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

Neuchatel, HES-SO//Master, February 11, 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

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 PoW consensus protocol to a 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 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 consctructing 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 succintly 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.

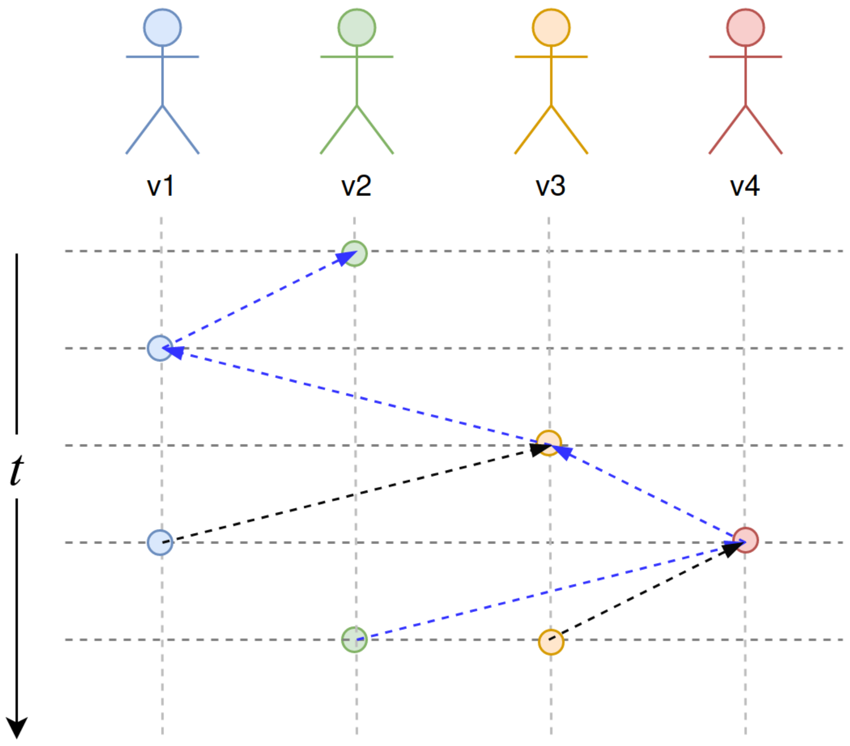

Figure 2.1 CBC blockchain example

The Figure 2.1 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 newtork, and would be the estimate of a new message sent by an honest validator that has this view.

Figure 2.2 examplifies validator v_1 producing a new message and the resulting new blockchain. An honnest 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 equivovate 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.

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

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

2.1.4 PoS vs PoW

TODO: talk about differences between pos and pow

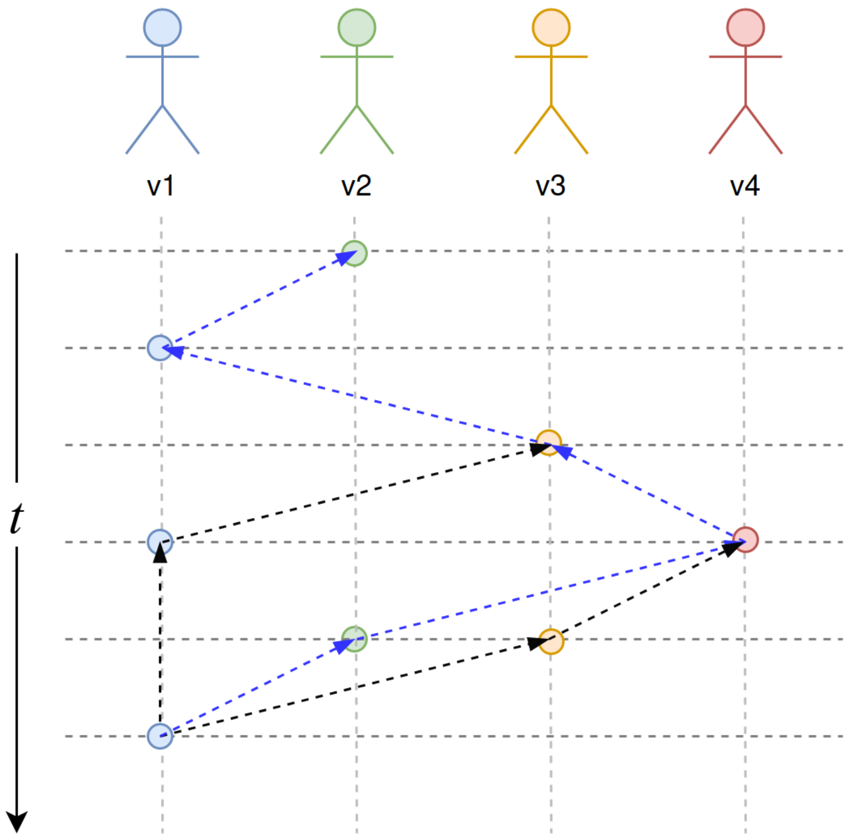

Figure 2.2 CBC blockchain example 2

	PoW	PoS
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 computationnal 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 for to build blocks	No clear incentive to build on the longest chain

Table 2.1 Summary of key differences between PoW and PoS

2.1.5 key differences

2.1.6 new problems

TODO: talk about key differences and what this project is about

2.2 CBC casper

TODO: describe the protocol in my own words

TODO: describe liveness, safety

2.3 `core_cbc`

`core_cbc` a Rust implementation of the CBC-Casper, made by TrueLevel. 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 the other casper?

2.5 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.

2.5.1 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

2.5.2 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 together. Each node has an 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`.

2.5.3 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. TODO: include schemas of fully connected and ring layouts 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.

3 Strategies Evaluation

3.1 Modelisation

3.1.1 Metrics

A way to rationaly 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 less messages are needed in order to confirm a transaction, and therefore less time. TODO: schema for latency

Number of nodes

The number of nodes is quite straightforward; it is the number of nodes that can be included in the validator set. This number should be as high as possible to guarantee decentralization and therefore safety. TODO: schema for number of nodes

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. TODO: schema for overhead

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 entierty of the triangle and

3.1.3 Model

The 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 preponderant in the strategy and therefore the closer to a corner of the triangle the strategy is. TODO: why 1/n, why everything linear?

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 entierty of the trade-off triangle:

- round-robin;
- randomness;
- double round-robin;
- overhead.

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. TODO: talk about real life implications? aka synchronize all nodes?,... and that for every strategy

3.2.2 Randomness

The next strategy is the simplest to think of: complete randomness. Using fixed probability density functions, nodes chose when to create messages and to which other validator to send them.

3.2.3 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 at opposite places in the set of validators TODO: explain better, the latency to finality is supposedly half as much as the simple round-robin strategy. The overhead is however doubled.TODO: add that we could have a triple rr, quadrr, ...

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.

3.2.5 Bottom-up strategies

TODO: talk about strategies that are obtained from a bottom-up point of view, instead of a global vision, each node decides when and why it has to send a message TODO: talk about incentives, slashing and such

3.3 Experimentations

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;
- Ending condition;
- Sender strategy;
- Receiver strategy.

Number of validators

This parameter is quite straightforward, it is the number of nodes that can validate blocks.

Ending condition

The ending 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 an height of 4. TODO: why 4

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 Section 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. 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.

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

3.4 Visualization

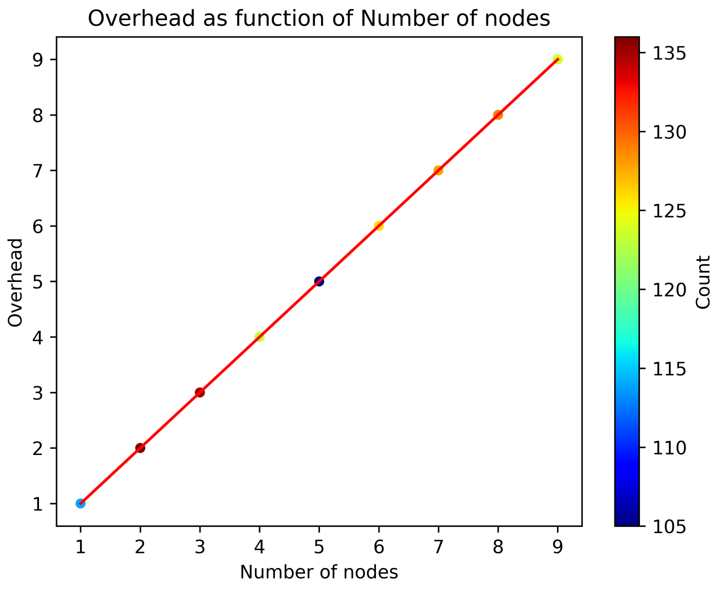

Figure 3.1 *exmaple*

4 Conclusion

TODO: insert conclusion

List of Figures

List of Tables

Glossary

CBC	Correct by Construction. 1, 3–5, 7
DAG	Directed Acyclic Graph. 3
GHOST	Greedy Heaviest Observed Sub-Tree. 3
PoS	Proof-of-Stake. vii, 1, 3, 4, 13
PoW	Proof-of-Work. vii, 1, 3, 4, 13