



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

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
Advisor

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

Abstract

TODO: add abstract

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 (Proof-of-Work (PoW)) consensus protocol to a Proof of Stake (Proof-of-Stake (PoS)) one. 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.

1.3 Contents

TODO: explain what is in this

2 | Background

2.1 PoW and PoS

In Ethereum, PoS is aiming at replacing PoW as a distributed consensus algorithm. This section succinctly describe both methods as well as the main differences between them, and then explains which problems arise when you replace PoW with PoS.

2.1.1 What is PoW?

PoW is the current consensus protocol used to decide on a blockchain 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 have -on average- a block created at a set interval. 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. Miners are therefore incentivised 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 said work is a proxy for timing; a miner cannot create an arbitrary large number of blocks in a short time because it's inherent to the protocol.

2.1.2 What is PoS?

PoS, on the other hand, selects a new block creator according to its weight (or stake). This weight can be 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 TODO: cite somewhere is an abstract consensus protocol family which is PoS-ready. Nodes, 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, messages each point to one 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 returns a blockchain.

Fig. ?? shows a small example of a CBC execution over a blockchain. Four nodes are pictured. Colored circles are messages sent by validators, dotted arrows show their justifications. The selected blockchain is depicted with blue arrows. It is obtained by running the GHOST algorithm on the pictured view of the network, and would be the estimate of a new message sent by an honest validator that has this view.

Fig. ?? 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 justification, it is considered as an *equivocator*, does not follow the protocol, and could be punished by the network for such a behavior. Note that validator v_1 does not equivocate because its first message is in the justification of v_2 's first message, and v_1 has this message in the justification of its second message. It is said that v_1 's first message is in the *dependency* of its later messages.

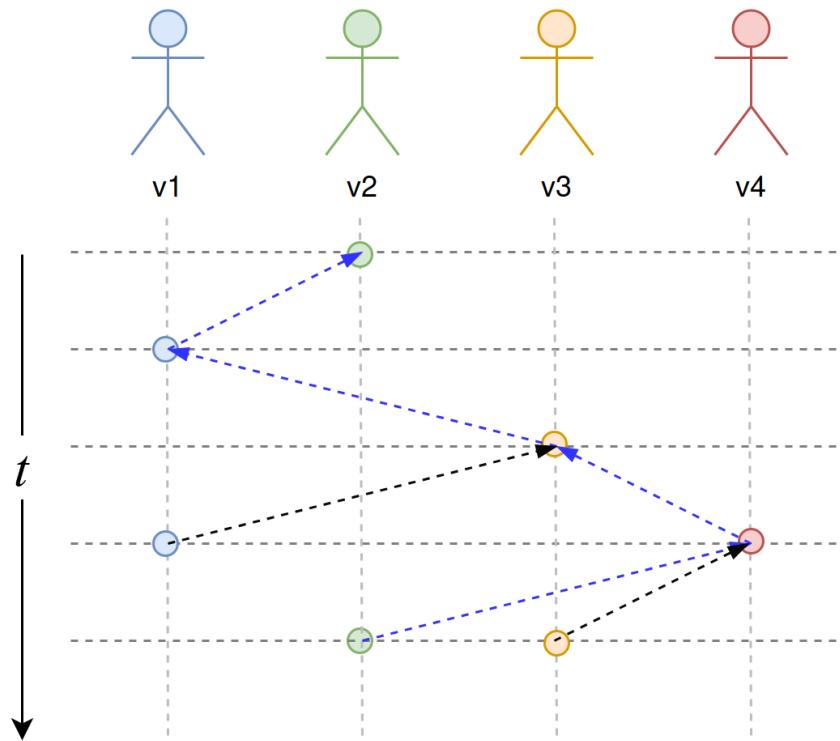


Figure 2.1 CBC blockchain example

TODO: include a schema showing validators, justifications, estimates, ... in more depth

TODO: describe liveness, safety
 TODO: talk about finality, safety oracles, ...
 TODO: change structure; make comparison not between POS and POW but between POW and CBC

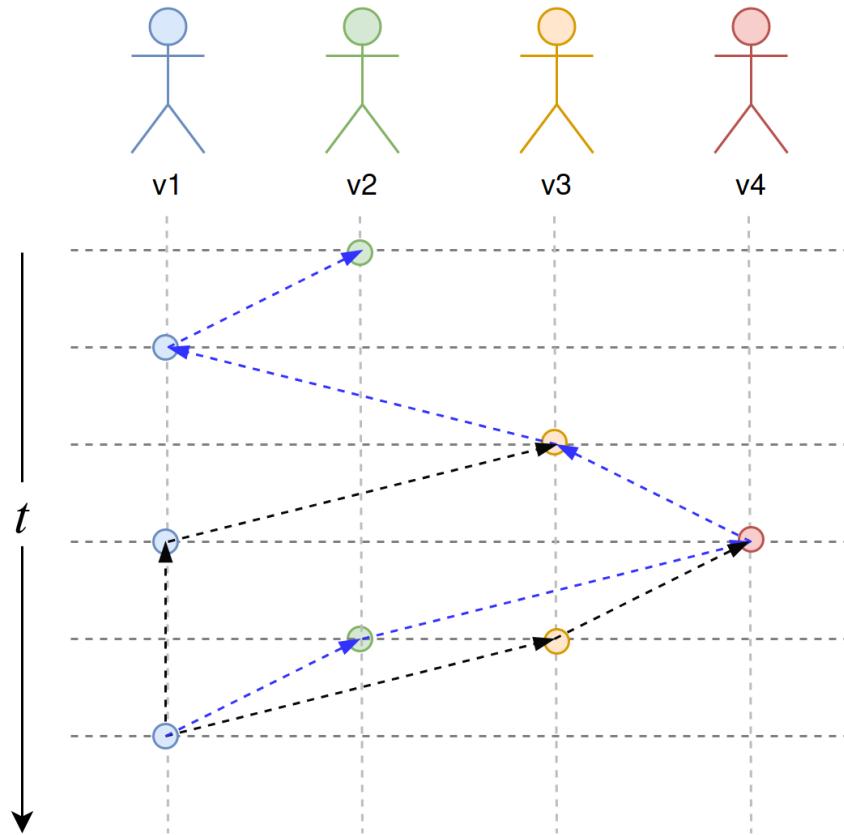


Figure 2.2 CBC blockchain example 2

2.1.4 CBC Casper vs PoW

Block production

TODO: cite ref In a PoW setting, a miner can broadcast any block to the network, but the block will only be picked up by the other nodes if it is valid. As the miner has to work to produce a valid block, it cannot create blocks on the fly at any time, whereas in a CBC Casper setting, a node can broadcast blocks at any time.

Timing assumption

In a PoW protocol, the work a miner has to do to create a block is parametrized by the difficulty of the cryptographic puzzle it has to solve. This difficulty is set in such a way that it takes a certain time -on average- to solve the puzzle. On the contrary, 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 to produce Casper messages, a node could easily spam the whole network.

Economic majority

TODO: cite ref, and better explanation As machines that are better at solving the cryptographic puzzle in a PoW context are more expensive to buy and to operate, **TODO: nah** Economic majority is however built in the CBC Casper protocol, in the form of the weight that validators have.

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 they 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

TODO: rename section As seen in Subsec. ??, 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` a Rust implementation of the CBC-Casper, made by TrueLevel **TODO: introduce somewhere**. It implements the consensus algorithms proposed in the paper and offers an abstract structure that can be used to create consensus on any value.

2.4 Parity Ethereum

TODO: remove redundancy in this chapter **TODO: parity networking/gossiping layer research?** 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 test block creation strategies in a pseudo real-like manner. 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, **TODO: rephrase "a lot of"** a lot of 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. The choice of using Docker was made because there was a possibility to work on the UniNE clusters, which happen to work well with containerized software. 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 main disadvantage of doing the experimentations 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. **TODO: add that the lib was not fully tested at this point and that further testing was needed before being able to test parity**

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.

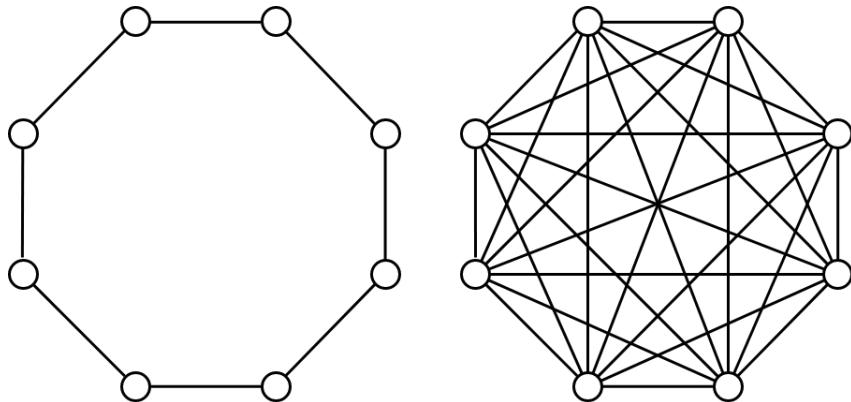


Figure 2.3 Parity nodes topologies implemented in the Docker testnet. Ring layout (left), fully connected (right)

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 than adding and removing topologies is easily done through abstract structures.

3 | Testing Framework Implementation

3.1 Modelisation

3.1.1 Metrics

A way to rationally compare strategies is needed in order to discuss their relative strengths and weaknesses. Three main characteristics will be used for that:

- latency;
- number of nodes;
- overhead.

Latency

The latency is the number of messages needed to finalize a block. Ideally, you want to have a latency as low as possible to reach finality as soon as possible. The latency is a way to measure liveness in a blockchain system. If it is low, then the system is considered more "live", as fewer messages are needed in order to confirm a transaction, and therefore less time.

Number of nodes

The number of nodes is quite straightforward; it is the number of nodes that are in the validator set. This number should be as high as possible to guarantee decentralization and therefore 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.

Example

Fig. ?? describes how the computation of the metrics takes place. The left half of the figure shows an initial view of the system. At time t_1 , message m_1 is finalized. The right half of the figure shows a view of the system at t_2 , time at which m_2 is finalized. The latency of m_2 is l_2 , the difference of heights between m_3 and m_2 . In this example, $l_2 = 4$. o_2 , the overhead for m_2 , is the number of messages that have been sent between t_1 and t_2 , t_1 being the time at which the previous message has been finalized and t_2 the time at which the current message is finalized. In the example, $o_2 = 3$.

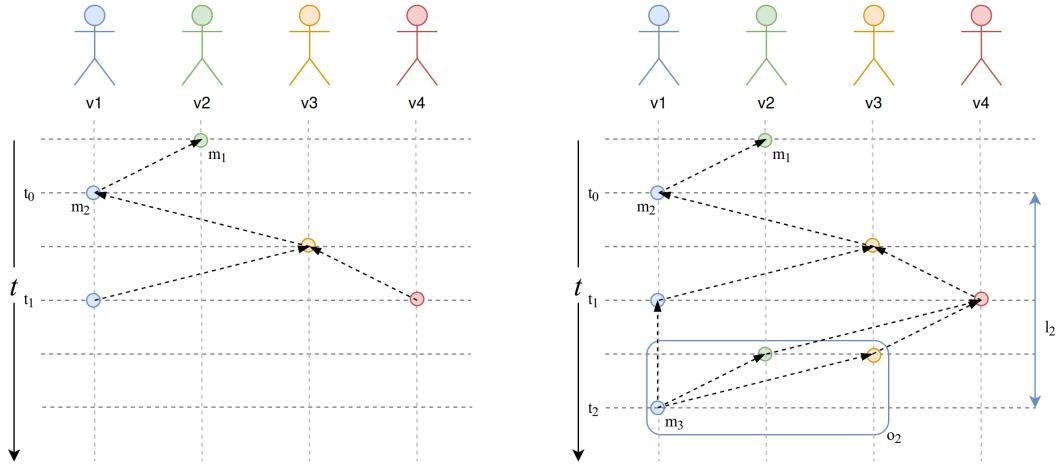


Figure 3.1 Metrics computation example

3.1.2 Tradeoff triangle/Trilemma

In a standard consensus protocol, the three metrics form a trade-off triangle in kind of a "pick two" fashion. **TODO: not a pick two** CBC-Casper has no assumptions on timings, sources, contents, destinations of the messages that are exchanged, and can therefore explore the whole trade-off space. This project aims to find strategies that span the entirety of the triangle and **TODO: finish sentence what**

3.1.3 Basic model

The first model that has been chosen for the evaluation is the following:

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

This model binds 3 scores s_x to their respective variables x . Variables are as follows:

- n the number of nodees;
- l the latency;
- o the overhead.

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. This model is a basic one, where everything is linear. It is probably not accurate but will serve as a base for a closer to reality one once data have been plotted using this model.

3.1.4 Model Evaluation

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

3.2 Strategies

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

- round-robin;
- double round-robin;
- triple round-robin;
- 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.

3.2.1 Round-robin

The first strategy that comes to mind is a simple round-robin. Nodes send messages one after the other, in a fixed order.

3.2.2 Double Round-robin

In this setting, two nodes send messages at the same time, in a fixed order. 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. The overhead is however doubled.

3.2.3 Triple Round-robin

This strategy is similar to the Double round-robin, except that three nodes send a message a 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-uple 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.

3.2.4 Maximal Overhead

This strategy is the most expensive in terms of bandwidth; at each step, each validator sends a message to the others. 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 **TODO: cite** protocol, as each node sends a message to every other one over two steps.

3.2.5 Arbitrary

The next strategy is the simplest to think of: complete randomness. Using fixed probability density functions, one node is arbitrarily selected at each step to create a message.

3.2.6 Bottom-up strategies

All the previous strategies (except the Arbitrary one) can be viewed as "top-down" strategies, as there has to be a way for 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.

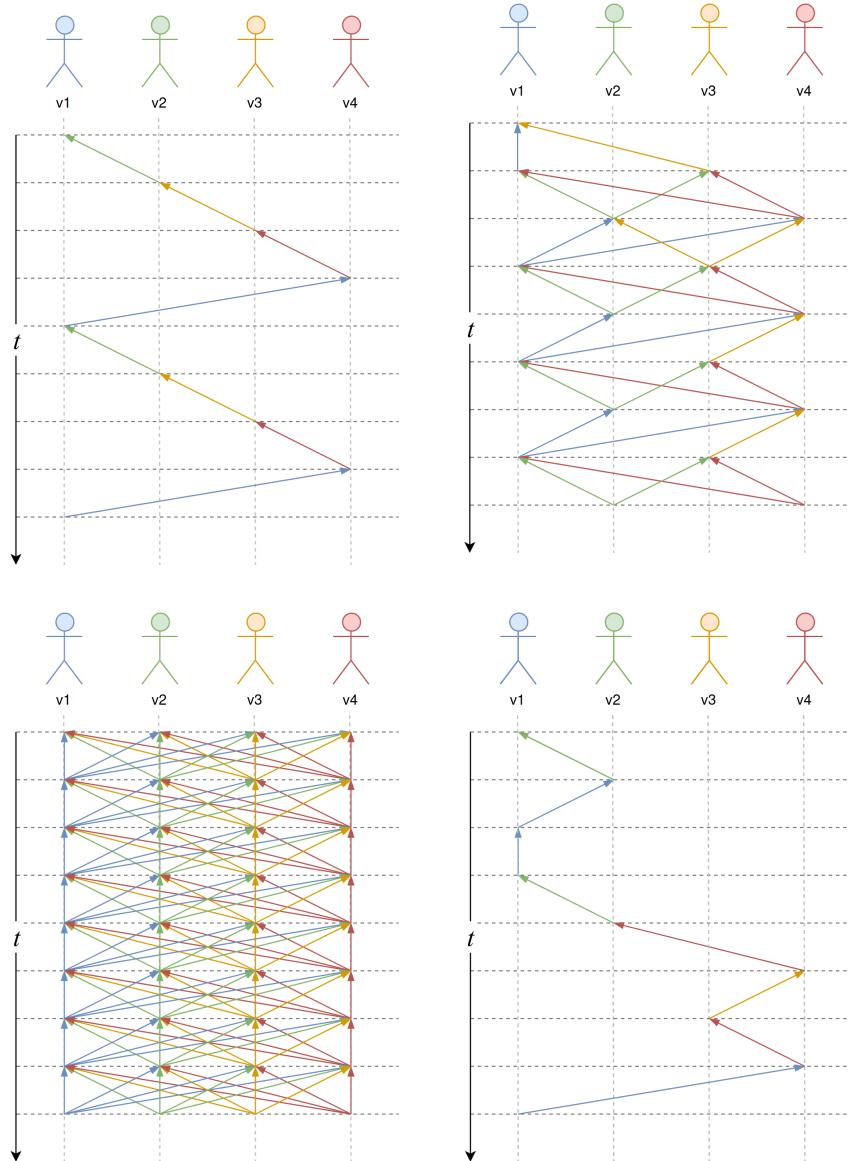


Figure 3.2 Strategies examples with 4 validators. Top-left: Round-robin. Top-right: Double round-robin. Bottom-left: Maximal overhead. Bottom-right: Arbitrary. Arrows here show justifications for all messages.

3.3 Methodology

Over the duration of this thesis, the `core-cbc` library has 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 **TODO: numbers** of blocks. At the time of the writing, the simulations do not include networking 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

This parameter is quite straightforward, it is 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 our 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. 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 *sec : strategies 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. Two strategies have been implemented for now:

- All 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 an 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.

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. **TODO: what data** This two steps processing permitted 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

4.0.1 Tests

Table ?? summarizes which tests have been run. All of them have the same terminating condition, which is reached when at least one node finds a safety oracle at height 4. The number of nodes is chosen at random in the interval [1, 20]. It has been decided to perform tests on this interval because it is large enough to give an insight on the behavior of the relations between the metrics, and small enough to reach the terminating condition in a reasonable time (1 run with a number of nodes chosen at random takes on average 1 hour with the Maximal overhead strategy).

TODO: remove table

	All receivers	Some receivers
Round-robin	✓	✓
Double round-robin	✓	✓
Maximal overhead	(✓)	(✓)
Arbitrary	✓	✓

Table 4.1 Summary of tests that have been run. A tick in parenthesis implies that tests were run but that there are not many data points or big disparities in the distribution of the number of nodes

TODO: schema with what to measure TODO: how the measurements take place in the code

4.1 Visualization I: All receivers

This chapter 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 each other. These graphs also show a simple linear regression as an attempt to find a relation between each metric. At the end of the discussion of each strategy a linear regression is performed in order to find the scores of each variable according to the model presented in Subsec. ??.

TODO: 3D plotting? TODO: undersampling issue?

4.1.1 Round-robin

The distributions of the number of nodes and the latency (Fig. ?? left and center) 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.

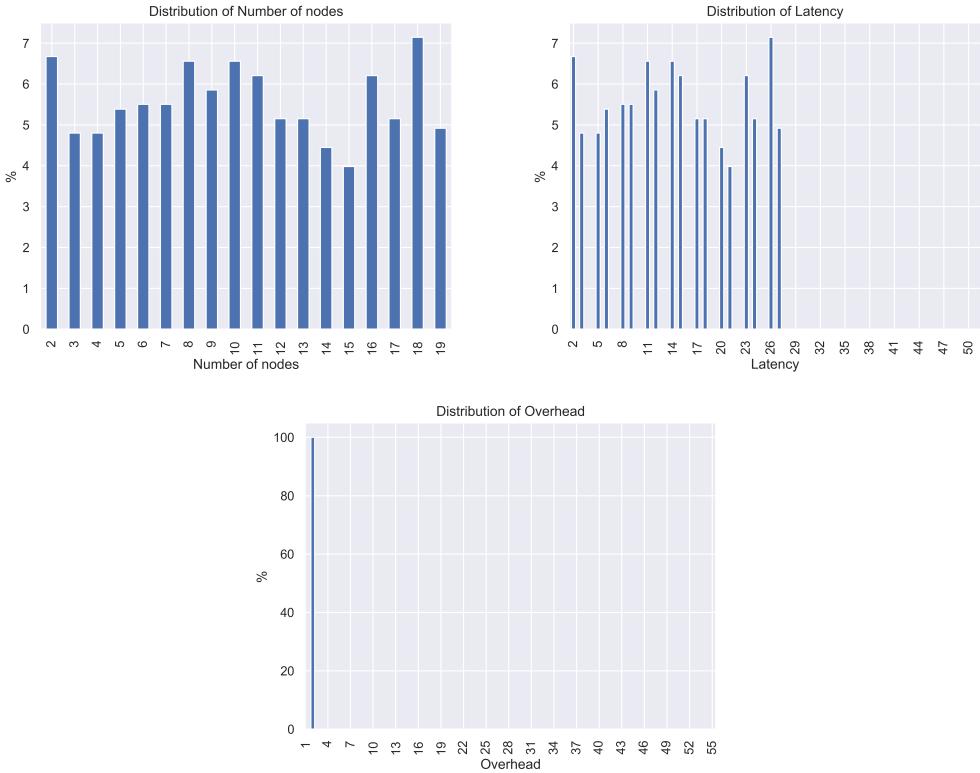


Figure 4.1 Distributions of the dataset for the round-robin strategy and all receivers. As the latency (top-right) presents a linear correlation with the number of nodes (top-left), the shapes of the histograms are similar. The overhead (bottom) always equals to 1 because only one message is sent per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

The right and center graphs on Fig. ?? 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 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. ??, the fitted line has a slightly small 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. **TODO: insert line with $1.5 \cdot n$?** The gaps in the histogram for latency showed in Fig. ?? 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.

4.1. Visualization I: All receivers

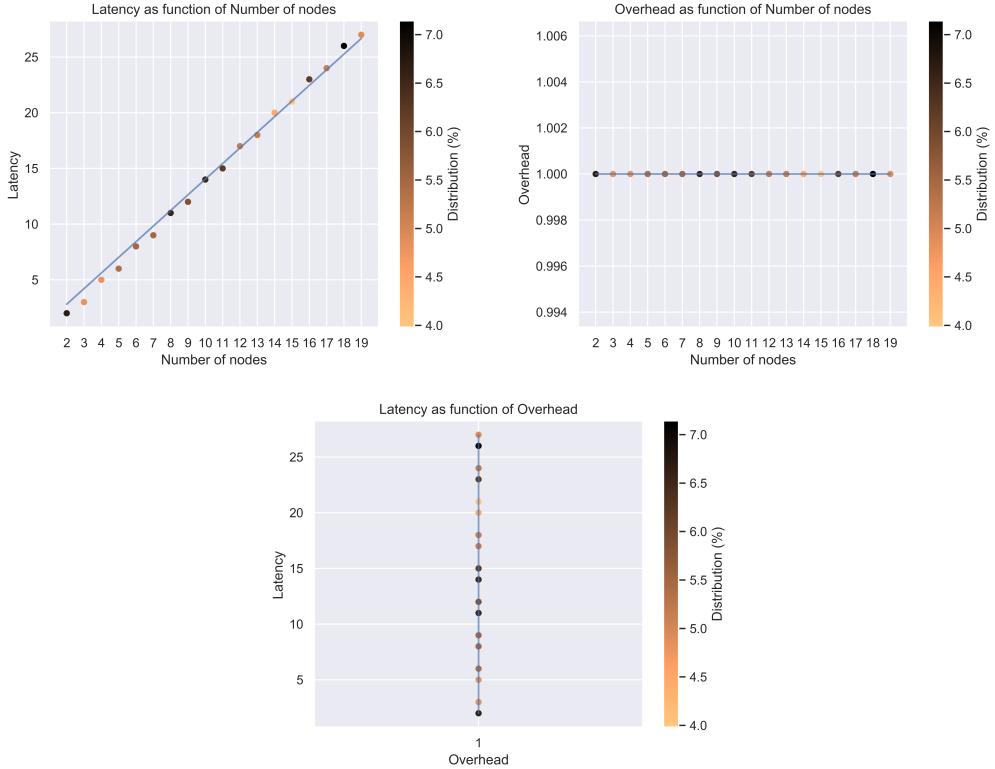


Figure 4.2 The round-robin strategy shows a linear relation between the number of nodes and the latency (top-left). The overhead is a constant for this strategy (top-right, bottom).

4.1.2 Double round-robin

As for the simple round-robin case, the latency and number of nodes distributions show a similar shape. The latency is less spread than for the singl round-robin, which explains some dissimilarities between the distributions. The distribution of the overhead presents a peak at 2, which is expected but also shows that there are some messages that have double this overhead, at 4. This will be explained using the following graphs. **TODO: explain outliers**

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. ?? shows a linear relation between the number of nodes and the latency:

$$l = 0.721750 \cdot n$$

This value is about half of the slope obtained for the same relation with the round-robin strategy. 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.

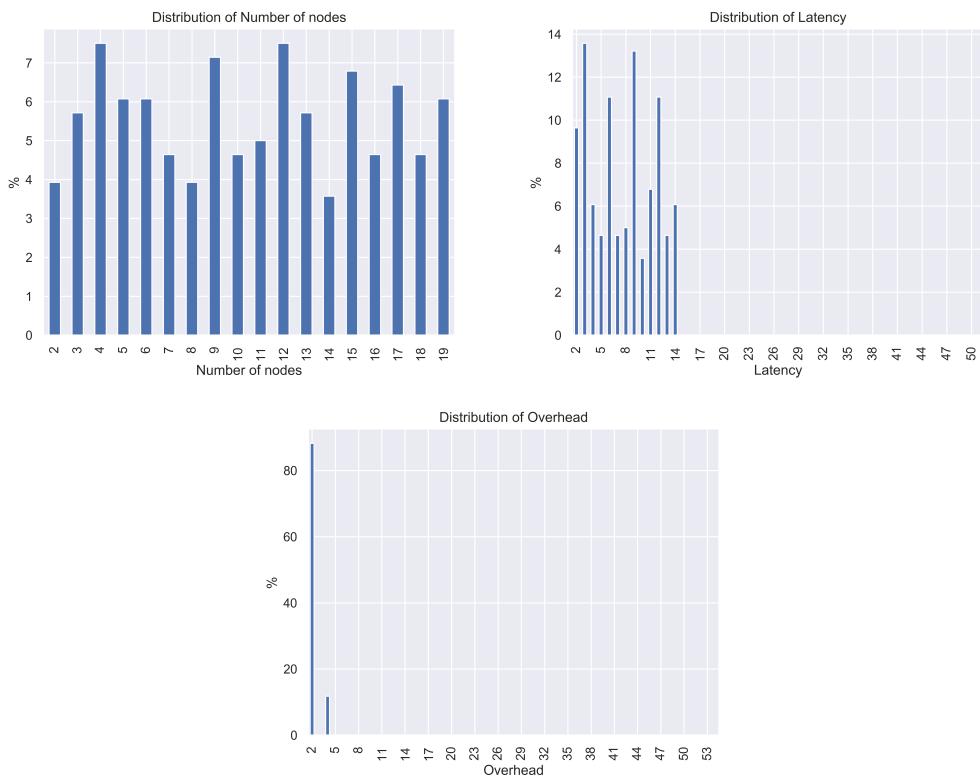


Figure 4.3 Distributions of the dataset for the double round-robin strategy and all receivers. As the latency (top-right) and the number of nodes (top-left) are linearly correlated, their distributions are of similar shapes. The overhead (bottom) equals 2, as expected and shows an outlying value at 4.

4.1. Visualization I: All receivers

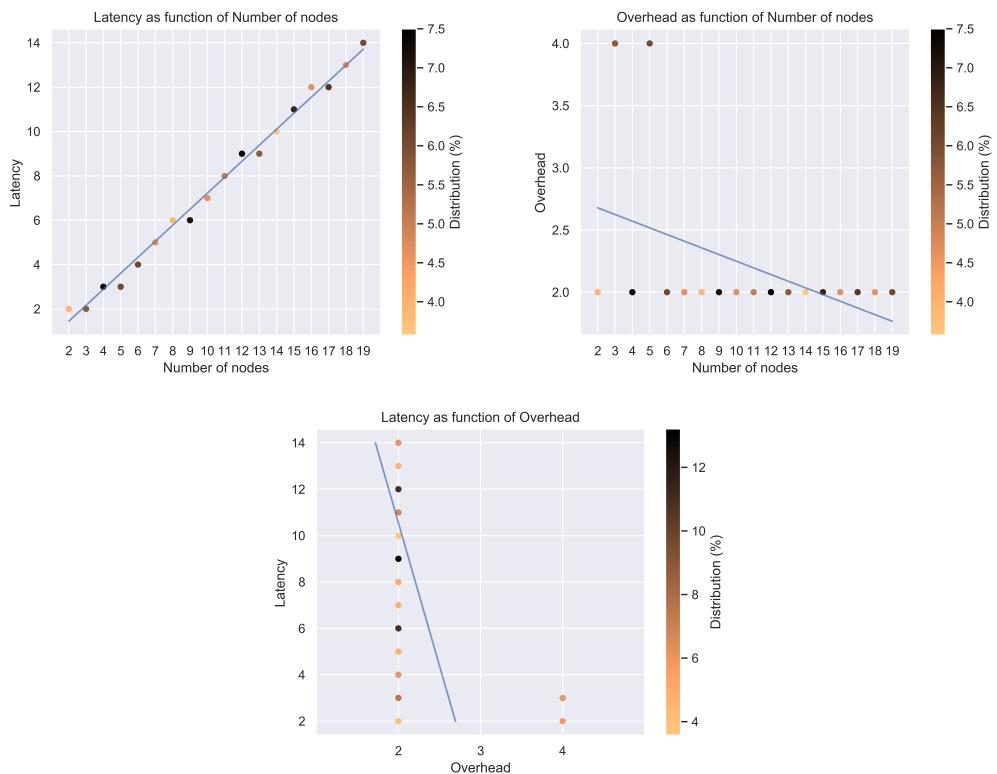


Figure 4.4 The double round-robin strategy shows a linear relation between the number of nodes and the latency (top-left). The overhead is a constant for this strategy (top-right, bottom). The top-right plot indicates that the outliers only emerge when the tests are run with 3 or 5 nodes.

4.1.3 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.

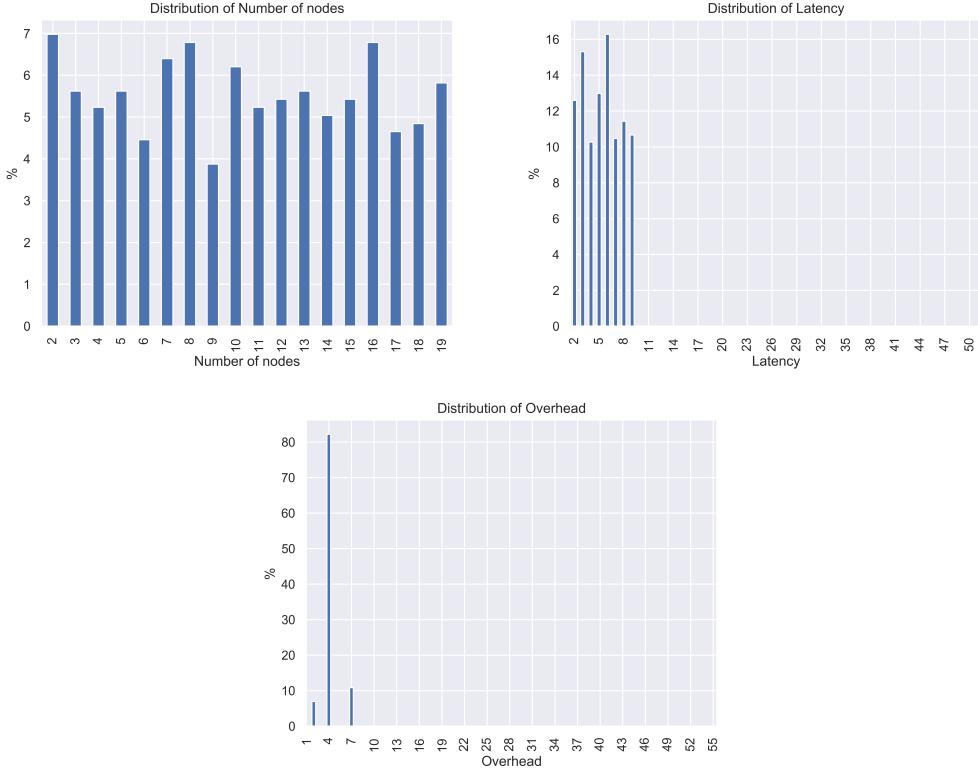


Figure 4.5 Distributions of the dataset for the triple round-robin strategy and all receivers. As the latency (top-right) and the number of nodes (top-left) are linearly correlated, their distributions are of similar shapes. The overhead (bottom) equals 3, as expected and shows two outlying values at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

$$l = 0.496152 * n$$

The slope is a third as the slope for the simple round-robin strategy. Moreover the overhead is three times larger for the triple round-robin than it is for the simple. Fig. ?? 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.

TODO: explain other outliers

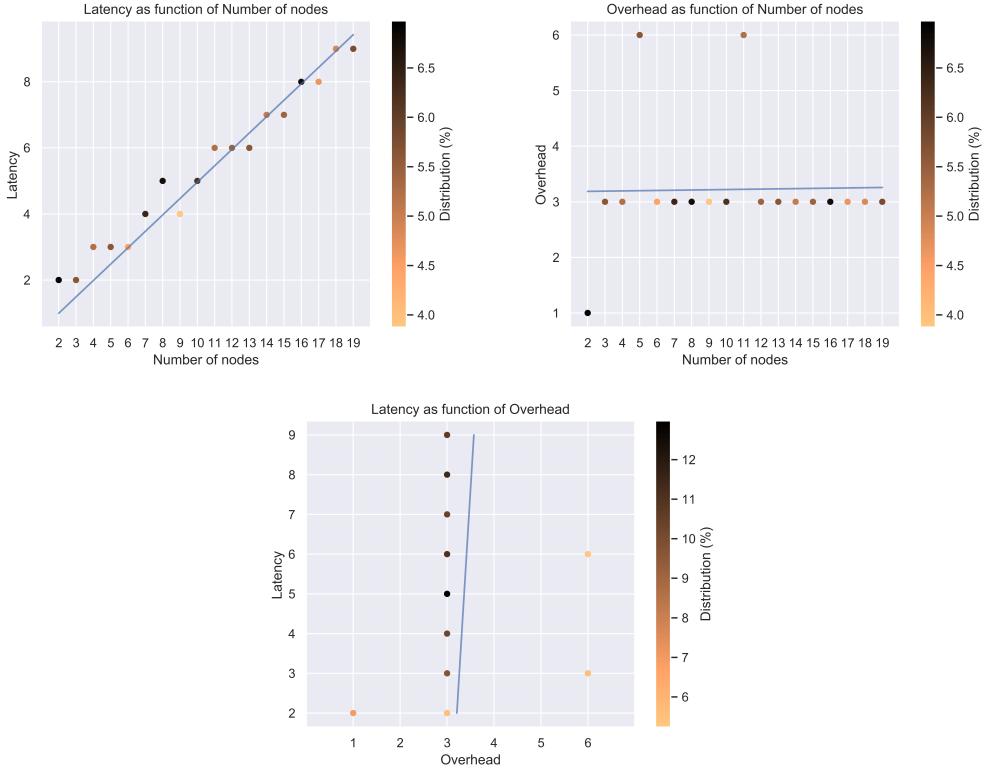


Figure 4.6 The triple round-robin strategy shows a linear relation between the number of nodes and the latency (top-left). The overhead is a constant for this strategy (top-right, bottom) except for some outliers. The top-right plot indicates that the overhead is lower for 2 nodes, which is expected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

4.1.4 Maximal overhead

This strategy aims to reduce latency to its minimum, which is shown in the top-right plot on Fig. ???. 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 have been 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 **TODO: VLADS PAPER** and not yet optimized. For completeness's sake, 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. ?? clearly shows a linear dependency between the number of nodes and the overhead. The relation is $o = n$, as it is 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.

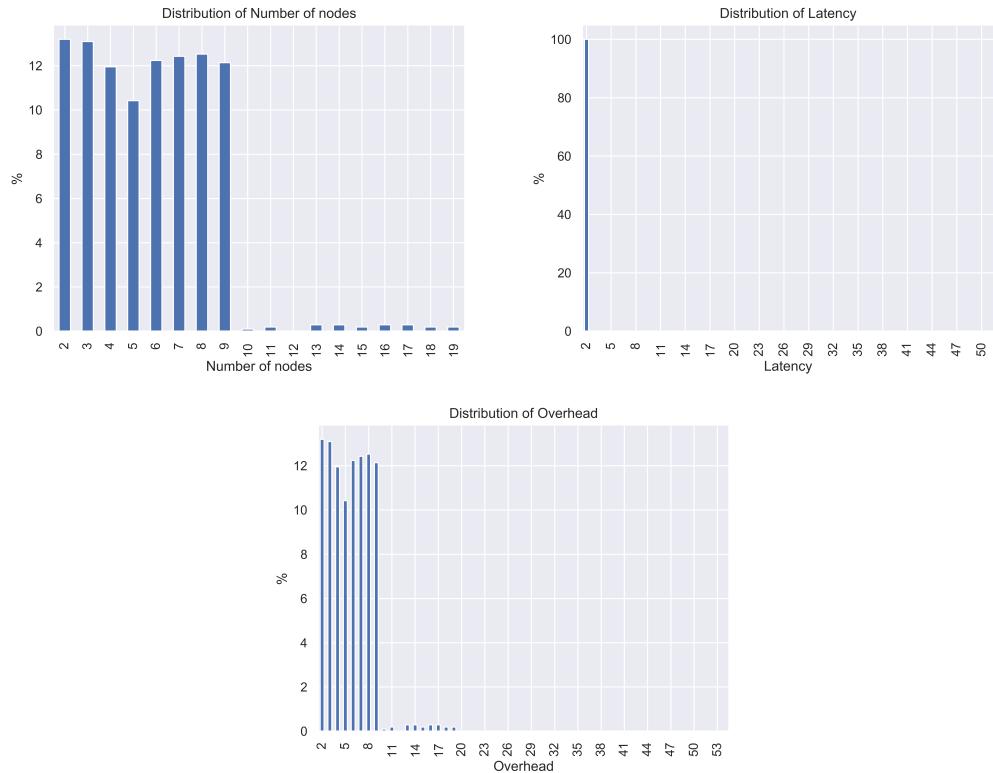


Figure 4.7 Distributions of the dataset for the maximal overhead strategy and all receivers. As the overhead (bottom) and the number of nodes (top-left) are linearly correlated, their distributions are of similar shapes. The latency (top-right) equals 2, as expected. Note that because the maximal overhead 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.

4.1. Visualization I: All receivers

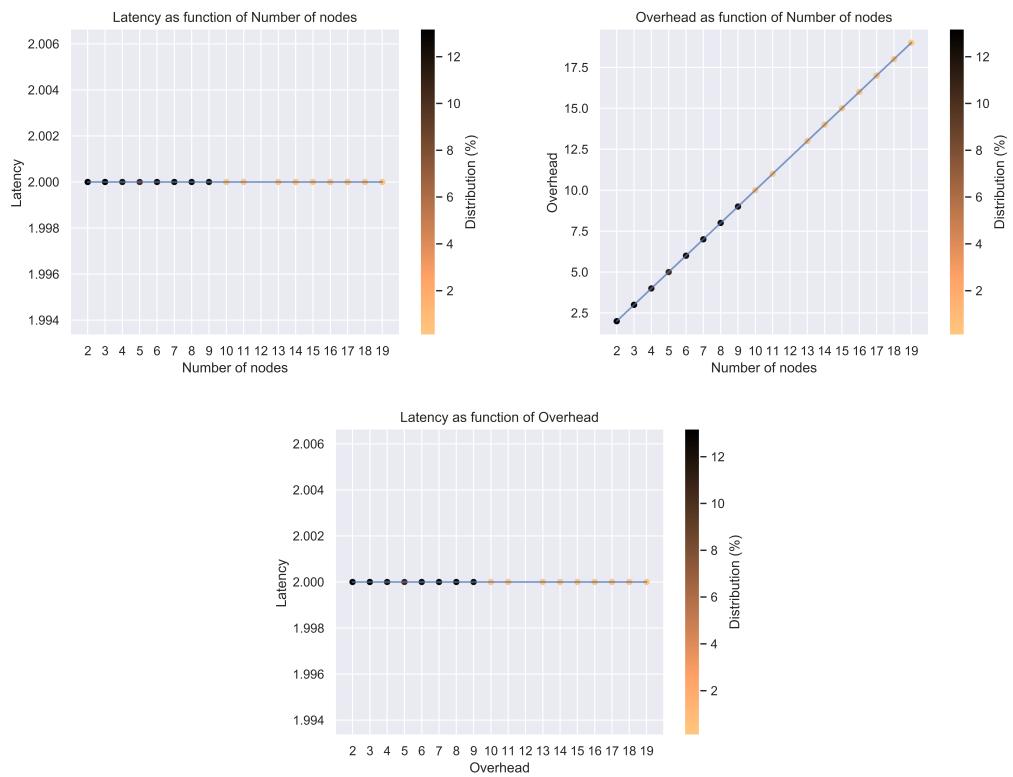


Figure 4.8 The maximal overhead strategy shows a linear relation between the number of nodes and the overhead (top-right). The latency is a constant for this strategy (top-left, bottom).

4.1.5 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 distribution **TODO: what is this shape? how to compute characteristics?**

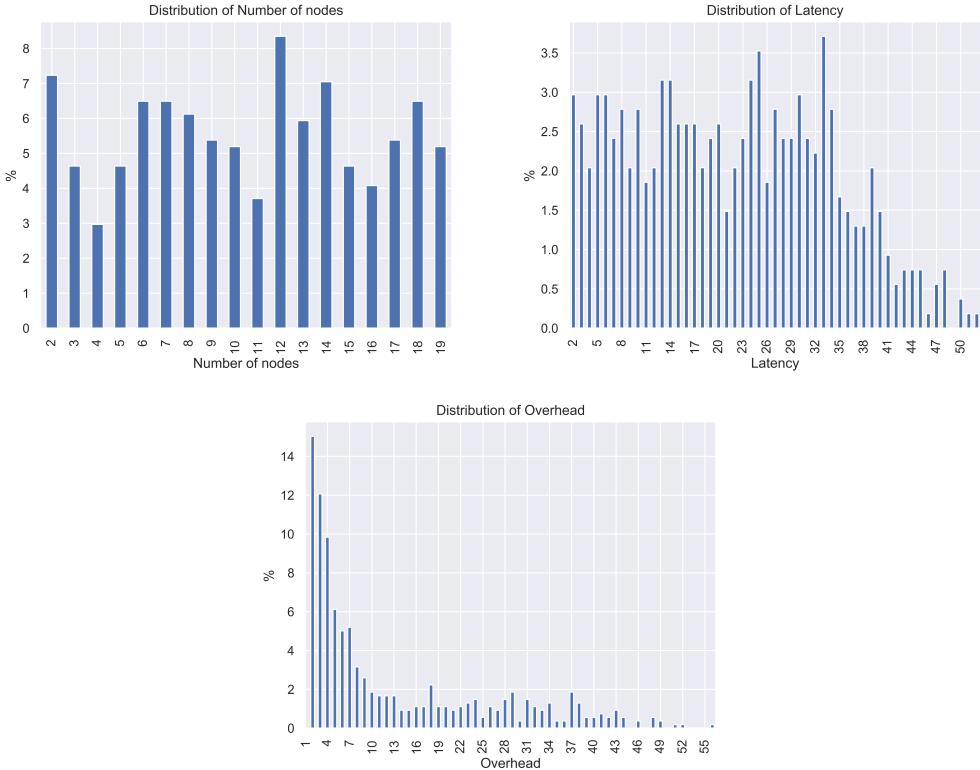


Figure 4.9 Distributions of the dataset for the arbitrary strategy and all receivers. Only the overhead seems to be distributed in a **TODO: name of distrib** manner.

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

$$l = 2.091255 \cdot n$$

. Even though it shows some variance, the slope is not that far from a simple round-robin strategy. **TODO: rephrase <-**

The two other graphs on the Figure are less straightforward to explain than the previous strategies'. The plot that pictures 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:

4.1. Visualization I: All receivers

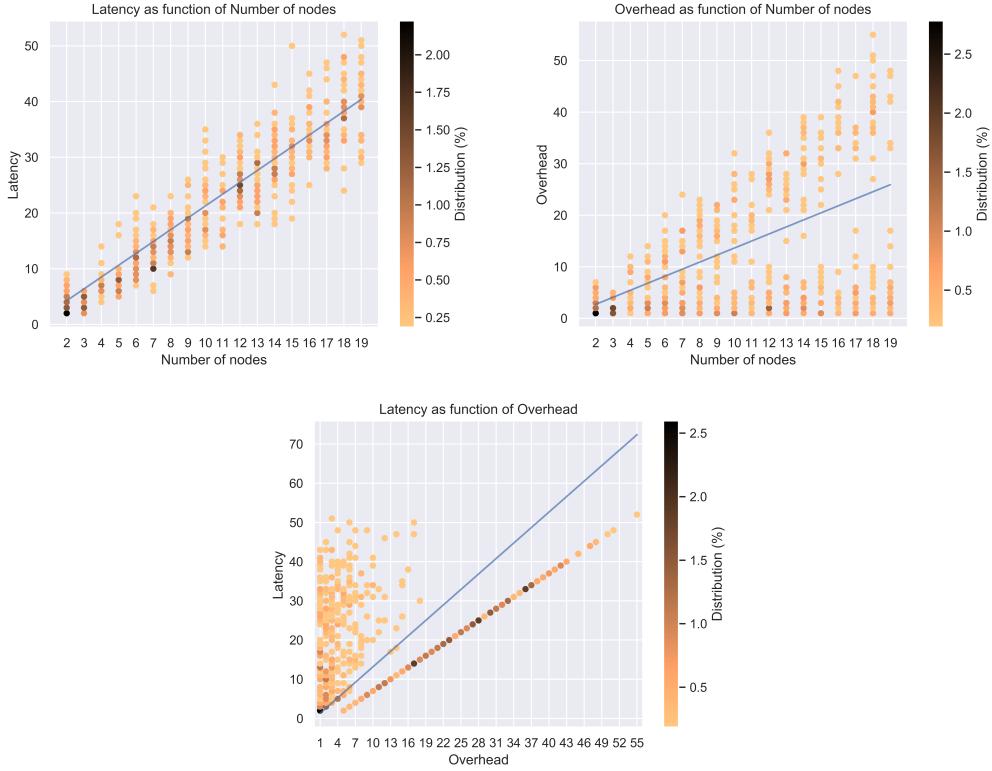


Figure 4.10 The arbitrary strategy shows a linear relation between the number of nodes and the latency (top-left). However, the other relations between variables both seem to be split into two linear functions with a clear gaps between them.

4.1.6 Fitting of the basic model

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

TODO: add error value

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

Sending Strategy	s_n	s_l	s_o
Round-robin	0.0	0.0	1.0
Double round-robin	1.435	0.055	0.162
Triple round-robin	1.686	0.084	0.091
Maximal overhead	0.0	0.5	0.0
Arbitrary	2.556	0.026	0.002

Table 4.2 Raw fitted values for the simple model, rounded to the 3rd decimal place

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 round augments and the overhead increases along with the number of nodes sending messages in a single round.

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. ??, the arbitrary strategy clearly is worse than the other ones for the latency and overhead. That is reflected with the values obtained for their respective scores. The main issue with it is that the score for the number of nodes is used as a buffer for how well the two other variables perform in a set 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.

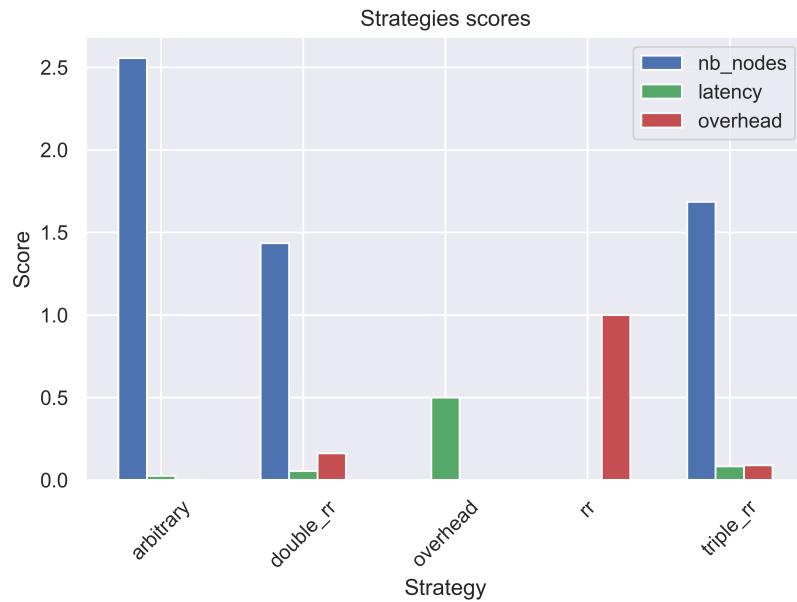


Figure 4.11 Raw fitted values for the simple model. The strategies are listed on the x-axis and their scores are plotted along the y-axis. TODO: prettify the x-axis and bar legend

4.1. Visualization I: All receivers

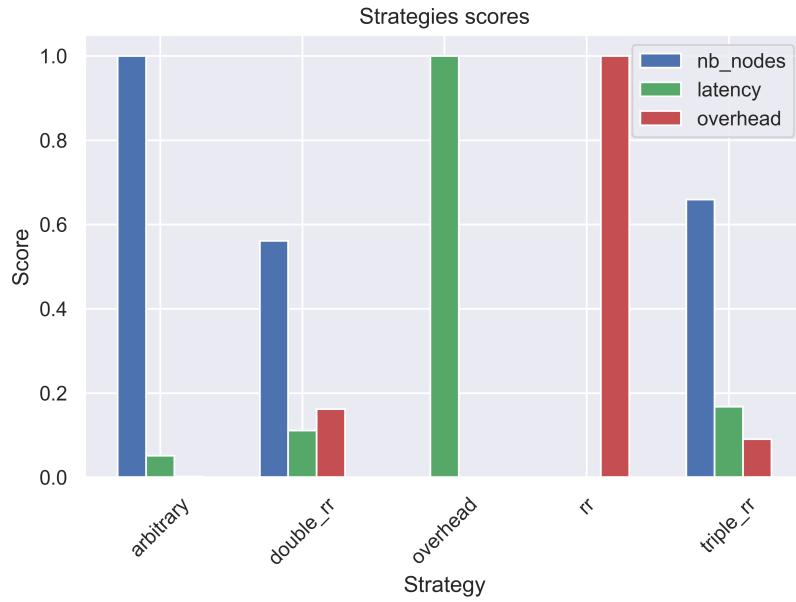


Figure 4.12 Normalized fitted values for the simple model. The strategies are listed on the x-axis and their scores are plotted along the y-axis. *TODO: prettify the x-axis and bar legend*

4.1.7 Improvements of the basic model and new fitting

Show normalized output/radar graphs Explain why changing exponents could lead to a better model

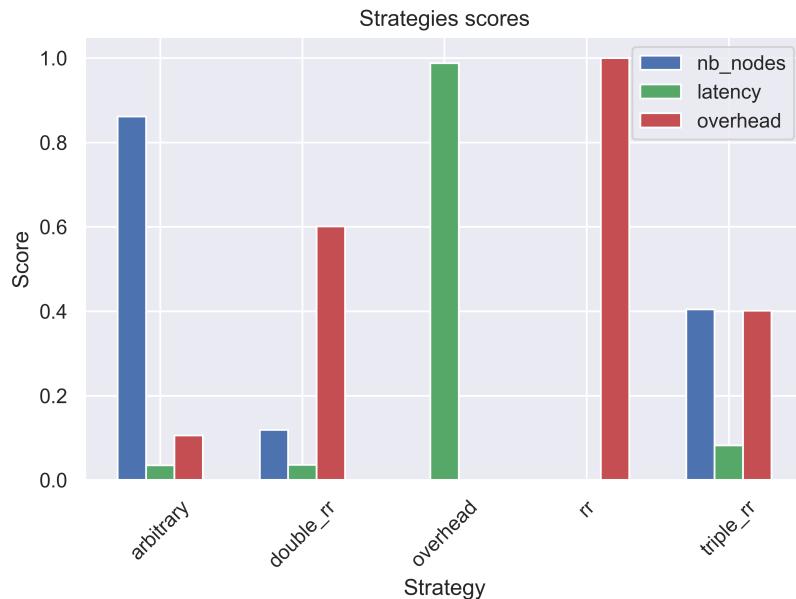


Figure 4.13 Raw fitted values for the improved model. The strategies are listed on the x-axis and their scores are plotted along the y-axis. *TODO: prettify the x-axis and bar legend*

4.2 Analysis

TODO: why the model is bad according to the number of nodes

TODO: here will be a table containing all the fitted lines values and a comparison and explanation of the behaviors that are observed

4.3 Further research

4.3.1 Half set strategy

A new sending strategy that could be interesting to implement is a strategy where 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.

4.3.2 Bottom up strategies

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

4.3.3 Optimize and simplify model

As seen in the final comparison plots TODO: section, the model might be too restrictive for the actual problem. It might be worth trying to find a better fitting model to better reproduce the real life use case. One could argue that tying the number of nodes to the rest of the variables might be too restrictive, and getting rid of the parameter could offer other insights while comparing strategies.

4.3.4 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 permitted to validate 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, ...

5 | Conclusion

working comparison framework easy to implement new strategies, new terminating conditions, ... well new test cases

however: model not that good no networking assumptions bottom up strategies to be implemented **TODO: insert conclusion**

List of Figures

List of Tables

List of Tables
