DEPARTMENT OF
COMPUTER SCIENCE

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

RICARDO FILIPE MENDES LOUREIRO

Bachelor in Computer Science and Engineering

REASONING ABOUT CONSENSUS PROTOCOLS:

SIMULATION AND VALIDATION ENVIRONMENT

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

REASONING ABOUT CONSENSUS PROTOCOLS:

SIMULATION AND VALIDATION ENVIRONMENT

RICARDO FILIPE MENDES LOUREIRO

Bachelor 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
February, 2024

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 are blockchain protocols, usually defined as distributed ledgers with a growing list of records (blocks), linked together by cryptographic hashes. Records are permanent and distributed across a peer-to-peer computer network where participants adhere to the consensus protocols to validate and add transactions.

There are a wide range of different blockchain protocols and some of them are not set in stone, Like Ethereum which moved from proof of work into a proof of stake solution or Tezos where stakeholders are capable of proposing and agreeing on changes to the consensus protocol allowing it to evolve over time. Because of this a number of tools focused on assisting with the development of these protocols have emerged.

One of these tools is MOBS, Modular Blockchain Simulator, built with extensibility and modularity in mind, allows for simulation of consensus and blockchain protocols as well as parametrize multiple scenarios for their study which include bandwidth limits, network layout and Byzantine and adversarial behaviour of participant nodes.

The goal of this dissertation is twofold, first we want to extract statistics and information from MOBS' logs so that we can provide developers with qualitative information about the protocols' execution to empirically verify the protocols' properties (it does not avoid the need for formal verification, it is only a support during the prototyping phase). Secondly we also propose to improve the behaviour and parameterization of the network layer to allow for specific membership protocols to be selected and determine the network layout and re-configuration when participants join or leave the network at runtime.

Keywords: Blockchain, Networking, Consensus, Simulation, Validation of protocol properties

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

List of Figures

List of Tables

1 Introduction

This thesis aims to address practical challenges in designing, implementing and maintaining blockchain consensus protocols by providing a simulation environment to analyse their behaviour and offering empirical metrics of their runtime in different environments and conditions. To achieve this we intend to take Modular Blockchain Simulator (MOBS) and expand the previously done work to help better test and validate these protocols with tools to extract statistics and qualitative data about their runtime. We also tested the viability to provide a more dynamic and independent network layer to better simulate real-world conditions, this would allow programmers to more quickly prototype protocols or solutions in a simulated, modular and parameterizable environment where executions can be repeated and a wide range of scenarios can be used for testing.

1.1 Context

With each passing day the amount of distributed applications like E-banking and social networks increases, these applications leverage state machine replication to offer distributed and reliable services. A subset of these are applications that have strict requirements about the integrity of its data, transactional operations by authenticated 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. Blockchain protocols allow for simultaneous access, integrity check and update of records across a distributed database, each node has its own copy of the ledger that it uses to keep data integrity and reach a consensus about its accuracy.

Around 2008, blockchain protocols appeared with the motivation of a distributed ledger for cryptocurrency transactions, they offer decentralization enhanced security and transparency given that the history of transactions usually being public. These protocols are not static, being because vulnerabilities, flaws that need to be corrected or changes proposed by the stake-holders. One of these dynamic protocols is Tezos [1], which relies on the stake-holders that participate in the system to propose and agree on changes and upgrades to the protocol. These changes are done with Tezos' self-amendment, which

enables the network to undergo changes without the need for a network fork, which in most blockchains is the common practice. Another example is Ethereum [2] that started by using a blockchain protocol based on proof of work and in 2022 migrated towards an implementation based solely on proof of stake for Ethereum2. This change opened Ethereum to bouncing attacks that hindered the finalization of blocks[3]. Due to the nature of data that these handle, errors and vulnerabilities like these can be costly. This opens a necessity for tools that aid in support these evolutions in a faster, more seamless and 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 are further explained in Chapter 2.

The challenges in developing blockchain protocols motivated the development of MOBS, a modular and extensible simulator that provides the ability to simulate different families of blockchain and consensus protocols. MOBS provides a parametrizable execution of the selected protocols, exhibiting a modular and extensible structure and offers detailed logs for the qualitative evaluation for the study of implemented protocols. However, this simulator has limitations such as a network layer that only allows for static parameterizations, network layout and node behaviours are defined before runtime and there is no mechanism to re-structure the network after a node fails or leaves the system, there is also a lack of qualitative data making it difficult to evaluate the protocols' execution.

1.2 Problem

Consensus protocols are inherently complex to design and implement correctly [4, 5]. In blockchain systems, where dynamic behavior and financial transactions are predominant, protocol correctness becomes critical, as even minor errors can result in significant financial losses and system failures. One example of this is Ethereum moving from Proof of Work to a Proof of Stake consensus; this opened the protocol to vulnerabilities to bouncing attacks on liveness [3]. This type of attack prevents the chain from being finalized because the main selected chain in the fork choice rule is continually bouncing between two alternative branches.

There is also Solana [6], a new blockchain protocol that relies on Proof-of-History to build its chain, where repeated testing results showed that the protocol does not fully

achieve consensus and a single malicious validator can halt the Solana blockchain [7]. These tests also showed that there are inconsistencies in the behaviour between what is described in the documentation and what the protocol showed since Solana's implementation has deviated in undocumented ways from the available protocol design descriptions.

Consensus and blockchain protocols are usually described with pseudocode and model checked with idealized languages that not reflect the implementations. Since validating correctness is very costly[8], before planning an implementation, developers should test their ideas and prototypes. There is a lack of methodologies to develop, evaluate, tune and replicate these protocols, opening a need for extensible and modular tools for consensus analysis and experimental labs for rapid prototyping. With this in mind MOBS was developed with the aim to give the ability to test and validate these protocols under different conditions and settings by changing the execution parameters, aiming to catch vulnerabilities or even logic errors before deploying changes to these protocols. Since verification of protocols is very costly, developers can use MOBS to get confidence in their implementations from experimentations first.

Right now the parameters to the network are set before the execution of the simulation, and regarding the network behaviour, we can set the network topology and 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.

Currently, in MOBS there is no dynamic parameterization of the network layer, making it hard to simulate real world conditions, together with the lack of qualitative data provided by the simulator, validating a protocol's executions becomes hard.

1.3 Goal

With the different properties that different network overlays provide besides the parameterizations that are already provided by MOBS, we can offer a more complete simulator that allows for the study of the behaviour in different membership protocols. This will allow us to leverage the modularity of this layer to better simulate the behaviour of new participants coming into the system, existing ones leaving and how changes to the network topology of the network affects their execution.

Another aspect we improve is the logging module of MOBS, by providing a template for logging of consensus protocols and an after execution analysis tool to validate their execution and extract metrics for comparison between executions in different environments, like average time to reach consensus, consensus agreement percentage or number of agreements messages received after consensus.

The goal of this thesis is twofold: first 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 parameterization regarding the behaviour of the network,
- Better qualitative data by providing scenarios that better mimic real world execution;

and secondly we want to improve the logs to provide better qualitative data so that we can quickly analyse the protocols properties and check their execution, allowing the programmer to spot early flaws or even vulnerabilities by extensive simulation, gaining confidence in the execution of the protocol before moving to formally prove their correctness. Thus, we provide concise, meaningful and transversal data like:

- Average time to consensus,
- Quorum agreement percentage for each consensus,
- Number of late agreement messages;

This will allow users to better evaluate the protocols' execution and combined with the more modular and more parameterizable simulator provide a better environment to quickly prototype protocols and solutions while testing in a wide range of parameterizable scenarios.

To achieve this we extended MOBS (MOBS-Fork) in the following ways:

- Implement and study what properties are needed to evaluate the execution of consensus protocols, these include Paxos, Chandra-Toueg, PBFT and Ethereum.
- Evaluate the execution of blockchain protocols and replicate known vulnerability scenarios, namely Ethereum Probabilistic Bouncing attack[3] and Solana Halting Problem[7].

1.4 Contributions

The contributions to this thesis focues on the implementation and evaluation of consensus protocols in MOBS, to verify that MOBS can be used has a tool for prototyping and validation. The protocols implemented and evaluated were:

- Chandra-Toueg [9], a simple consensus protocol to ensure the logs from MOBS can be used to evaluate consensus properties,
- Paxos [4], a more complex consensus protocol that allows for greater flexibility in the face of network partitions,
- PBFT [10], a practical Byzantine fault-tolerant protocol that is widely used in permissioned blockchain systems,

- Ethereum [2], a decentralized platform that enables the creation of smart contracts and decentralized applications and specifically replicated the Probabilistic Bouncing attack and the patch that was implemented validating its executions.

2

Background

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 some consensus protocols.

In Section 2.3 we will discuss some different types of blockchain protocols.

2.1 Membership Protocols

This section introduces membership protocols that have been considered to be implemented as part of the work that will be developed. We selected these memberships to study due to their wide differences in properties and their popularity in usage.

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 usually 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 tend to grow at a logarithmic rate instead.

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

- **Reactive:** Using this strategy, the partial view undergoes alteration solely when there are changes in membership, typically occurring when a node joins or leaves the membership.

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

This membership protocols are usually represented by a graph where the vertices represent the participants of the network 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. The node's position is usually based on a unique identifier for each node of the network and different protocols organize the nodes by their identifier by

However, having a predefined topology come at the cost of a more costly re-structuration of the network every time a there is a change to the membership. This process is usually slow and costly since a change of one node 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 [11] is a Distributed Hash Table based algorithm where each node is assigned a unique identifier based on the output of a cryptographic hash function like SHA-1 or MD-5. Usually this hash 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 determine key assignment in the distributed hash table: each node is responsible for all keys in the range from its predecessor's identifier (exclusive) to its own identifier (inclusive). More formally, a node with identifier n is responsible for keys in the range *]predecessor(n), n]*. This ensures that every possible key in the identifier space is assigned to exactly one node, with keys hashed using the same hash function used for node identifiers.

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 and each node maintains a direct link to their predecessor and successor. Each node also maintains a routing table with around O($log\ n$) distinct entries called a finger table. Using this routing table we can improve the lookups from O(n) to O($log\ n$) since we can search in our finger table for the node with the closest preceding identifier instead of navigating the ring node by node.

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 neighbours. In order to increase fault tolerance, a periodical stabilization protocol is run in background, making sure the neighbours are still active and restructuring the overlay accordingly as well as updating the entries on the finger table.

2.1.2.2 Pastry

Pastry [12], 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 *neighbourhood set* and a *leafing set*. The routing table is organized in levels, where each level l contains entries for nodes that share an l-digit prefix with the local node's identifier, typically using base-2^b encoding (commonly b=4), the routing table has up to 2^b entries per level, allowing efficient prefix-based routing.

The neighbourhood set contains the M nodes that are closest to the local node in terms of network proximity (measured by latency or network distance), not identifier space proximity.

The leaf set contains the $L/2$ numerically closest nodes with smaller identifiers and $L/2$ numerically closest nodes with larger identifiers in the circular identifier space.

When a new node joins, it contacts an existing nearby node and obtains initial routing state by querying nodes along the path to its final position and updates routing tables, leaf sets, and neighbourhood sets of affected nodes.

Pastry runs periodic maintenance protocols to ensure routing correctness. Nodes periodically exchange keepalive messages with their leaf set and neighbourhood set members, update routing table entries by probing nodes with longer common prefixes, and repair any detected inconsistencies in their routing state when there are changes in the memberhsip.

Pastry uses a three-step routing algorithm: (1) if the destination is within the leaf set range, route directly to the destination or the numerically closest node; (2) if not, forward to a node from the routing table that shares a longer common prefix with the destination; (3) if no such node exists, forward to a node from the leaf set or neighbourhood set that is numerically closer to the destination. With this algorithm Pastry achieves O($log\ N$) expected routing hops for message delivery and maintains O($log\ N$) routing state per node, making it highly scalable for large networks.

2.1.3 Unstructured Overlays

Unstructured overlays place few constrains as on how the nodes chose their neighbours. 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 [13] 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.

SWIM uses a three-state model for nodes: alive, suspect, and failed. When a direct probe fails, the target node enters the suspect state rather than being immediately marked as failed. During the suspicion period, other nodes can refute the suspicion by providing evidence that the suspected node is actually alive. This mechanism significantly reduces false positives while maintaining rapid failure detection.

SWIM uses incarnation numbers to handle false suspicions: each node maintains an incarnation number that it can increment to refute suspicions about itself. The protocol uses specific message types including *PING* (direct probe), *PING-REQ* (indirect probe request), *ACK* (acknowledgment), and membership update messages that carry node states and incarnation numbers.

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.

SWIM achieves excellent scalability properties: failure detection time is O(*1*) with respect to group size, and each node generates a constant message load per time period regardless of the total number of nodes. The protocol provides eventual consistency of membership views across all nodes, with new nodes joining by contacting any existing member and receiving the current membership list. The weak consistency model ensures that all correct nodes eventually converge to the same membership view, though temporary inconsistencies may exist during periods of high churn.

2.1.3.2 HyParView

HyParView [14] 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 two distinct views: an *active view* used for reliable message dissemination, and a *passive view*, usually 3 to 5 times larger than the active view, that serves as a backup for network restructuring when the active view changes.

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 with periodic shuffle operations every t seconds (typically 10-30). During a shuffle, a node exchanges a subset of its passive view with a randomly selected active neighbor. The neighbour responds with its own partial passive view. Both nodes integrate received entries while maintaining view size limits through age-based eviction. This mechanism ensures O($log\ N$) mixing time for achieving uniform random sampling across the network and maintains connectivity with high probability.

The protocol uses TCP connection failures as an implicit failure detector: when a TCP connection to an active neighbor fails, the node immediately replaces it by promoting a node from the passive view to maintain the target active view size, providing excellent fault tolerance properties and avoiding partitions even under high churn rates.

Tests done in [14] 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 [15] 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 neighbour. Each node maintains a partial view that contains the addresses of nodes that are not its immediate neighbours, 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 [16] 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 follows, 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 Consensus Protocols

Consensus protocols address the fundamental challenge of enabling distributed participants to reach agreement on a specific value, this is essential for building reliable distributed systems such as replicated databases, distributed file systems, and blockchain networks. These algorithms must ensure agreement is reached even in the presence of faulty nodes, network partitions, and asynchronous communication delays.

Any consensus protocol must satisfy the following four fundamental properties [17]:

- **Termination:** Eventually, every correct process decides some value.
- **Agreement:** If all correct processes propose the same value v, then any correct process that decides a value must decide v.
- **Integrity:** No correct process decides more than once.

Consensus protocols differ in their assumptions about the system model, including synchrony, failure types (crash-fault vs. byzantine-faults), and the number of faults tolerated. The famous FLP impossibility result shows that deterministic consensus is impossible in asynchronous systems with even one crash failure, making practical protocols rely on additional assumptions like partial synchrony or randomization.

We will now examine specific protocols that solve consensus under different system models.

2.2.1 Chandra-Toueg

The Chandra-Toueg consensus protocol [9] solves consensus in partially synchronous systems using an eventually strong failure detector. This failure detector abstracts the timing assumptions needed for consensus, providing an oracle that can make mistakes about process failures but eventually becomes reliable. An eventually strong failure detector is defined by the following two properties:

- **Strong Completeness:** Eventually, every process that crashes is permanently suspected by every correct process.
- **Eventual Strong Accuracy:** There is a time after which no correct process is suspected by any correct process (i.e., eventually the failure detector stops making false accusations about correct processes).

The algorithm operates in the crash-fault fail model, it assumes that fewer than half of the processes can fail *(f < n/2)* and guarantees termination in O(*f+1*) rounds, where *f*

is the number of process failures. The protocol proceeds in asynchronous rounds with a rotating coordinator selected in round-robin fashion. Each round consists of four phases:

1. **Phase 1:** All processes send their current estimate and timestamp to the coordinator.
2. **Phase 2:** The coordinator waits to receive messages from a majority of processes. If successful, it selects the estimate with the highest timestamp and broadcasts this chosen value to all processes. If the coordinator is suspected of failure, it proceeds to the next round.
3. **Phase 3:** Each process waits to receive the coordinator's proposal or for its failure detector to suspect the coordinator as failed:
    - If the proposal is received, the process adopts it as its new estimate and sends an *ACK* to the coordinator.
    - If the coordinator is suspected as failed, the process sends a *NACK* and proceeds to the next round with a new coordinator.
4. **Phase 4:** If the coordinator receives *ACKs* from a majority of processes, it broadcasts a *DECIDE* message with the chosen value. Upon receiving a *DECIDE* message, processes decide on the value and terminate. The *DECIDE* message is also relayed to ensure all correct processes eventually decide.

2.2.2 Paxos

Paxos [4] is a family of protocols for solving consensus, for this example we will take a look at what is commonly referred as basic Paxos, that decides on a single value, and after take a look at MultiPaxos which gives a constant stream of agreed values. For this algorithm we make the following assumptions regarding processors:

- Operate at arbitrary seed.
- May experience failures.
- Have stable storage and may re-join the protocol after failures.
- Byzantine failures do not occur.
- The maximum number of failing processors is less than half of the total processors.

And the following assumptions regarding the network:

- Processors can send messages to any other processor.
- Messages are asynchronous and take an arbitrary time to deliver.
- Messages may be lost, re-ordered or duplicate.

- Messages when delivered are delivered without corruption.

Each participant can act as a Proposer, an Acceptor and a Learner. Each execution of basic Paxos decides on a single value and operates in two phases, each with two secondary phases.

- **Phase 1**

    1. **Prepare:** A Proposer creates a message which we call Prepare identified with a number n, this works as an identifier and has to be greater than any previous number used in previous Prepare messages. The Prepare message does not contain the proposed value. This message is sent to a quorum of acceptors, and a proposer shouldn't initiate Paxos if it cannot communicate with a quorum of acceptors.
    2. **Promise:** Acceptors wait for a Prepare message from any of the proposers, if they receive a message, depending on the value of n two flows can happen:
        a) If n is higher than every previous proposal received from any Proposer the Acceptor returns a Promise message and ignores all future proposals with a value less than n, if a proposal was accepted at some point in the past, the message includes the previous n value and the corresponding accepted value.
        b) Otherwise, the Acceptor can ignore the Prepare message or sending a not acknowledge to tell the Proposer to stop its attempt to create the Proposal.

- **Phase 2**

    1. **Accept:** If the Proposer receives Promises from a Quorum of Acceptors, it sets a value for its proposal. If any Acceptors that accepted a previous proposal they would have sent the previous accepted value, the Proposer will agree on the return value sent by the Acceptors with the highest n. In none of the acceptors had previously accepted a value then the Proposer chooses the value it initially wanted to propose. The proposer sends an Accept message with the chosen value and the n value.
    2. **Accept:** If an Acceptor receives and Accept message from a Proposer, it must accept if it hasn't proposed previously Proposals with an identifier grater than n. If the value is accepted, an Accepted message is sent to every proposer and Learner. Learners will only learn the decided value after receiving Accepted messages from a majority of Acceptors.

MultiPaxos is another algorithm in the Paxos family and is used when a continuous stream of agreed values is needed. If the leader is relatively stable phase 1 becomes unnecessary. To achieve this we include the round number along with each value which

is incremented in each round by the same leader. We still need the phase 1 for the first round but in consequent rounds if the Leader does not change nor fails we can skip it, reducing the overhead for each round.

2.2.3 Practical Byzantine Fault Tolerance (pBFT)

Practical Byzantine Fault Tolerance (pBFT) [10] provides a solution for achieving consensus in partially synchronous networks supporting f nodes with Byzantine behaviour in a network of *3f+1* nodes. Unlike probabilistic consensus mechanisms, pBFT guarantees immediate finality and strong consistency, making it suitable for applications requiring deterministic transaction confirmation.

The pBFT protocol operates through a three-phase process for each consensus round:

1. **Pre-prepare:** The primary replica broadcasts a pre-prepare message containing the proposed operation sequence number and client request.
2. **Prepare:** Upon receiving a valid pre-prepare message, backup replicas broadcast prepare messages, indicating agreement with the proposed operation ordering.
3. **Commit:** Once a replica receives *2f* prepare messages from different replicas, it broadcasts a commit message and executes the operation after receiving *2f+1* commit messages.

The protocol ensures safety through view changes when the primary is suspected of failure, and liveness through eventual synchrony assumptions. pBFT variants are widely used in permissioned blockchain networks such as Hyperledger Fabric [18] and various consortium blockchain implementations, where the set of validators is known, and network communication is more reliable than in public networks.

While pBFT provides strong consistency guarantees, its $O(n^2)$ message complexity limits scalability to networks within hundreds rather than thousands of participants, making it more suitable for a smaller number of participants like in consortium and private blockchain deployments.

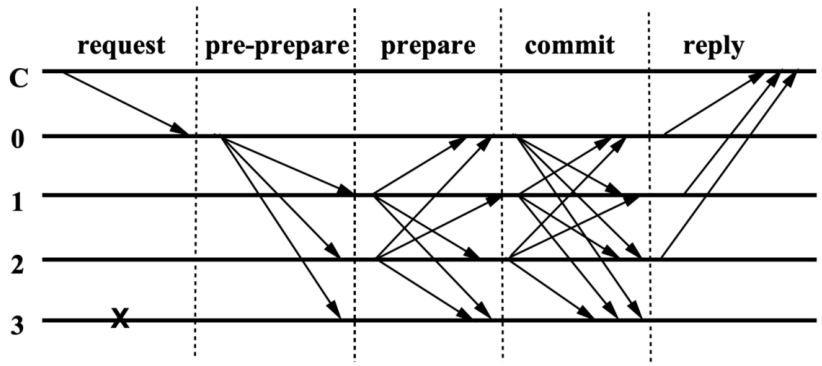

Figure 2.1: Flow of PBFT

2.3 Blockchain Protocols

Blockchain protocols are the set of rules that govern how data is secured, recorded and shared with a blockchain network, usually a distributed ledger that maintains a continuously growing list of records (blocks) linked and secured using cryptographic hashes. These blocks are linked in a chain that is immutable and tamper-evident, making it suitable for applications that require trustless consensus among distributed participants.

Blockchain protocols are usually designed with the goal to achieve four key properties:

- **Decentralization:** No single point of control or failure, data is replicated across multiple nodes.
- **Immutability:** Historical records cannot be altered due to cryptographic linking.
- **Transparency:** All transactions are visible to all network participants
- **Consensus:** All nodes maintain consistent state despite potential Byzantine faults.

Blockchain systems can be categorized into three main types based on access control and governance models [19]:

- **Public Blockchains (Permissionless):** Open to all participants without restrictions. Anyone can join the network, validate transactions, and participate in consensus. Examples include Bitcoin, Ethereum, and Litecoin. These systems prioritize decentralization and censorship resistance but often sacrifice performance and energy efficiency.
- **Private Blockchains (Permissioned):** Controlled access restricted to specific organizations or entities. Only authorized participants can join, validate, or access data. Examples include enterprise solutions like Hyperledger Fabric deployments within corporations and central bank digital currencies (CBDCs). These offer higher performance and privacy but reduce decentralization benefits.
- **Consortium Blockchains (Semi-decentralized):** Controlled by a pre-selected group of participants, typically industry partners or allied organizations. Examples include R3's Corda for financial institutions, IBM Food Trust for supply chain management, and Energy Web Chain for the energy sector. These balance decentralization with performance and regulatory compliance.

Each type presents distinct trade-offs between decentralization, performance, governance, and trust assumptions, fundamentally influencing the choice of consensus mechanism and system architecture.

In this section, we examine popular blockchain consensus protocols that address Byzantine behavior under different system models and trust assumptions.

2.3.1 Proof of Work (PoW)

Proof of Work, adopted by Bitcoin [20] and Ethereum [2] (pre-2022), represents the first successful solution of a blockchain protocol. This consensus mechanism selects the next block producer through a computational competition where nodes (miners) compete to solve a cryptographic puzzle with adjustable difficulty.

The PoW algorithm operates as follows: miners collect pending transactions into a block candidate, then iteratively modify a nonce value while computing the block's hash using a cryptographic function (typically double SHA-256). The objective is to find a hash that meets the current difficulty target that automatically adjusts to maintain average block times (e.g., 10 minutes for Bitcoin).

PoW provides several security guarantees: it ensures eventual consistency through the longest chain rule, requires attackers to control more than 50% of the network's computational power for successful attacks, and creates an economic incentive structure through block rewards and transaction fees. However, this comes at the cost of significant energy consumption and limited transaction throughput.

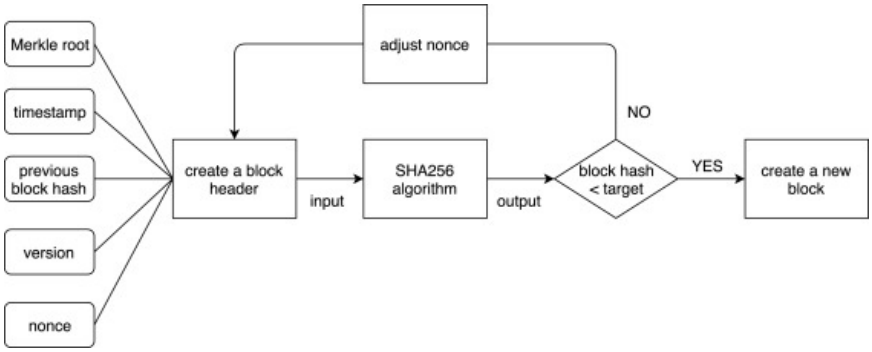

Figure 2.2: Flow of PoW

2.3.2 Proof of Stake (PoS)

Proof of Stake replaces computational competition with economic stake as the basis for consensus participation and block production rights. In PoS protocols, validators are selected to create new blocks based on their stake rather than their computational power, significantly reducing energy consumption compared to Proof of Work protocols.

The selection mechanism typically employs randomization weighted by stake size where validators with larger stakes have proportionally higher chances of being selected to produce the next block. Modern PoS implementations like Ouroboros [21], Ethereum 2.0's Gasper [22], and Tendermint [23] use selection algorithms that provide cryptographic proofs of randomness and prevent manipulation.

PoS systems enforce honest behavior through economic penalties: validators must deposit stake as collateral, which can be partially or fully confiscated for provable misbehavior such as double-signing or violating protocol rules. This creates strong economic

incentives for honest participation while enabling faster finality and higher transaction throughput than PoW systems.

Ethereum successfully transitioned from PoW to PoS in 2022, demonstrating the practical viability of this approach for large-scale blockchain networks.

Ethereum's transition to PoS although successful has opened the protocols up to vulnerabilities, one of which will be further discussed in Section 5.2.

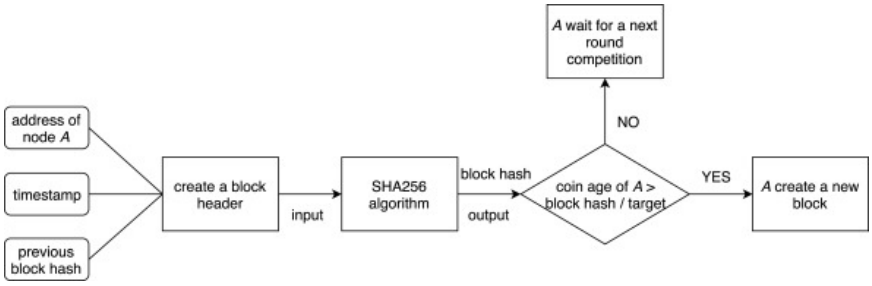

Figure 2.3: Flow of PoS

2.3.3 Delegated Proof of Stake (DPoS)

Delegated Proof of Stake introduces a representative democracy model where token holders vote to elect a limited set of delegates responsible for block production and network governance.

In DPoS systems, token holders continuously vote for delegates using their stake as voting power. The top-voted delegates form a rotating committee that takes turns producing blocks in a round-robin fashion, producing deterministic block times and higher transaction throughput compared to probabilistic consensus mechanisms like PoW or traditional PoS.

DPoS implementations include EOSIO [24], Tron [25], and the BitShares [26]. These protocols offer several advantages: faster block confirmation times, typically one to three seconds, higher transaction throughput, and lower energy consumption. However, this comes at the cost of increased centralization, as only a few delegates control block production, potentially making the system more susceptible to collusion and censorship attacks.

Delegate accountability is maintained through continuous voting: poorly performing or malicious delegates can be voted out by token holders, creating ongoing incentives for honest behavior and competent network operation.

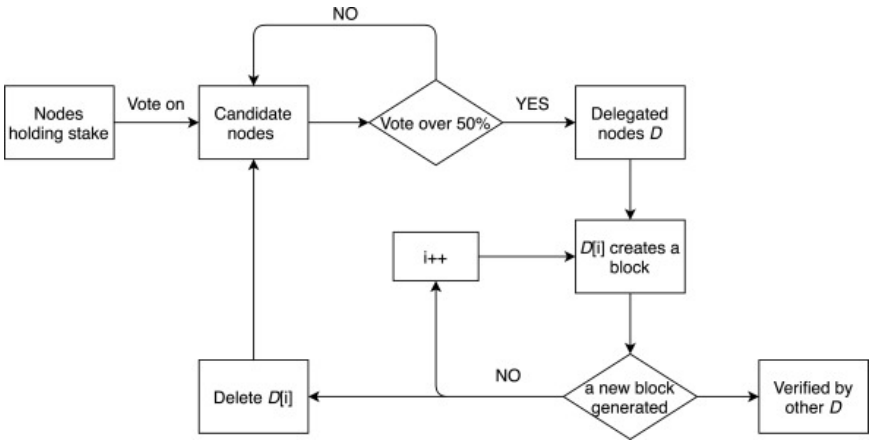

Figure 2.4: Flow of DPoS

3

Related work

(TODO passar tudo para o presente (tese inteira))

In this chapter we will talk about work that has been developed and we found that is most in line with the work we want to develop. We will use them either as a comparison to what we want to develop or as a part of the solution that we are proposing.

In this Section we will discuss some simulators that we think are relevant to study since they provide tools that are in line to what we also want to provide with MOBS and are the state of the art for blockchain simulators.

3.1 VIBES

Vibes [27] is a message driven blockchain simulator developed with the goal of providing configurable, scalable and fast network simulations. It also provides a GUI for visualization of some extracted metrics and allows users to view a time-lapse of the processed events.

Vibes is developed in Scala and as such inherits the Actor language paradigm, the actors in Vibes can have one of three roles:

- **Node:** Follows the protocol to replicate the behaviour of the blockchain network.
- **Coordinator:** This actor acts as an application-level schedule. It receives requests from all nodes to fast-forward the network to the point in time when each node has completed his current task and once it receives a request from all nodes it moves the entire network to the earliest timestamp, guaranteeing a correct execution order of all tasks.
- **Reducer:** Once the simulation ends the reducer gathers the state of the network and produces the simulation results to be processed by the user.

3.2 BlockSim: Blockchain Simulator

BlockSim [28] is a simulation framework that assists in the design and Evaluation of blockchain protocols. Developed in Python it uses a probabilistic distributions model that can be specified by the user to model random phenomena such as time taken to validate a block and network latency.

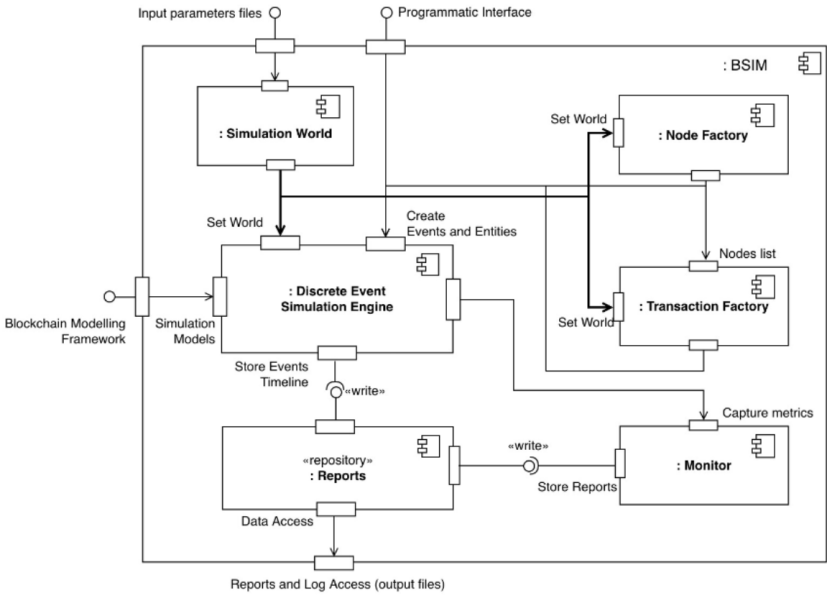

Figure 3.1: BlockSim Architecture [28]

BlockSim's architecture, as shown in the figure, is made up by the following components:

- **DESE:** Discrete Event Simulation Engine, based on SimPy
- **Simulation World:** Responsible for handling input/configuration parameters of the simulations.
- **Transaction and Node Factory:** Responsible for creating batches of transactions modelled as random phenomena. Node Factory creates nodes that are used during the simulation
- **Programmatic interface and Simulation Example:** Main interface available to the user
- **Monitor and reports:** Monitor captures metrics during the simulation, i.e. number of transactions broadcasted or received, transactions added to queue. These metrics are stored in the reports component
- **Blockchain modelling Framework:** Has several layers like the Node layer, the Consensus, the Ledger, Transaction and block, Network and Cryptographic.

3.3 BlockSim: Extensible simulation tool for blockchain systems

Although with a similar name of the previously presented simulator, BlockSim is a different framework designed to build and simulate discrete-event dynamic systems models for blockchain systems.

BlockSim has three main modules:

- **Simulation:** Is in charge of setting up the scheduling of events and compute the simulation statistics.
- **Base:** Consists of the layers for the implementation of the blockchain protocol that will be simulated
- **Configuration:** Is the main user interface where the simulation can be selected and parameterized.

The base module has three layers that can be extended by the user:

- **Network:** represents blockchain nodes and the underlying peer-to-peer protocol to exchange data.
- **Consensus:** encompasses algorithms and rules adopted to reach an agreement about the current state of the blockchain ledger.
- **Incentives:** contains the economic incentive mechanisms adopted by a blockchain to issue and distribute rewards among participating nodes.

3.4 SimBlock:A Blockchain Network Simulator

SimBlock was developed in Java to preform experiments on blockchain protocols with a large amount of nodes. It allows users to easily change the behaviour of nodes and study their overall impact on the network. It also provides users with a simple GUI for users to load the output file of a simulation and observe evolution of the simulated protocol. Simblock is divided into three components:

- **Network:** Creates the network topology which is configurable in the number of nodes, number of neighbours for each node and latency for each node to its neighbours.
- **Node:** Defines the behaviour of each node in accordance to the simulated protocol.
- **Node:** Defines the rules for creating and validating blocks.

3.5 JABS

JABS [29] was developed in JAVA and is aimed at researching large-scale blockchain consensus algorithms, with a main focus on simulating consensus, network, and ledger-data layers. The simulator is designed to be modular and extensible, optimized for performance and scalability.

JABS is developed to be used as a discrete-event simulation tool for benchmarking, evaluating, adjusting, and comparing consensus algorithms, especially for global-size public blockchains

JABS is composed of five main components:

- **Scenario:** Serves as a template for designing and adjusting simulation parameters.
- **Network:** Describes the connections between nodes and their bandwidth.
- **Simulator:** Responsible for processing each node's events in the correct order.
- **Node:** Is where the logic for the protocol is implemented and defines each node's behaviour.
- **Logger:** Handles all the outputs in the simulation into either a standard output or a CSV file.

3.6 Critical Analysis

We analysed the presented simulators, either by reading the papers presented by the creators, using the publicly available ones to run their implemented protocols or in the case of JABS, trying to implement a new protocol to see how the creators provided modularity and extensibility. From that analysis we compiled this data:

In Vibes [27] 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 [28], BlockSim [30], SimBlock [31] nor JABS support adversarial or Byzantine behaviour 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.

	Adversarial Behavior	Offline Nodes	Bandwidth Limits	Network Topology	Proof of Stake	Proof of Work	GUI
VIBES [27]	yes	not modeled	not modeled	generated via parameters	not modeled	bitcoin	yes
BlockSim [28]	not modeled	not modeled	Throughput calculated from distributionn	fully linked	not modeled	bitcoin ethereum	no
BlockSim [30]	not modeled	not modeled	not modeled	fully linked	not modeled	bitcoin ethereum	no
SimBlock [31]	not modeled	not modeled	specify expected available bandwidth	generated via params	simple example	bitcoin dogecoin litecoin	yes
JABS [29]	not modeled	not modeled	configurable	generated via network layer	simple example	bitcoin DAGsper	no
MOBS	yes	yes	parametrizable	generated via params	tenderbake	bitcoin algorand ouroboros	yes

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

JABS designed to implement simple blockchain protocols, and is not ready to implement pure consensus protocols. Furthermore, it lacks the means to leverage the already implemented logic in other protocols to be re-used in new implementations, making the development of new protocols costly and time-consuming.

4

MOBS

4.1 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 separately.

4.1.1 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 4.1 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.

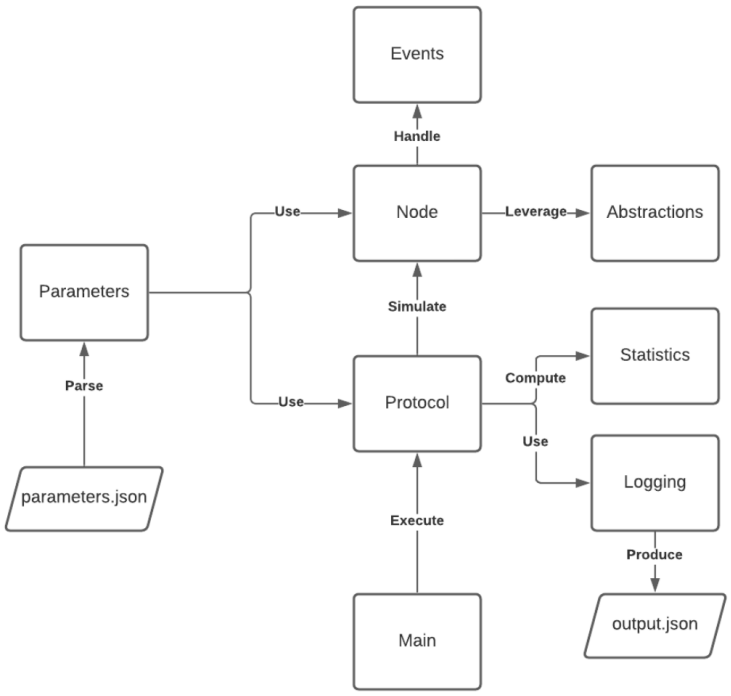

Figure 4.1: Illustration of top-level module interactions

- **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.

4.1.2 Graphical User Interface

The graphical user interface was implemented in NodeJS, Vue3 and ElectronJS. The choice for web technologies enables the future deployment 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.

4.1.2.1 Parametrization

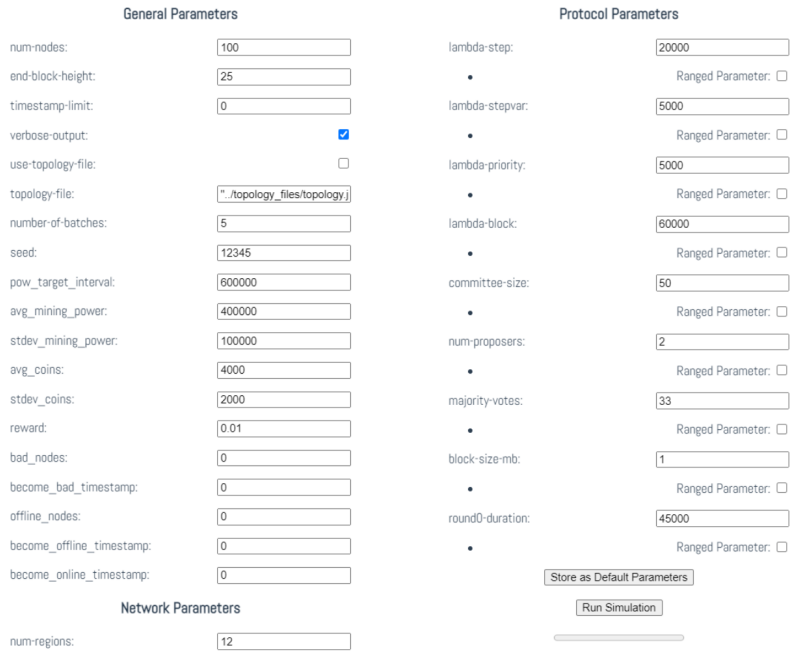

Figure 4.2: Parameters window

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.

4.1.2.2 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.

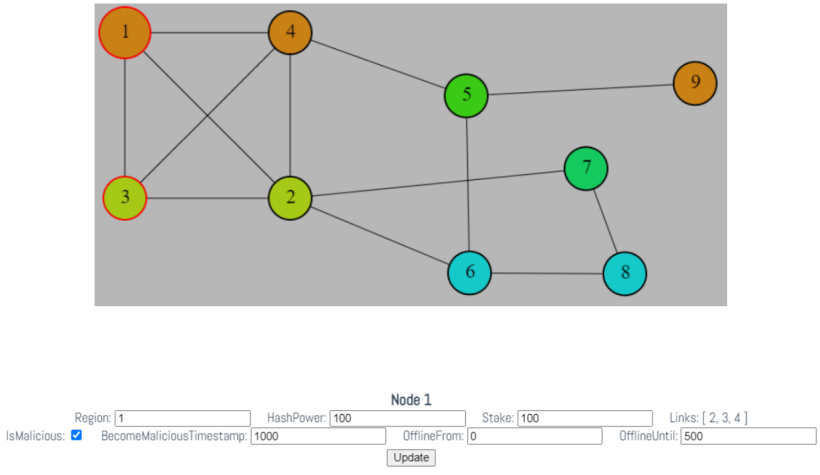

Figure 4.3: Topology window and possible parametrizations for each individual node

4.1.2.3 Visualizer

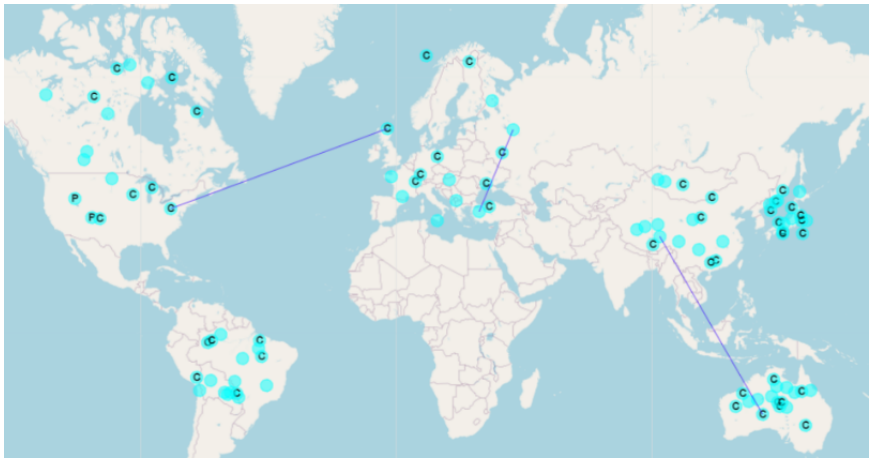

Figure 4.4: Time-lapse in the Visualizer window

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

4.1.2.4 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.

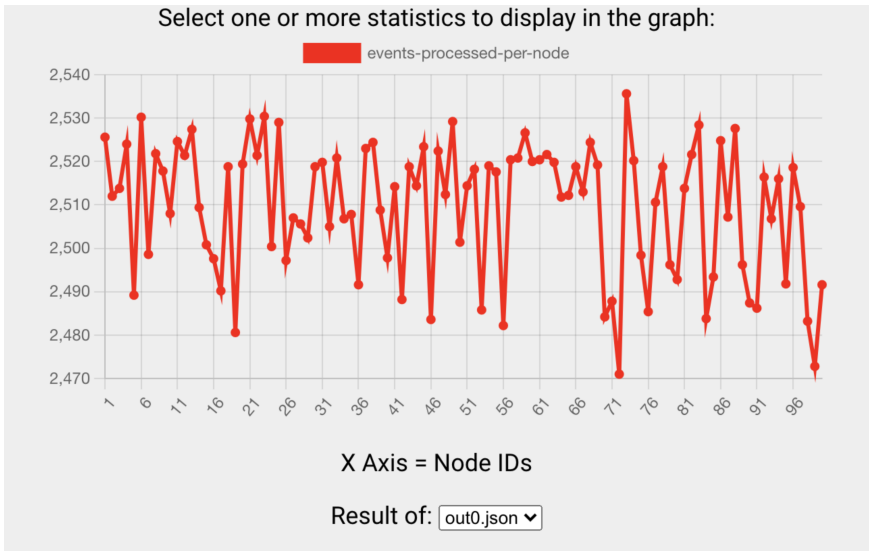

Figure 4.5: Per Node Statistics window

4.2 Initial Network Exercise (TODO -> Implementation of membership protocol)

DESCRIÇÃO E CRITICA

As an initial exercise we chose to implement Chord in MOBS; this served twofold:

- To understand how MOBS worked and how to use it to implement protocols
- To see the capabilities of MOBS in providing qualitative metrics to evaluate a membership protocol.

To achieve this implementation we made use of MOBS Node module and implemented the following events:

- **Join:** An event that a node receives when another node first joins the network. The receiving node will evaluate if the identifier of the new node makes him a candidate to be that node's predecessor in the ring.
- **JoinResponse:** A node that has previously received a join will reply with this event if the new node has been accepted as their predecessor. This contains the identifier of the old predecessor so that the new node can make it its own.
- **Uft:** This event stands for update finger table, this event shares an identifier known to a node, being from their finger table, their own, or one of its neighbors with a random node. The receiver node will add this identifier to their finger table.
- **UpdateSuccessor:** This event is only used when a node updates their successor this event is sent to itself as a way to log the new link.
- **UpdatePredecessor:** This event works the same way as UpdateSuccessor but for the predecessor node.
- **RebalanceNetwork:** When a node receives this event it comes with an identifier, the receiver node will see if the identifier is a better candidate for their successor. This serves as a cyclic membership managing strategy.
- **SendRebalanceNetwork:** Periodically a node will propagate this message to neighboring nodes to trigger a RebalanceNetwork event with their identifier.

The implementation of this protocol at the Node level turned out to not be successful.

One of the reasons is that when implementing this at the Node module, the network module implements a network with a random graph, and as an initial operation, one node would trigger a RebalanceNetwork with the identifier of another node and through an epidemic broadcast fashion trigger all nodes into trying to join to other nodes. This resulted in one of two scenarios:

- Several Chord rings would be created instead of one single ring with all nodes, making further communications since at the Chord protocol level, nodes from different rings would be unknown from each other.
- After making nodes send periodic SendRebalanceNetwork events to neighboring nodes at the Network module level, if the first couple of events made them unable to integrate the ring they would become dormant, since the trigger to send new SendRebalanceNetwork events is triggered by receiving events from other nodes

Another reason this protocol failed was that even leveraging overlay created at the Network layer to send SendRebalanceNetwork events to know nodes at that level to ensure that all nodes would converge to the same ring, this would create an enormous amount of events that would slow down the simulator. This showed some promising results, and we were able to see signs of a converging network, but the search for better neighbors at the Chord level turned out to be a blind search, and the closer the network became to converge the longer it took for new updates to take place.

To solve these issues we proposed move the implementation to the network module level. This will make possible to initiate one node at a time and reduce blind lookups, solving all three problems at once:

- Only one ring will be created
- Since nodes would always ping a known member of the ring to join, the protocol would ensure they would be placed on the correct place of the network and not be left out isolated
- Join operations in a formed chord ring would be less costly in the amount of messages generated since we don't have to broadcast events and can instead target to specific nodes.

We can also leverage this implementation at the network level to make the network module even more modular and extensible, making it easier to implement new network protocols.

4.3 Consensus protocols implemented

——————————TODO We need to add PBFT and Chandra-Toueg to this section as well seperar em secções para cada protocol mais secção de comparação——————————

One of the problems we want to tackle with this thesis is getting better qualitative information out of blockchain and consensus protocols logs so that we can evaluate their correctness at the end of the simulation. To achieve this we implemented two simple and well known consensus protocols so that we can extract the runtime information and use them as a simple example to see MOBS' limitations and if further work needs to be done in Statistics and Logging module.

The first of these protocols was Paxos. This was implemented as referenced in 2.2.2, with only main changes to the protocol made taking an optimistic approach and as a first phase not worrying about scenarios where malicious nodes are present or messages are lost. The second was having only 1 proposer through the whole simulation, making this an implementation closer to MultiPaxos than regular Paxos, this was done for the sake of simplicity an ease of implementation and testing.

With and implementation of Paxos done and validated, a script in Python was done that allowed us To scrub the logs and extract runtime metrics:

```
Value: 6816668 Reached at: 96928ms nodes agreed before: 6 nodes agreed after:3
Value: 1261359 Reached at: 97595ms nodes agreed before: 7 nodes agreed after:2
Value: 2731005 Reached at: 97984ms nodes agreed before: 6 nodes agreed after:3
Value: 4043237 Reached at: 98564ms nodes agreed before: 6 nodes agreed after:3
```

Figure 4.6: Output from the first iteration of the log analyser script

This was the first iteration of this script which allowed us to see what value was accepted, at what tie it was accepted, how many nodes accepted the value and how many accept messages reached the proposer after the value was accepted. On the second iteration of this script we added a section with condensed global data, where we can more easily observe metrics like number of values accepted, total simulation runtime, average time per consensus and the average acceptance percentage of consensus.

```
==============================GLOBAL DATA=====================================

No of consensus reached: 26
Runtime: 9723ms
Average time per consensus: 373.96153846153845ms
Average acceptance percentage: 98.07692307692307%
```

Figure 4.7: Output from the second iteration of the log analyser script

The implementation of both the protocol and the script can be consulted at this pull request.

With this work done we implemented Chandra-Toueg as described in 2.2.1, we chose this protocol for the ease of implementation and again focused on an optimistic implementation where we didn't take into account malicious nodes and lost messages. We also took into account and implemented the logs of this protocol so that the same script developed for Paxos could be used without changes to analyse this protocol. The implementation can be consulted at this pull request.

After the implementations were completed we compared their executions. We found that while both protocols reached near 100% agreement, Paxos was on average twice as fast as Chandra-Toueg reaching consensus. After manually analysing the logs we came to the conclusion that the missing percentage of acceptance came from the protocol runtime being limited to 10000ms, and the simulation ending before all the Accept messages reached the proposer, since all values before accounted for 100% acceptance.

Table 4.1: Average results from 20 executions at 10000ms runtime

	Average Time per Consensus	**Average acceptance percentage**	**Number of consensus reached**
Paxos	101.76 ms	98.35%	97
Chandra-Toueg	201.755ms	98.85%	49

5 Ethereum

5.1 Protocol Description

5.1.1 PoS to PoW

5.2 Bouncing attack

5.2.1 Proposed Solution

5.2.2 Critical analysis of the Proposed Solution

5.3 Implementation In Mobs

5.3.1 Specification

5.3.2 Implementation

5.3.3 Validation

5.3.4 Results analysis

Bibliography

[1] V. Allombert, M. Bourgoin, and J. Tesson. "Introduction to the Tezos Blockchain". In: *CoRR* abs/1909.08458 (2019). arXiv: 1909.08458. URL: http://arxiv.org/abs/1909.08458 (cit. on p. 1).

[2] V. Buterin. "Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform". In: (2013). URL: https://github.com/ethereum/wiki/wiki/White-Paper (cit. on pp. 2, 5, 17).

[3] U. Pavloff, Y. Amoussou-Guenou, and S. Tucci-Piergiovanni. *Ethereum Proof-of-Stake under Scrutiny*. 2022. arXiv: 2210.16070 [cs.CR] (cit. on pp. 2, 4).

[4] L. Lamport. "Paxos Made Simple". In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001-12), pp. 51–58. URL: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/ (cit. on pp. 2, 4, 13).

[5] H. Howard and R. Mortier. "Paxos vs Raft: Have we reached consensus on distributed consensus?" In: *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. 2020, pp. 1–9 (cit. on p. 2).

[6] A. Yakovenko. "Solana: A new architecture for a high performance blockchain v0.8.13". In: *Whitepaper* (2018) (cit. on p. 2).

[7] J. Sliwinski et al. "Halting the Solana Blockchain with Epsilon Stake". In: *Proceedings of the 25th International Conference on Distributed Computing and Networking*. ICDCN '24. <conf-loc>, <city>Chennai</city>, <country>India</country>, </conf-loc>: Association for Computing Machinery, 2024, pp. 45–54. ISBN: 9798400716737. DOI: 10.1145/3631461.3631553. URL: https://doi.org/10.1145/3631461.3631553 (cit. on pp. 3, 4).

[8] G. J. Holzmann. "Design and validation of computer protocols". In: 1991. URL: https://api.semanticscholar.org/CorpusID:61218849 (cit. on p. 3).

[9] T. D. Chandra and S. Toueg. "Unreliable failure detectors for reliable distributed systems". In: *Journal of the ACM (JACM)* 43.2 (1996), pp. 225–267 (cit. on pp. 4, 12).

[10] M. Castro, B. Liskov, et al. "Practical byzantine fault tolerance". In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186 (cit. on pp. 4, 15).

[11] I. Stoica et al. "Chord: a scalable peer-to-peer lookup protocol for Internet applications". In: *IEEE/ACM Transactions on Networking* 11.1 (2003), pp. 17–32. DOI: 10.1109/TNET.2002.808407 (cit. on p. 7).

[12] A. Rowstron and P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. Vol. 2218. Springer Berlin Heidelberg, 2001-11. Chap. Middleware 2001, pp. 329–350. ISBN: 978-3-540-45518-9. URL: https://www.microsoft.com/en-us/research/publication/pastry-scalable-distributed-object-location-and-routing-for-large-scale-peer-to-peer-systems/ (cit. on p. 8).

[13] A. Das, I. Gupta, and A. Motivala. "SWIM: scalable weakly-consistent infection-style process group membership protocol". In: *Proceedings International Conference on Dependable Systems and Networks*. 2002, pp. 303–312. DOI: 10.1109/DSN.2002.1028914 (cit. on p. 9).

[14] J. Leitao, J. Pereira, and L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast". In: *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. DSN '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 419–429. ISBN: 0-7695-2855-4. DOI: 10.1109/DSN.2007.56. URL: https://doi.org/10.1109/DSN.2007.56 (cit. on p. 10).

[15] M. Jelasity, A. Montresor, and O. Babaoglu. "T-Man: Gossip-based Fast Overlay Topology Construction". In: *Comput. Netw.* 53.13 (2009-08), pp. 2321–2339. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2009.03.013. URL: http://dx.doi.org/10.1016/j.comnet.2009.03.013 (cit. on p. 11).

[16] J. Leitão et al. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 99.PrePrints (2012). ISSN: 1045-9219. DOI: http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.29 (cit. on p. 11).

[17] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design (International Computer Science)*. 4th rev. ed. Addison-Wesley Longman, Amsterdam, 2005. ISBN: 0321263545 (cit. on p. 12).

[18] H. Sukhwani et al. "Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)". In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 2017, pp. 253–255. DOI: 10.1109/SRDS.2017.36 (cit. on p. 15).

[19] S. Zhang and J.-H. Lee. "Analysis of the main consensus protocols of blockchain". In: *ICT Express* 6.2 (2020), pp. 93–97. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2019.08.001. URL: https://www.sciencedirect.com/science/article/pii/S240595951930164X (cit. on p. 16).

[20] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (2009-05). URL: http://www.bitcoin.org/bitcoin.pdf (cit. on p. 17).

[21] C. Badertscher et al. "Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 913–930. ISBN: 9781450356930. DOI: 10.1145/3243734.3243848. URL: https://doi.org/10.1145/3243734.3243848 (cit. on p. 17).

[22] G. Pititto. "The gasper protocol: A proof of stake era for ethereum". PhD thesis. Politecnico di Torino, 2022 (cit. on p. 17).

[23] J. Kwon. "Tendermint : Consensus without Mining". In: 2014. URL: https://api.semanticscholar.org/CorpusID:52061503 (cit. on p. 17).

[24] G. Lee. *EOS.IO Technical White Paper v2*. 2018. URL: https://academy.bit2me.com/wp-content/uploads/2021/05/eos-whitepaper.pdf (cit. on p. 18).

[25] T. Network. *TRON: Advanced Decentralized Blockchain Platform*. 2025. URL: https://tron.network/static/doc/white_paper_v_2_1.pdf (cit. on p. 18).

[26] S. L. Dan Larimer and C. Hoskinson. *BitShares 2.0: The Decentralized Exchange*. 2023. URL: https://github.com/bitshares/whitepaper (cit. on p. 18).

[27] L. Stoykov, K. Zhang, and H.-A. Jacobsen. "Vibes: fast blockchain simulations for large-scale peer-to-peer networks". In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*. 2017, pp. 19–20 (cit. on pp. 20, 23, 24).

[28] C. Faria and M. Correia. "BlockSim: blockchain simulator". In: *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2019, pp. 439–446 (cit. on pp. 21, 23, 24).

[29] H. Yajam, E. Ebadi, and M. A. Akhaee. "JABS: A Blockchain Simulator for Researching Consensus Algorithms". In: *IEEE Transactions on Network Science and Engineering* (2023) (cit. on pp. 23, 24).

[30] M. Alharby and A. van Moorsel. "Blocksim: An extensible simulation tool for blockchain systems". In: *Frontiers in Blockchain* 3 (2020), p. 28 (cit. on pp. 23, 24).

[31] Y. Aoki et al. "Simblock: A blockchain network simulator". In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2019, pp. 325–329 (cit. on pp. 23, 24).