have different types of nodes and validators:

> $Validator\ Node$ or $Consensus\ Node$, which is a node that can participate in the consensus algorithm.
> $Normal\ Node$, which are nodes that can only send transactions to the blockchain.
> $Spekaer$ or $Primary$, which is the leader of the consensus algorithm round.
> $Delegate$ or $BackUp$, which are the nodes responsible of voting on the consensus stages.
> $Candidate$, which are the nodes that are nominated for validator election

In addition, we list some of the most important messages used in the algorithm.

> $Prepare\ Request$, which is sent by the leader of the round.
> $Prepare\ Response$, which is the message that validator nodes broadcast if the $Prepare\ Request$ is correct.
> $Commit$, which is the message to inform that enough Prepare Response have been collected.
> $Change\ View$, message for requesting a view changing attempt.

4.2. Algorithm Description

A quick look at the algorithm can be seen in the following **Figure 4.1**.

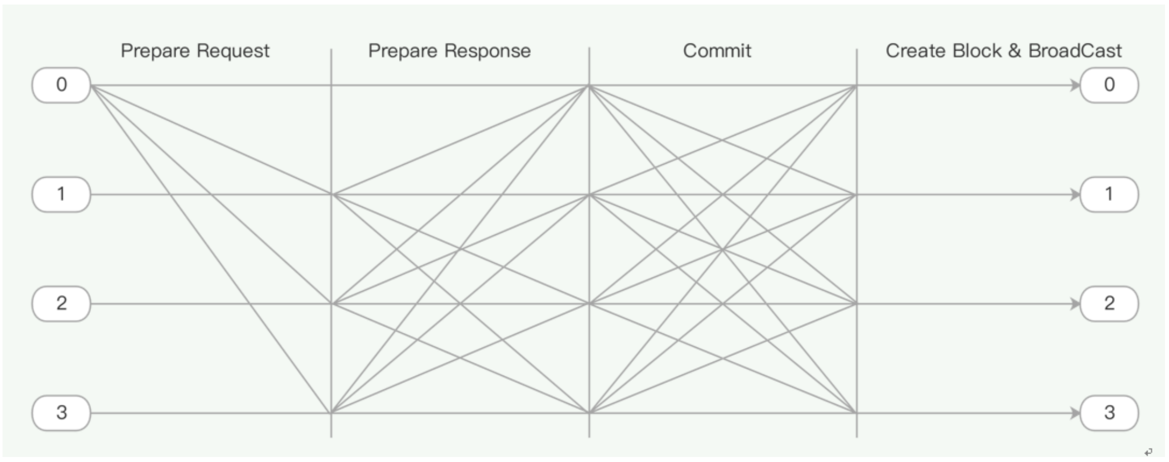


Figure 4.1 Neo Algorithm Overview

The process of consensus takes place along three steps:

4.2.1. Leader Election and Broadcast Prepare Request

Once we move to a new height, we need a leader in each new round. In this case, the leader is called $Speaker$ or $Primary$ and the following algorithm determines it:

$$speaker_{id} = (height - view) \bmod N$$

where N is the number of validator nodes and view always start from 0.

Once the $Speaker$ has been selected it will broadcast a $Prepare\ Request$ message to the other validator nodes. This $Prepare\ Request$ is composed of the hashes of the transactions that are in the memory pool of the leader.

4.2.2. Prepare Response Stage

Once the $Delegate$ nodes receives a $Prepare\ Request$ message, they check if all the transactions are collected and no errors can be found in any of them. If it is the case, they will broadcast a $Prepare\ Response$ message. If there is no $Prepare\ Request$ message before the timeout or the transactions are not correct, it broadcasts a $Change\ View$ message.

When the validator node collects enough $Prepare\ Response$ messages, which means that it receive more than 2/3 messages with respect to the total amount of validator nodes, it broadcasts a $Commit$ message.

4.2.3. Commit Stage

Once the validator node, $Speaker$ or $Delegate$, gets more than 2/3 $Commit$ messages creates a new block and broadcast it. If there are not enough Commit messages before the timeout time, broadcast a $ChangeView$ message.

4.2.4. Change View

The change view timeout is set using the following formula

$$2^{v+1} * Tblock$$

where $Tblock$ can be set by each validator node configuration.

Then, if the transaction verification fails or there are not enough messages, this will trigger this request. However, there are some cases that the $Change\ View$ will not be applied. If the sum of nodes with Commit send is greater than the 2/3 of the validator numbers, or there are no 2/3 Change View messages, it will force to send a $Recovery\ Request$ message by this node. Nevertheless, if there are enough $Change\ View$ messages it will change the local view, initialize a new local consensus context and determine the next round speaker in these new view.

4.3. Distinguishing features

In this section, we are going to explain which are the main features that Neo has that are different from other blockchains.

4.3.1. Transport Protocol

When a consensus message enters the P2P network, it is transmitted like other messages. This is because the consensus nodes do not have the IP address of other consensus nodes. Consensus nodes are not directly connected. That is, ordinary nodes can also receive consensus messages. The consensus message broadcast flow it is not straightforward. The intermediate nodes, which receive a consensus message, forward them to other nodes. Before this, the nodes send an invitation to the neighbours to see if they do not yet have the information in this consensus message. If they do not have the information yet, they will reply with a $getdata$ to get the information from the message of the message. This will follow the same process again to make it easier for the message to reach a node that does participate in the consensus and finally achieve consensus.

4.3.2. Voting

There is a voting process where any node can be part of the consensus. The idea is that any node can initiate a transaction to ask to be a candidate (validation node), as long as they want to pay the fee, and anyone with Neo can vote for any candidate to be part of the consensus node set and the consensus number that form it. When voting for validator candidates, it actually includes two parts: the number of consensus nodes, and name list of consensus nodes. It goes through a special transaction called $StateTransaction$.

Following a discrete probability function, in which the probability of the x consensus node equals its proportion of votes [19], we calculate the number of consensus nodes as $Count$, and take validators from the candidate list ranked by votes in descending order. When there is not enough candidates, it will be add validators from $StandbyValidators$. With that, we have set the new validator nodes, which are a consensus between all the nodes that have Neo.

5. Performance Evaluation

Once we have brush up the necessary background to understand the basics of the blockchain technology and, in particular, of Exonum and Neo, we can analyse in a more practical way the performance of these blockchains in different scenarios. In this chapter, we describe the tools used to analyse them and the scenarios considered.

5.1. Scenario setup

In order to evaluate the performance of the blockchain, we run a test network with several nodes on a VirtualBox VM (Virtual Machine) with Ubuntu 19.10 (64 bits). In its turn, the VM is running on an 8th Generation Intel Core i5-8250U processor laptop and 5GB RAM were allocated to it. In a second set of tests, four VMs were used, each of them running one Exonum node and 1.5 GB RAM was allocated to each one. All VMs were connected through a VirtualBox NAT-Network.

We need to take into account that:

- Most nodes running on real blockchains are required to have a minimum of 2GB of RAM and preferably, 4GB RAM. This means that our nodes will not perform under ideal conditions but we still get good results in terms of performance.
- By recommendation, Oracle VM VirtualBox recommends not to use more than 65% of the available RAM in the system. If we do not leave enough free memory for the host operating system, it will have to start swapping and running slowly. This might in turn, paradoxically slow down our virtual machine, since it needs resources from the host OS as well. Our laptop has theoretically 8GB RAM so we should not exceed 5.2 GB of RAM. In the scenario with 4 VMs we are actually using 6GB, which slows down the system.

5.2. Exonum test network

Everything needed to run and test the blockchain can be found in the Exonum documentation [12] where the ins and the outs are explained in detail. Some personal problems installing it can be found in the **Appendix 1**.

Exonum is coded in a very modern language called Rust [11], which despite remaining a great unknown to many programmers, it is the programming language most loved by programmers in 2019, for fourth year in a row [20]. Rust emerged in 2010 by Mozilla, looking for a language to code extremely fast, at the same level as C or C++, but without the memory management problems of these languages, which have risen issues regarding memory access and faulty conditions during runtime.

5.3. Neo test network

Neo's current documentation refers to Neo 2.x [21] although recently they have released version 3.x, which is an unstable trial version. The main problem that we have find is that is mostly written in the NET framework. NET is a runtime framework that interprets MSIL bytecode and translates it to the native CPU instruction set on-the-fly [22]. In addition, the framework does the management of memory for variables for you. Therefore, it is always going to be slower and require far more memory than C/C++ or other modern languages like Rust, which compiles code directly to native CPU instructions. Because of this, the performance in scenarios of low available RAM is worse than expected. The advantage is that NET programmers do not have to worry about issues like buffer overflows so it is a much safer environment for running a blockchain node, in which remote exploitation by hackers is a constant concern.

It must be noticed that the documentation is not very friendly, and we have only been able to identify a few parameters that can be configured to run the tests. These will be explained in the next chapter 5.5.

5.4. Scenarios

We have considered three different scenarios to test the performance of both blockchains:

In the first one, we run the entire blockchain in a single VM, in which all nodes communicate by means of the loopback interface. We vary the number of validators and analyse the performance of the blockchain under the presence of a number of dishonest nodes.

In the second scenario, we analyse how the blockchains performance gets worse in the presence of selfish validator nodes, which do not participate in the consensus algorithm.

Finally, the aim of the third one is to test the blockchain in a more realistic scenario by adding a given delay to the communication between nodes. In order to achieve that, we use several VMs and some Linux kernel. We use the tool tc [23] to add the delays. The tool is used to configure Traffic Control in the Linux kernel. We will use filters, which is used by a classful $qdisc$ to determine in which class a packet will be enqueued. Then we can attach any node of the tc tree to one network interface, add a priority to that rule and determine which is the condition) to apply the filter (destination port, source port, maximum bandwidth) if it matches the rule.

5.5. Results

In this chapter, we will discuss and analyse the results obtained by analysing each blockchain and finally we will provide a comparison between them. All the following results are made by running the blockchain from scratch, generating a new Genesis Block [24] and reaching up approximately two hundred blocks. By doing so, we can ensure that the life of the Exonum blockchain is long enough and it is likely that some especial events have happened. For analysis, we remove some of the first blocks to avoid the synchronization process of all the validating nodes, especially the last nodes that we run because they rarely manage to participate in the consensus of the first blocks.

The results are based on the comparison of two blockchains in terms of commit time in different scenarios. We are going to vary the number of validator nodes, the number of byzantine validating nodes, knowing that a byzantine node is one that does not participate honestly in the consensus (it does not send messages, it sends different messages to different nodes, it sends incorrect proposals ...) and varying the quality of the network using different packet error rates.

5.5.1. Limitations

As explained in section 5.1, we do not have the resources to run many virtual machines on the same laptop. Because of this, we cannot simulate scenarios with a large amount of nodes.

The Linux kernel tc tool has an important limitation, since due to its tree routing design applying filters, it puts us as limit up to 6 different filters. Considering that a filter would count as a link between one node and another, it means that we can only have six different connections, which means that we are only allowed to interconnect 4 nodes.

An attempt has also been made to discover what would be the highest transaction per second (tps) that each blockchain can support. Then we can compare it with what they assure us in their benchmarks, and see that although we have less computing capacity, we can reach reasonable tps numbers and that the proportion of tps when adding more nodes is similar to theirs. To generate this, we should somehow generate a constant flow rate above the transaction limit supported by each node, to see how many transactions the blockchain is capable of absorbing. This has been attempted with Exonum, but due to the limited amount of available RAM, it was not possible to generate the flow of more than 2000 transactions to each node as tested by developers in the whitepaper.

As explained in **4.1 Algorithm Overview**, the NET framework requires more memory and RAM power to run properly. After several attempts to run the Neo blockchain, I realize that when the numbers of nodes is greater than 10, the blockchain is not stable and repeatedly has trouble reaching consensus. This means that depending on the test, you can get different commit times without following a clear trend. Due to these scattered results, I consider that it would not be correct to draw valid conclusions from these tests, so for the Neo blockchain, the number of nodes has been limited to a maximum of 10, where it maintains very similar results between different attempts.

5.5.2. Exonum

As explained in the section **4.1 Algorithm Overview** we are going to analyse the performance of Exonum under three different scenarios. Among the different parameters of the nodes that affect the consensus algorithm, we will focus on those that have a higher impact on the algorithm:

- $first\_round\_timeout$ (ms), which is the interval between the actual round and the next rounds of the consensus algorithm. This parameter is used to estimate the timeout value for further consensus rounds as follows:

$$first\_round\_timeout = first\_round\_timeout + (r - 1) * round\_timeout\_increase$$

where r is the actual round and $round\_timeout\_increase$ is the 10% of the previous $first\_round\_timeout$.

- $max\_propose\_timeout$ (ms), which is the amount of time that the leader waits to broadcast a new block proposal, after a block has been committed to the blockchain.
- txs_block_limit (Number), which is the maximum transactions that can be included in a block. In order to speed it up even more, we are going to limit each block to only contain one transaction ($txs\_block\_limit = 1$). With that, we are able to reduce the verification time, which is the time that the other validators use to verify that all the transactions included in the proposal are known by this node.

We will repeat each test two times:

- First configuration: The fastest possible in Exonum.
- Second configuration: With timeouts in line with the Neo blockchain for a more reliable comparison.

5.5.2.1. Fastest time to commit

In this test, we are going to analyse the speed of the blockchain in terms of commit time. The commit time is understood as the time interval between the blockchain accepting a block and the next, taking into account that each block includes at least one valid transaction.

For this test, we evaluate it in the one Virtual Machine scenario and we will increase the number of validators that participate in the consensus from 4 up to 16, which is the maximum recommended [25]. To achieve the fastest scenario, we are going to set the minimum possible value in both parameters. After some testing, it has been observed that with the $first\_round\_tiemout < 1000\,ms$ the blockchain goes too fast, causing each node to advance to a new round too soon, causing the performance of the blockchain to decrease. The $max\_propose\_timeout$ is going to be the lowest one as we are in the fastest possible scenario. So finally, we set the parameters as $first\_round\_timeout = 1000$ and $max\_propose\_timeout = 0$ to avoid that the leader waits to propose a new block.

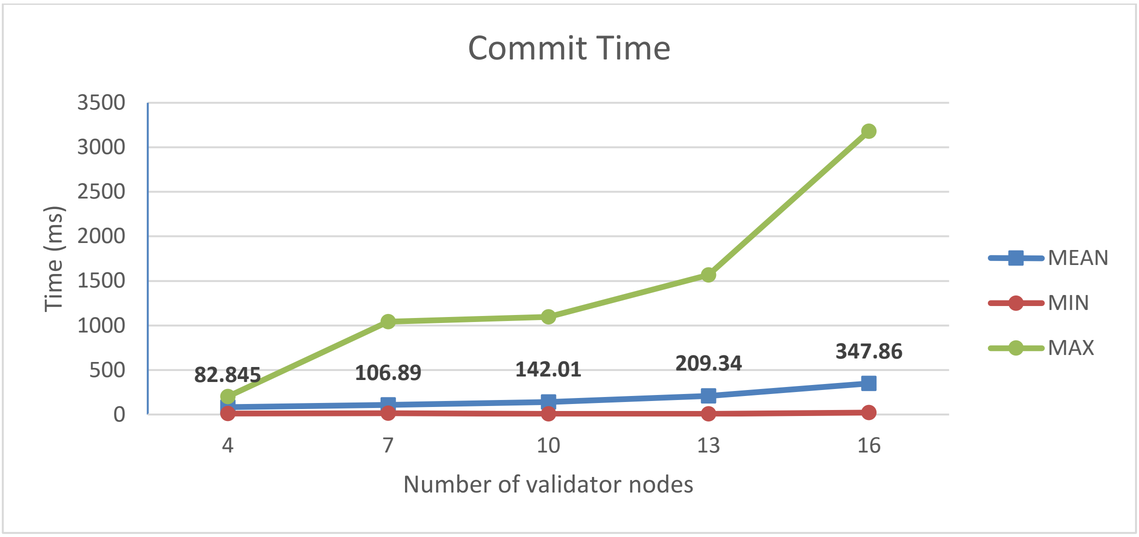


Figure 5.1 Commit time vs number of validator nodes (1st configuration)

Figure 5.1 shows the commit time as a function of the number of validators. It can be seen that, on average, the commit time does not considerably increase despite the increase in the number of validator nodes, from 82ms for 4 nodes to almost 350ms using 16 nodes. This can be modelled as a linear increasing. There is always an almost perfect round, where all the nodes manage to participate at the same time and commit very quickly. On the other hand, we also have one where several nodes have been left a little behind and then we have a lot of delay. We must bear in mind that the network is very fast due to the imposed timeouts and that they are all sharing the same RAM, making a block to take a little longer to commit. So if we have this situation in several nodes simultaneously, we begin to have these maximum values, which take us more than one second from 7 nodes to more than 3 seconds when we have 16 nodes. Obviously, both cases do not occur very frequently, since otherwise the mean would be greatly affected.

As mentioned in section **5.5.1 Limitations** we now repeat the tests with $first\_round\_timeout = 2000$ and $max\_propose\_timeout = 1000$ to have similar implementation of the blockchain similar to Neo.

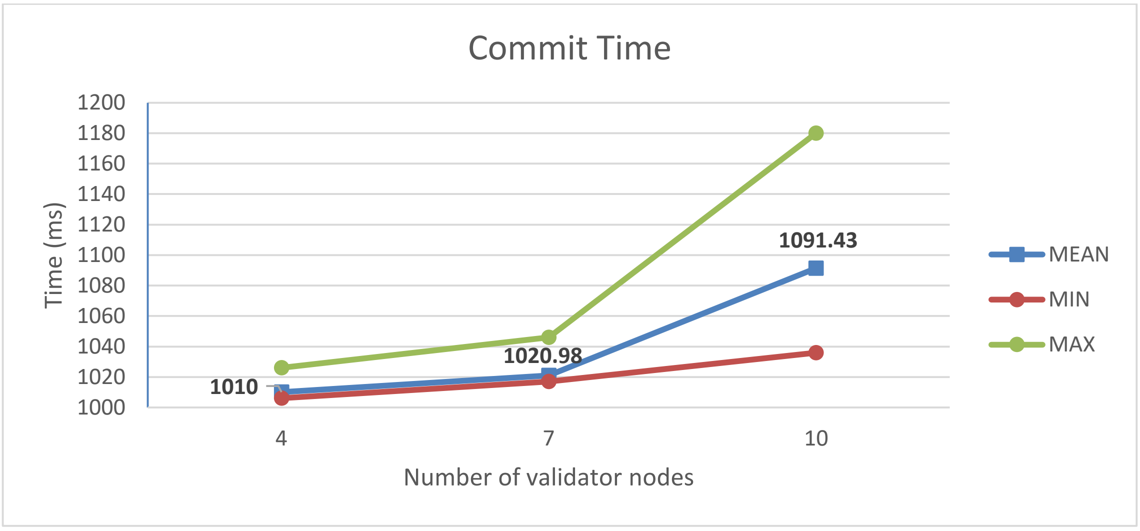


Figure 5.2 Commit time vs number of validator nodes (2nd configuration)

In the **Figure 5.2**, we can see that the mean value has risen above the level of one second due to the one second waiting timeout. The blockchain behaviour is almost the same, in terms of mean time respect to the previous case **Figure 5.1**. However, the overall behaviour we can see that, in the case of the maximum, this does not differ beyond 100 milliseconds with respect to the mean value, which makes a blockchain with a better performance. In the previous case the max values where far distance of the mean value.

5.5.2.2. Time to commit vs malicious nodes

In this test what we are trying to check is the impact of having some malicious nodes on the commit time. As in the previous section, we will increase the number of validators from 4 to 16, all of them running on a single Virtual Machine. For each case, we will disconnect as many dishonest nodes as possible while ensuring that consensus is reached. Again, we will set $first\_round\_timeout = 1000$ and $max\_propose\_timeout = 0$

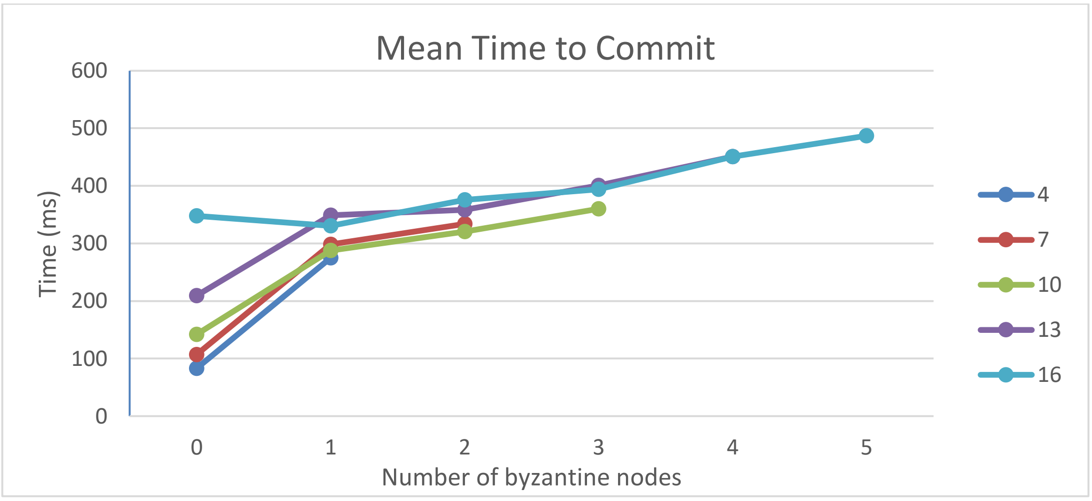


Figure 5.3 Mean time to commit vs number of byzantine nodes (1st configuration)

In **Figure 5.3,** we can see a similar trend of the commit time regardless of the number of nodes. Under the presence of two or more byzantine nodes, in the case of a small number of validators, the event of a byzantine node being the leader occurs more frequently than the in the case of large set of validators. With a large number of validators, the blockchain is slower and then the periods of time in which a byzantine node is a leader are smaller. It makes sense, that the most affected are when the set of validators is short. We also need to remark that depending on the number of validators, it supports having more byzantine nodes. For example, in the case of a network with 10 nodes, we can only have up to 3 Byzantine nodes. This is because the PBFT algorithm has to have a minimum of 2/3 of honest nodes. Therefore, for 10 nodes, we find that it must have at least $\left\lceil \frac{20}{3} \right\rceil = \lceil 6{,}6666 \rceil = 7$. Therefore, in this scenario, we can have 1, 2 and up to 3 byzantine nodes.

We now repeat the tests with $first\_round\_timeout = 2000$ and $max\_propose\_timeout = 1000$ to have similar deployment of the blockchain with Neo.

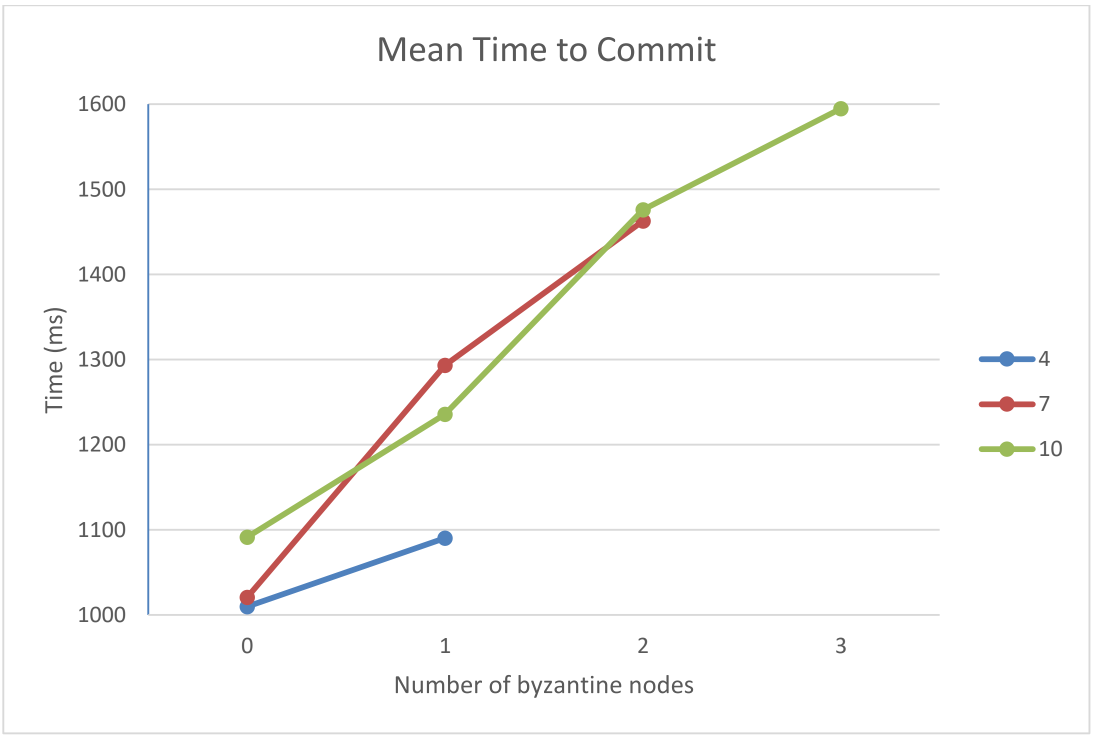


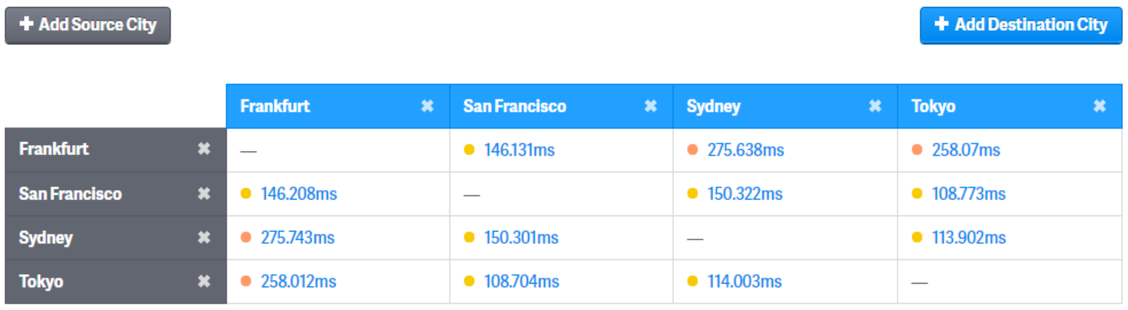
Figure 5.4 Mean time to commit vs number of byzantine nodes (2nd configuration)

In **Figure 5.4,** we can observe that the byzantine nodes now have more effect on the overall mean commit time. Compared to **Figure 5.3**, now that we have the 1 and 2 second extra waiting time to proceed to the next round, it makes the performance of the blockchain worse with the presence of byzantine nodes. From an around 1 second when the three blockchains work without any byzantine node to add half a second when there is two byzantine nodes. If we see the trend it has, it is predictable that for a higher validator set we would have large commit times in the presence of Byzantine nodes. That is why many have processes to be able to change the validator nodes.

5.5.2.3. Time to commit vs Packet Error Rate

So far we have evaluated the performance of the blockchain, in terms of commit time, in an ideal scenario where nodes are directly connected and there are no delays nor packet losses in their communications. In this section we aim at evaluating the impact of the packet error rate on the commit time.

In order to simulate a worldwide-extended network with realistic delays, we have used the tool provided by Wonder Network [26], which provides us with a global network of servers and leverages them to provide network testing [27] . Then we will locate one node in various locations, scattered around the world. One on each continent: Europe, America, Oceania and Asia **Figure 5.5**.



	Frankfurt ✖	San Francisco ✖	Sydney ✖	Tokyo ✖
Frankfurt ✖	—	🟡 146.131ms	🟠 275.638ms	🟠 258.07ms
San Francisco ✖	🟡 146.208ms	—	🟡 150.322ms	🟡 108.773ms
Sydney ✖	🟠 275.743ms	🟡 150.301ms	—	🟡 113.902ms
Tokyo ✖	🟠 258.012ms	🟡 108.704ms	🟡 114.003ms	—

Figure 5.5 Global Ping Statistics

In **Figure 5.5,** we can see which delays have our nodes. For example, if we are in the node located in Frankfurt, we will have a 146 milliseconds delay with San Francisco, a 275 milliseconds delay with the node in Sydney and 258.07 milliseconds with the one in Tokyo. As we can see, the links have almost the same delay in both directions and we will approximate it to the same millisecond.

For these tests, we are going to use four VMs connected through a NAT-Network. We will increase the Packet Error Rate (PER) see the network performance in terms of commit time. After doing some testing using the fastest possible values ($first\_round\_timeout = 1000$ and $propose\_timeout = 0$), we observe some synchronization problems among nodes due to the fact that the values of $first\_round\_timeout$ and link delay are very similar and they do not manage to work properly. There was continuously desynchronization between them. For greater network stability, we increase $first\_round\_timeout$ to 3000 milliseconds.

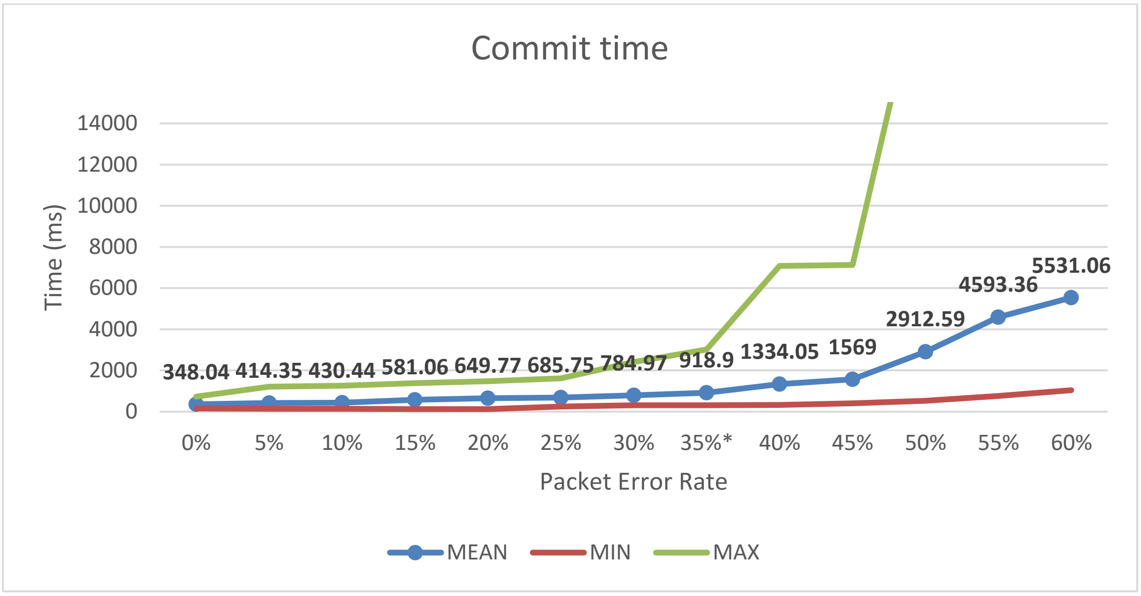


Figure 5.6 Commit time vs Packet Error Rate (1st configuration)

*From this point onwards, the nodes start to loss of synchronization and some of them need to ask their peers for information to reach the current round of the blockchain.

In **Figure 5.6,** we can observe that, as expected, the blockchain performance begins to decrease when the packet loss rate increases. From 35% PER on, the maximum time to commit starts to be considerably high, from 3 seconds to more than 14 seconds when the PER is greater than 45%. Even that, the overall mean time to commit does not degrade too much. Up to a PER of 35%, which can be considered high, the network can still commit in mean in less than a second. Moreover, in the case of going with a higher PER, we could achieve in a network of 60% PER reach a consensus in less than six seconds. If we think in terms of transaction per seconds, when we start to have loss of synchronization, namely from 35% PER, we have maximum times to commit really high which would be fatal for the network due to the tps will decrease and it can cause the pool of unconfirmed transactions to increase too much.

We repeat the test with $first\_round\_timeout = 2000$ and $max\_propose\_timeout = 1000$ to have similar implementation of the blockchain regarding Neo.

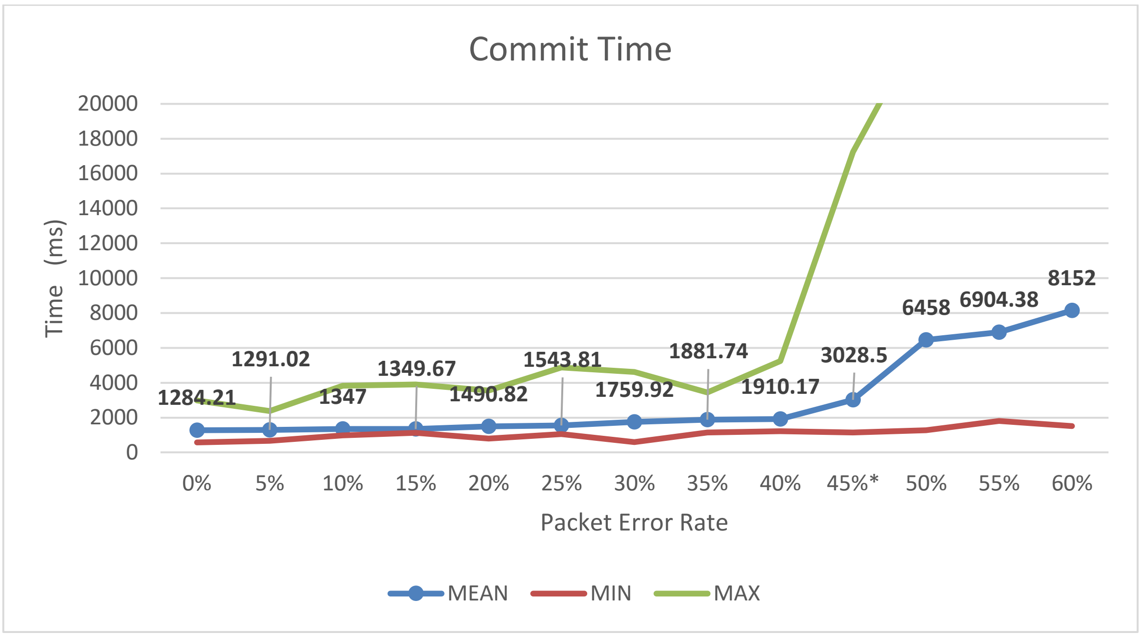


Figure 5.7 Commit time vs Packet Error Rate (2nd configuration)

*From this point onwards, the nodes start to loss of sync and some of them need to ask their peers for information to reach the current round of the blockchain.

In **Figure 5.7,** we see that the blockchain works pretty well with a lot of PER. Up to 40% PER, the mean commit time remains almost unchanged and it barely increases when we add more PER. From 45% and onwards, the blockchain begin to behave very bad due to this large number of messages being lost, but even with that, we could still add some blocks to the blockchain in about seven or eight seconds. Working with that high PER it is really worst for the performance of the network, due to it really decreases the transactions per second in the blockchain.

We would also like to highlight that these test can be affected if we considerably increase the number of transactions in each block. We have tested in the case that there is only one transaction in each block due to it was difficult to manage that our system kept the generation of transactions constant. In order for a node to accept the block proposal of another node, it must know all the transactions included and, if that does not happen, it will issue a message asking the leader node to send them. If these messages, as well as the request or the response messages are lost, the node could never accept the proposal and would not send a prevote message. Therefore, if there is no majority, the round must be changed, leading to a change of leader after $first\_round\_timeout$ seconds. Due to the limitations mentioned previously which were that was really complicated in terms of RAM to generate a constant flow for each node, it could not be tested, so they will be included in chapter **6 Conclusions and future development**

5.5.3. Neo

In this section, we provide the simulation results obtained for the Neo blockchain. In order to be able to perform a comparison analysis between Neo and Exonum, we will perform the same tests as the one performed with Exonum. The Neo blockchain offer fewer configuration parameters, so we have mainly focused on one parameter:

- $SecondsPerBlock$ (s), which is the amount of time that the leader waits to broadcast a new block proposal, after a new block is committed.

After some inspection in the code and reading carefully the documentation, it turns out that there are many more timeouts that are coded in the code [28]. It is not easy to identify which parameter could be relevant or similar to $first\_round\_timeout$ in Exonum, but after having done several tests with different $SecondsPerBlock$, we believe that the relationship between both can be expressed as $first\_round\_tmeout = 2 * SecondsPerBlock$. For this reason, we have tested the Exonum using 1 and 2 seconds respectively.

5.5.3.1. Fastest Time to commit

For this test, we run in the scenario of one virtual machine and we will increase the number of validators from 4 to 10. To test the fastest time to commit, we are going to set the minimum value that can be set, which is $SecondsPerBlock = 1$ second.

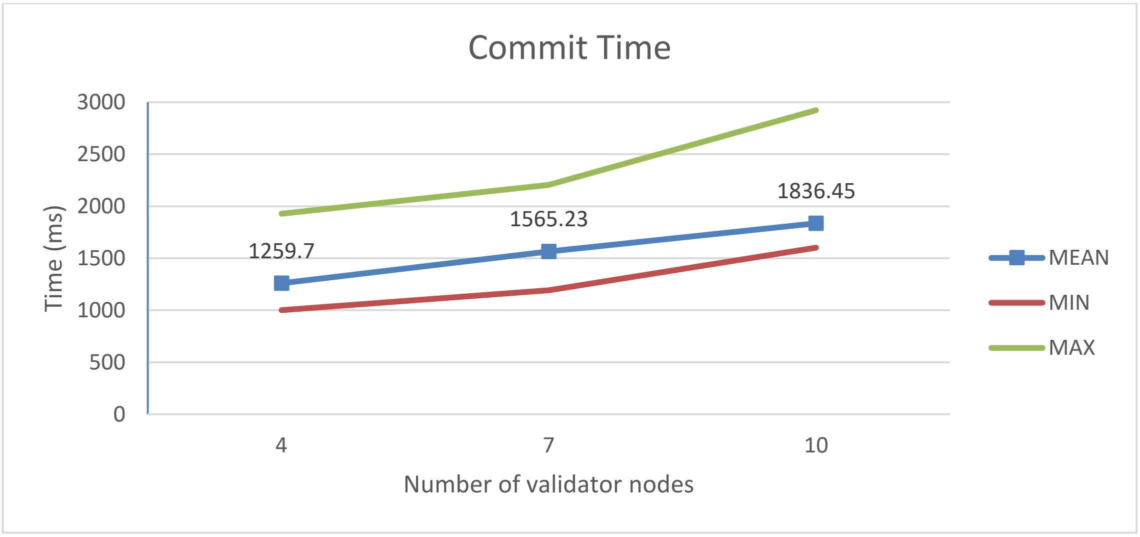


Figure 5.8 Commit time vs number of validator nodes

Figure 5.8 shows what the performance of our blockchain. We can see that the mean value constantly increases with the number of nodes. On the other hand, the minimum value is very close to the mean, which means most time the blockchain commits a block in less time, while we have a few cases in which the commit time can go up to two or three seconds. Seeing this trend, we do not recommend using it with many validation nodes. In fact, in the actual real deployment of the blockchain [29] they only have 7 validator nodes.

5.5.3.2. Time to commit vs malicious nodes

In this section we analyze how the presence of byzantine node affects the blockchain in terms of commit time. For this test, we are going to use the same scenario as we did with Exonum and disconnect as many nodes as possible while still being a consensus (2/3 of number of nodes behave honestly). We are going to use the same parameter which is $SecondsPerBlock = 1$ and keeping in mind that coded is the $first\_round\_timeout = 2$ seconds.

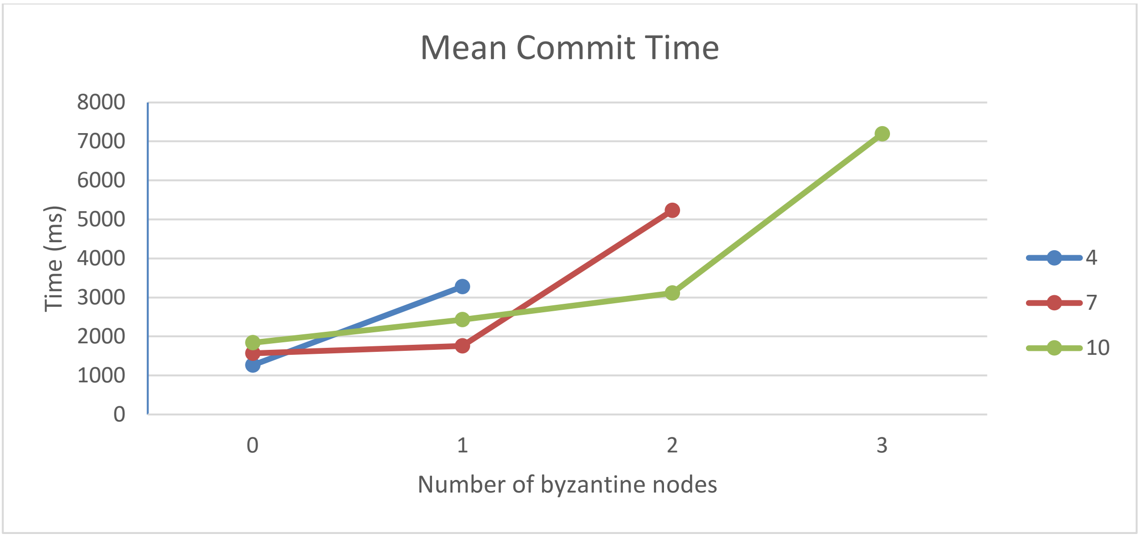


Figure 5.9 Mean commit time vs number of byzantine nodes

In **Figure 5.9,** we can see that there is a tendency that when we are close to the two thirds of the threshold, our blockchain slows down a lot. We can say that almost three times slower. In the cases in which a node fails, except for the four nodes case that we are in the previous case, it tolerates it quite well and does not impair the performance of the blockchain. That is probably why we have the election mode [30] when ad node fails, they can change them in a quickly and by consensus, to to regain a consistent mean commit time.

5.5.3.3. Time to commit vs Packet Error Rate

In this test, we are going to test the blockchain in a real environment, where there are delays and Packet Error Rates. Then we will se how it works in presence of external network factors. To achieve this,we will spread four nodes around the world: Europe, America, Oceania and Asia, applying delays as we did with Exonum and they are listed in **Figure 5.5**.

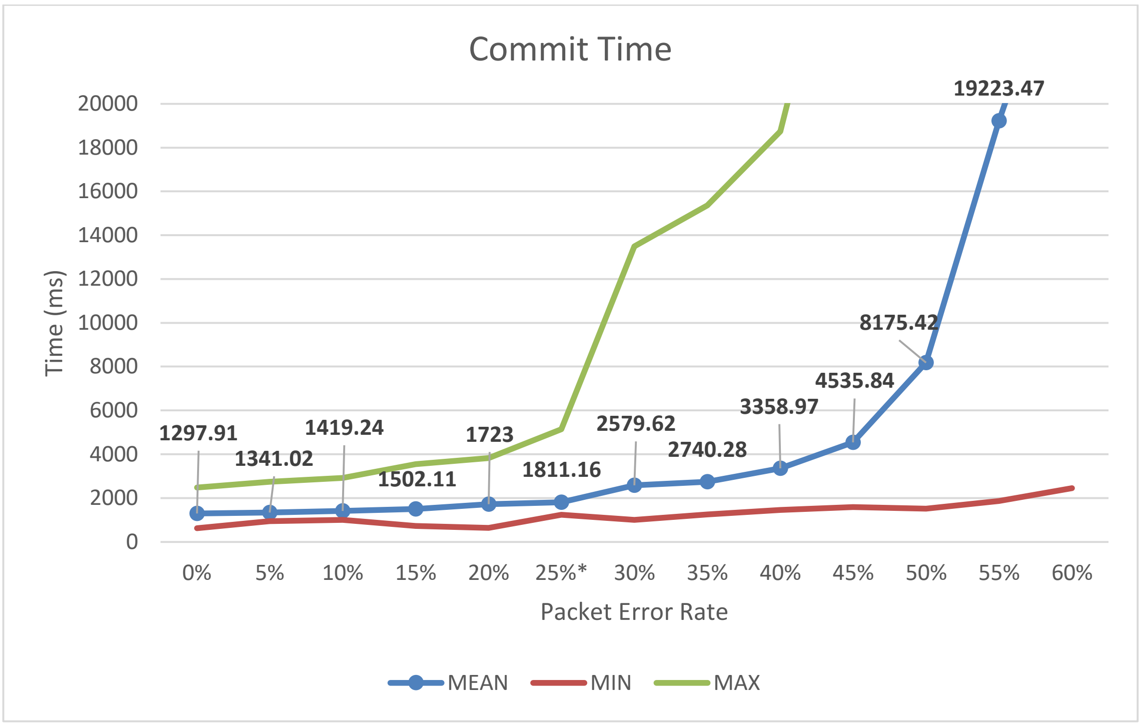


Figure 5.10 Commit time vs Packet Error Rate

*From this point onwards, the nodes start to loss of sync and some of them need to ask their peers for information to reach the current round of the blockchain.

In **Figure 4.10**, we can see that the blockchain can bear up to 25% of PER without worsening the blockchain performance. Once we go onwards, the blockchain starts to have really high maximum times, which may greatly affect if they happen one after another because they can last around fourteen seconds or more. This blockchain will not work properly once we exceed 40% of PER.

5.5.4. Exonum vs Neo

It is time to compare them both, Neo and Exonum, and see which one can behave better depending on the scenario. We must remember that by design, Exonum runs quickly, but we will put them in the same scenario where they run under the same conditions. Therefore, we will check test by test and see which blockchain performs best.

5.5.4.1. Fastest time to commit

If we put one against the other on a graph, we have:

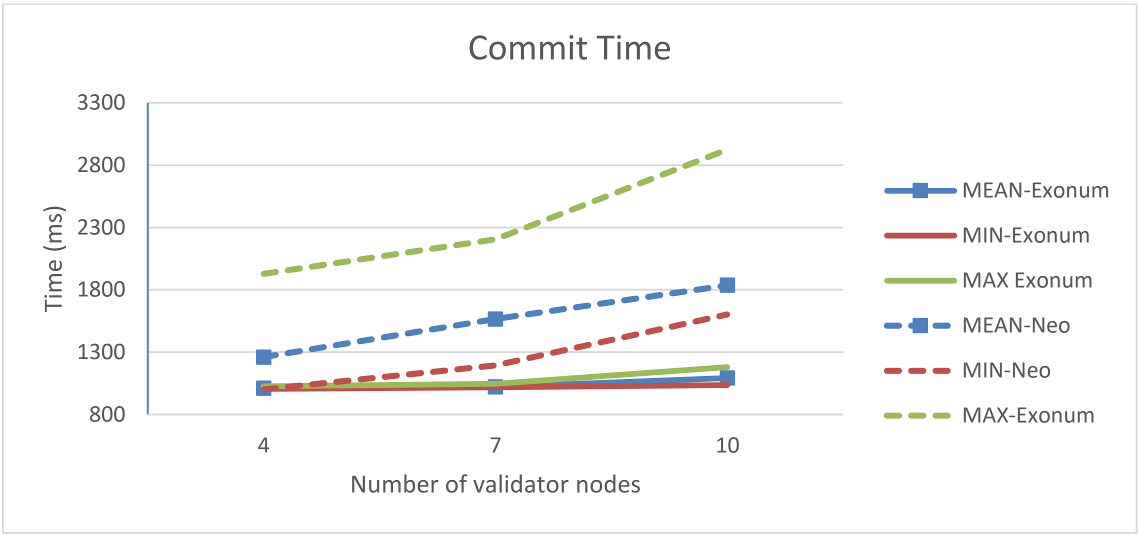


Figure 5.11 Commit time vs number of validator nodes

In **Figure 5.11** we can easily observe that the Exonum blockchain works faster than the Neo Blockchain does in all the different set of nodes. This is probably because Exonum is built in Rust language that is faster than other languages like .Net framework. Therefore, if we speak in terms of transactions per second, Exonum will probably always accept more blocks to commit than Neo.

5.5.4.2. Time to commit vs malicious nodes

If we put one against the other on a graph, we have:

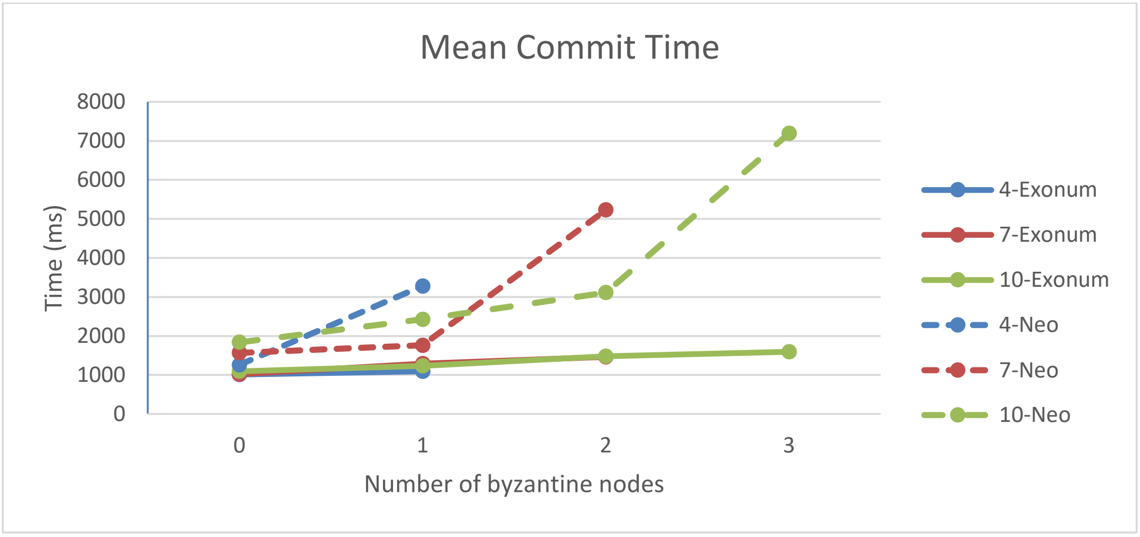


Figure 5.12 Mean commit time vs number of byzantine nodes

In **Figure 5.12** we can see that only when everything works properly, which means that all the nodes are honest and contribute to the consensus algorithm, is the only case where both blockchains almost perform in the same way with a very similar performance.

If we consider a larger set validators and some dishonest validator nodes, it is by far better the Exonum blockchain, as Neo mean commit time increases too much.

5.5.4.3. Time to commit vs Packet Error Rate

If we compare both blockchains:

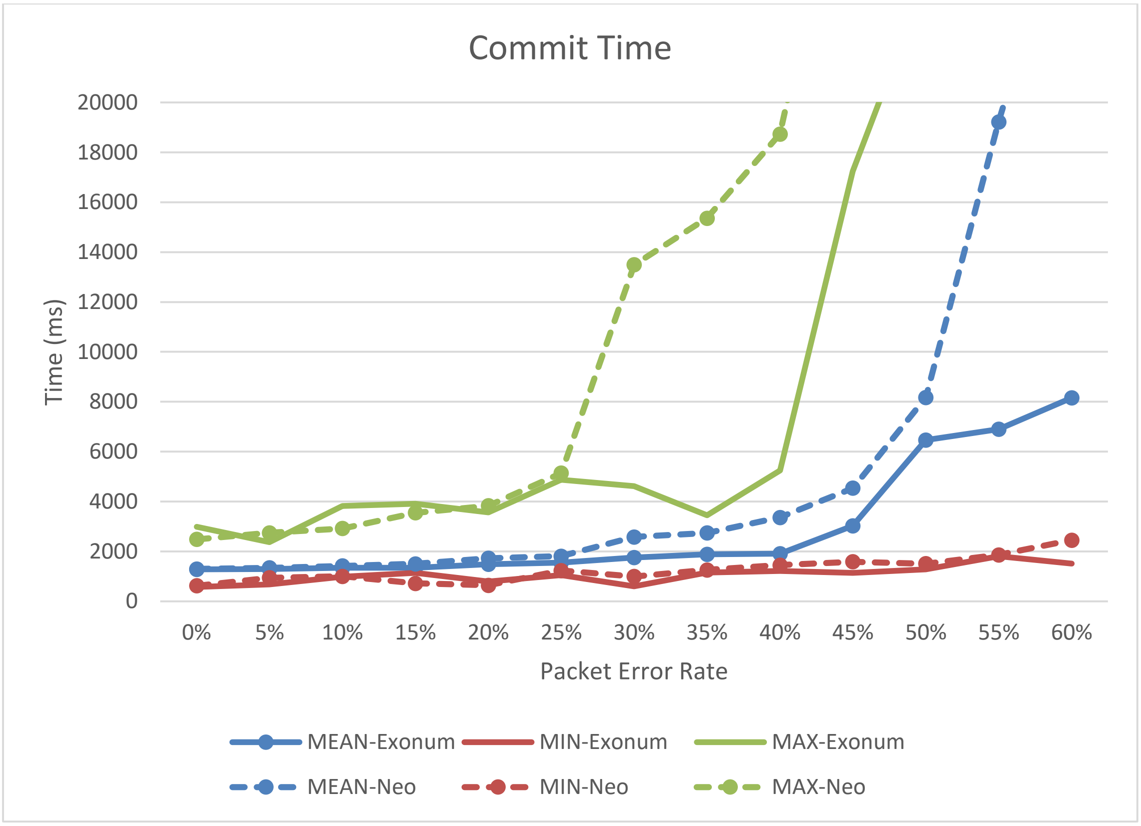


Figure 5.13 Commit time vs Packet Error Rate

In this test, which is shown in **Figure 5.13,** we can see that both blockchains work quite well using networks with less than 25% of PER. Up to 25% PER, both blockchains are able to commit with the same mean time and both have similar behaviours in the best and worst cases. Once the network begins to have a high PER, above 25%, the Exonum blockchain is able to commit faster the blockchain. This is probably due to the Exonum Request Algorithm [31], which efficiently determines which node on the network has the information regarding to consensus. Others use the gossip algorithm, which has much more time to be compute. Using the Request Algorithm, any node that has been left behind, because it has not received the previous messages, such as prevote or even that others have committed a block, can catch up quickly and efficiently.

5.5.5. Conclusions

As conclusions, it can be said that it is much better to use the Exonum blockchain compared to the Neo if speed is the main priority, since it allows to commit many more transactions per second. As we have seen, it is also much better if the network has a high rate of packet loss or if we need to incorporate a greater number of validation nodes. In this case, there is no reason to choose Neo, as they both require the same size of RAM and the same amount of available storage.

On the other hand, if we want to incorporate greater functionality in our blockchain, such as being able to inspect it or depending on a system that has already been implemented and using it, Neo could be a better solution. Neo came out a few years before Exonum, and it has more external developer groups programming new features and functionalities. However, Exonum is in an initial phase, where people who belong to the company that owns it make practically all changes and modifications.

In the end, to use one technology or another, it is convenient to value more aspects than just its speed. NET programmers do not have to worry about things like buffer overflows so it is much safer for writing something like a blockchain node, where remote exploitation by hackers is a constant concern. C++ is much faster and has much lower RAM requirements, but you must be very careful to manage your memory allocations to avoid introducing vulnerabilities.

5.6. <u>Modifying the actual Exonum consensus algorithm</u>

One of the initial goals of this thesis was to make some improvements to the way the blockchain does reach consensus. It is known by far that the consensus algorithm is the core of a blockchain, because without it the concept of distributed ledger makes no sense. If any node of the blockchain could accept different blocks, each copy of the ledger would look different depending on the node owning the copy.

It is proven that the PBFT algorithm, the one on which Exonum is based, works perfectly under the presence of byzantine peers. Nevertheless, what happens if there is no possibility of reaching consensus? What can the blockchain do if there are peers disconnected, with high transmission delays, or not behaving honestly? It seems that having the possibility of changing the node validators, as they take part in the consensus algorithm, could be an important feature to have. So, let us look at the internal behaviour of the Exonum blockchain and see if changing the validator nodes is a feasible solution.

The question is, how to modify the number of validators that are participating in the consensus algorithm when there is no consensus? We have to take into account that:

- The blockchain should wait a certain amount of time before triggering the validator change process. It might be that messages are not being received on time because nodes are experimenting high delays, due to network overload (we must bear in mind that the nodes may be spread throughout the world). Therefore, the network should wait for a minimum time interval before changing the validator.
- The set of possible validators should be agreed beforehand. We cannot set as new validator an auditor node, which has not the rights to participate in the consensus algorithm. Then, we can have a total number of validators and in each consensus process a set of this validators are chosen to participate iin it.

After reading carefully the code and verifying if these changes were feasible to made, some questions raised: are we were really considering all the constraints we should be aware of? Is it really possible to do it without violating the blockchain properties? After further study, the following answers were found:

- Exonum, and mostly all PBFT-based blockchains, assume that there is a partially synchronous network model. In this model, it is non-trivial to tell whether some nodes are slow, have shut down or behaving wrongly. It has been proved that using a synchronous model in permissioned blockchains is less accurate, due to if some node behaves dishonestly. Then there is no easy way to set an interval where events occur.
- The one and only moment in which the validator set can be changed is when transitioning to a new blockchain height, because this is the only state in which the consensus configuration can change, since by definition the configuration is part of the blockchain state. The state is how the blockchain is currently being (height, number of validators, set of validators ...). Therefore, this state must always be the same at the height that we are, which means that it cannot be changed in any round within the same height. This is done, because it automatically guarantees that the new validator set is agreed among all the nodes.
- Rounds in the Exonum consensus algorithm do not end until a new consensus height is reached. This means that it is impossible to simply change the validator set when it reaches a round X (where X is a positive integer). There may be undelivered messages on the previous rounds, which the node still needs to process. One of these messages could lead to consensus and to reach a new height, and it could be rejected because the node it is not a validator any more. This is fatal for the network liveness since rejecting valid blocks should never happen.
- The validator set cannot change in a pseudo-random way due to it is encoded inside a supervisor service [32], which is in charge for ensuring that nothing changes during a new block proposal without its approval. Moreover, in order to get the approval, supervisor actions must be authorized by the supermajority of 2/3 of the validator nodes.
- From the coding perspective, implementing some logic change is certainly doable, but it needs to take careful consideration because it can take risk to break basis consensus properties like safety and/or liveness, like previously mentioned. There is the need to get away the assumption that the validator set remains static throughout a consensus epoch, which is used in multiple places in the code.
- There also some other limitations out of the scope of the consensus algorithm. There is the possibility of deploying smart contracts, which have access to the validator set. Therefore, if the set of validators has change, in order for those contracts to be executed, the code must rerun all previous blocks proposals and modify that information.

Due to a lot of restrictions and issues and, to put it shorter: the Exonum consensus algorithm does not handle the scenario in which the validator set would change in a round X + 1 (where X is a positive integer); it does not need to and the developers have no plan to support such a scenario in the future.

6. Conclusions and future development

Blockchain is an emerging technology that, besides a monetary system without any central entity, offers a wide range of applications and it is being adopted as a secure solution by many companies. In this work we have focused on two permissioned blockchains: Exonum and Neo, in which transactions added to the blockchain are approved by consensus among its members and by using a PBFT-based algorithm.

After having analysed these blockchains, it can be concluded that this technology will come stomping on companies and that many of them will benefit from it, especially because of its transparency and decentralized features. In addition, because Exonum, does not depend on a specific cryptocurrency, fees associated to transactions do not depend on the fluctuations of the cryptocurrency, which constitutes an enormous advantage with respect to other technologies.

As mentioned above, we have tested two different blockchains based on PBFT: Exonum and Neo. Although Exonum is faster in terms of transactions per second than Neo, the latter has more extra functionalities already developed. Both have been found to be capable of withstanding Byzantine failure, which was expected due to their consensus algorithm. Furthermore, both can work with unstable networks. When these networks have low PER, which do not reach 30%, they both work correctly and the time to commit is not increased too much. When this rate is raised, from 30%, Exonum is able to work much better in this scenario. Neo instead, although you can continue to commit blocks, these are done in very long times which causes the performance of the blockchain to drop significantly. Finally, in terms of number of nodes, when the nodes run on devices with low computational capacity, that is, with low RAM around 2GB, the Exonum network due to the Rust language is capable of running much more lightly. This makes it possible to have much more validator nodes in the network. In practice, companies do not usually have this restriction and can work with several devices with much more RAM.

Finally, this work could be continued in several ways:

- To run a blockchain with more nodes, either by interconnecting several or by using Linux containers or a similar approach, so that more than one node can be running in the same computer without requiring so many resources as VMs.

- Test the environment with more transactions per block. Due to the previous point, this has been very difficult to prove because the results were not very reliable and varied with each attempt. If this is possible, it can be tested whether in a high-loss scenario (high PER), where the leader's proposal contains many transactions, see what happens on the other nodes. For example, the other node may not have all the transactions, and send a message asking you to send them to them, since if they do not have the transaction, they will not consider the proposal valid. This message and many others can be lost on the network due to high PER rates and can lead to scenarios where the blockchain really does not behave, as it should. Although many proposals are valid, they could be rejected and move to a next round.

- To use another tool to be able to add delay between nodes. The tc tool has had a great limitation in terms of the number of links to be able to simulate.

Bibliography

[1] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. [Online] Available: https://bitcoin.org/bitcoin.pdf

[2] Ethereum. Home | Ethereum.org. Ethereum.Org. https://ethereum.org/en/

[3] Wikipedia contributors. (2020b, June 30). Merkle tree. [Online] Available: Wikipedia. https://en.wikipedia.org/wiki/Merkle_tree

[4] Wikipedia contributors. (2020, July 3). Blockchain. [Online] Available: Wikipedia. https://en.wikipedia.org/wiki/Blockchain#Blocks

[5] Blockchain Industry Applications - IBM Blockchain. [Online] Available: https://www.ibm.com/blockchain/industries

[6] Fernandez, V. (2016, November 14). Blockchain y Ciberseguridad I. Security Art Work. [Online] Available: https://www.securityartwork.es/2016/11/14/blockchain-ciberseguridad-i/

[7] Konstantopoulos, G. (2020, February 6). Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance. Medium. [Online] Available: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419

[8] Zainuddin, A. (2018, December 1). Consensus Mechanism Makes Blockchain Revolutionary. Here is how. Medium. [Online] Available: https://medium.com/@azizzainuddin91/this-guide-to-consensus-algorithm-takes-a-look-at-a-common-question-on-what-is-consensus-mechanism-561fffe3f385

[9] Litecoin - La moneda electrónica. Litecoin. [Online] Available: https://litecoin.org/es/

[10] Exonum: Build trust into business with blockchain technology. (2018, June). [Online] Available: https://exonum.com/index

[11] Rust Programming Language. [Online] Available: https://www.rust-lang.org/

[12] Exonum Documentation. (2018). [Online] Available: https://exonum.com/doc/version/latest/

[13] Bitfury. (2018). Exonum: Byzantine fault tolerant protocol for blockchains. Appendix A page 31 [Online] Available: from https://bitfury.com/content/downloads/wp_consensus_181227.pdf

[14] Exonum Data Model. (2020). [Online] Available: https://exonum.com/doc/version/0.12/architecture/storage

[15] Wikipedia contributors. (2020, March 8). Gossip protocol. Wikipedia. [Online] Available: https://en.wikipedia.org/wiki/Gossip_protocol

[16] Neo Smart Economy. [Online] Available: https://neo.org

[17] What is Digital Identity and how can you protect yours? Digital Identity. [Online] Available: https://blog.signaturit.com/en/what-is-digital-identity

[18] Neo Documentation. [Online] Available: https://docs.neo.org/docs/en-us/index.html

[19] Neo Documentation Voting Election. Neo.Org. [Online] Available: https://docs.neo.org/docs/en-us/tooldev/consensus/vote_validator.html#voting

[20] Stack Overflow Developer Survey 2019. (2020). Stack Overflow. [Online] Available: https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted+

[21] Neo Documentation. Neo.Org. [Online] Available: https://docs.neo.org/docs/en-us/index.html

[22] (2012, November 16). Compiling to MSIL. Microsoft Docs. [Online] Available: https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/c5tkafs1

[23] Sutter, P. (2016, January 18). QoS in Linux with TC and Filters. Linux.Com. [Online] Available: https://www.linux.com/training-tutorials/qos-linux-tc-and-filters/

[24] T. (2019, September 18). What is Genesis Block and why Genesis Block is needed? Medium. [Online] Available: https://medium.com/@tecracoin/what-is-genesis-block-and-why-genesis-block-is-needed-1b37d4b75e43A

[25] Exonum Network Structure. Exonum. [Online] Available: https://exonum.com/doc/version/latest/advanced/network/#network-structure

[26] Rd.Io, M. C. Network Testing Solutions. Wonder Network. [Online] Available: https://wondernetwork.com/

[27] Wonder Network. Global Ping Statistics. [Online] Available: https://wondernetwork.com/pings

[28] neo-project/neo. GitHub. [Online] Available: https://github.com/neo-project/neo/blob/master-2.x/neo/Consensus/ConsensusService.cs

[29] Consensus Nodes - Neo Smart Economy. Neo.Org. [Online] Available: https://neo.org/consensus

[30] Neo Documentation Voting Election. Neo.Org. [Online] Available: https://docs.neo.org/docs/en-us/tooldev/consensus/vote_validator.html#voting

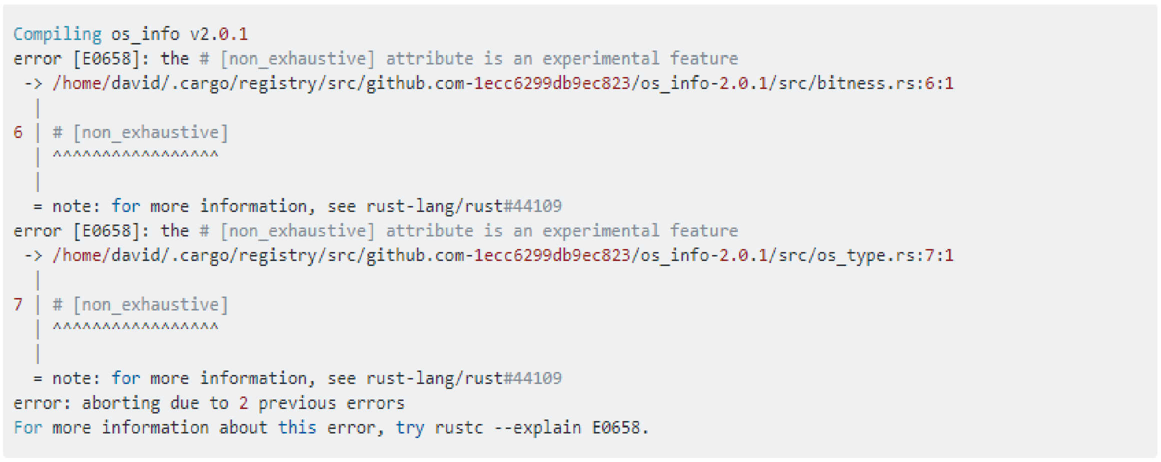
[31] Exonum Request Algorithm. [Online] Available: https://exonum.com/doc/version/latest/architecture/consensus/#requests-algorithm

[32] Exonum Supervisor Service. [Online] Available: https://exonum.com/doc/version/latest/advanced/supervisor/

Appendices

1. Exonum Installation

Following the actual guide of installing Exonum, the following error is obtained:



```
Compiling os_info v2.0.1
error [E0658]: the # [non_exhaustive] attribute is an experimental feature
 -> /home/david/.cargo/registry/src/github.com-1ecc6299db9ec823/os_info-2.0.1/src/bitness.rs:6:1
  |
6 | # [non_exhaustive]
  | ^^^^^^^^^^^^^^^^^
  |
  = note: for more information, see rust-lang/rust#44109
error [E0658]: the # [non_exhaustive] attribute is an experimental feature
 -> /home/david/.cargo/registry/src/github.com-1ecc6299db9ec823/os_info-2.0.1/src/os_type.rs:7:1
  |
7 | # [non_exhaustive]
  | ^^^^^^^^^^^^^^^^^
  |
  = note: for more information, see rust-lang/rust#44109
error: aborting due to 2 previous errors
For more information about this error, try rustc --explain E0658.
```

Figure Appendices 1.1 Installation Exonum Error

The error that the cmd finally threw was the following: https://doc.rust-lang.org/error-index.html#E0658. I read the explanation, it is understood that you are building the project using the stable version of $rustc$ and you should really use the latest version available even if it is not stable (that is why $\#[non\_exaustive]$). In order to do this, a rust tool called $nightly\ mode$ is used. For this, this option must be included previously using $rustup\ default\ nightly$. The bug is that $rustup$ is a standalone module and does not come installed when you install $rustc$ (It is a bug that has been around since 2016 https://github.com/rust-lang/rustup/issues/686). To do this you reinstall it by activating the flag --y that forces you to install all the packages it finds derived. Once installed you can use the nightly option and correctly install Exonum.

Issues opened by me can be found at:

https://github.com/exonum/exonum/issues/1808

https://github.com/exonum/exonum/issues/1810

Glossary

BFT:	Byzantine Fault Tolerance
dBFT:	Delegated Byzantine Fault Tolerance
P2P:	Peer-to-peer
PBFT:	Practical Byzantine Fault Tolerance
PER:	Packet Error Rate
PoL:	Proof-of-Lock
PoS:	Proof of Stake
PoW:	Proof of Work
tps:	transactions per second
VM:	Virtual Machine