NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

RICARDO FILIPE MENDES LOUREIRO

Masters in Computer Science and Engineering

REASONING ABOUT CONSENSUS PROTOCOLS:

SIMULATION AND VALIDATION ENVIRONMENT

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
Jully, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

REASONING ABOUT CONSENSUS PROTOCOLS:

SIMULATION AND VALIDATION ENVIRONMENT

RICARDO FILIPE MENDES LOUREIRO

Masters in Computer Science and Engineering

Adviser: António Ravara
Associate Professor, NOVA University Lisbon

Co-adviser: Simão Melo de Sousa
Associate Professor, University of Beira Interior

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
Jully, 2023

Abstract

The number of services and applications that require and rely on transactional, replicated and verifiable data to function is increasing with each passing day, from banking and financial applications to online voting. With these requirements also come challenges, like availability, consistency, and security mechanisms that allow for integrity, non-repudiation and encryption of messages.

A common solution that these applications use to satisfy these requirements is Distributed Ledger Technologies or DLT. These systems are characterized as having a decentralized database, consensus mechanisms to validate transactions and immutable data once verified. But with the wide range of different requirements come a wide range protocols. These require testing, validation, and inevitably come vulnerabilities or errors that need to be corrected. Since the use of DLTs is recently new there is a lack of tools for the testing and validation of these protocols, which means that problems with the logic of the algorithm or vulnerabilities are often discovered in live applications.

One of these tools is MOBS(meter referência), which stands for Modular Blockchain Simulator, a simulator built with the extensibility and modularity in mind, allowing users to simulate any family of protocols as well as parametrize multiple scenarios for their study. These parameterizations include bandwidth limits, byzantine behavior of the participant nodes and adversarial behavior. After the execution the desired statistics and information needed to validate the execution of the protocols can also be parametrized.

The goal of this dissertation is twofold we propose an extension to this tool to better allow the simulator to provide different sets of execution environments by allowing parameterization of the network layer of the simulator. This will allow for the definition of membership protocols in which this layer will be built, either with structured or unstructured overlays or with the help network optimization algorithms, we propose Chord, HyParView and X-BOT respectively. We also propose the implementation of tools that will allow us to validate the correctness of the implemented protocols through the logs of the simulator.

Keywords: Distributed Ledger Technology, Blockchain

Resumo

O número de serviços e aplicações que necessitam e utilizam dados replicados, verificáveis e transacionais para funcionarem estão a aumentar com cada dia que passa, desde aplicações bancárias e financeiros a sistemas de voto online. Com estes requisitos também vêm desafios, como disponibilidade, consistência dos dados e mecanismos de segurança que permitam a integridade, a não repudiação e a encriptação de mensagens.

Uma solução comum que estas aplicações usam para satisfazer estes requisitos são Tecnologias de Registos Distribuídos ou TRD. Os sistemas de TRD são caracterizados por terem uma base de dados descentralizada, mecanismos de consensos para validar transações e dados imutáveis após verificados. Mas com a vasta diversidade de requisitos vem uma quantidade diversa de protocolos. Estes necessitam de ser testados, validados e inevitavelmente corrigir os erros de lógica e as vulnerabilidades descobertas 'a posteriori'. Como o uso de TRDs é relativamente recente, há uma falta de ferramentas para testar e validar estes protocolos, o que implica que problemas de lógica ou vulnerabilidades são muitas vezes descobertos depois das aplicações que os utilizam serem lançadas.

Uma destas ferramentas é o MOBS(referencia), Modular Blockchain Simulator, este simulador foi construído com a extensibilidade e modularidade em mente, permitindo os utilizadores simular qualquer família de protocolos tal como parametrizar múltiplos cenários para o estudo destes. Estas parametrizações incluem limites de bandwidth, comportamentos bizantinos dos nós participantes e comportamento adversarial. Após a execução as estatísticas escolhidas e informações necessárias para a validação da execução destes protocolos também pode ser parametrizada e estendida.

Neste documento propomos uma extensão desta ferramenta para melhor simular e fornecer diferentes conjuntos de ambientes de execução permitindo a parametrização da camada de rede do simulador. Isto vai permitir a definição de protocolos onde esta camada vai ser construida, tanto com uma rede estruturada ou não estruturada, ou permitindo à rede a execução de algoritmos de otimização para a mesma, permitindo assim o estudo do desempenho, a correção destes protocolos num ambiente dinâmico, em diferentes camadas de rede que vão independentemente responder a comportamentos bizantinos ou mudanças na camada de rede.

Palavras-chave: Tecnologia de Registo Distribuído

Contents

Appendices

Annexes

List of Figures

List of Tables

1

Introduction

This thesis aims to address the practical challenges in designing, implementing and maintaining Distributed Ledger Technologies protocols by providing a simulation environment so that results can be extracted and the behavior of the protocols can be verified before going into a live environment. This thesis involves taking MOBs and expanding the previously done work to help better test and validate these protocols by providing a wider range of networks and a more dynamic and independent network layer.

1.1 Context

With each passing day the amount of distributed application is increasing, and a subset of these are applications that deal with data that requires validation, support transactional operations by validated users and simultaneous access for updating and consulting the records. For this specific subset there are a group of protocols that have been created to meet and ensure these requirements. Distributed Ledger Technologies allow for simultaneous access, validation and update of records across a distributed database, each node has its own copy of the ledger that it uses to validate information and reach a consensus about its accuracy.

Around 2008 began the rise of a new set of protocols that takes DLTs as their basis. Blockchain protocols appeared with the motivation to serve as a distributed ledger for cryptocurrency transactions. The main difference from Blockchains to DLTs is that the log of records is created in blocks, each blocked is closed by a hash and the next beginning with the closing hash of the previous.

These protocols are not static, being because vulnerabilities or flaws need to be correct or due to the very nature of the protocol itself. One of these dynamic protocols is Tezos(reference miguels thesis), which relies on the stake-holders that participate in the system to propose and agree on changes and upgrades to the protocol. Another example is Ethereum(reference see miguels thesis) that uses a blockchain protocol based on proof of work and their current goal is to migrate towards an implementation based solely on proof of stake for Ethereum2 (reference see miguels thesis). This opens a necessity for

tools that aid in support these evolutions in a faster, more seamless, secure fashion.

Blockchain protocols operate on top of membership layers that dictate how the topology of the network is configured. Different membership environments come with different properties and trade-offs. Structured membership overlays allow for faster lookups for specific nodes and a pre-defined and predictable structure to the network. Non-structured membership offer a more resilient overlay when new nodes are introduced or existing ones leave, albeit by choice or failure. And overlays that operate by building a partially structured overlays allow for the benefits of a non-structured overlay at the cost of slower re-structuration since optimization procedures are regularly executed to improve routing and lookup operations. The trade-offs and some of these protocols will be further explained in Chapter 2.

This motivated the development of MOBs, a modular and extensible simulator that provides the ability to simulate different families of protocols, parameterizable, a well-defined structure and a qualitative evaluation for the study of implemented protocols. But this simulator can be further improved by giving conditions closer to real life, like a dynamic and parameterizable network layer, allowing for the study and validation of the behavior of these protocols in different scenarios and abstracted from the intricacies of the arrangement of the participants.

1.2 Problem

Consensus protocols are not trivial to define or understand correctly and in blockchain systems, where the behavior is dynamic and mostly financial transactions are dealt with, their correctness is crucial and errors can be costly. To help with this MOBs was developed giving the ability to test and validate these protocols under different conditions and settings by changing the execution parameters.

Right now the parameters to the network are set before the execution of the simulation, and regarding the network behavior we can set the network topology, how many nodes will fail and when. In the real world a network's topology is dynamic, new nodes can enter as new participants, existing ones can leave, either by choice or by failure, and even when the participants are static the network can suffer changes to its topology as a result of optimizations performed by this layer.

The simulation of how this layer behaves and the parametrization of the different protocols that can be used as its basis are the problems that we aim to address with this thesis.

1.3 Goal

With the different properties that different network overlays provide besides the parametrizations that are already provided by MOBs, we can further provide an even more modular

simulator that allows for the study of the optimal conditions of executions of the protocols. We can also leverage the modularity of this layer to better simulate the behavior of new participants coming into the system, existing ones leaving and how changes to the topology of the network affects their execution.

The goal of this thesis is to improve the networking layer of MOBs by making it more modular and therefore giving the following advantages:

- Different environments for more diverse testing scenarios.
- Stronger parametrization regarding the behavior of the network.
- Better qualitative data by providing scenarios that better mimic real world execution.

1.4 Document Organization

The remainder of this document is organized in the following manner:

Chapter 2 studies related work: in particular we will cover several membership protocols and some implementations; Babel which is a framework for implemented distributed system protocols;

Chapter 3 describes in more detail the work that has already been conducted and the proposed work, including an evaluation and work plan

2

Related work

In this chapter we will discuss relevant work considering the goals of the work to be conducted in this thesis. In particular, we will focus on the following topics:

In section 2.1 we discuss the differences between the different kinds of membership systems as well as some implementations.

In section 2.2 we will discuss Babel(insert reference), a framework to develop distributed protocols.

In section 2.3 we will present and compare some protocol analysis tools and compare them to MOBs.

2.1 Membership Protocols

This section introduces some membership protocols that have been considered to part of the work that will be developed.

2.1.1 Membership

A protocol where each node knows every other node in the network might work well for small networks, but it is not a scalable solution since each node would need to have *n - 1* communications channels open at all time, being n the amount of nodes in the system, and each node needs to be following any and all changes to the system. This is not feasible since the number of links between nodes would rise quadratically and networks can easily reach thousands of participants. In order to overcome the challenges that arise from large networks, we use partial view membership protocols. In these protocols each node only knows and maintains information about a small selection of nodes in the systems, making it a more scalable strategy than a total view protocol, since the number of connections grow at a logarithmic rate instead.

When maintaining a partial view, protocols usually follow one of two strategies when managing their memberships:

- **Reactive:** With this strategy the partial view only changes when the membership suffers changes, usually by a node leaving or joining the membership

- **Cyclic:** With this approach, the partial view changes periodically. Every t seconds nodes exchange information that may lead to changes to the partial view.

This membership protocols may be represented by a graph where the vertices represent the participants of the networks and the edges represent the communication links. Depending on how the graph ends up we can classify the membership as structured, unstructured or partially structured. In the next sections we will go over these types of memberships and some implementations.

2.1.2 Structured Overlays

Structured overlay memberships follow a pre-defined structure by having the nodes routed to a specific logical position in the network. This known structure allows improvement on search primitives, enabling efficient discovery of data and process. This position is usually determined by giving a unique identifier for each node of the network.

However, having a predefined topology come at the cost of more costly re-structuring usually by a node leaving the network or a new participant joining. This process is usually slow and costly since a change of one node in the membership may affect the network at a global level. This drawback becomes more relevant in high churn rate scenarios.

2.1.2.1 Chord

Consistent Hashing and Random Trees, or Chord(referencia) is a Distributed Hash Table based algorithm where each node is assigned a unique identifier based on the output of a hash function. Usually this has is based on the IP address of the node since this is unique for every node of the network, and making it so that once a node joins the network their identifier is set in stone. These identifiers are also so that each node knows which keys of the distributed hash table are assigned to them, since the range of keys that belong to a certain node is represented by the range defined by their identifier up to the identifier of the next node.

The graph generated based on the membership overlay will be a cyclic graph with a ring shape, with the nodes ordered by their generated identifier. Each node also maintains a routing table with around O(log n) distinct entries called a finger table. This routing table is used in order to navigate the overlay in a more efficient manner.

The tables maintained by these nodes are automatically updated when a new node joins or leaves the system making it always possible to find a node responsible to a given key. However, simultaneous failures may break the overlay since the correctness of the membership depends on each node knowing their correct neighbors. In order to increase fault tolerance, a periodical stabilization protocol is run in background, making sure the neighbors are still active and restructuring the overlay accordingly as well as updating the entries on the finger table.

2.1.2.2 Pastry

Pastry(insert reference), similarly to Chord is a structured overlay protocol that defines a distributed hash table. A Pastry system is a self-organizing overlay network of nodes where every node has a 128-bit identifier. This identifier is assigned randomly when a node joins the system and is used to indicate a node's position in a circular space that ranges from 0 to $2^{128}-1$. It is assumed that the hash function that gives the node's identifier generated a uniform distribution of identifiers around the 128-bit space.

Each Pastry node maintains a *routing table*, a *neighborhood set* and a *leafing set*. The routing table contains the entries based on substrings of the own node's identifier. The neighborhood set contains the node identifiers and the corresponding IP address of the M the closest nodes to the local node, M being parameterizable. The neighborhood set contains the nodes that are used to maintain local properties. The leafing set contains the $L/2$ closest large node identifiers and $L/2$ smallest node identifiers, L also being a parametrized value. The leaf set is used in message routing.

Pastry makes use these tables in the following fashion: when a message is received for another node in the network we check the leaf set, if the node is in range of the leaf set we route directly to the destination node, otherwise we check the routing table for a node that shares a common prefix with the key.

2.1.3 Unstructured Overlays

Unstructured overlays place few constrains as on how the nodes chose their neighbors. This results in a random graph that is hard to predict and describe.

By not having a structured overlay these memberships end up being more fault-tolerant in high churn rate scenarios due to the cost of the network restructuring itself is fairly low.

2.1.3.1 SWIM

SWIM(referencia) stands for **S**calable **W**eakly-consistent **I**nfection-style Process Group **M**embership Protocol. This protocol is composed of two distinct components, a failure detector and a dissemination component.

Unlike traditional gossip based overlays that rely on a heartbeat strategy, the failure detector in SWIM is independent of the rest of the protocol. The failure detector is fully decentralized and is executed in a randomized probe-based fashion, the authors later suggest an optimization via a round-robin fashion instead. Every t seconds, parameterizable value, a random node that belongs to the membership is checked to see if its still alive, if no awnser is received the request is forwarded to k other members to execute an indirect probe. If the nodes responds to at least one of the indirect requests the suspicion is removed and an *alive* message is spread to fasten the process in case other nodes suspect from it. This mechanism reduces The number of false positives in failure detection, which can usually be due to a temporary unavailability from a network partition or from a

slow-down of a node due to a high load of requests. As a drawback failure detection is slower.

The other component of this protocol is the gossip dissemination system which maintains a partial view of the network. This component updates whenever a member joins or leaves the system by an infection style dissemination protocol. In order to make this more efficient, the updates piggyback the messages that are sent during the failure detection procedure.

2.1.3.2 HyParView

HyParView (referencia) is a gossip based membership protocol that offers high resilience and high delivery reliability of messages while being highly scalable. It relies on a hybrid approach by maintaining, besides the active view that is used by the nodes to maintain the communication overlay, a passive view, in case the networks needs to be restructured.

HyParView uses different approaches when it comes to maintaining each of the views, for active view a reactive strategy is used, nodes react to events that require the network to be restructured such as new nodes joining the membership or existing ones leaving, either by failing or by choice.

The nodes in the active view are the ones with which each node maintains a communication link. In case the active view needs to be changed the nodes in the passive view may be promoted to active nodes the same way nodes in the active view may be demoted to the passive view. All the nodes in the active view of a given node have that node in their active view, making the connection graph that represents the overlay be a bidirectional graph.

For the passive view, HyParView uses a cyclic strategy by periodically exchanging information with other nodes regarding the contents of its views. This information is exchanged by means of a shuffle operation with another node from its active view. After the exchange, the nodes integrate the new received members in its passive views, even if for that, some older members have to be dropped. This mechanism helps maintaining a homogeneous overlay in the number of connections each node has.

The gossip strategy relies on the use of TCP as a reliable transport protocol and fail detector. Since all members are tested at each step of the gossip protocol, failed nodes do not stay unnoticed for long. This way, HyParView provides with fast self-healing properties that are characteristic of gossip protocols.

Tests done in (referencia hyparview) show that the algorithm is able to recover from as much as 80% node failure, as long as the overlay stays connected. By being able quickly react to failures in the system, the protocol was shown to be able to maintain 100% reliability for message dissemination.

2.1.4 Partially Structured Overlays

Partially structured overlays aim to get the best of both strategies. We can leverage the easy to maintain and fault-tolerant unstructured overlays and by applying some optimization procedure to the network we can achieve a more efficient search and application level routing.

2.1.4.1 T-MAN

T-MAN (referencia) was created with the motivation to give the ability of taking some random overlay, and *evolve* it into another one.

The logic behind the algorithm is giving each node a ranking value that every node can use to apply a function to determine how desirable a node is as a neighbor. Each node maintains a partial view that contains the addresses of nodes that are not its immediate neighbors, much like the HyParView overlay described in 2.1.3.2. Periodically each node exchanges its partial view with the first node in its active view, according to the ranking values which depends on the target overlay. The receiver will execute the same procedure as the sender so they can later merge their local views and apply the ranking function. Using their peers' views to improve their own the overlay will gradually become closer the desired overlay.

Experimentally this algorithm is shown to be scalable and fast, with the convergence times growing approximately at a logarithmic rate in function of the number of nodes in the network. The problem that surges with this, is that the network only becomes as fault-tolerant as the desired overlay, since T-MAN doesn't aim to maintain a balanced degree between the nodes, this might create uneven load balance or even node isolation during or after the procedure

2.1.4.2 X-BOT

X-BOT (referencia) stands for **B**ias the **O**verlay **T**opology according to some targeting criteria **X**. This protocol is completely decentralized, and the nodes do not require any prior knowledge of where they will end up in the final topology. This protocol strives to preserve the degree of the nodes that participate in the 4-node coordinated optimization technique, described in greater detail below, this is essential to preserve the connectivity of the overlay. X-BOT is built in a way that every modification that is done by the protocol increases its efficiency and due to the dynamic nature of the model, its ensured that the overlay does not stabilize in a local minimum. These optimizations are done in a way that key features of the overlay, such as low clustering coefficient and low overlay diameter, are preserved. The protocol is highly flexible because it relies on a companion oracle to estimate the link cost and therefore bias the network according to different cost metrics.

The companion oracle is accessible by all nodes, and its sole purpose is to give the link cost from the node that invokes it to a given node.

The 4-node coordinated technique that has been referred above works as follow, a node *i*, the initiator, starts the optimization round selecting a node from its passive view. Node *O* is a node from *i*'s active view that will be replaced. Node *c*, the candidate, is a node from *i*'s passive view that is going to be upgraded to its active view. And finally node *d* is the node to be removed from the candidate's view so that *i* can be accepted. These nodes are always selected based upon the link cost values provided by the oracle, ensuring that every time the optimization procedure is called the network increases its efficiency

2.2 Babel

In this section we will discuss Babel (referência), a framework that simplifies the development of distributed protocols within a process that executes in real hardware. Babel simplifies the development process by abstracting the developer from the low level complexities associated with typical distributed systems implementations like interactions with other protocols, communication details, handling timers and concurrency control. The communication comlpexities are hidden behind abstractions called *channels* that can be extended by the developer. The Babel framework is composed by three main components:

- **Protocols:** This is the component the developer implements encoding the behavior of the distributed system being designed. Protocols are modeled as state machines whose state changes by action of external events, for this purpose each protocol contains an event queue that keeps track of these events.
- **Core:** This is the central component that coordinates the execution of all protocols and mediates all interactions within the framework. Core also keeps track of timers setup by the protocols and delivers an event to the correct protocol when the timer is up.
- **Network:** This component handle the channel abstraction that was mentioned previously. Protocols can interact with channels by opening and closing connections and sending messages as well as receiving relevant events for the protocol.

Babel is provided as a java library and the developer can implement a protocol by extending the abstract class *GenericProtocol* that provides the needed methods to generate events, register callbacks to process received events and the initial behavior for a node for the protocol that is being implemented. The provided API can be split into three categories:

- **Timers:** This allows for the execution of periodic actions.
- **Inter-protocol communication:** Babel supports multiple protocols executing concurrently in the same process so mechanisms for these protocols to interact with each other exist. These mechanisms take the form of requests and notifications.

- **Networking:** Babel offers different network channels with different capabilities but keeping the same model of interactions between protocols and the different channel implementations.

Evaluation testing in (referencia) show that Babel can reduce the effort for programmers in creating prototypes of distributed protocols, comparing the number of lines programmed using Babel and standalone versions of the same protocols in java. This is achieved mostly by the logic that is covered in the network and core layers, relieving the developer from having to worry about handling timers, message ordering and most communication besides events.

Regarding performance tests conducted by implementing MultiPaxos (referencia no artigo do babel) with the framework when comparing to standalone implementations, the implementation using Babel showed comparable performance with less implementation overhead.

2.3 Protocol Analysis Tools

3

MOBs

3.1 Overview

MOBs standing for Modular Blockchain Simulator is divided into two main components, the simulator a and the graphical user interface, we will look into them seperatly.

3.1.1 Simulator

The simulator makes use of OCaml's *modules* and *functors* to provide modularity and extensibility. MOBs adopts a *Discrete-Event Simulation Model* making the state of the system only change in discrete points in time when events occur. These events are stored in a queue ordered with two main values:

- **Timestamp:** This value dictates the order in which the events are stored in the queue and is based on the simulator's internal clock. When getting a new event the simulator will fetch from the queue the one with the smallest timestamp and move its internal clock to match that event's timestamp.

- **Target:** The entity that should process this event.

Events are fetched from the queue until no more events remain or a predefined stopping condition is reached.

The simulator is built in a module based architecture, Figure (referencia) illustrates how the different modules interact. These modules are:

- **Main:** Entry point for the simulator, manages the execution of protocols.

- **Protocol:** Top-level loop of the simulation, initializes the different nodes, network topology, event queue and performs event handling and delegation.

- **Node:** This is a user defined module that describes the behaviour of an entity in the simulated protocol.

- **Abstractions:** This module provides primitives to aid in the development of new simulation such as proof of stake sortation, proof of work mining, timers, alarms and message exchanges.

- **Statistics and Logging:** Extract metrics and values from the execution to be processed by the GUI.

3.1.2 Graphical User Interface

The graphical user interface was implemented in NodeJs, Vue3 and ElectronJS. The choice for web technologies enables the future deployement of simulator as a web application. The GUI was developed with the goal of allowing the users to use it with their own custom simulators as long as the following conditions are met:

1. The simulator uses a parameters.json as an input with three categories, General, Network and Protocol, the actual parameters inside each category are user defined.

2. The output of the simulator produces log with two top-level entries, kind and content. Kind can be one of ten values, each with their specific content, *parameters, add-node, add-link, flow-message, add-block, node-committee, node-proposer, create-block, statistics* and *per-node-statistics*.

The GUI is composed of 4 pages, described in the following sections.

3.1.2.1 Parametrization

(referencia)

The Parameters window parses the parameters.json file and produces a form where the user can customize the values or ranges of values for every parameter.

3.1.2.2 Topology Specification

The GUI also allows user to specify the topology of the network without needing to manually write the JSON file. The Topology window offers a canvas to construct a network topology as well as set individual parameters for each node.

(referencia imagens de topology window e parametrizations)

3.1.2.3 Visualizer

(referencia imagiem de timelapse e informação dos nos)

The Visualizer window allows user to play the state of each node in a time-lapse manner and visualize exchanged messages.

3.1.2.4 Statistical Analysis

The Statistics window aids in the analysis of the metrics produced by the simulator. The GUI will parse the output.json file and display it in an easy to read format. These formats can come as graphs, displaying minimum and maximum values observed for each metric that was produced and a graph with per node statistics.

3.2 Critical Analysis

In (referencia tese) we can see a detailed state-of-the-art study on other simulator tools for Blockchain protocols. In this study it is noted that each of the tools studied is mostly built with a specific purpose and not in being modular or extensible to allow for the study of new scenarios and protocols, nor in wide parametrization for different testing scenarios. The table bellow shows an overview of the functionalities that each existing simulator and MOBs offer.

	Adversarial Behavior	**Offline Nodes**	**Bandwidth Limits**	**Network Topology**	**Proof of Stake**	**Proof of Work**	**GUI**
VIBES	yes	not modeled	not modeled	generated via parameters	not modeled	bitcoin	yes
BlockSim	not modeled	not modeled	Throughput calculated from distributionn	fully linked	not modeled	bitcoin ethereum	no
BlockSim	not modeled	not modeled	not modeled	fully linked	not modeled	bitcoin ethereum	no
SimBlock	not modeled	not modeled	specify expected available bandwidth	generated via params	simple example	bitcoin dogecoin litecoin	yes
MOBs	yes	yes	parameterizable	generated via params	tenderbake	bitcoin algorand ouroboros	yes

Table 3.1: Feature comparison between existing blockchain simulators and MOBS.

VIBES (referencia), is scalable, fast and capable of simulating Bitcoin-like protocols however, there is a lack of separation between the code that defines the simulator and the codes that defines the protocols, which makes implementing new protocols difficult and time costly. On the GUI provided this lack of separation also exists, having the statistics that are displayed tied to the protocols being simulated, which means that if the user wants to implement a new protocol that has different metrics/statistics, changes would need to be done to the GUI to support them.

Neither BlockSim, BlockSim nor SimBlock support adversarial or byzantine behavior of the nodes, making it impossible to test the protocols when the network is not in perfect conditions.

VIBEs, BlockSim and BlockSim do not model Proof of Stake protocols which hinders their extensibility, since as a result, these simulators don't offer abstractions for timers and alarms commonly used in proof of stake.

MOBs addresses these concerns like described above offering a more modular and extensible environment, an independent GUI and the ability to simulate different families of protocols.

3.3 Initial Exercise

4

Work to be developed

4.1 Proposed work

4.2 Evaluation

4.3 Work Plan