Miguel Cerdeira Alves

Bachelor in Computer Science and Engineering

Consensus Protocols Environments and Specifications

Dissertation plan submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
⟨**Computer Science and Engineering**⟩

Adviser: António Ravara,
Associate Professor, NOVA School of Science and Technology

Co-adviser: Marco Giunti,
Researcher, NOVA School of Science and Technology

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

⟨**February**⟩, ⟨**2021**⟩

Abstract

The dissertation must contain two versions of the abstract, one in the same language as the main text, another in a different language. The package assumes that the two languages under consideration are always the main language and English. And if the main language is English, it assumes English and Portuguese. You may change this behaviour by adding

```
\abstractorder(<MAIN_LANG>):={<LANG_1>,...,<LANG_N>}
```

e.g.,

```
\abstractorder(de):={de,en,it}
```

The package will sort the abstracts in the appropriate order. This means that the first abstract will be in the same language as the main text, followed by the abstract in the other language, and then followed by the main text. For example, if the dissertation is written in Portuguese, first will come the summary in Portuguese and then in English, followed by the main text in Portuguese. If the dissertation is written in English, first will come the summary in English and then in Portuguese, followed by the main text in English.

The abstract should not exceed one page and should answer the following questions:

- What's the problem?
- Why is it interesting?
- What's the solution?
- What follows from the solution?

Keywords: Keyword 1, Keyword 2, Keyword 3, ...

Resumo

Independentemente da língua em que está escrita a dissertação, é necessário um resumo na língua do texto principal e um resumo noutra língua. Assume-se que as duas línguas em questão serão sempre o Português e o Inglês.

O *template* colocará automaticamente em primeiro lugar o resumo na língua do texto principal e depois o resumo na outra língua. Por exemplo, se a dissertação está escrita em Português, primeiro aparecerá o resumo em Português, depois em Inglês, seguido do texto principal em Português. Se a dissertação está escrita em Inglês, primeiro aparecerá o resumo em Inglês, depois em Português, seguido do texto principal em Inglês.

O resumo não deve exceder uma página e deve responder às seguintes questões:

- Qual é o problema?
- Porque é que ele é interessante?
- Qual é a solução?
- O que resulta (implicações) da solução?

E agora vamos fazer um teste com uma quebra de linha no hífen a ver se a LaTeX duplica o hífen na linha seguinte...

zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz comentar--lhe zzz zzzz zzz zzzz

Sim! Funciona! :)

Palavras-chave: Palavra-chave 1, Palavra-chave 2, Palavra-chave 3, ...

Contents

Chapter 1

Introduction

1.1 Context

- brief overview of what blockchain is and why it is relevant
- different aspects involved in developing blockchain technologies
- current challenges (verification, validation, simulation, DSLs for clear and concise presentation of the protocols)

1.2 Problem

- dividing blockchain protocols into essential components, to better abstract and extract their commonalities and key differences
- the need for blockchain simulation
  - a blockchain protocol is composed of several underlying components/algorithms, and each of which may have several parameters that can influence the performance and execution of the overall system. It is therefore important to be able to reason about the choice of these parameters before deploying to production
  - finding the fundamental aspects and the best building blocks to structure the presentation of the protocols (at the right level of abstraction, to allow reusability) and develop an extensible blockchain simulator

1.3 Goal

- modular and extensible blockchain simulator, providing the ability to make changes to protocols, and the ability to simulate different families of protocols
- ability to parameterize the different components (underlying algorithms) of the protocols
- structured according to the 5-component framework presented in (Survey Of Distributed Consensus Protocols for Blockchain Networks)
- ability to perform some level of validation/evaluation on the simulator's results?
- study the evolution of the protocols when adjusting different parameters (the goal is not to accurately simulate a real system, but rather to support decisions with respect to new versions/protocols)
- the intended simulator should provide a qualitative evaluation of the protocol, rather than evaluate its efficiency
- should provide researchers results that allow them to reason about certain implementation details
  - how many adversaries are supported
  - "fairness" when handling stakeholders with different weights

Chapter 2

Background

This chapter presents core concepts regarding consensus protocols, blockchain protocols and simulation. It also describes three consensus and blockchain protocols that will be further addressed in chapter 4.

2.1 Consensus Protocols

Consensus protocols enable coordination between multiple machines, allowing them to maintain a replicated state and are therefore crucial in building reliable, large-scale distributed systems.

These protocols are widely used in the context of state machine replication, where different processes execute the same operations on the same state.

It is expected that a consensus protocol ensures the following properties [8]:

1. Termination - eventually, every correct process accepts a value.
2. Agreement - all correct processes accept the same value.
3. Integrity - if all correct processes proposed the same value, then all correct processes accept that same value. A stronger integrity constraint for Byzantine fault tolerant consensus is: if a correct process accepts a value v, then v must have been previously proposed by some correct process.

We will now present Single-Decree Paxos [15], that allows different machines to agree on a single value out of several proposed values. Then we will present Raft [14][17] and Multi-Decree Paxos [14][18][6], which introduce the concept of a replicated log of operations and ensure that all correct processes will execute these operations in the same order on the same state.

2.1.1 Single-Decree Paxos

The Single-Decree Paxos protocol [15] is used to achieve consensus on a single value, and it has no notion of a replicated log. Each entity involved in the protocol is called a peer, and each peer can assume one, or multiple, of the following roles: Proposer, Acceptor and Learner. The protocol progresses through the exchange of messages between peers as demonstrated, in a simplified way, in figure x. The description of the peer's behavior will be divided among the different roles, to improve readability.

When a peer receives a value from a client, it begins behaving like a Proposer. A Proposer executes the following steps [15]:

1. Sends a PREPARE message to all the other peers, which contains the proposal number that it intends to use in a future proposal.

2. Waits for the responses to it's PREPARE message, which are called PROMISES. Upon receiving a PROMISE from a majority of peers, the Proposer proceeds to send a PROPOSE message to all the peers - note that if any of the received PROMISES contained the information of an already accepted value, this Proposer will have to propose that same accepted value.

3. Finally, if the Proposer receives an ACCEPT response from a majority of peers, then it has achieved consensus on it's proposal and can inform the Learners - implementation dependent - and the client.

An Acceptor keeps a record of the received PREPARE and PROPOSE messages, and whether or not a value was already accepted. It's behavior consists in processing messages from Proposers and sending the appropriate responses.

Upon receiving a PREPARE message, an Acceptor will respond in one of three ways [8]:

1. If the proposal number in the PREPARE message is the largest the Acceptor has seen so far and it hasn't accepted values yet, the Acceptor will respond with a PROMISE message containing the received proposal number, and from that moment forward the Acceptor will ignore any messages that contain a smaller proposal number.

2. If the proposal number is not the largest the Acceptor has seen so far and it hasn't accepted any values yet, it will simply ignore the PREPARE message.

3. If the Acceptor has already accepted a value, it will respond with a PROMISE message containing the highest proposal number it has seen so far, and the value that it has accepted.

Upon receiving a PROPOSE message, an Acceptor will respond in one of two ways [15]:

1. If it hasn't accepted any proposals yet, and the proposal number is larger than or equal to the largest proposal number it has seen so far, the Acceptor will accept the value and will respond with an ACCEPT message.

2. If it has already accepted a value, or it has seen a larger proposal number, the Acceptor will respond with a REJECT message or simply ignore the proposal - implementation dependent.

Finally, the Learner role represents the entity that will "learn" the value for which consensus was reached. In real systems the Learners are, for example, databases. Therefore, in the articles that describe the protocol, there is very little information about the Learner's behavior.

figure demonstrating the sequence of messages that leads to progress in the protocol

It is important to note that, in the presence of concurrent proposals, the Single-Decree Paxos protocol may not terminate as different proposers might send consecutive PREPARE messages with higher proposal numbers for an indefinite amount of time. However, it does guarantee that if consensus is reached, all Learners will "learn" that same agreed upon value.

2.1.2 Raft

The Raft protocol [17] is used in the context of log replication, ensuring consensus on the order in which the operations contained in the log entries will be applied. Each entity involved in the protocol is called a peer, and in any given instant a peer can have one of three roles: Follower, Candidate or Leader.

Each peer stores a term and a log. The term is used alongside the size of the log to keep track of how up-to-date each peer is. The base structure of the log can be seen in the figure below:

figure showing the log and log entries' structure

The protocol can be implemented using message exchange or RPCs (Remote Procedure Calls).

The following are the descriptions of each role's behavior, as presented in the official paper [17]:

When a peer has the Follower role, it's behavior is mostly based on the execution of RPCs, with a few exceptions.

Upon receiving a client request, a Follower redirects that message to the current leader of the protocol.

When a Candidate calls a RequestVote RPC on a Follower, the result will correspond to the Follower's vote. This vote will be positive if the Candidate's term and log entries are up-to-date, and the Follower hasn't voted in favor of another candidate. Otherwise, the vote will be negative.

A Leader can execute AppendEntries RPCs on all Followers, not only to send them new log entries, but also to inform them that there is an active Leader. The execution of this RPC on a Follower will succeed if the sent log entries are successfully appended

to the Follower's log. The execution fails if the Follower is missing some previous log entries or if it detects that the Leader is outdated - if the Leader's term is smaller than the Follower's term. If the RPC's execution succeeds, the Follower will also compare it's commit index to the Leader's, apply the newly committed operations, if there are any, and the RPC's result will include the index of the last entry in the Follower's log.

Finally, a Follower will change its role to Candidate when a heartbeat timeout occurs - when the Follower hasn't received a message from the valid Leader recently.

When a peer changes its role to Candidate, the election process begins. The Candidate will increment its current term, and will execute RequestVote RPCs on every other peer.

Upon receiving a negative result from a RequestVote RPC, the Candidate will check if that follower's term is larger than his current term and if that is true, then the Candidate will revert its role back to Follower since there is another peer more up-to-date than him.

If the Candidate receives a majority of positive results from the RequestVote RPCs, the Candidate will change its role to Leader.

The Leader is in charge of executing AppendEntries RPCs on all peers periodically, regardless of whether or not there are new entries in its log, because these messages also serve the purpose of informing the peers that the Leader did not fail.

When an AppendEntries RPC is unsuccessful, the Leader will decrement the index of the next message that needs to be sent to that peer, to try and solve the conflict. Worst case scenario, the conflict is at the start of the log and therefore the request will only succeed when the Leader decrements the index to be sent to 0.

Every time an AppendEntries RPC is successful, the Leader will check what is the latest entry that it can commit, which will be the entry with the largest index that is replicated in the log of a majority of peers.

Upon receiving any message that contains a term larger than its own, the Leader will revert its role to Follower since that means there is another Leader more up-to-date.

The Raft protocol guarantees that peers will reach a consensus on the order of the entries in the log and execute those operations in the same order, however it does not guarantee that all client requests will be added to the log, as some might get lost during leader failures.

*diagram with the role changes ? *

2.1.3 Multi-Decree Paxos

The Single-Decree Paxos protocol describes how to reach a consensus on a single value however, in a majority of scenarios, this is not enough to maintain a proper state of a system. Multi-Decree Paxos is essentially a sequential execution of multiple Single-Decree Paxos rounds, to achieve log replication.

There are several papers that present somewhat different descriptions of Multi-Decree Paxos, such as [**paxos_comple**], [6] and [14]. For now, we will present the protocol as described in "Paxos Made Live - An Engineering Perspective" [6].

The state kept by each peer is very similar to the state kept in Single-Decree Paxos, with the addition of the log entries and the identity of the last known leader.

At the start of the protocol, one peer is elected to act as the coordinator (also known as leader or distinguished proposer). This election can be done through one round of Single-Decree Paxos, where a peer generates a new sequence number and sends a PREPARE message to the remaining peers. If he receives a majority of valid PROMISES then he can start acting as the coordinator. Note that to ensure no out-of-date peers, the PREPARE message must include the peer's log.

The coordinator can then receive requests from clients or from the other peers, selecting one of those entries to be broadcasted. This is the PROPOSE message sent in Single-Decree Paxos. This message will now also include the coordinator's log, so that peers that failed in a previous iteration of the protocol may catch up to the current state of the system.

Once a majority of replicas acknowledge the coordinator - ACCEPT his proposal - consensus is reached, and the coordinator proceeds to send a COMMIT message to notify the replicas.

The previously mentioned steps are then repeated. In practice, the election step only needs to happen again if the current coordinator fails. This failure detection can be achieved in a similar way as in Raft, by leveraging timeouts and heartbeats.

Like Raft, Multi-Decree Paxos guarantees that peers will reach a consensus on the order of the entries in the log, but in the occasion of a coordinator failure some client requests may not be added to the log.

2.2 Blockchain Protocols

Blockchain, as the name suggests, is an append-only data structure composed of a chain of blocks. At the very least, each block contains a cryptographic hash that identifies the previous block in the chain.

A blockchain protocol is the composition of steps executed by each node in a network to maintain a replicated copy of a blockchain among multiple nodes.

Blockchain protocols often function under the assumption that nodes do not necessarily trust each other, however they still need to reach a consensus on the ordering of the blocks in the blockchain.

The object of the consensus is a chain of blocks, as depicted in figure x. However, in practice, it is not uncommon for the data structure stored by each node to be a tree of

insert the two figures here

blocks (figure y) as forks may occur, which will be further explained in later sections.

There are two main types of blockchain networks, permissioned and permissionless.

A permissioned blockchain, also referred to as a private blockchain, is a closed network where only authorized nodes can join and participate in the blockchain protocol. This does however require a centralized authority that regulates who can join the network. An example of a permissioned blockchain is Hyperledger Fabric [3].

A permissionless blockchain, also referred to as a public blockchain, allows any node to join the network and execute the blockchain protocol, and is therefore a fully decentralized environment. It is common for these blockchains to employ some type of incentive mechanism to strengthen its security. Some examples of permissionless blockchains are Bitcoin [16], Ethereum [4] and Algorand [13].

Several permissionless blockchains can also execute on permissioned environments, often more efficiently and securely.

2.2.1 Five-Component Framework

To further explain blockchain protocols, we will first describe the five-component framework presented in "A Survey of Distributed Consensus Protocols for Blockchain Networks" [21], as it is the abstraction mechanism that we adopted to specify these protocols.

The authors identify five core components of blockchain protocols: block proposal, information propagation, block validation, block finalization and incentive mechanism.

Block Proposal

The block proposal component encompasses how each node generates a new block, and the corresponding proof of generation.

Examples of block proposal mechanisms include proof of work (PoW), proof of stake (PoS) and proof of authority (PoA), among others.

Information Propagation

The information propagation component relates to how nodes communicate with each other in the network for sending and receiving transactions, blocks, etc.

Gossiping, broadcast and flooding are examples of information propagation mechanisms.

Block Validation

Block validation includes all operations related to verifying the validity of a propagated block, by checking the corresponding generation proof and the validity of the block's content.

For example, in proof of work protocols, verifying the generation proof of a block usually consists in verifying the block's hash and the chain it extends, whereas in proof of stake protocols, verifying the generation proof consists in checking if the node that proposed the block is in fact eligible to do so.

Block Finalization

The block finalization component encompasses how the nodes in the network reach an agreement on the acceptance of validated blocks.

Some protocols use block finalization mechanisms that involve communication between nodes, such as Byzantine Fault Tolerant agreement algorithms and checkpointing, while others use local finalization mechanisms such as the longest-chain rule and the Greedy Heaviest Observed Subtree (GHOST) rule.

Incentive Mechanism

The incentive mechanism includes the protocol's functionalities that promote the honest participation of nodes in the network, through block creation rewards and transaction fees, for example.

2.2.2 Bitcoin

Bitcoin [16] is a permissionless blockchain network that manages a decentralized digital currency (which is also referred to as bitcoin), allowing online payments to be sent directly between two entities, without the need for a third party central authority. Honest peers work together to make the system trustworthy by validating transactions, creating blocks through proof of work and appending them to the blockchain.

The following is a description, divided according to the five component framework [21], of the protocol executed in the bitcoin network [16] [9].

Block Proposal

Transactions are broadcasted through the network. Nodes running mining software validate these transactions, pack them into a block and proceed to compute a nonce that, when hashed together with the block's header, produces a hash that begins with a predefined number of zero bits (the amount of zero bits is periodically adjusted to allow an average of 6 new blocks per hour). This nonce is the proof object of this protocol and it is hard to compute, but rather simple to verify.

The described process is referred to as proof of work.

Information Propagation

Each block or transaction has an origin node where it is first created. After creation, this node immediately sends the corresponding data to its neighbours which is then propagated to the entire network via the gossip protocol.

In order to avoid sending transactions and blocks to nodes that have already received them from others, they are not forwarded directly. Once a node verifies a received block or transaction, it sends an inv message to neighbour nodes. This message contains a set of transaction and block hashes that the node has verified, and upon receiving an inv message other nodes can request transactions and blocks that they don't have by sending a getdata message to the sender of the inv message.

Block Validation

Upon receiving a block, a node validates the transactions and the proof associated with the block. If the transactions are valid, and if the nonce indeed produces value with the required number of zero bits when hashed together with the block's header then the block is accepted and added to that node's local chain.

Block Finalization

Nodes consider the longest chain in the blocktree to be the correct one, and will work towards extending it. Forks may occur if two nodes simultaneously broadcast different blocks that extend the same chain, leading different subsets of nodes to extend different chains. Eventually this fork will be solved when one chain becomes longer than the other and all nodes will extend a common chain once again.

When a block is sufficiently deep in the chain, it can be considered final with a high probability, since the more blocks are built on the chain that extend it, the more work would be required to alter the chain.

Incentive Mechanism

A node is incentivized to mine a valid block, as he not only receives transaction fees from the transactions that get included in the block, but also a fixed amount of bitcoins as a reward for the creation of the block itself which is included in the block as a special transaction that creates those bitcoins.

2.2.3 Algorand

Algorand [13] [7] is a blockchain network that manages a cryptocurrency, designed to confirm transactions on the order of one minute. At its core, Algorand uses a Byzantine Fault Tolerant agreement protocol, that is not only scalable to many users, but also allows the network to reach consensus on a new block with low latency and with a low risk of creating forks. The protocol also satisfies user replaceability, allowing for a different subset of users to participate in different steps of the agreement protocol, providing tolerance against targeted denial of service attacks.

The following is a description, divided according to the five component framework [21], of the protocol executed in the algorand network [13] [7].

Block Proposal

Nodes collect pending transactions that they learn about into a block, in case they are chosen to propose the next block in the chain.

The nodes then proceed to execute cryptographic sortition that allows them to privately check if they are selected to propose a block. Cryptographic sortition ensures that only a small set of nodes are selected at random, with consideration to their weight (amount of money the node holds in the system). If a node is selected to propose a block,

cryptographic sortition also produces a priority (used as tie breaker when several nodes are selected) and a proof of the node's ability to propose and of its priority.

The described process of selecting a proposer randomly, based on its weight (stake), is referred to as proof of stake.

Information Propagation

Similarly to Bitcoin, new transactions are propagated through the network via the gossip protocol, as well as the messages and blocks sent during the agreement protocol.

Algorand avoids forward loops by not allowing nodes to forward the same message twice, and mitigates pollution attacks by selecting neighbour nodes based on how much money they hold and each node only relays messages after validating them.

Block Validation

Upon receiving a block, a node will validate the transactions contained in the block, and will perform a proposer eligibility check to ensure that the node that sent the block was indeed selected to do so.

Block Finalization

Cryptographic sortition may select multiple nodes to propose a block in a given round. To reach consensus on a single block to be appended to the chain, the agreement protocol is executed.

Each node begins the agreement protocol with the block with highest associated priority they have received. Each node then executes the following steps [7], in sequential rounds (also referred to as periods), until consensus is reached:

1. Execute cryptographic sortition to check whether the node was selected to be part of the committee for that period. Note that a new period begins when a node receives a majority of next-votes for some block in the previous period.

2. If the node was indeed selected to be part of the committee, it broadcasts (to all nodes, not just the ones that are part of the committee) the block with highest priority it has seen (if round=1) or the block that received the most next-votes in the previous round (if round>1), as well as the proof it belongs to the committee. This initially broadcasted block is known as the node's starting block.

3. If the node sees a majority of next-votes for an empty block, it sends a soft-vote for the non-empty block with most votes that it has seen. Otherwise, if the node sees a majority of next-votes for one non-empty block, it sends a soft-vote for that same block.

4. If the node sees a majority of soft-votes for a non-empty block, then it sends a cert-vote for that block.

5. If the node has certified some value in the current period, it sends a next-vote for that same value. Else, if it has seen a majority of next-votes for an empty block, it

also sends a next-vote for an empty block. Otherwise, it sends a next-vote for its starting block.

6. If the node sees a majority of soft-votes for a non-empty block, then it sends a next-vote for that block. Otherwise, if it sees a majority of next-votes for an empty block, and it has not certified a block in this period, it sends a next-vote for an empty block.

7. If a node sees a majority of cert-votes for one block for the same period, it sets that block as the output of its agreement protocol. The set of those cert-votes form a certificate for the block, meaning consensus was reached and it can be appended to the blockchain.

Note that regardless of whether or not a node belongs to the committee, it keeps track of the messages (votes) exchanged by the committee members. This is important because it allows committee members to be replaced every period, while maintaining the progress achieved in previous periods which helps protect the network against targeted denial of service attacks.

Incentive Mechanism

Nodes receive a reward upon creating a block that gets successfully added to the blockchain.

2.2.4 Ethereum

Ethereum [4] [20] is a permissionless blockchain network that manages a cryptocurrency, ether, and supports a built-in Turing-complete programming language that users can leverage to create smart contracts and decentralized applications that can specify state transition functions. The execution of these pieces of code can be triggered by messages and transactions.

The following is a description, divided according to the five component framework [21], of the protocol executed in the Ethereum network [4] [20]. For clarification, this is a description of the Ethereum 1 protocol. An overview of the key functionalities introduced by the currently deployed version of Ethereum 2 [x] will be presented afterwards.

add ethereum2 citation

Block Proposal

Miner nodes gather received transactions and pack them into a block. Since transactions may trigger state transition functions, it is necessary for the block to also include the most recent state - the state reached after the miner applied the transactions contained in the block, which may trigger contract code that changes the state.

The miner also includes 0 or more stale blocks (a visualization of stale blocks can be seen in figure x), also referred to as orphan or uncle blocks, into the created block. The

included uncle blocks must be different than all uncles included in the previous blocks that belong to the chain that is being extended.

Finally, nodes execute a process similar to Bitcoin's proof of work. The difference is that the resulting proof is a nonce that when hashed together with the block header and a dataset obtained by downloading the full blockchain, must be lower than a target value.

Information Propagation

Ethereum nodes communicate using a gossip protocol. It is worth highlighting that the messages are exchanged using the RLPx Transport protocol, which allows nodes to send encrypted messages.

Block Validation

When a node receives a block, it will start by verifying if it extends a valid chain. It will then validate the contained transactions, gas limit and block header, and finally will verify if the nonce generated is a valid proof of work.

Then it will apply the transactions contained in the block. If any transaction returns an error or the total gas consumed exceeds the limit, the validation fails. Otherwise, if all transactions successfully terminate, it will check whether or not the state reached after their execution is the same as the one contained in the block.

Block Finalization

In the Ethereum network, nodes follow the Greedy Heaviest Observed Subtree (GHOST) rule. This is similar to the longest chain rule, where nodes work to extend the longest chain, but it also considers the uncle blocks in the calculation of which chain is the "longest".

As with the longest chain rule, when a block is sufficiently deep in the chain, it can be considered final with a high probability.

Block Finalization in Ethereum 2

The current state of Ethereum 2 is a hybrid proof of stake and proof of work protocol. It builds on top of the Ethereum 1 proof of work implementation, adding a checkpointing mechanism to block finalization, using the CasperFFG protocol [5].

CasperFFG [5] allows nodes to maintain a checkpoint tree and vote on a chain of checkpoints, resolving forks and finalizing the blocks on the agreed chain.

Every time a node appends a block to the chain, with that block's height being a multiple of 100, that block is considered a valid checkpoint block.

Nodes assume the role of validators. Each validator has an associated deposit of ether, its stake. Validators broadcast vote messages, containing two checkpoints and their respective heights.

Each node's checkpoint tree also includes a vote count, where the votes are weighted according to the voter's stake. Consensus on a chain is reached as follows:

1. If an ordered pair of checkpoints (a,b) has received 2/3 of weighted votes, it is called a supermajority link.

2. A checkpoint a is justified if it is the checkpoint tree's root, or if there is a supermajority link (b,a) such that b is justified.

3. A checkpoint a becomes finalized if it is the checkpoint tree's root, or if it is justified and there is a supermajority link (a,b) where b is a direct child of a.

4. Honest validators always cast votes to extend the chain with the chain with the highest justified checkpoint.

Once a checkpoint becomes finalized, all the blocks it extends not only in the checkpoint tree, but also in the overall block tree, can be considered final.

Incentive Mechanism

Nodes get rewarded for successfully creating a block that gets appended to the main chain and for the associated transaction fees. Besides that, a node also is also rewarded for the creation of stale blocks that get included as uncles in blocks that extend the main chain.

figure showing what a stale/uncle block is

2.3 Simulation

the goal isn't to simulate the protocol's behavior when compared to reality, but to simulate the influence of parameter values in the overall behavior of the protocol

Chapter 3

State of the Art

Blockchain simulation isn't necessarily a new concept, however it is still rather unexplored. Although there have been several simulators developed to test and validate the performance of specific blockchain systems, such as Blockchain Simulator [12], Bitcoin Network Simulator [2] and eVIBES [10], only a small number of simulators have been developed with the goal of providing extensibility to simulate several families of blockchain protocols.

We will describe existing extensible blockchain simulators, namely BlockSim [11], BlockSim [1] and VIBES [19], since their scope is similar to this work.

3.1 BlockSim: Blockchain Simulator

BlockSim [11] is a simulation framework developed in Python that assists in the design, implementation and evaluation of blockchain protocols. It provides insights regarding how a blockchain system operates and allows the examination of certain assumptions on the chosen simulation models, without incurring in the overhead of developing and deploying a real network.

BlockSim uses probabilistic distributions to model random phenomena, such as the time taken to validate a block and network latency to deliver a message, among others. These probability distributions can be specified by the user, in configuration files.

The core of the simulator is its Discrete Event Simulation Engine, providing primitives for event generation, scheduling and execution, and allowing different processes to communicate with each other through events. The Transaction and Node Factories are responsible for creating batches of random transactions and instantiating nodes, respectively. Finally there is the Monitor, whose purpose is to capture metrics during the simulation.

Blocksim uses different detached layers to provide the needed abstraction level to support different blockchain protocols, namely:

- Node layer - specifies the responsibilities and behaviour of a node.
- Consensus layer - specifies the algorithms and rules for a given consensus protocol.
- Ledger layer - defines how a ledger is structured and stored.
- Transaction and block layer - specify how information is represented and transmitted.
- Network layer - establishes how nodes communicate with each other.
- Cryptographic layer - defines what cryptographic functions will be used and how.

For each of these layers, a base model is provided which the user can extend, and BlockSim also provides examples of how these models were extended to simulate Bitcoin and Ethereum1.

3.2 BlockSim: An Extensible Simulation Tool for Blockchain Systems

BlockSim [1] is a discrete-event simulation framework developed in Python, with the purpose of exploring the effects of configuration, parameterization and design decisions on the behavior of blockchain systems, by providing intuitive simulation constructs.

BlockSim has three main modules: the Simulation Module, the Base Module and the Configuration Module. The Simulation Module is composed of four classes - Event, Scheduler, Statistics and Main - and is in charge of setting up the simulation, scheduling events and computing simulation statistics. The Configuration Module acts as the main user interface, where users can select and parameterize the models used in the simulation. Finally, the Base Module consists in the base implementation of the blockchain protocol that will be simulated.

To support a variety of different blockchain protocols, the Base Module is divided according to the following abstraction layers:

- Network layer - defines the blockchain's nodes, their behavior and the underlying peer-to-peer protocol to exchange data between them.
- Consensus layer - defines the algorithms and rules adopted to reach agreement about the current state of the blockchain ledger.
- Incentives layer - defines the economic incentives mechanisms adopted by a blockchain to issue and distribute rewards among the participating nodes.

For each of these abstraction layers, an extendable implementation is provided.

3.3 VIBES: Fast Blockchain Simulations for Large-scale Peer-to-Peer Networks

VIBES [19] is a message-driven blockchain simulator developed in Scala, with the goal of enabling fast, scalable and configurable blockchain network simulations on a single computer.

Architecturally, VIBES is composed of Actors. These Actors can have one of three main types: Node, Reducer or Coordinator.

A Node follows a simple protocol to replicate the behavior of a blockchain network, whether it is a full-node or a miner node. The Reducer, once the simulation ends, gathers the state of the network and produces an output with the simulation results that the user of the simulator can process.

The Coordinator is essential for providing scalability and speed. This is done by acting as an application-level scheduler that fast-forwards computing time. In practice, each Node estimates how long it will take to complete a certain task, such as mining a block, and asks the Coordinator to fast-forward the entire network to that point in time. Once the Coordinator has gathered fast-forward requests from all Nodes, it will fast-forward the entire network to the time referred by the request with the earliest timestamp thus guaranteeing a correct order of execution of tasks.

3.4 Critical Analysis

	Adversarial Behaviour	Abstraction Layers	Proof of Stake	Proof of Work
VIBES [19]	not modeled	none	not modeled	bitcoin
BlockSim [11]	not modeled	node layer consensus layer ledger layer transaction layer block layers network layer cryptographic layer	not modeled	bitcoin ethereum
BlockSim [1]	not modeled	network layer consensus layer incentives layer	not modeled	bitcoin ethereum

Table 3.1: Overview of the provided functionalities of the existing simulators, in regard to properties we consider relevant. If adversarial behaviour is modeled, what are the adopted abstraction layers for extending the simulator to other protocols, and what proof of stake and proof of work protocols have been modeled.

VIBES [19] is scalable, fast, and is capable of being extended to blockchain protocols other than bitcoin, although requiring some changes to be made to the simulation engine

[10]. However, it does not follow any abstraction mechanisms to specify the protocols to be simulated, which would increase it's extensibility.

Neither BlockSim [11] nor BlockSim [1] model adversarial behaviour, nor do they provide a concrete foundation for simulating families of blockchain protocols based on proof of stake.

Both BlockSim [11] and BlockSim [1] define different abstraction layers to facilitate the simulation of different blockchain protocols. However, we believe that using finer-grained abstraction layers is advantageous.

Hence, this thesis intends to fill this gap by providing a simulator that models adversarial behavior, models more families of protocols besides proof of work and defines concrete, finer-grained abstraction layers for modeling blockchain protocols, thus achieving better code structure and reusability, and enhancing the readability and understandability of the simulated protocolos.

Chapter 4

Use Cases

1. paxos and raft in rust
2. explain why ocaml was the language of choice, as opposed to Rust
3. modularizing blockchain protocols
    - nakamoto, algorand and ethereum
    - maybe include code-snippets, for example, of the top-level modular function

Chapter 5

Work Plan

5.1 the different steps/phases of development

One paragraph per topic, explaining what will be done, and the timeframe to do it in.

- ability to simulate multiple nodes
- network operations (communication between nodes)
- base implementation for PoW
- base implementation for PoS
- base implementation for blocktree/blockchain
- base implementation of a node
- way of gathering metrics/results from the different nodes, and producing some results
- simulating nakamoto
- simulating algorand
- simulating ethereum
- simulating tezos

Bibliography

[1] M. Alharby and A. van Moorsel. "BlockSim: An Extensible Simulation Tool for Blockchain Systems". In: *Frontiers Blockchain* 3 (2020), p. 28. DOI: 10.3389/fbloc.2020.00028. URL: https://doi.org/10.3389/fbloc.2020.00028.

[2] L. Alsahan, N. Lasla, and M. Abdallah. "Local Bitcoin Network Simulator for Performance Evaluation using Lightweight Virtualization". In: *IEEE International Conference on Informatics, IoT, and Enabling Technologies, ICIoT 2020, Doha, Qatar, February 2-5, 2020*. IEEE, 2020, pp. 355–360. DOI: 10.1109/ICIoT48696.2020.9089630. URL: https://doi.org/10.1109/ICIoT48696.2020.9089630.

[3] E. Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. Ed. by R. Oliveira, P. Felber, and Y. C. Hu. ACM, 2018, 30:1–30:15. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538.

[4] V. Buterin. *Ethereum Whitepaper*. online. 2013. URL: https://ethereum.org/en/whitepaper/.

[5] V. Buterin and V. Griffith. "Casper the Friendly Finality Gadget". In: *CoRR* abs/1710.09437 (2017). arXiv: 1710.09437. URL: http://arxiv.org/abs/1710.09437.

[6] T. D. Chandra, R. Griesemer, and J. Redstone. "Paxos made live: an engineering perspective". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*. Ed. by I. Gupta and R. Wattenhofer. ACM, 2007, pp. 398–407. DOI: 10.1145/1281100.1281103. URL: https://doi.org/10.1145/1281100.1281103.

[7] J. Chen et al. "ALGORAND AGREEMENT: Super Fast and Partition Resilient Byzantine Agreement". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 377. URL: https://eprint.iacr.org/2018/377.

[8] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems - concepts and designs (3. ed.)* International computer science series. Addison-Wesley-Longman, 2002. ISBN: 978-0-201-61918-8.

[9] C. Decker and R. Wattenhofer. "Information propagation in the Bitcoin network". In: *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*. IEEE, 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704. URL: https://doi.org/10.1109/P2P.2013.6688704.

[10] A. S. Deshpande. "Design and Implementation of an Ethereum-like Blockchain Simulation Framework". MA thesis. Technical University of Munich, 2018.

[11] C. S. F. Faria. "BlockSim: Blockchain Simulator". MA thesis. Instituto Superior Técnico, 2018.

[12] A. Gervais et al. "On the Security and Performance of Proof of Work Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by E. R. Weippl et al. ACM, 2016, pp. 3–16. DOI: 10.1145/2976749.2978341. URL: https://doi.org/10.1145/2976749.2978341.

[13] Y. Gilad et al. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017, pp. 51–68. DOI: 10.1145/3132747.3132757. URL: https://doi.org/10.1145/3132747.3132757.

[14] H. Howard and R. Mortier. "Paxos vs Raft: have we reached consensus on distributed consensus?" In: *7th Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC@EuroSys 2020, Heraklion, Greece, April 27, 2020*. Ed. by A. D. Fekete and M. Kleppmann. ACM, 2020, 8:1–8:9. DOI: 10.1145/3380787.3393681. URL: https://doi.org/10.1145/3380787.3393681.

[15] L. Lamport. *Paxos Made Simple*. online. 2001. URL: https://lamport.azurewebsites.net/pubs/paxos-simple.pdf.

[16] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. online. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[17] D. Ongaro and J. K. Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*. Ed. by G. Gibson and N. Zeldovich. USENIX Association, 2014, pp. 305–319. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[18] R. van Renesse and D. Altinbuken. "Paxos Made Moderately Complex". In: *ACM Comput. Surv.* 47.3 (2015), 42:1–42:36. DOI: 10.1145/2673577. URL: https://doi.org/10.1145/2673577.

[19] L. Stoykov. "VIBES: Fast Blockchain Simulations for Large-scale Peer-to-Peer Networks". MA thesis. Technical University of Munich, 2018.

[20] G. Wood. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER*. online. 2021. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[21] Y. Xiao et al. "A Survey of Distributed Consensus Protocols for Blockchain Networks". In: *IEEE Commun. Surv. Tutorials* 22.2 (2020), pp. 1432–1465. DOI: 10.1109/COMST.2020.2969706. URL: https://doi.org/10.1109/COMST.2020.2969706.