MASTER OF SCIENCE
IN ENGINEERING

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation: Information and Communication Technologies (ICT)

IMPROVING COSTS, SAFETY AND LIVENESS IN BLOCKCHAIN SYSTEMS

Author:
Nicolas Huguenin

Under the direction of:
Prof. Dr. Marcelo Pasin
HE-Arc

External expert:
Dr. Hugues Mercier
Institutes of Computer Science and Mathematics - UniNE

Neuchatel, HES-SO//Master, March 8, 2019

Information about this report

Contact information

Author: Nicolas Huguenin

MSE Student

HES-SO//Master

Switzerland

Email: *nicolas.huguenin@master.hes-so.ch*

Declaration of honor

I, undersigned, Nicolas Huguenin, hereby declare that the work submitted is the result of a personal work. I certify that I have not resorted to plagiarism or other forms of fraud. All sources of information used and the author quotes were clearly mentioned.

Place, date: _____

Signature: _____

Validation

Accepted by the HES-SO//Master (Switzerland, Lausanne) on a proposal from:

Prof. Dr. Marcelo Pasin, project advisor

Place, date: _____

Prof. Dr. Marcelo Pasin	Prof. Dr. Philippe Passeraub
Advisor	Dean, HES-SO//Master

Acknowledgments

I want to thank the whole team at TrueLevel, especially Robert Hambrock and Daniel Lebrecht, for their support, helpful ideas and inputs as well as their proofreading.
Acknowledgement to Prof. Marcelo Pasin for his feedback on the work and his support during all the stages of the thesis.
Special thanks to Karim Luy for his throrough proofreading and help.

Abstract

The Ethereum blockchain is meant to move from a Proof-of-Work consensus protocol to a Proof-of-Stake one. Correct-by-Construction Casper delivers an abstract family of mathematical Proof-of-Stake consensus protocols but does not provide block production strategies for nodes which this project aims to cater to and compare them. A testing framework based on an existing Casper library has been implemented in order to simulate blockchains ruled by the proposed basic strategies and latency, node count and overhead were measured during the simulations. The implemented framework has been validated using the basic strategies and permits the implementation of more complex ones through a generic code architecture. The proposed model offers a good way to compare latency and overhead for strategies but is not efficient to evaluate the way a strategy scales with the number of nodes in the network.

Keywords: Algorithms; Blockchain; Distributed Systems; Sharing Economy

Contents

1 Introduction

1.1 Context

The Ethereum blockchain aims to move from the current Proof of Work (PoW) consensus protocol to a Proof of Stake (PoS). Multiple teams are currently researching ways to describe and implement such a protocol. One of these teams, lead by Vlad Zamfir, is working on a protocol family called Correct by Construction (CBC) Casper. CBC Casper describes an abstract set of protocols that can achieve consensus between nodes on any kind of value, for example an integer, a vote, or a blockchain. This project aims to find and compare block publishing strategies for a CBC Casper blockchain consensus.

1.2 Objectives

As the CBC Casper paper only describes an abstract way of constructing PoS consensus protocols and does not make any assumptions on synchrony, one of the main challenges of the actual implementation is to find incentive mechanisms and strategies telling the nodes when to produce blocks. The main goals of this project are:

- to propose multiple block producing strategies;
- to create a model that allows one to easily compare said strategies;
- to discuss the advantages and disadvantages of each strategy.

2 Background

2.1 PoW and PoS

In Ethereum, PoS is aiming to replace PoW as a distributed consensus algorithm. This section succinctly describes both methods, their main differences, and which problems arise when you replace PoW with PoS.

2.1.1 What is PoW?

PoW is the current consensus protocol used to decide between blockchains in Ethereum. In order to create a new valid block, a node has to solve a cryptographic puzzle and include its solution in the newly created block. The difficulty of the puzzle is parametrized in order to output a block on an average time interval and a reward is given to the creator of each block. The consensus rule states that the chain with the greatest total difficulty is to be considered the main one which incentives miners to build on the main chain if they want to get rewards for their work. The fact that the difficulty changes to keep a certain interval between blocks means that work is a proxy for timing; a miner cannot create an arbitrarily large number of blocks in a short time because it's inherent to the protocol.

2.1.2 What is PoS?

PoS selects a new block creator according to their weight -or stake- which is determined through the node's age, wealth, etc. In this report, we will mainly discuss a specific PoS protocol, CBC Casper.

2.1.3 CBC Casper

CBC Casper [1] [2] is an abstract consensus protocol family which is PoS-ready. Miners, called validators in this context, send messages to each other, acknowledging they saw other messages by including them in a *justification*, that is attached to each message. Based on its justification as well as a weighted list of validators, each message defines an *estimate*, which is the consensus value proposed by the sender of the message. In the case of a blockchain, each message is a block, which points to a single older message as their estimate and form a *block-Directed Acyclic Graph (DAG)*. Running a slightly modified version of the Greedy Heaviest Observed Sub-Tree (GHOST) algorithm on the DAG [1] [3] returns a blockchain.

The picture on the left of Fig. 2.1 demonstrates a small example of a CBC execution over a blockchain with four nodes, messages sent by validators as colored circles, justifications as dotted arrows, and the selected blockchain with blue arrows. The blockchain is obtained by running the GHOST algorithm on the state of the network as shown in Fig. 2.1 and the result of this algorithm would be the estimate of a new message sent by an honest validator within this network state.

The picture on the right of Fig. 2.1 exemplifies validator v_1 producing a new message and the resulting new blockchain. An honest node includes all the latest messages it has received (including its own last message). When a validator does not include its own messages in its justifications, it is considered as an *equivocator* which does not follow the protocol, and may be punished by the network for such a behavior. Note that validator v_1 does not equivocate because its first message is in

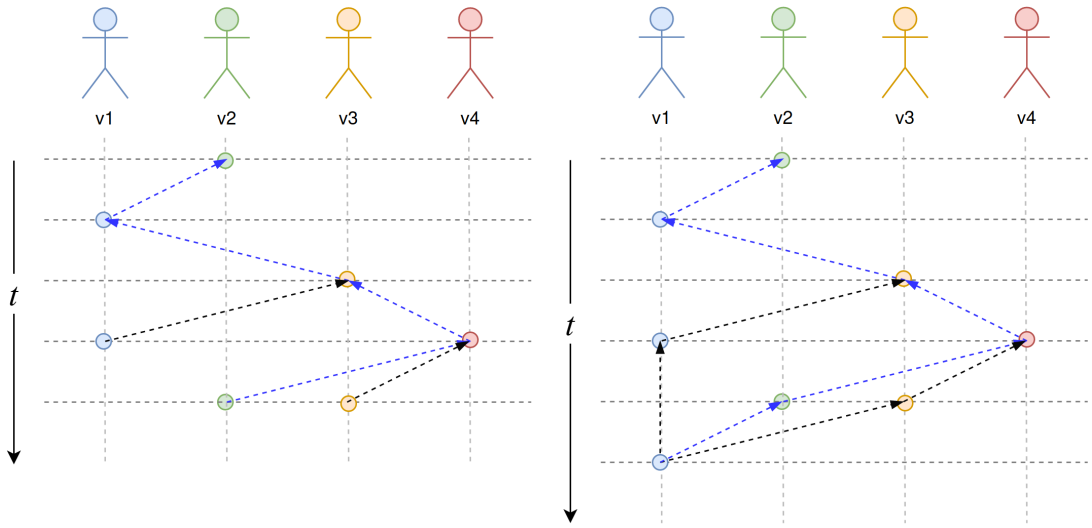

Figure 2.1 *CBC example on a blockchain consensus.Four nodes are represented horizontally. Messages are colored circles, justifications are dotted lines and the selected blockchain is pictured as blue arrows. Time is represented as the vertical axis.On the left, an initial state of the network is pictured. On the right, validator v_1 publishes a new message to the network and includes the latest messages it has seen in its justification. The estimate of the newly published message is the output of the GHOST algorithm run on said message and its justification.*

the justification of v_2's first message, and v_1 has this message in the justification of its second message. This is legal because v_1's first message is in the dependency of its later messages.

Finality

Finality is a key concept to address for this project whereby, currently a block is considered finalized in a network state if it cannot be removed by receiving new messages from honest nodes. For CBC Casper, a block is considered finalized when a safety oracle is detected. There are multiple parameters and ways to compute safety oracles and the one that has been used for this project is that a block is considered safe when every validator in a subset of the network has seen each other as a member of the same subset agreeing on said block. Two validators agree on a block when they both include it in the blockchain of their estimate. In this project, the subset will be any 50% of the network. This implies that at least two rounds of messages are needed in order to finalize a block; the first one for nodes to acknowledge they have seen a message by including it in their justification, and the second to acknowledge that they have seen other nodes seeing it.

2.1.4 Differences between CBC Casper and PoW

Block production

In a PoW[4] setting, a miner can broadcast any block to the network, which will only be picked up by the other nodes if the block is valid, but the miner has to work to produce said valid block. This restriction implies that the block miner cannot create blocks at any given time, which is different to the CBC Casper setting that allows nodes to broadcast whenever.

Timing assumption

Time is assumed in a PoW environment by the amount of work a miner has to provide to solve a cryptographic puzzle; the difficulty of this puzzle is determined in such a way that it would require an amount of time to solve that averages over the network. On the other hand, the CBC Casper protocol family does not make any timing assumption.

Spam

The two previous points imply that spamming issues are mitigated by the need to work to produce blocks in a PoW chain. However, as there is a negligible computational cost, a node could easily spam the Casper network.

Economic majority

As machines that are better at solving the cryptographic puzzle in a PoW context are more expensive to buy and to operate, therefore the work is a proxy for the economic majority. If a miner can do more work than another, it has more computational power and therefore more at stake. Economic majority is however built in the CBC Casper protocol, in the form of the stake that validator have in the consensus mechanism.

Building strategy

In order to receive a reward for its work, a miner wants a block it has produced to be included in the main chain. A PoW protocol usually states that the chain with the most total difficulty is the main one. Therefore, a miner is incentivised to build on top of the main chain. In a Casper setting, as there is no such incentive, a validator could build on any chain, or even on its own chain.

	PoW	CBC Casper
Block production	Miners can publish blocks if the can prove they worked for it	Nodes can publish blocks at any time
Timing assumption	Work is a proxy for timing	None
Spam	Work removes the possibility to spam	Negligible computational costs to produce blocks imply potential spam
Economic majority	Work is a proxy for economic majority	Economic majority
Building strategy	Nodes are incentivised to build on the longest chain because they have to work to build blocks	No clear incentive to build on the longest chain

Table 2.1 Summary of key differences between PoW and PoS

2.2 Main problematic

As seen in Subsec. 2.1.4, many real-life implementation points are still to define in order to have a practical CBC Casper blockchain. One main problem that arises is the selection of a block building strategy. This project will try to propose a model to compare strategies, as well as a framework that allows one to easily implement new strategies and discuss their performances based on the proposed model. Basic strategies that have predictable outputs will be implemented as well, in order to validate the functionality of the framework.

2.3 `core_cbc`

`core_cbc` is a Rust implementation of the CBC Casper, made by TrueLevel, the company with which this project takes place. It implements the consensus algorithms proposed in the CBC Casper paper and offers an abstract structure that can be used to create consensus on any value. This project uses `core_cbc` as a base layer to implement tests and run simulations.

2.4 Parity Ethereum

Parity is a Rust Ethereum client. It includes a *Pluggable Consensus* module that allows one to easily add new consensus protocols by implementing an interface. At first, the goal of this project was to implement a small bridge between the Parity module and the `core_cbc` implementation to study block creation strategies in a realistic testbed. The implementation of the bridge was not as straight forward as planned so it has been decided to cut it out and test strategies without mimicking the network and client settings.

2.4.1 Background work

During the early stages of this project, a clear objective was set: to be able to run a Casper *testnet* with Parity custom nodes. Some work has been done for the implementation of the `core-cbc` library into Parity, but that was left to people that had a better understanding of the underlying library. Furthermore, considerable effort had been injected in the creation of a Docker infrastructure in order to easily deploy, connect and monitor multiple custom Parity nodes on a single machine in order to experiment with different message building strategies. Using Docker simplified access to the UniNE machine clusters for testing purposes as software on their clusters are heavily containerized. After seeing that the Parity implementation would take too much time to be completed, and therefore might be unusable for this thesis, it has been decided to evaluate strategies inside the `core-cbc` library instead of the more real-life-like setting that is a Parity testnet. The core library was still being implemented at the time and further testing was needed. The main disadvantage of doing the experimentation in the library is that the whole network latencies and topology are not taken into account. However, a non-negligible advantage of implementing strategies in the core library is that it will be easier to test them on consensus values that are not only blockchains.

2.4.2 Local testnet

Scripts that create and manage two local nodes have been written. They launch 2 Parity instances with the CBC Casper consensus engine, and connect them to each other. Each node has a user account to send transactions, as well as a sender account, that can act as a validator in a Casper sense. Currently, each node produces a block every 10 seconds. This is a basic strategy that enabled further testing of the Parity inclusion of the `core-cbc`.

2.4.3 Docker testnet

Testing at a larger scale than two local instances was needed. The possibility to access a cluster at the UniNE was discussed and the more straightforward way to run programs on the cluster is to have containers. It was therefore decided to create a more complex infrastructure using Docker containers. `docker-compose` scripts were created to achieve this goal. An arbitrary number of containers can be created at once and inter-connected in two different ways:

- fully connected;
- ring.

The created nodes have the same types of accounts as for the local testnet. In the fully connected setting each node is connected to every other node. In the ring case, each node is connected to two other nodes in a circular manner. Those were the two first layouts that were implemented for simplicity. It was thought to add new layouts afterwards in order to match more precisely the real network topology of the Ethereum *mainnet*. However, because the Parity implementation was deemed too time consuming to be used before the end of this project, no further efforts have been put in that direction. Nevertheless, the software architecture is in such a state that adding and removing topologies is easily done through abstract structures.

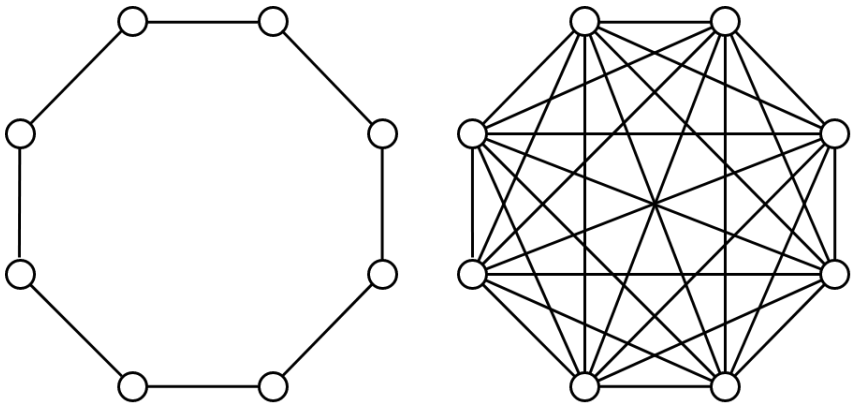

Figure 2.2 *Parity nodes topologies implemented in the Docker testnet. Ring layout (left), fully connected (right)*

3 Testing Framework Implementation

This chapter describes the modelling of the problem, the block building strategies that have been implemented in order to validate the correctness of the proposed model, and the methodology that has been employed to test the framework.

3.1 Modelling

3.1.1 Metrics

A way to rationally compare strategies is needed in order to discuss their relative strengths and weaknesses. The three main characteristics which will be used are:

- latency;
- number of nodes;
- overhead.

Latency

The latency is the number of messages required to finalize a block and is a measure of a blockchain's liveness. Latency should be as close to 2 as possible so as to reach finality as soon as possible. The closer the latency is to the ideal value, the more live the system is as transactions take less time to be confirmed. The minimal value for latency is 2 because it is the minimal number of steps to finalize a block, as seen in Subsec. 2.1.3:Finality.

Number of nodes

The number of nodes in the validator set; should be as high as possible to guarantee decentralization and safety.

Overhead

The overhead is the number of messages that are sent over the network between one step of the consensus and the next. It should be as low as possible to keep the costs in bandwidth low, the minimum being 1 as at least one message is needed to move consensus forward.

Example

Fig. 3.1 portrays how the metrics are computed: the left half of the Figure shows the initial states of the system at time t_1 where message m_1 is finalized. The other half of the Figure is the same system at time t_2 when m_2 is finalized. The latency of m_2 is l_2 -represented by the difference in heights between m_3 and m_2- which is worth 4 in this case. The overhead is measured by counting the number of messages that have been sent between t_1 and t_2, worth 3.

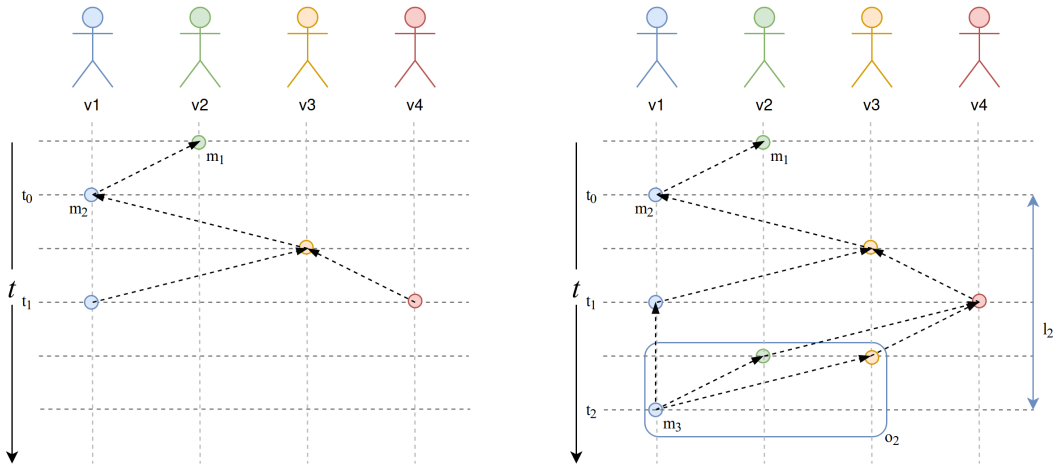

Figure 3.1 *Metrics computation example. At time t_1, m_1 is finalized (left). At time t_2, m_2 is finalized. The latency l_2 for m_2 is the difference of height between t_1 and t_2, worth 4 in this case, and its overhead o_2 is worth 3 and is the number of messages sent between the finalization of the previous message (m_1) and m_2.*

3.1.2 Basic model

The proposed variables can be considered as forming a trade-off triangle or trilemma. As CBC Casper does not make assumptions on timing, sources, contents, destinations of the messages that are exchanged, the whole trade-off space can be explored by varying any of those parameters.

The first strategy evaluation model which represents such a trilemma that was chosen is:

$$1 = s_n \cdot \frac{1}{n} + s_l \cdot l + s_o \cdot o$$

This model binds each 3 variables: n (number of nodes), l (latency) and o (overhead) to their score $s_{variable}$.

The higher the score s_x is, the more its related variable is dominant in the strategy and therefore the closer to a corner of the triangle the strategy is. $\frac{1}{n}$ is used because a large number of nodes is wanted. The basic model has linear dependencies between scores and variables and is probably highly inaccurate but serves as a decent starting point for analysing the simulation dataset.

3.1.3 Model Evaluation

After running the strategies in the simulation environment, metrics for n, l and o have been recorded. Then, for multiple runs, a linear regression has been performed in order to find the scores s_x.

3.2 Strategies

The following strategies were proposed in order to visit known parts of the trade-off triangle:

- round-robin;
- double round-robin;

- triple round-robin;
- maximal overhead;
- arbitrary.

These strategies should allow one to visit the whole triangle and to discuss their respective strength and weaknesses. The following sections describe the strategies as well as their expected locations in the triangle.

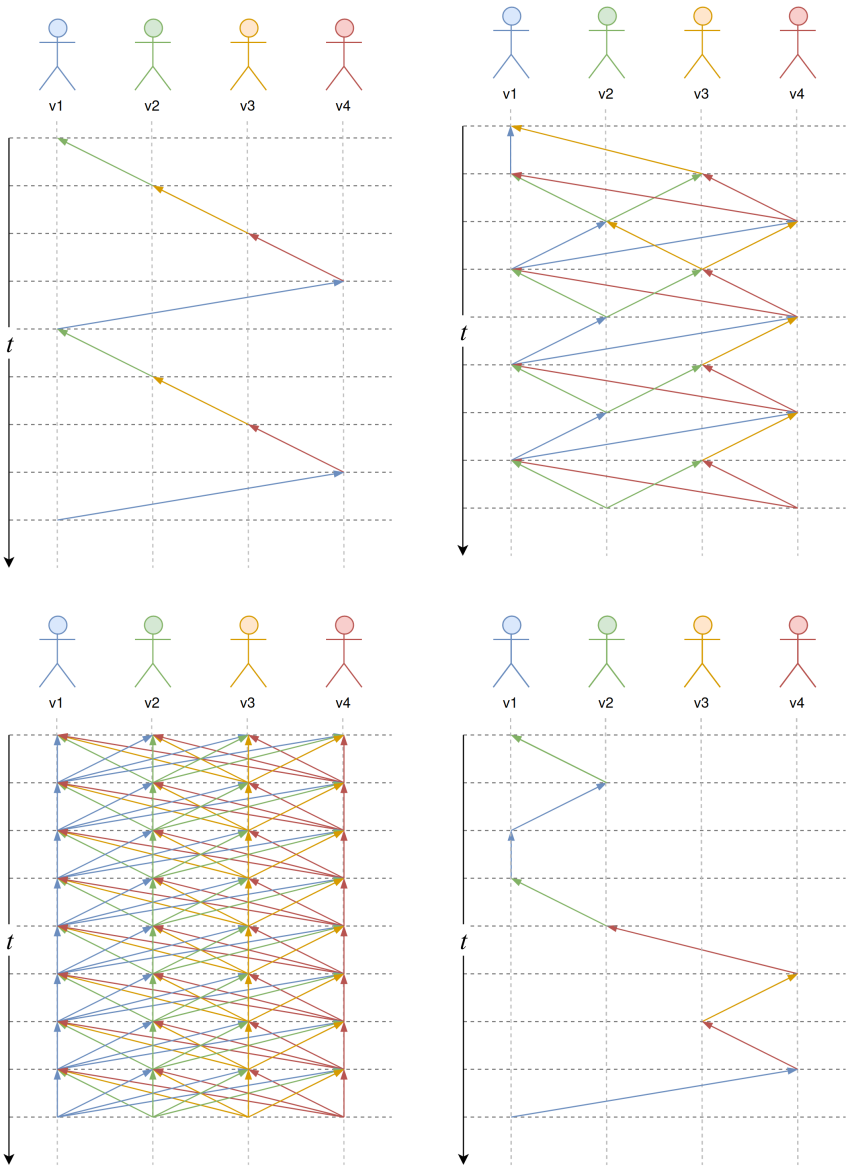

Figure 3.2 *Strategies examples with 4 validators. Top-left: Round-Robin. Top-right: Double Round-Robin. Bottom-left: Maximal Overhead. Bottom-right: Arbitrary. Arrows here show justifications for all messages.*

3.2.1 Round-Robin

Nodes send messages one after the other in a fixed order (Fig. 3.2, top-left). The expected overhead for this strategy should be the lowest of all the strategies as only one message is sent at a time and each sent message should finalize a new block. The latency, on the other hand, should be quite high because as seen in Subsec. 2.1.3 50% of the validators have to acknowledge the messages to finalize them. Since the Round-Robin has a fixed rotation order for the message production, finalization happens only a rotation and a half later, where the first rotation acknowledges the message's existence and the subsequent half rotation confirms acknowledgment.

3.2.2 Double Round-Robin

In this setting, two nodes send messages at the same time, in a fixed order (Fig. 3.2, top-right). If the two nodes that send messages at the same step are as far as possible from each other in the set of validators, the latency to finality is supposedly half as much as the simple Round-Robin strategy, as the whole validator set sends a message in half the number of steps. The overhead is however doubled because two messages are sent at each step.

3.2.3 Triple Round-Robin

This strategy is similar to the Double Round-Robin, except that three nodes send a message at each step. It was decided to add this strategy at a later stage of the project in order to have intermediate comparison points between the Double Round-Robin and the Maximal Overhead strategy. As for the Double Round-Robin strategy, the latency should be divided by three compared to the simple Round-Robin and the overhead should be three times higher. Note that a N-parallelized round-robin could be implemented in order to change the position in the trade-off triangle but it would lose meaning for a number of nodes smaller than N so there is no quadruple (or bigger) round-robin implementation.

3.2.4 Maximal Overhead

This strategy is the most expensive in terms of bandwidth; at each step, each validator sends a message to all others(Fig. 3.2, bottom-left). This example strategy should give a baseline value for the maximum overhead that is reachable in the tradeoff triangle. It also minimizes the latency as only two steps are needed in order to finalize a block. This strategy is similar to the pBFT[5] protocol because each node sends a message to every other one over two steps.

3.2.5 Arbitrary

The Arbitrary strategy is the simplest to think of: pick a sender at random(Fig. 3.2, bottom-right). Using fixed probability density functions, one node is arbitrarily selected at each step to create a message. This strategy could be modified in the same way the simple Round-Robin has evolved into the Double and Triple Round-Robin strategies to show the evolution of the metrics with the number of messages sent at each step.

3.2.6 Bottom-up strategies

All the previous strategies except Arbitrary require an overarching view ("top-down" view) of the system and the nodes to be synchronized. Bottom-up strategies are more likely to be implemented from a real-life point of view. Each node has its own view of the messages and acts in its own interest based on that view. For example a node could keep track of the whole state of the finalization of messages, and only publish a message itself when it facilitates the finalization. That kind of strategy has not been implemented but the testing and scoring framework allows one to easily add them. Such strategies should implement incentives for the nodes not to spam the network and include mechanisms to slash nodes not following some rules the strategies dictate.

3.3 Methodology

Over the duration of this thesis, the `core-cbc` library included a test framework called `proptest`. The testing framework that has been implemented includes ways to simulate the behavior of the Casper protocol over multiple nodes and thousands of blocks. At the time of writing, the simulations do not include network latencies.

3.3.1 `proptest`

The `proptest` implementation is able to run blockchain simulations off the following parameters:

- Number of validators;
- Terminating condition;
- Sender strategy;
- Receiver strategy.

Number of validators

The number of nodes that can validate blocks.

Terminating condition

The terminating condition is a predicate that tells whether or not the simulation has reached an end. In this case, the end of the simulation is reached when at least one node finds a safety oracle for a blockchain that has a height of 4. This height was chosen because it offered a balance between the computation speed and the variance in output values. If the chosen value was smaller the generated blockchains would all look alike and some side effects would have been observed near the genesis block whereas a bigger value would imply more computation time.

Sender strategy

The sender strategy selects one or more nodes that will create new messages and forward them to the rest of the network. All the basic strategies that have been presented in Sec. 3.2 are implemented as Sender strategies. New strategies (including bottom-up ones) can be easily implemented as well.

Receiver strategy

Receiver strategies select a set of validators that receive messages created by Sender strategies. Three strategies have been implemented for now:

- All receivers;
- Half receivers;
- Some receivers.

The *all receivers* strategy broadcasts messages to each other validator. The *some receivers* strategy sends a message to 1 or more validator, using a uniform probability density function. As of now, none of the implemented strategies are a good modelisation of a typical Ethereum network and this will be fixed at a later iteration. Nonetheless, these strategies offer two extreme points on the spectrum of the network topology: a fully connected one without latency (*all receivers*), and a random one (*some receivers*) and will be both taken into account to compare the sender strategies. The *half receivers* strategy sends a message to half the validator set, chosen at random which has been implemented at a later stage of the project because the *some receivers* strategy ended up modifying multiple parameters at once rendering the analysis of the output too complex for the proposed model and tools.

3.3.2 Metrics measurements

Metrics are not computed as-is in the `proptest` implementation. Simpler data are logged into `csv` files, and are processed by a `Python` script. The recorded data are:

- the number of nodes;
- the identity of the current node;
- the height of the chain;
- the height of the highest finalized block;
- and the total number of messages that have been sent.

This two step processing enabled the generation of data before having a precise definition of the metrics. It also permits to vary the way metrics are recorded. For example, from the same log file, you could extract metrics only when the last step of the consensus is reached, or have an average of the metrics at each step of the consensus.

4 Framework and Strategies Evaluation

This chapter visualizes the data collected from the test runs with the various proposed strategies. The first part of the chapter explains the data obtained and compares them to the expected results for each strategy described in Sec. 3.2. The second part of the chapter shows the scores of the strategies when the obtained data are fitted to the model discussed in Subsec. 3.1.2. The latter part of the chapter discusses suggested improvements to the model for a better strategy juxtaposition.

4.1 Tests

Tests that cover all the combinations between sending strategies and receivers strategies have been run. All of them have the same terminating condition that is reached when at least one node finds a safety oracle at height 4 (in other words, that the block at height 4 in a node's view is finalized). The number of nodes is chosen at random in the interval $[2, 20[$. Performing tests on this interval is large enough to give an insight on the interactions between all the metrics and small enough to reach the terminating condition in a reasonable amount of time (1 run with a number of nodes chosen at random takes on average 1 hour with the Maximal Overhead strategy). Because the Maximal Overhead strategy takes way more time than the others to give results, this strategy has first been run with a number of nodes in the interval $[2, 10[$, then with the whole $[2, 20[$ a lower number of times to confirm the tendencies observed with the lower number of nodes.

4.2 Visualization All receivers

This section presents a visualization of the data obtained through the multiple test runs with the All receivers strategy. Each subsection shows the results using 3 histograms, representing the raw data, followed by 3 scatter plots picturing each metric against one another. These graphs also show a simple linear regression as an attempt to find a relation between each metric. At the end of the chapter a linear regression is performed in order to find the scores of each variable according to the model presented in Subsec. 3.1.2.

4.2.1 Round-Robin

The distributions of the number of nodes and the latency (Fig. 4.1 top row) have the same shapes except for the gaps every 3 steps. The resemblance in shape is explained by the fact that the latency is strictly correlated to the number of nodes, as expected. The gaps will be explained at the end of the section, using the graphs that show relations between variables.

The histogram for the overhead is easy to analyse as well; the overhead is expected to be 1 because once consensus is obtained for the genesis block, only one message from the next validator is needed to finalize the next block.

The right and center graphs on Fig. 4.2 do not give more insight on the data, as the overhead is always 1. On the other hand, the graph on the left shows a clear linear relationship between the latency and the number of nodes. The fitted line has

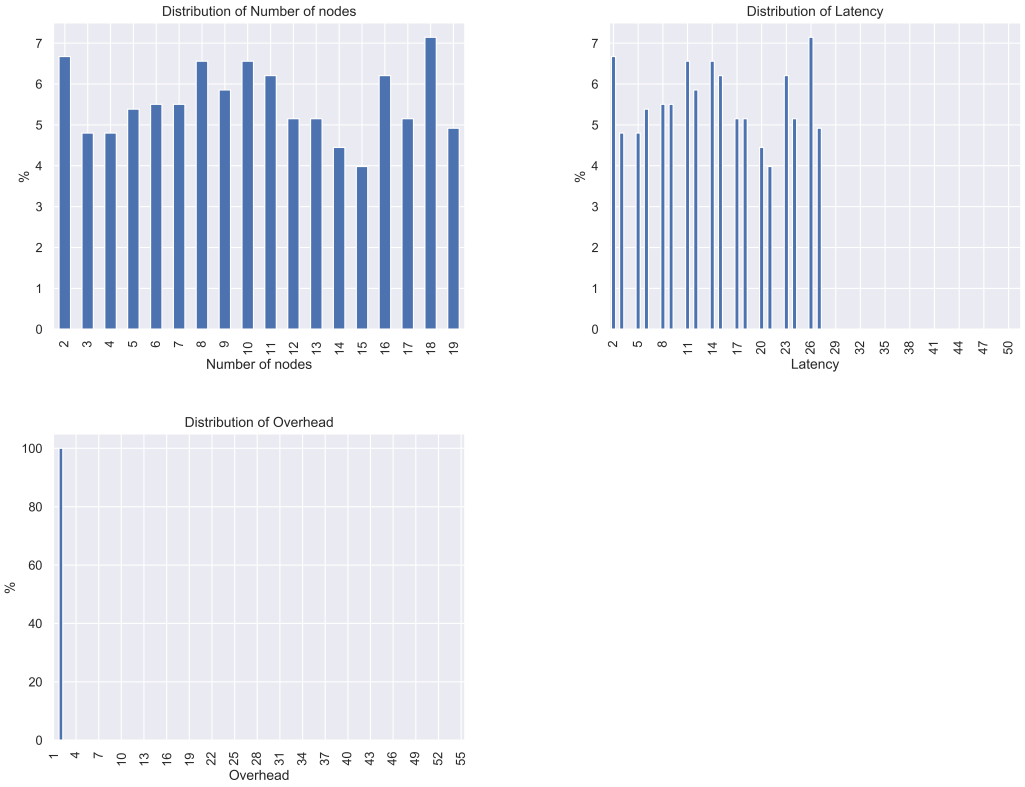

Figure 4.1 *Distributions of the dataset for the Round-Robin strategy and all receivers. As the latency (top-right) presents a linear correlation with the number of nodes (top-left), the shapes of the histograms are similar. The overhead (bottom) always equals to* 1 *because only one message is sent per step, and each step finalizes a block.*

the following equation:

$$l = 1.403402 \cdot n$$

The latency is expected to be around $1.5 \cdot n$ because the finality is reached when at least 50% of the network has acknowledged that the whole network has the finalized message in its justification. As pictured in the top-left graph of Fig. 4.2, the fitted line has a slightly gentler slope compared to the data points due to the fact that the line is fitted to be linear and the data points are close the origin. The gaps in the histogram for latency showed in Fig. 4.1 are explained by the fact that we expect the relation to be $l = 1.5 \cdot n$, and therefore the span of the latency distribution is bigger than the one for the number of nodes. The regularity of the gaps (which appear to be every 3 units of latency), is caused by the regularity of the sending and receiving strategies. As all nodes receive the messages at the same time they all finalize the same blocks together and there cannot be a non-integer average value for the latency in this case.

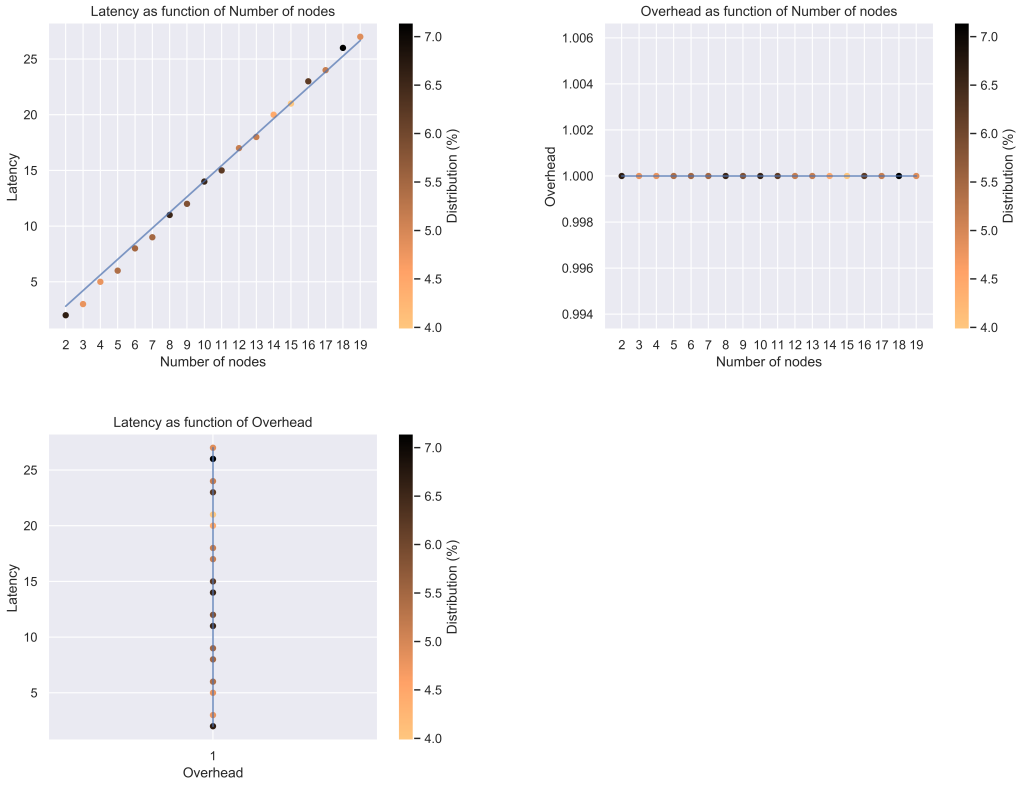

Figure 4.2 *The Round-Robin strategy shows a linear relation between the number of nodes and the latency (top-left). The overhead is a constant for this strategy (top-right, bottom).*

4.2.2 Double Round-Robin

As for the simple Round-Robin case, the latency and number of nodes distributions show a similar shape. The latency is more localized than for the single Round-Robin and demonstrates a difference between distributions. The distribution of the overhead presents a peak at 2, which was expected, but also shows that there are some messages that have double this overhead at 4. The appearance of these outliers is explained further in Subsec. 4.2.5.

As for the simple Round-Robin strategy, the plots including the overhead are not of much use here, except they show that its value is 2, as it is expected. On the other hand, the plot on the top-left of Fig. 4.4 shows a linear relation between the number of nodes and the latency:

$$l = 0.721750 \cdot n$$

The coefficient of this latency in relation to the number of nodes is about half that of one single Round-Robin method. The Double Round-Robin strategy was supposed to show half the latency with respect to the simple Round-Robin and this value is therefore correct. The value of 2 for the overhead is also twice the overhead for the single Round-Robin and confirms the hypothesis.

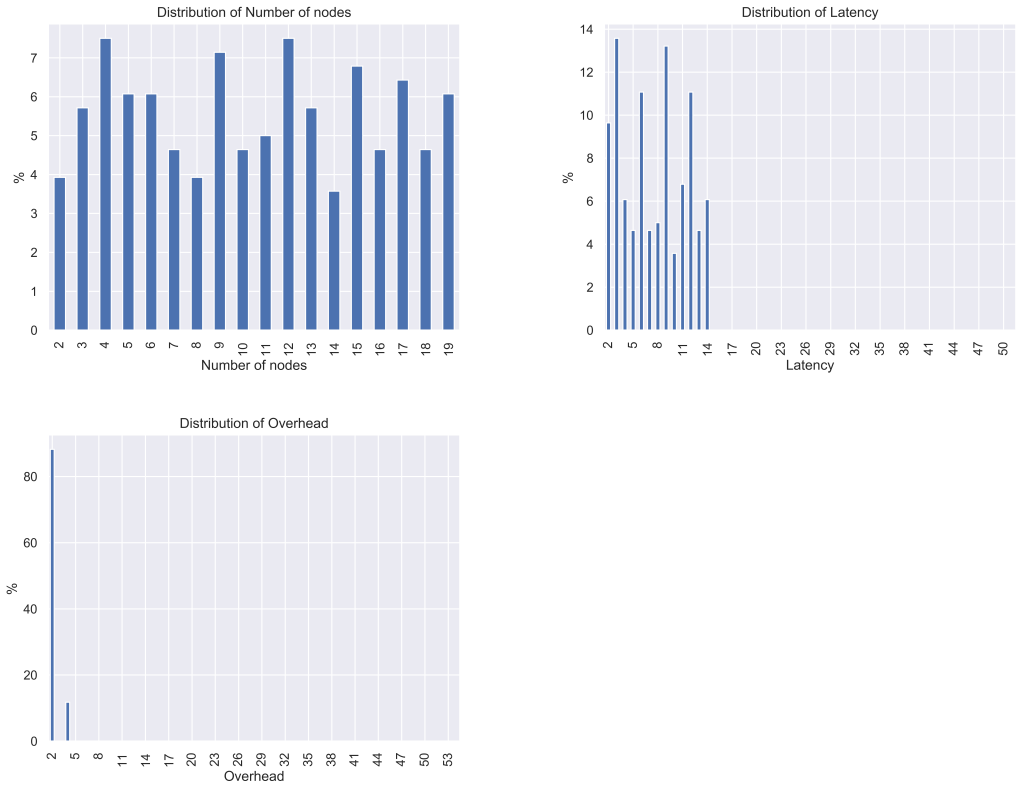

Figure 4.3 *Distributions of the dataset for the Double Round-Robin strategy and all receivers. As the latency (top-right) and the number of nodes (top-left) are linearly correlated, their distributions are of similar shapes. The overhead (bottom) equals 2, as expected and shows an outlying value at 4.*

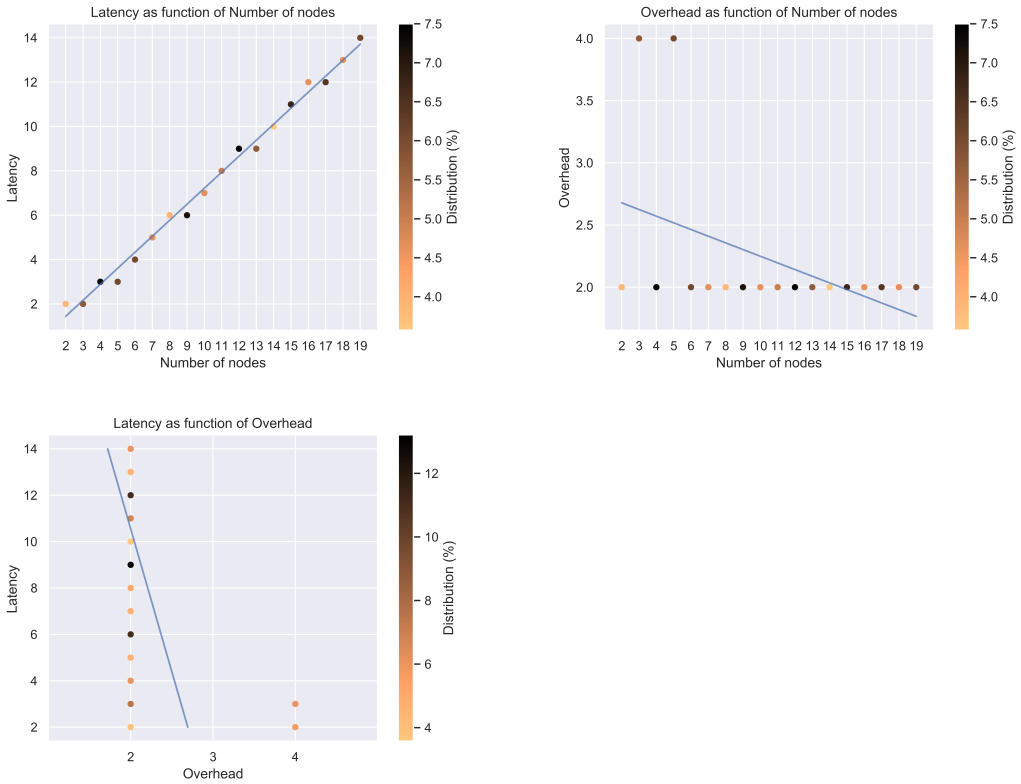

Figure 4.4 *The Double Round-Robin strategy shows a linear relation between the number of nodes and the latency (top-left). The overhead is a constant for this strategy (top-right, bottom). The top-right plot indicates that the outliers only emerge when the tests are run with 3 or 5 nodes.*

4.2.3 Triple Round-Robin

The Triple Round-Robin strategy shows the same kind of information as the Double Round-Robin one. The overhead is 3 for almost all cases (this will be discussed later), and the number of nodes and latency are linearly correlated.

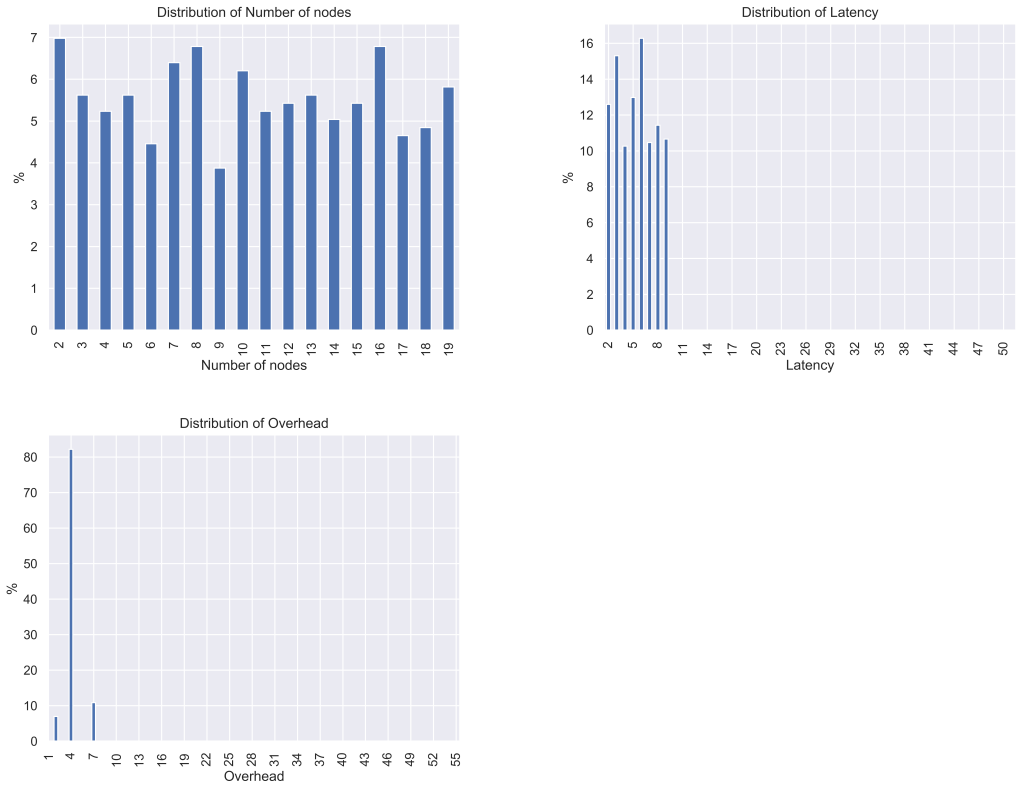

Figure 4.5 *Distributions of the dataset for the Triple Round-Robin strategy and all receivers. As the latency (top-right) and the number of nodes (top-left) are linearly correlated, their distributions are of similar shapes. The overhead (bottom) equals 3, as expected and shows two outlying values at 1 and 6.*

The relationship with the overhead are again not of much use because its value is always 3. Again, there is a linear relationship between the latency and the number of nodes, which follows this equation:

$$l = 0.496152 \cdot n$$

This coefficient is about a third that of the single Round-Robin method and the overhead is three times larger. Fig. 4.6 shows clear outliers for the overhead value when running 2, 5 and 11 nodes. When running 2 nodes, which is an obvious edge case for this strategy (as there are only 2 nodes and there should be at least 3 to send different messages), the overhead is only 1 because only one node sends a message in this case. The appearance of the other two outliers is explained further in Subsec. 4.2.5.

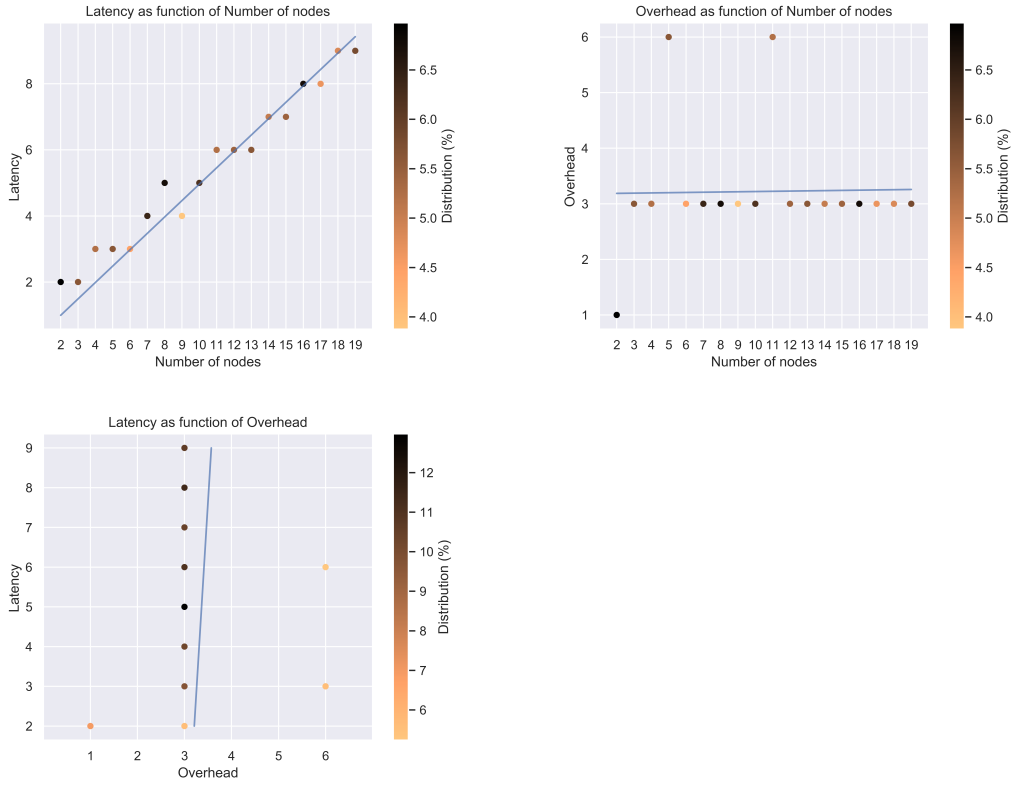

Figure 4.6 *The Triple Round-Robin strategy shows a linear relation between the number of nodes and the latency (top-left). The overhead is a constant for this strategy (top-right, bottom) except for some outliers. The top-right plot indicates that the overhead is lower for 2 nodes, which is expected because fewer than 3 messages are sent in this case. Outliers are also found when running 5 and 11 nodes.*

4.2.4 Maximal Overhead

This strategy aims to reduce latency to its minimum, as shown in the top-right plot on Fig. 4.7. The distributions of the number of nodes and overhead are of the same shape because they are linearly correlated, as will be shown in the next paragraph. Note that the experiments were run with a lower number of nodes because the simulations for a large number of nodes takes a large amount of time. The simulations are currently implemented following the mathematical formulae found in [1] and not yet optimized. For the sake of completion, a few runs have been performed with the same amount of nodes as for the other experiments to show whether or not there were divergences. It turned out that there was not such divergences.

The top-right plot on Fig. 4.8 shows a linear dependency between overhead and number of nodes of $o = n$, as was expected. The two other plots on the same Figure only confirm that the latency equals 2, which is the minimum that can be reached, as there needs to be two steps to finalize a message. During the first step, all nodes acknowledge they have seen a message, and the second step confirms that everyone has seen the other nodes' acknowledgments.

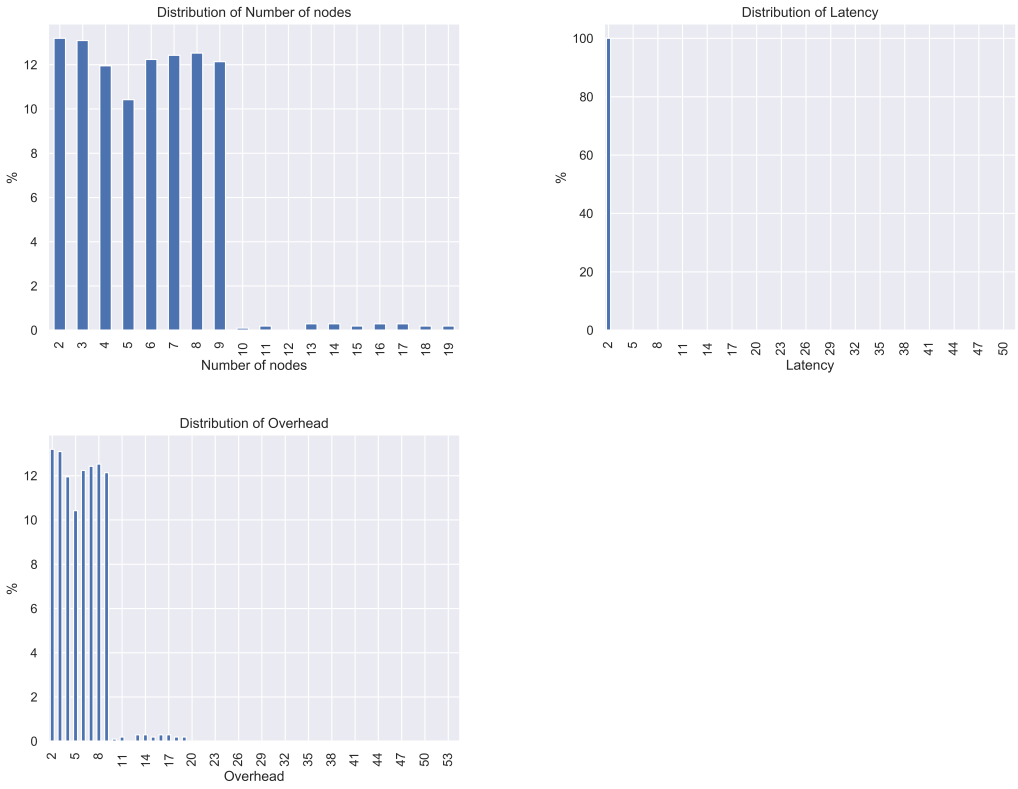

Figure 4.7 *Distributions of the dataset for the Maximal Overhead strategy and all receivers. As the overhead (bottom) and the number of nodes (top-left) are linearly correlated, their distributions are of similar shapes. The latency (top-right) equals 2, as expected. Note that because the maximal overhead strategy is computationally heavy, tests were run for 10 nodes and a few of them were run for 19 nodes, which is reflected in the shapes of the bottom and top-left histograms.*

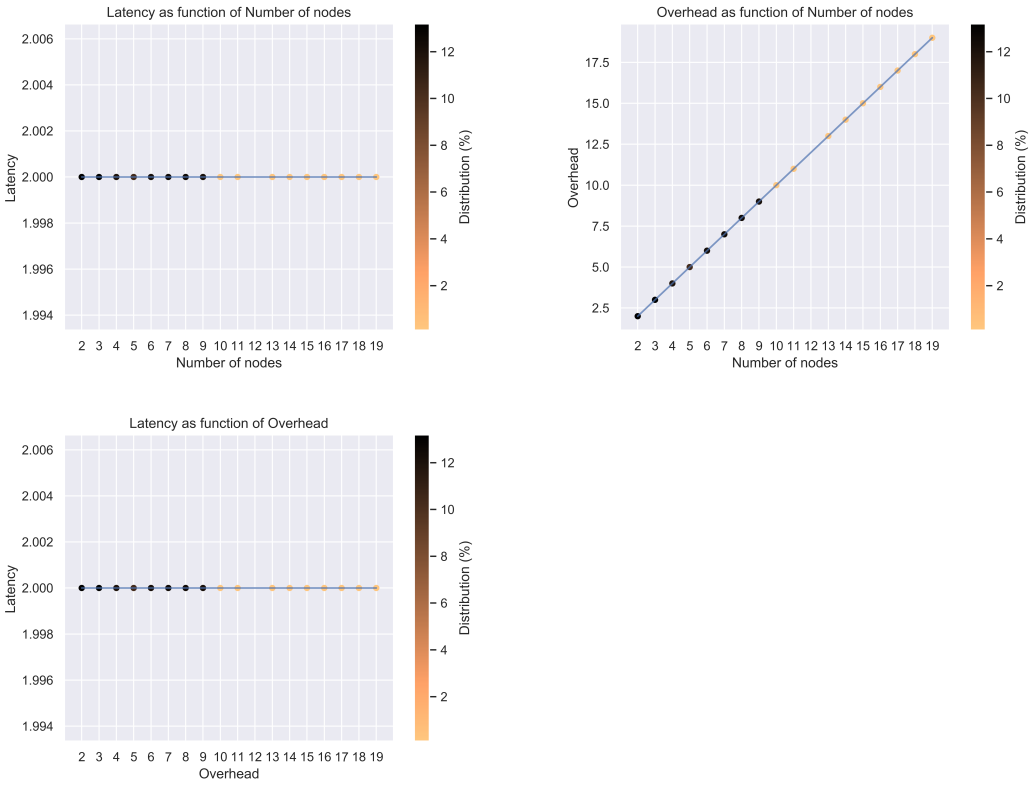

Figure 4.8 *The Maximal Overhead strategy shows a linear relation between the number of nodes and the overhead (top-right). The latency is a constant for this strategy (top-left, bottom).*

4.2.5 Arbitrary

This strategy is the only one that uses a random factor and that can be considered as a "bottom-up" strategy. The main goal of this strategy is to have a reference point when comparing new strategies. The overhead and latency distributions both seem to follow a gamma distribution explained by the senders being selected at random.

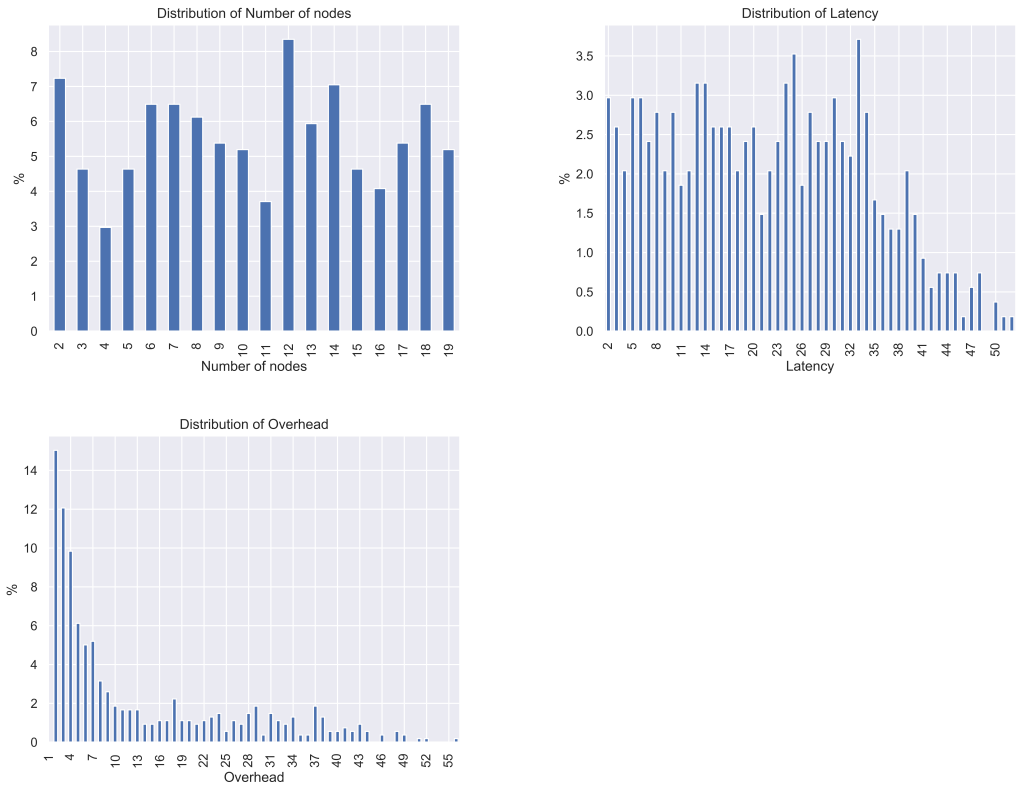

Figure 4.9 *Distributions of the dataset for the arbitrary strategy and all receivers. The latency and overhead seem to follow a gamma distribution.*

The top-left graph on Fig. 4.10 shows a linear relationship between the latency and the number of nodes. The equation of the fitted line is:

$$l = 2.091255 \cdot n$$

Though the slope varies a bit, it is still very similar to that of the simple Round-Robin strategy.

The two other graphs on the figure are less straightforward to explain than the previous strategies'. The plot that illustrates the relationship between the overhead and the number of nodes shows two linear branches that split quite clearly, and the same goes for the plot of the latency against the overhead. The latter one shows an even weirder artifact:

During the analysis phase of this strategy and particularly when trying to explain the straight line appearing on the bottom-left graph on Fig. 4.10, a critical bug was discovered. This bug happens when two or more messages are finalized during the

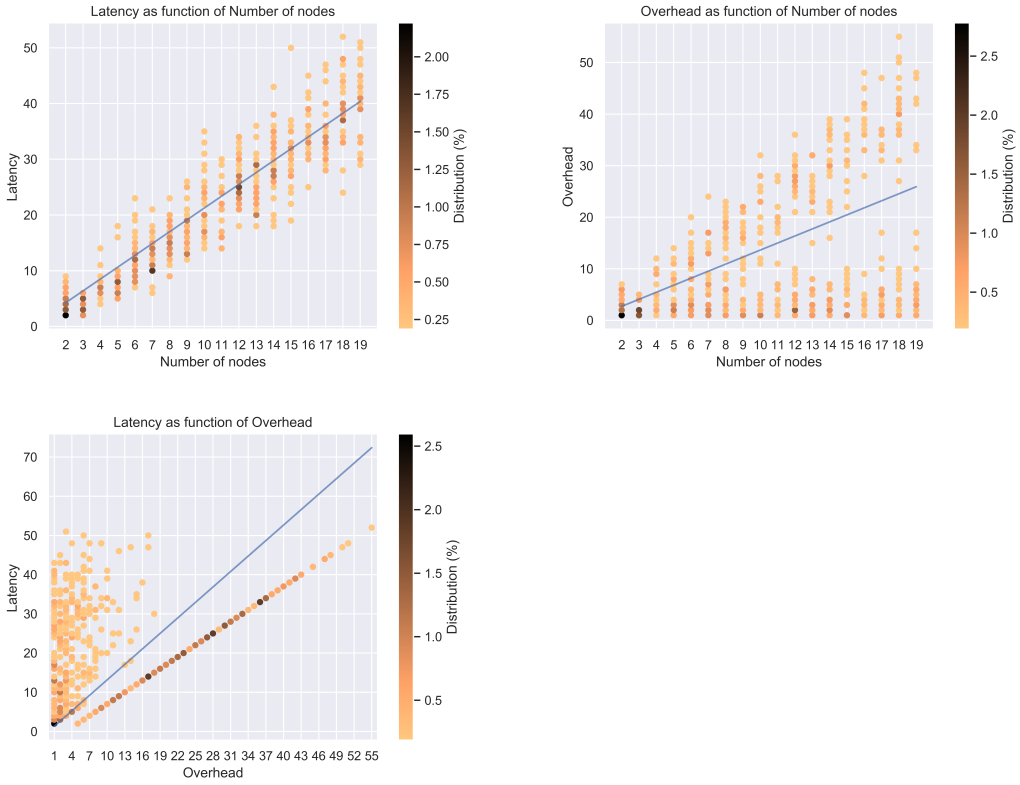

Figure 4.10 *The arbitrary strategy shows a linear relation between the number of nodes and the latency (top-left). However, the other relations between variables both seem to be split into two linear functions with a clear gaps between them.*

last step of the simulation. As the bug was identified the near end of the project's timeframe, it was not corrected, but will require to be fixed in order to continue research on the subject. As the data collection process is segregated from the metrics computations (Subsec. 3.3.2), data that has already been generated are not lost and can be used again when the bug is corrected. Note that this bug also explains the outliers observed for the Double Round-Robin and Triple Round-Robin strategies as well (Subsec. 4.2.2 and Subsec. 4.2.3). Nonetheless, a version of the graphs with the edge cases filtered out was made in order to better benchmark the strategy (Fig. 4.11). The filtered data show a linear dependency between the number of nodes and the latency with the same slope as the unfiltered datapoints. There is no clear relation between number of nodes and overhead as the overhead is distributed evenly across all number of nodes which is confirmed by both the top-left graph on Fig. 4.11 and the histogram on Fig. 4.9. The relationship between latency and overhead seem to be spread around a center of mass around the point $(4, 15)$ but more data is needed to show a clear correlation.

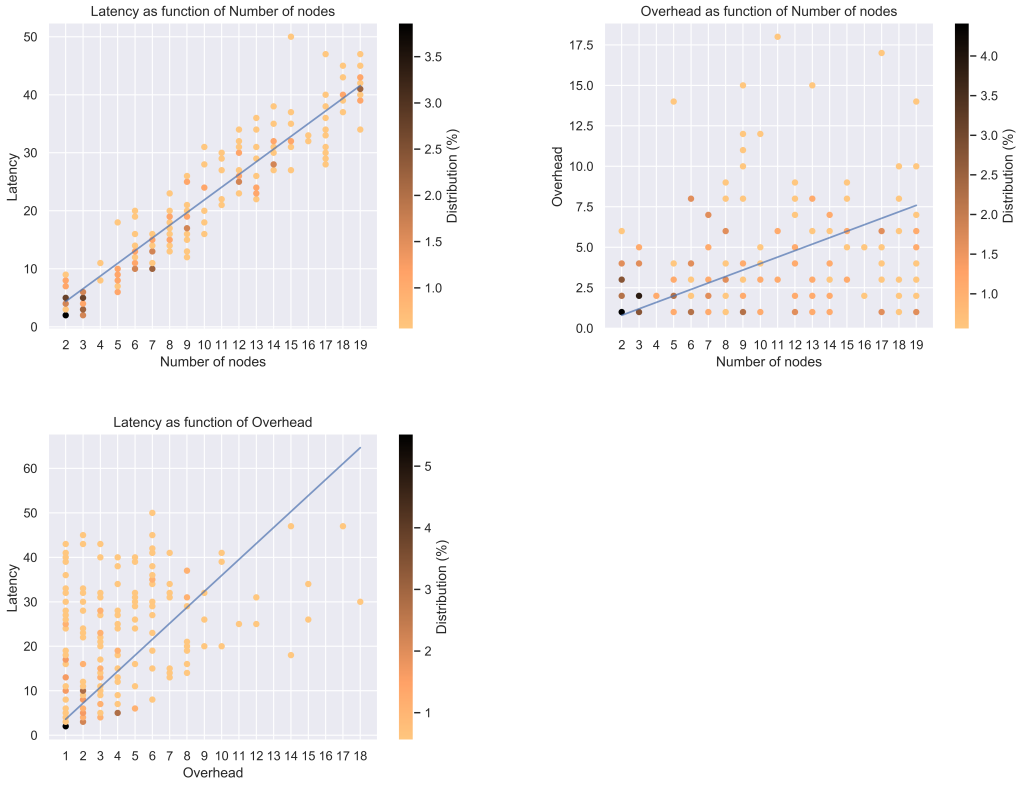

Figure 4.11 *The filtered arbitrary strategy shows a linear relation between the number of nodes and the latency (top-left). The top-right graph, in combination with the histogram on Fig. 4.9 shows that the overhead is evenly spread with no correlation with the number of nodes.*

4.2.6 Fitting of the basic model

Fitting the basic model described in Subsec. 3.1.2 ($1 = s_n \cdot \frac{1}{n} + s_l \cdot l + s_o \cdot o$) to the data using a simple linear regression gives the results as shown as raw values on Table 4.1 and in a bar graph on Fig. 4.12.

The scores obtained for the Round-Robin and the Maximal Overhead strategies are the expected ones. The Round-Robin strategy optimizes the overhead (one message per new finalized block) and the latency evolves linearly with the number of nodes and is therefore bad. The score for the number of nodes will be discussed later in this chapter. The Maximal Overhead strategy shows that it maximizes the latency (which is 2, the minimal value for the latency) and therefore the rest of the scores are non influential. For the Double and Triple Round-Robin strategies, the variation for the overhead and the latency scores, compared to the ones for the Round-Robin are also in accordance with what is expected; the latency decreases when the number of rounds augments and the overhead increases along with the number of nodes sending messages in a single round.

Sending Strategy	s_n	s_l	s_o	RMSE
Round-Robin	0.000	0.000	1.000	0.000
Double Round-Robin	1.435	0.055	0.162	0.141
Triple Round-Robin	1.686	0.084	0.091	0.115
Maximal Overhead	0.000	0.500	0.000	0.000
Arbitrary	2.556	0.026	0.002	0.233

Table 4.1 Raw fitted values for the simple model, rounded to the 3rd decimal place. The RMSE column shows the error value for the linear regression. The rows for both Round-Robin and Maximal Overhead are both expected; the Round-Robin strategy maximizes the overhead score and the Maximal Overhead strategy maximizes the latency score. The score for the latency increases with the number of message per step as the score for the overhead decreases. The score for the number of nodes seems to be a buffer that depends on how bad the two other scores are, which is confirmed by the error that increases with it.

Number of nodes

The main problem with these results is the evolution of the score for the number of nodes, particularly for the arbitrary strategy. Based on the observations made in Subsec. 4.2.5, the arbitrary strategy is clearly worse than the others ones for the latency and overhead which is reflected with the values obtained for their respective scores. The main issue with the arbitrary strategy relies in the fact that s_n serves as a buffer for how well the other two variables perform in a given strategy. Indeed, the model forces the relation between the scores and the variables, and if the scores for both the latency and the overhead are low, the score for the number of nodes is forced to be high. Furthermore, the number of nodes is the variable with the lowest variance so the fitting shows a lower error by using it as a buffer instead of the two others. This is confirmed by looking at the error column in Table 4.1; the bigger the score for number of nodes is, the higher the error.

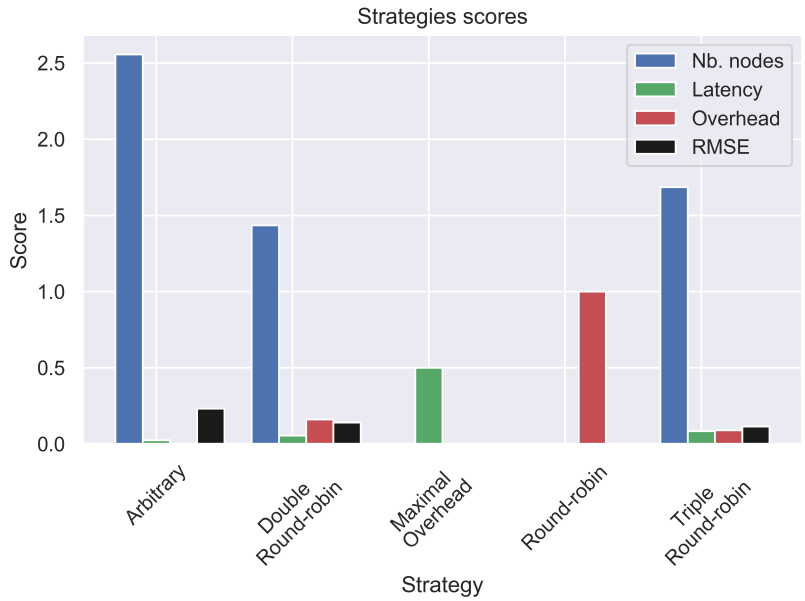

Figure 4.12 *Raw fitted values for the simple model. The strategies are listed on the x-axis and their scores are plotted along the y-axis. The RMSE column shows the error value for the linear regression. The rows for both Round-Robin and Maximal Overhead are both expected; the Round-Robin strategy maximizes the overhead score and the Maximal Overhead strategy maximizes the latency score. The score for the latency increases with the number of message per step as the score for the overhead decreases. The score for the number of nodes seems to be a buffer that depends on how bad the two other scores are, which is confirmed by the error that increases with it.*

4.2.7 Normalized model

Fig. 4.13 presents a normalized visualization of the same data in Fig. 4.12. The normalization is made per-class, i. e. each class is normalized against the best score for that class. This normalization mainly permits to better visually compare the strategies on a per-class basis. It also shows a correlation between the scores for the number of nodes and the error values, which further confirms that the score for the number of nodes is used as a buffer when the two other scores are sub-par and don't fit the model.

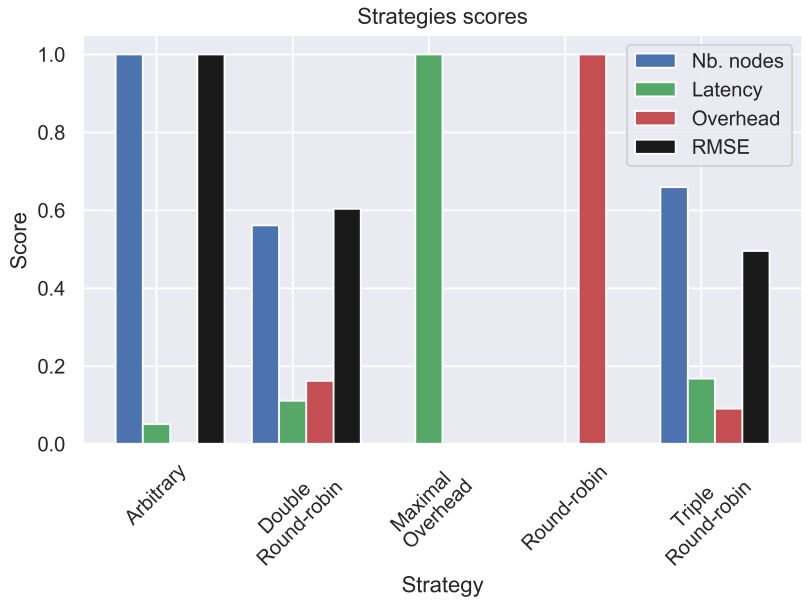

Figure 4.13 *Normalized fitted values for the simple model. The strategies are listed on the x-axis and their scores are plotted along the y-axis. This per-class normalization facilitates the comparison of strategies on one particular score. The correlation between the error score and the score for the number of nodes is also clearer when presented this way than, compared to the raw values.*

4.2.8 Improvements of the basic model and new fitting

Now that there are exploitable results for the basic model, some improvements can be made. The first step is to standardize the scale by imposing the range $[0, 1]$ which simplifies the visual juxtapisition as seen in Fig. 4.13. The next step is to try to give a meaning to relative values for the same score when comparing two strategies. This is obtained by finding better exponents for the variables in the model.

Improving factors

As seen in Subsec. 4.2.6, the best score for the latency is 0.5, which is due to the fact that latency is at least 2. A 0.5 factor can therefore be added to the model to obtain a value of 1 as the maximum score for the latency:

$$1 = s_n \cdot \frac{1}{n} + \frac{1}{2} s_l \cdot l + s_o \cdot o$$

In the same manner, to scale the possible scores for the number of nodes to the chosen interval of $[0, 1]$, a factor of 2.556 can be added to the model:

$$1 = 2.556 \cdot s_n \cdot \frac{1}{n} + \frac{1}{2} s_l \cdot l + s_o \cdot o$$

Both models are represented on Fig. 4.14

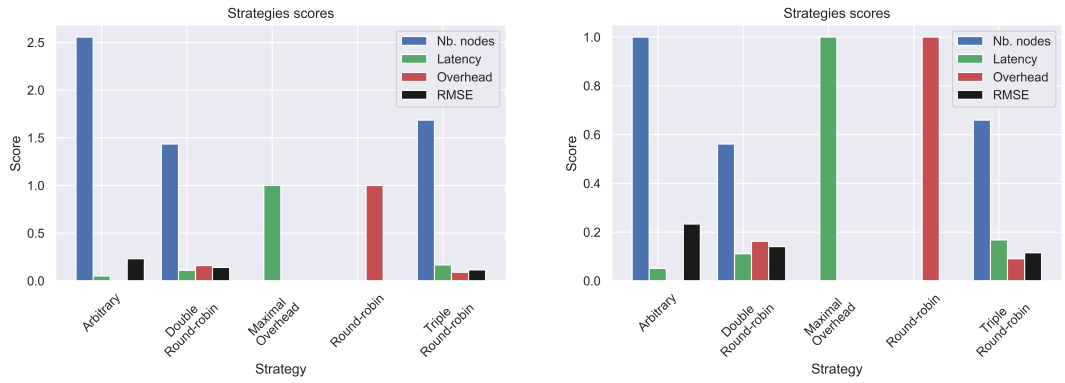

Figure 4.14 *Improvement of the factors of the basic model. On the left, the latency score has a maximum of* 1.0. *On the right, the score for the number of nodes is also normalized to a maximum of* 1.0.

Improving exponents

Now that the model gives a normalized output, it is possible to improve it by changing the exponents applied to the variables which could produce scores that are easier to reason about. For example, as described in Subsec. 4.2.2 and Subsec. 4.2.3 the overhead for the Double Round-Robin is 2/3 of the overhead for the Triple Round-Robin and, therefore, its score should be 3/2 of the overhead score for the Triple Round-Robin. However it seems it isn't the case as the score values in Table 4.1 for the Double an Triple Round-Robins are worth 0.162 and 0.091 and their ratio amounts to 1.780 but by performing a bisection on the function $\frac{0.162}{0.091}^{e}$ with a goal value of 3/2, an exponent of $\frac{1}{2}$ is revealed. As in Fig. 4.15 and Table 4.2,using this newly calculated exponent gives a much better ratio to the overhead score for each Round-Robin strategy pair: Single-Double(2.092), Double-Triple (1.598), and Single-Triple (3.344). Indeed, the overhead score for the single Round-Robin strategy should be twice that of the Double Round-Robin strategy and three times as much as the score for the Triple Round-Robin. By looking at the error values obtained by the fitting of the new model on the data (Fig. 4.3), it can be seen that those are marginally lower than the original basic model, which means the model better fits the data that output predictable results.

The exponent for the latency is not to be changed because the scores already match the expectations (the score for the Triple Round-Robin is 3/2 of the score for the Double Round-Robin strategy).

Again, the exponent for the number of nodes will not be modified because its score's role in the model is to buffer.

The final model is therefore:

$$1 = 2.556 \cdot s_n \cdot \frac{1}{n} + \frac{1}{2} s_l \cdot l + s_o \cdot \sqrt{o}$$

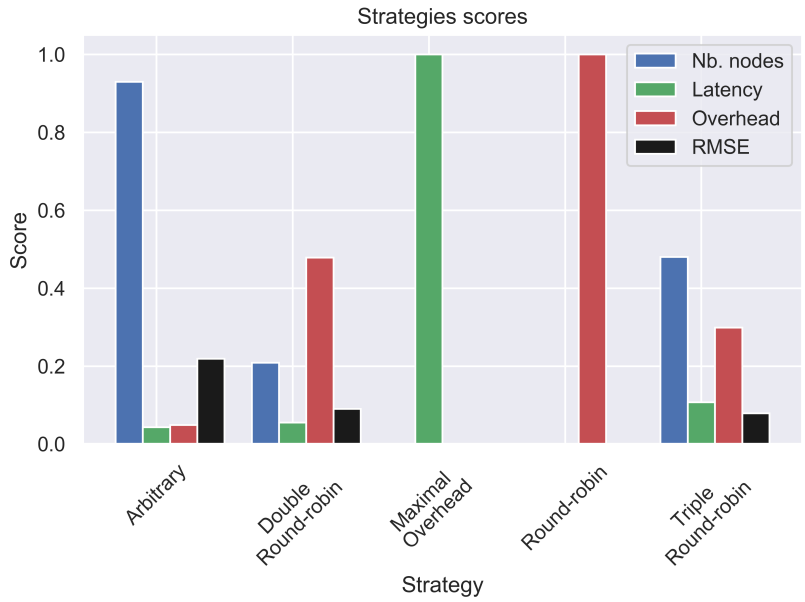

Figure 4.15 *Raw fitted values for the simple model, rounded to the 3rd decimal place. The RMSE column shows the error value for the linear regression. The scores are now normalized to the* $[0, 1]$ *interval. The new model also improves the relatives values for the overhead score, the Double Round-Robin score is roughly half of the simple Round-Robin and the Triple Round-Robin score is roughly a third of the overhead score for the simple Round-Robin strategy.*

Sending Strategy	s_n	s_l	s_o	RMSE
Round-Robin	0.000	0.000	1.000	0.000
Double Round-Robin	0.209	0.055	0.478	0.090
Triple Round-Robin	0.480	0.107	0.299	0.079
Maximal Overhead	0.000	1.000	0.000	0.000
Arbitrary	0.929	0.043	0.049	0.219

Table 4.2 *Raw fitted values for the simple model, rounded to the 3rd decimal place. The RMSE column shows the error value for the linear regression. The scores are now normalized to the* $[0, 1]$ *interval. The new model also improves the relatives values for the overhead score, the Double Round-Robin score is roughly half of the simple Round-Robin and the Triple Round-Robin score is roughly a third of the overhead score for the simple Round-Robin strategy.*

Sending Strategy	RMSE Basic model	RMSE Improved model
Round-Robin	0.000	0.000
Double Round-Robin	0.141	0.090
Triple Round-Robin	0.115	0.079
Maximal Overhead	0.000	0.000
Arbitrary	0.233	0.219

Table 4.3 Raw error values for the simple and improved models, rounded to the 3rd decimal place. The error values are lower for the improved model which implies that it better fits the data.

4.3 Further research

4.3.1 Resolve edge case bug

As discussed in Subsec. 4.2.5, an edge case bug has been identified at the end of the project and it is necessary to fix this implementation error before researching further on the subject.

4.3.2 Half set strategy

Another strategy that could be interesting to implement consists in half the validator set sends a message at each step. That would give a new comparison point with a fixed latency and an overhead that would be half the one for the Maximal Overhead strategy. It would be useful to have it to find better coefficients for the refined model.

4.3.3 Bottom up strategies

Now that the framework is functional and can give some reference points and has ways to compare strategies, the next step would be to implement bottom up strategies (see Subsec. 3.2.6) that could include rewarding/slashing the stake of the nodes.

4.3.4 Optimize and simplify model

As seen in Subsections 4.2.6 and 4.2.8, the model might be too restrictive for the actual problem and it might be worth trying to find a better fitting model to more precisely reproduce the real life use case, by, for example, trying out new exponents for the number of nodes for example. Removing the number of nodes as a whole in the model might also give simpler data to analyse.

4.3.5 Better network modeling

Currently, the `proptest` implementation does not include a good model for the network layer. The three proposed receiving strategies are naive but allowed the validation of the framework from the metrics measurements to the actual model fitting. A step forward would be to create better models for the network, based on real life network topologies, latencies, number of nodes, etc.

4.3.6 Further analyse the data

As seen in Subsec. 3.3.1, more than one sending strategy have been implemented, but only one has been thoroughly analysed. The some receivers strategy was implemented early in the project for other purposes but it turned out during the analysis phase that choosing both the number of nodes that receive the message as well as which ones get them introduced too many parameters varying simultaneously. The half receivers strategy was then implemented but data points were generated too late into the project's lifetime to fully analyse them as well. A logical next step would to use the half receivers strategy to validate that the chosen model does not overfit the all receivers strategy data.

5 Conclusion

This project's target is to setup a methodology to compare block building stratgies for a CBC Casper protocol particularly for blockchain use. The project results in a comparator framework based on a core library implementing the CBC Casper blockchain abstract structure which is generic enough to allow the implementation of new block building strategies, network topologies, and improved termination conditions. The framework oversees three variables for comparison: latency, number of nodes, and overhead; due to their explicit relative performances through core strategies, the framework is valid. After adapting the model to achieve target performances for each strategy, a new model is proposed. This model shows a better means of comparison though still lacking when estimating the scaling of the number of nodes for different strategies despite this variable's lesser impact on the Trilemma model.

Future improvements include: modeling using the number of nodes variable in a different way, modeling in a network topology implementation that reflects a real-life setting, and dipping into "bottom-up" strategies that are more likely to be implemented on a deployed blockchain. Furthermore, data that has been generated with other sending strategies have not been analyzed yet and could be used to optimize the current model.

A | Codebase

This project's codebase spans multiple repositories listed below.

A.1 Local and Docker testnets

`https://github.com/TrueLevelSA/parity-dev-chain`

A.2 Custom Parity implementation

`https://github.com/TrueLevelSA/parity-ethereum/tree/feature/instant_seal_consensus`

A.3 Simulations

`https://github.com/TrueLevelSA/cbc-casper-msg/tree/metrics_extraction`

A.4 Casper Visualization

`https://github.com/TrueLevelSA/casper-visualization/tree/metrics_extraction`

List of Figures

List of Tables

Bibliography

[1] Vlad Zamfir et al. *Introducing the "Minimal CBC Casper" Family of Consensus Protocols*. 2018. URL: https://github.com/cbc-casper/cbc-casper-paper/raw/master/cbc-casper-paper-draft.pdf (visited on 03/03/2019).

[2] Vlad Zamfir. *A Template for Correct-by-Construction Consensus Protocols*. 2017. URL: https://github.com/ethereum/research/raw/master/papers/cbc-consensus/AbstractCBC.pdf (visited on 03/03/2019).

[3] Yonatan Sompolinsky and Aviv Zohar. *Secure High-Rate Transaction Processing in Bitcoin*. 2013. URL: https://fc15.ifca.ai/preproceedings/paper_30.pdf (visited on 03/03/2019).

[4] Dr. Gavin Wood. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER*. 2019. URL: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 03/03/2019).

[5] Miguel Castro and Barbara Liskov. *Practical Byzantine Fault Tolerance*. 1999. URL: http://pmg.csail.mit.edu/papers/osdi99.pdf (visited on 03/03/2019).

Glossary

CBC Correct by Construction. 1, 3–7, 10, 35, 39

DAG Directed Acyclic Graph. 3

GHOST Greedy Heaviest Observed Sub-Tree. 3, 4, 39

PoS Proof-of-Stake. ix, 1, 3, 5, 6, 43

PoW Proof-of-Work. ix, 1, 3, 5, 6, 43