TÉCNICO LISBOA

SCALEET: A Scalable and Performant Permissionless Blockchain

João Paulo da Costa Campos

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Rodrigo Seromenho Miragaia Rodrigues

Examination Committee

Chairperson: Prof. David Manuel Martins de Matos
Supervisor: Prof. Rodrigo Seromenho Miragaia Rodrigues
Member of the Committee: Prof. Miguel Ângelo Marques de Matos

January 2021

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor, Professor Rodrigo Rodrigues, for welcoming me as its student and for his dedicated guidance and support throughout this work. It was an amazing adventure and a delightful journey to be working with such a good professional and person.

I would also like to thank my friends for the endless support and encouragement throughout all these years. Carlos Carvalho, Tiago Romero and Sara Esperto - without you, it would not have been the same.

Thank you to my precious girlfriend, Joana Ribeiro, for all her love and friendship. Thank you for holding me tight, especially on my most vulnerable days.

Lastly, I want to thank my family, especially my parents, grandparents, sister and brother in law for helping me to grow as a person and for always being by my side during the good and bad times. To my nephews Gonçalo and Lourenço, who were born during this 5-year walk.

To each and every one of you – Thank you.

Abstract

Blockchains, which initially gained attention in 2008 as the underlying technology of Bitcoin, have become one of the most disruptive technologies of our days, by solving the problem of transferring the trust originally held by a single third-party, to a decentralized model where this trust is split among numerous entities. However, to maintain the disruption and growth of blockchains, these systems need to ensure the ability to securely scale out to thousands of participants in an open-membership setting. The design of the existing deployed blockchains encodes an implicit dichotomy on the key aspect of deciding on a total order of the transactions that form the blockchain: existing deployments have either permissionless designs that are based on Proof-of-Work (PoW) schemes, or permissioned designs based on Byzantine Fault Tolerance (BFT) consensus. In this thesis, we intend to deconstruct this dichotomy in a principled way. To this end, we present a novel blockchain design called SCALEET. SCALEET still allows for using PoW, but leverages representative committees, sharding, and BFT consensus to move PoW out from the critical path. SCALEET exhibits high levels of modularity, allowing an easy integration of newer and more performant BFT algorithms that can be used as a drop-in replacement. Finally, SCALEET also introduces a novel and fairer approach for assigning rewards. An evaluation of our experimental prototype shows that SCALEET can scale linearly in the number of participants, enabling a throughput on the order of thousands of Transactions Per Second (tps) and confirming transactions in a few seconds.

Keywords

Blockchain; Consensus; Scalability; Permissionless Setting; Committees; Sharding

Resumo

As blockchains tornaram-se uma das tecnologias mais importantes e disruptivas dos dias de hoje no contexto dos sistemas distribuídos, ao ser uma solução para o problema de transferência de confiança originalmente detida por uma só autoridade central para um conjunto de entidades. Contudo, para acompanhar o seu elevado crescimento, estes sistemas necessitam de ser capazes de escalar até um número elevado de participantes de forma eficiente e segura, mantendo ao mesmo tempo a sua característica pública, onde se mantém a filiação de membros aberta. Os designs existentes codificam uma dicotomia aquando da decisão sobre uma ordem total das transações que formam a blockchain: implementações existentes ou apresentam designs com características públicas baseados em esquemas PoW ou designs fechados através do uso de algoritmos de consenso clássicos de tolerância a falhas arbitrárias. Nesta tese, pretendemos desconstruir esta dicotomia. Para tal, apresentamos um design inovador de blockchain chamado SCALEET. O SCALEET aproveita o uso de comités, de técnicas de repartição de estado bem como uma realocação do PoW, movendo-o para fora do caminho crítico da validação de transações. O SCALEET exibe altos níveis de modularidade, permitindo assim uma fácil integração de algoritmos BFT novos e com melhor desempenho. Por fim, apresenta uma abordagem mais justa para atribuição de recompensas de participação. A avaliação do protótipo demonstrou que o SCALEET escala linearmente com o aumento do número de participantes, permitindo assim uma taxa de confirmação de transações na ordem das milhares de transações por segundo e a confirmação destas na magnitude dos segundos.

Palavras Chave

Blockchain, Consenso, Escalabilidade, Blockchains Públicas, Comitês, Fragmentação de Estado

Contents

List of Figures

List of Tables

Acronyms

BFT	Byzantine Fault Tolerance
tps	Transactions Per Second
UTXO	Unspent Transaction Output
PoW	Proof-of-Work

1 Introduction

Contents

1.1 Motivation

A blockchain is a decentralized, immutable, replicated, tamper-evident and tamper-resistant digital log. At its essential level, it provides a set of users with the ability to record transactions in it. With no centralized or controlling authority, blockchains have the potential to eliminate intermediaries by replacing them with cryptographically secure protocols, which results in a peer-to-peer network that guarantees *security*, *transparency* and *immutability*. Thus, this class of systems, which first gained attention as the underlying technology of Bitcoin [1], is being increasingly used in favor of centralized systems that need to rely on a single authority entity to properly work. Nowadays, Blockchains have a huge set of applications [2] beyond cryptocurrencies, expanding into different areas such as voting [3], social media [4] and government records [5].

One of the key design aspects of any blockchain design is the mechanism that is used to determine what is the sequence of transactions (or blocks of transactions) that forms the blockchain, or, more precisely, the protocols that allow for incrementally and securely appending a new block to the end of this chain. In this context, the design of the protocols that implement this aspect of the system specification needs to employ a set of mechanisms that are secure against participants that do not follow the prescribed protocols, and work even in an open and large-scale environment like the Internet.

The design of the blockchains that are deployed today encodes an implicit dichotomy on the above mentioned design aspect. In particular, existing blockchain deployments can be split into two groups.

The first group, including systems such as Bitcoin [1], Repucoin [6] and Btcoin-NG [7], assumes an open membership (or permissionless) deployment, and uses a design based on Proof-of-Work (PoW), where all participants can concurrently attempt to solve cryptographic puzzles in a race to append transactions. Such schemes are able to cope with Sybil attacks [8] and offer an easy way to determine which is the current (longest) blockchain, but suffer from wasting a lot of energy, having very low throughput and giving weak guarantees regarding termination. (A recent variant is a research proposal of using *Proof-of-Stake (PoS)* [9], but this suffers from the drawback that the probability of the richest stakeholders controlling the system can become very significant.)

The alternative choice in this dichotomy is geared towards systems with a closed system membership (permissioned deployments), where it is possible to enforce some form of access control to the system membership. The deployed systems of this type resort to Byzantine Fault Tolerance (BFT) consensus protocols [10], which are the basis for BFT state machine replication [11], which in turn allows for implementing arbitrary deterministic services (including the blockchain specification) in a way that tolerates arbitrary faults from a subset of the protocol participants. However, consensus protocols are very complex, involving communication among all the participating nodes, which makes them difficult to scale to a large system membership. In addition, these protocols are normally designed under the assumption of a static membership, which is not the case in an open Internet deployment. Finally, this sort of design would not work in a permissionless setting, since it raises the possibility of *Sybil* attacks, where a single entity can have multiple identities in order to gain control of the system.

Recent designs, like the one proposed in Omniledger [12], also tried to disassemble this dichotomy, but, as we will discuss in Chapter 3, they continue to present crucial limitations that we tackle in SCALEET. More specifically, they either lack a concrete solution to counter Sybil attacks or they introduce liveness issues that stem from the way they use PoW. SCALEET improves on these proposals by completely modularizing the use of PoW, which overcomes these limitations.

1.2 Contributions

The goal of this thesis is to deconstruct this dichotomy, by demonstrating, in a principled way, that is possible to build a highly scalable, secure, and performant blockchain for permissionless deployments. In particular, this is achieved through an approach that takes existing building blocks from the security and distributed systems literature, and assembles them in a judicious and novel way. One of the key insights behind this design is the fact that it is possible to use both PoW and BFT consensus, but it is crucial to carefully separate their use so that PoW is not in critical path of appending new transactions to the chain, but is still employed in a way that allows us to extract its key benefits of controlling the amount of power that each individual machine can have over the system, and providing a strong cryptographic proof over the current state of the system.

As result, this thesis makes the following contributions:

- Delivers a systematic and comprehensive analysis over the techniques that current systems use to match these properties in order to scale consensus, most of which are well-studied, existing building blocks from the security and distributed systems literature.
- Proposes a novel blockchain system, which was then materialized in a prototype called SCALEET. SCALEET combines these building blocks in a novel way, namely by leveraging the use of representatives committees and sharding, and proposing a reallocation of PoW, moving it out from the critical path, which allows us to achieve high performance. Furthermore, given the high level of modularity in the design of SCALEET, it allows for an easy integration of newer and more performant BFT algorithms that can be used as a drop-in replacement. SCALEET also introduces a novel and fairer approach for assigning rewards.
- An experimental evaluation of the prototype of SCALEET, showing that it can achieve much better performance than traditional PoW-based schemes, measured in terms of throughput and latency for executing transactions, graceful scalability as the number of system nodes increases, while also retaining the benefits of PoW that make it well-suited for open membership settings.

1.3 Document Outline

The remainder of this document is structured as follows. Chapter 2 sets the background of the work explaining the functioning of a typical blockchain. It gives a description of each main component of a blockchain as a building block to better understand the whole system.

Chapter 3 introduces the problems that emerge when one tries to strike the balance between efficiency, *scalability*, *security* and *open-membership* when designing a blockchain. It also presents and analyses several solutions that have been proposed by some authors, which constitutes the *state-of-the-art*. Finally, the limitations of the previous ideas are discussed to properly introduce ours on the next Chapter.

In Chapter 4 we propose and describe the design of SCALEET and Chapter 5 details the implementation. Chapter 6 presents the results of the experimental evaluation of SCALEET.

Finally, Chapter 7 concludes this document by summarizing our main takeaways and presenting the logical directions for future work.

2

Background

Contents

In this Chapter, the blockchain technology will be detailed. The functioning of blockchains can be better understood by examining each of their component individually as a building block. At a very high level, blockchains use well-studied technologies from computer science and cryptographic primitives together with record keeping concepts. Therefore, and having as an overarching goal this conceptual separation between the various goals and components, the rest of this Chapter is devoted to presenting and describing the functioning and the internals of a blockchain.

2.1 Blockchain Overview

Blockchains serve as a distributed ledger that represents a sequence of cryptographically signed, agreed, synchronized, distributed and replicated data called *blocks*, each one representing a set of transactions. The chain is implemented as a reverse linked list in which standard back pointers were replaced by cryptographic hash pointers. These pointers are virtual pointers since it is simply the hash of the parent block, thus also serving for integrity purposes. Apart from the first block of a blockchain, which is called genesis block and has no parent, each block has exactly one parent block, the previous block in the chain.

This technology embraces a set of useful and desirable features that are the main reason for it to be an important new building block in the design of global scale distributed systems:

- **Decentralization**: In traditional centralized transaction systems, each transaction is validated by a central trusted party, which leads to a likely performance bottleneck and necessity of trust on the central servers. In contrast, in blockchain technology there is no need for a trusted third party since transaction validation is done and agreed among a set of participants.

- **Persistence**: Each transaction that is confirmed and recorded in the ledger and spread across the network is nearly impossible to be deleted or rolled-back.

- **Pseudo-anonymity**: Each user can participate in the network with a generated address, which does not reveal the user real identity. This means that users are anonymous but their account identifier (address) is not. Perfect privacy preservation is difficult to guarantee since the values of all the transactions and balances for each address are publicly known. This aspect can be overcome if some techniques are used even if these can compromise other features, as will be discussed further in Chapter 3.

- **Auditability**: The techniques that are employed to keep track of the sequence of transactions enables easy verifiability and trackability. More details on this in Section 2.5.

2.2 Blockchain Categorization

Current blockchain networks can be categorized based on their permission model, which determines who can participate and maintain the ledger (e.g., publish and verify blocks). There are three types of systems according to this category: *permissionless* blockchains, *consortium* blockchains and *permissioned* blockchains.

- **Permissionless blockchains**: A permissionless blockchain is one where any node can join and leave freely without needing permission from any authority, meaning that (1) everyone can create transactions and expect to see them committed if they are valid and (2) everyone can participate in the consensus process. This type of blockchains rests on the opposite side of the traditional centralized systems and are generally called fully decentralized. Bitcoin [1] is an example of a permissionless blockchain.
- **Permissioned blockchains**: In opposition to permissionless blockchain, there are permissioned ones where restrictions are placed on who is allowed to participate in the network. These regulations are imposed by a single organization/authority. Gosig [13] is an example of a permissioned blockchain.
- **Consortium blockchains**: These blockchains stand on the middle of the previous two, where the consensus process is controlled by a pre-selected set of nodes. The control over the selection of the participants nodes is not granted to a single authority like in permissioned blockchains, but rather to a group. These blockchains are considered to be partially decentralized. Corda [14] is an example of a consortium blockchain.

2.3 Blockchain Participants

In a blockchain peer-to-peer network each participant is called a node. There are two main types of nodes: *full* nodes and *lightweight* nodes.

- **Full nodes**: These nodes act as servers in a decentralized network. Each of them saves the entire blockchain and their main tasks are verify transactions and propose blocks. In order words, they are responsible for maintaining the consensus with other full nodes.
- **Lightweight nodes**: They are the ones that make use of the blockchain network but do not really act as a full node. Therefore, lightweight nodes do not contribute to the security of the system because they do not keep a copy of the blockchain and do not participate in the process of verifying and validating transactions. They are used as a entry point to the network. In cryptocurrency scenarios, they are often used by wallet applications.

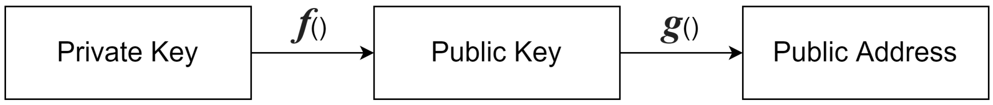

Figure 2.1: Blockchain address derivation.

2.4 Participant Addresses

Blockchain networks make use of addresses, which are a short alphanumeric string, to identify each node. An address is the hash of the user's public key and the latter is derived from the private key, as depicted in Figure 2.1. Consequently, only the private key need to be backed up, because everything else can be derived from it. The most common use of addresses are in ”to” and ”from” fields of a transaction.

There are two main reasons for the design choice behind this naming mechanism in blockchains. First, these addresses are short, thanks to their use of hashing. Second, they have interesting properties in security terms: not only the public key that can be used to authenticate data and communication becomes uniquely tied to a given node address, but also, in case the algorithm that manages the pair of keys is broken (e.g., Elliptic Curves), since the public key is hashed to obtain the node address, that key does not need to be known until a transaction is issued. As such, users still have an extra time window for keeping their belongings safe until a new algorithm is employed. The public key is only revealed when a transaction is issued because it is necessary to validate the digital signature.

2.5 Transactions

A transaction represents an interaction between users that results in some transformation on the state of the ledger. Under our example, a transaction may represent a transfer of a certain amount of currency between two users.

In cryptocurrencies, a commonly used model used to represent the relation between transactions was first referred in Bitcoin [1] as *Unspent Transaction Output (UTXO)*. This model represents a chain of ownership based on digital signatures where the owner signs the transaction, which proves the change of the ownership of some of his UTXO to other user. The total of UTXO can be seen as a set that represents all the coins in the system. Each transaction consumes some UTXO (inputs of the current transaction) from this set and produces others (outputs), representing the ownership change. An UTXO is the output of a transaction that has not been spent yet, and so, can be used as input for other transaction.

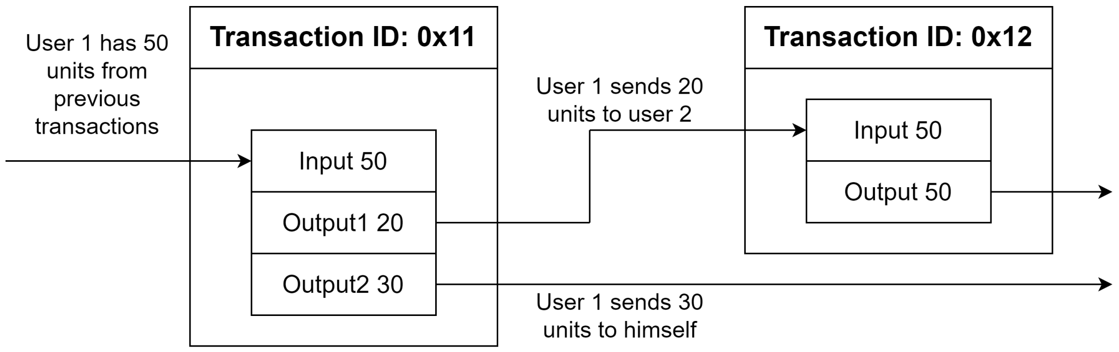
User 1 has 50 units from previous transactions
Transaction ID: 0x11
Input 50
Output1 20
Output2 30
User 1 sends 20 units to user 2
User 1 sends 30 units to himself
Transaction ID: 0x12
Input 50
Output 50

Figure 2.2: *UTXO* model.

2.6 Blocks

Publishing nodes are constantly collecting transactions that were issued by clients. After checking the validity and authenticity of these transactions, they create candidate blocks that will be proposed to be appended to the chain. A consensus algorithm is then run to agree on which is the subsequent block to be appended to the end of the chain.

A block consists of the block header and the block body. In spite each blockchain implementation defining its own data fields, most of them have, at least, the following:

- **Block header**:
  - *Block number*: The height of the block in the chain.
  - *Merkle tree root hash*: The hash value of all of the transactions present in the block's body.
  - *Timestamp*: Block's creation time as seconds in universal time since January 1, 1970.
  - *Size*: The size of the block.
  - *Parent block hash*: Hash value of the previous block. Represents the back pointer to the parent block.
- **Block body**: It is composed by a transaction counter and a list of transactions. The maximum number of transactions that a block can have depends on block size and the size of each transaction.

2.7 Consensus

The consensus algorithm is probably the most important component in allowing a blockchain to function and exist, since it is through a consensus model that the next legitimate block to be appended to the chain is established and agreed. It creates an irrefutable system of agreement between various nodes across the network while preventing exploitation of the system. Since solving consensus while maintaining an open-membership and achieving both scalability and performance is the goal of our work and it will be further discussed in Section 3, in this Section we are just going to introduce consensus in a broad sense.

The main need for consensus protocols in distributed systems comes from the need to provide resilience against failures across multiple nodes holding state (e.g, replicas of databases). This primitive is closely related with the one of *state machine replication* [11], SMR, which is a general method for implementing fault-tolerant services where the state is replicated across different nodes where clients send requests to servers, which are expected to execute the same order of requested operations (i.e., maintain a common state). In fact, to implement the latter it is necessary to implement the former because the next operation to execute against the state of the replicas need to be agreed among the nodes.

There are different variations of consensus in the literature, such as binary consensus, vector consensus, multivalue consensus, amongst others. All variations of consensus can be defined in terms of three properties: *validity*, *agreement* and *termination*. Normally, the difference between each variation is how the *validity* property is defined. As an example, the definition of the binary consensus is:

- **Validity**: If all correct processes propose the same value v, then any correct process that decides, decides v.
- **Agreement**: No two correct processes decide differently.
- **Termination**: Every correct process eventually decides.

The first two properties, *validity* and *agreement*, are *safety* properties, whereas *termination* is a *liveness* property. The notions of *safety* and *liveness* properties have been first introduced by Lamport [15]. Informally, a *safety* property expresses that ”something (bad) will never happen” during a system execution. A *liveness* property expresses that eventually ”something (good) must happen” during an execution.

These consensus protocols are described and implemented following a given communication model, as well as a failure model.

2.7.1 Communication Models

Regarding the communication model, according to the taxonomy of Dwork et al. [16], networks can be *synchronous*, *asynchronous* or *partially synchronous*.

- **Synchronous model**: In the synchronous model, there is some known finite time bound Δ in which a message can be delayed. So an adversary can only delay its delivery by at most Δ.

- **Asynchronous model**: Resting on the other side of the spectrum, there is the asynchronous model where messages may be delayed arbitrarily and do not exist any reliable bound Δ for their delay. Therefore, a adversary can delay its delivery with no bounds.

- **Partially synchronous model**: Lastly, partially synchronous networks represent a middle ground between the previous models. This model assumes that exists some known finite time bound Δ and a special event called GST (Global Stabilization Time) such that: (1) The adversary may cause the GST event to eventually happen after some unknown finite time; (2) Any message sent at time x must be delivered by time $\Delta + \max(x, GST)$. Otherwise stated, the system behaves asynchronously until the GST and synchronously after it.

2.7.2 Failure Models

Consensus is trivial and straightforward to solve if, and only if, there are no failures. To exemplify how such protocol could work, suppose the existence of a well-defined set of processes where every process broadcasts (propose) its value. After that, each of them wait to receive all values that were proposed. Finally, processes apply a deterministic function that take as inputs the values and outputs one of them.

A bigger and more challenging task is to solve consensus in the presence of failures. A failure occurs whenever a process does not behave accordingly to the algorithm that it has proposed to follow. In respect to these types of faults, a system can have different assumptions that differ according to the nature of the faults that can cause the process to fail. Based on the strength of the assumptions, the failure models form a hierarchy, ranging from the *crash-stop* failure model (strongest assumptions) to a *Byzantine* failure model (weakest assumptions) passing by *omission* model, *performance* model, among others. Thus, as we move through the models, the closer we get to the *Byzantine* model, the harder it is to solve consensus under its assumptions.

Actually, it was proven that in an *asynchronous* system where at least a single process may crash, there is no deterministic distributed system protocol that can solve consensus. In other words, in those conditions, it is impossible to simultaneously satisfy *liveness* and *safety*. This result was showed by Fischer et al. [17] and it is known as the *FLP* impossibility. In spite of the paper explicitly specifying the asynchronous setting with one crash failure, since the crash fault model is the one with the strongest assumptions, the result can be extrapolated to all other failure models. The result is also true in the presence of more than one faulty nodes given that, regardless of how many nodes can be affected, there is no algorithm to reliably detect the failure. Despite of the *FLP* impossibility, there are numerous algorithms to solve consensus, either by (1) extending/changing the algorithm with randomized algorithms, other timing assumptions, stronger primitives or assuming failure detectors; or (2) loosen the algorithm requirements.

In the *crash-stop* failure model, a process is treated as correct if it executes the algorithm correctly, including the exchange of messages with other processes. A crash failure occurs when a process stops the execution and never recovers after that time. Therefore, a process is said to be faulty if it crashes during its execution. Some of consensus protocol in this setting are Paxos [18], view-stamped replication [19], Raft [20] and Zab [21].

On the other end of the spectrum, there is the *Byzantine* failure model, which subsumes all the other failure models, and it is relevant to security-critical settings as the one that this work falls into. To be reliable and secure, a system should be able to function even in the presence of arbitrary failures. These failures can go from faults in the system (such as hardware or software bugs) to ones from malicious intent. Thus, a node is said to fail arbitrary if it fails to execute the algorithm in any possible way, including sending and receiving sequences of messages that are specially crafted do defeat properties of the consensus protocol. Arbitrary failures are also referred as *Byzantine* failures due to the initial paper about the theme, called ”The Byzantine Generals Problem” [10]. This paper represents a colorful allegory in which a group of army generals formulate a plan to attack a city. Some generals prefer to attack while other generals prefer to retreat. However, the attack would fail if only part of the generals attack. Thus, they have to reach an agreement. The generals can be *Byzantine* and they have to all agree on a common plan to attack or retreat. Together with some additional results, they present how many nodes are needed to guarantee that a conspiracy of f faulty nodes could not break consensus: $3f + 1$ (tolerate f *Byzantine* failures and remain available). This result had been also presented in a previous paper of the same authors [22]. A well-known representative protocol that conform to this setting is called *Practical Byzantine Fault Tolerance* [23], and it will be discussed in Section 3.2.0.B.

2.8 Forks

There are some blockchain implementations where the chance of multiple nodes making concurrent block proposals exists. The possibility of such events occurring is related with the type of consensus that is used and further discussed in Section 3.2. The concurrent proposals can cause the appearance of transient branches in the chain, known as *forks*, in which the proposed blocks share the same parent block and can continue on their own separated path/chain. At this point, the burden to choose what are the correct chain to extend through consensus is on the nodes of the network. These *forks*, if not accurately dealt with, can impact transaction finality, i.e., the moment when the parties involved in a transaction can consider the transaction to be completed or unfeasible to be reverted, and can lead to *Double Spending Attacks*, where the same currency is spent numerous times on different branches.

However, forks can also be used for blockchain software updates. On the one hand, if the software update is compatible with the previous, it is called a *soft fork*, and some nodes can upgrade while others may not, but all deal with the same data. On the other hand, there are *hard forks*, where the upgrade is non-compatible with the previous. Thus, the nodes that decide for not upgrading, stay in the older branch, while the others follow the new branch. A notable example of a *hard forks* is Bitcoin Cash [24], which was the first *hard fork* of the well-known cryptocurrency, Bitcoin [1].

3
Related Work

Contents

To conceive a system like what we proposed to, we cannot proceed without considering existing work. Since we aim to strike a good trade-off between *scalability*, *open-membership* and *security*, this Chapter goes over relevant work regarding how existing systems try to achieve that.

For a better understanding, we structured our state-of-the-art survey into two simple but critical components that are common to all blockchain systems: *membership management* and the *consensus protocol*. For each of these, we present the approaches chosen by existing systems. Finally, the set of identified techniques used by *state-of-the-art* solutions is presented and discussed.

3.1 Membership Management

The *membership management* component determines the set of nodes that participates in each consensus run. We focus this discussion on *permissionless* blockchains, given that it is there that SCALEET wants to fit in. The biggest threat for selecting the membership under such assumption comes in form of *Sybil Attacks* [8], where an attacker generates multiple identities, enhancing the probability of controlling the consensus. To address this problem, in most cryptocurrencies, incentives (e.g., some currency) are given to the node(s) that are selected to manage the consensus run, therefore encouraging the communities of participants to cooperate and create the value that will ensure the success of the network.

3.1.1 Proof-of-Work (PoW)

PoW is a mechanism that uses the process of solving a puzzle to express the membership selection. If a node wants to participate in the consensus run, it must find a solution to a puzzle specific for that run. This solution should be probabilistically difficult to discover, but has to be easily verifiable by others. The nodes that are accepted to participate are not limited as long as they have provided a correct solution.

Bitcoin [1] was the first blockchain system to adapt and incorporate PoW in their system. It was heavily derived from a prior proposal by Dwork et al. to withstand with junk mail [25]. In Bitcoin and its variants, the process of solving a puzzle is called *mining* and it is done by nodes that are called *miners*. When a *miner* decides to make a new proposal, it takes up a set of transactions that it has received, validates them and assembles them in a block. Then, the puzzle that it has to solve succeeds by finding a *nonce* such that, when the nonce is concatenated with the block's bytes, its *hash* satisfies a given pre-defined criteria. The properties of criptographic hashes make it impossible to speed up the process. The criteria used to control the rate of block proposals consists of the following: the harder it is to match the criteria the lowest the proposal rate will be. Other systems besides Bitcoin also use PoW-based membership selection [7, 26–28], making PoW the most widely used method.

Repucoin: An extension to standard PoW is the one used by J.Yu et al. in RepuCoin, named *Proof-of-Reputation* [6]. This type of membership selection allays the risks associated with the miners ability to rapidly gain computational power. Therefore, rather than considering sheer instantaneous mining power, it considers what the authors called the node's *integrated power*. This new metric is calculated taking into account the total amount of valid work that the miner has contributed to the system until that moment as well as its regularity. So, if an attacker joins the network at time t, even if it has a very strong mining ability, it would have no *integrated power* at time t, or even shortly after, since it did not contribute to the system before t. Moreover, when a miner deviates from the system's expected behavior its reputation will be penalized and hence so will its *integrated power*.

The biggest strength of PoW lies in its ability to provide a mechanism for validity to an arbitrary set of unknown nodes. This way, PoW-based systems overcome *Sybil Attacks* despite of coming with a downside of a non-negligible cost, such as gigantic electricity consumption due to large amounts of computation, a big waste of time and the need of powerful hardware. It assumes an honest distribution of the hashing power in order to work, where the honest nodes control the majority. If an adversary is able to gain control of the majority of the mining power, known as *51% Attack*, it can produce invalid transactions and/or censor the entire network. Having the majority of mining power means the ability, w.h.p., to produce blocks faster than any other node and with it comes the ability to create forks to its liking, allowing arbitrary double spending. Another problem is the centralization of the mining power. To permit for more profitable mining, nodes often aggregate resources and collude together to form *mining pools*, allowing cost and rewards sharing. The way it works is simple: members of *mining pools* do not inherently trust one another, but instead submit cryptography proofs, called *shares* to the pool operator, in order to demonstrate that they are contributing with work to the pool (e.g., work that is tied to the pool operator's public key). One possible solution to such problem is presented in the work of Miller et al. [29] in which they discourage mining pools by proposing non-outsourceable PoW puzzles. Using this new type of puzzles, any pool operator that wished to outsource mining work takes the plunge of losing its entitle mining reward, which creates a disincentive for hosting *mining pools*.

3.1.2 Proof-of-Stake (PoS)

Proof-of-Stake tries to improve the previous mentioned drawbacks of PoW, while still handling *Sybil Attacks*, by replacing the mining power selection with node's capacity of investment, such as the amount of currency held in the blockchain. The main idea is that a node that wants to be selected must prove its share of participation in the system by staking some of its assets. Therefore, this scheme assigns a weight (seen as voting power or probability of being chosen) to each possible participant that is proportional to the currency held by that node. *Safety* is guaranteed as long as a weighted fraction of nodes, $2/3$, is honest. With such schemes, the way of punishing a node, if it is caught misbehaving, is by withdrawing its stake.

There are several variants of *PoS*, which are introduced below.

Deposit based: In deposit based implementations, miners express their willingness to participate by ”locking” a certain amount of currency that they possess, which they cannot spend for the duration of their participation. One such implementation is used in Tendermint [9], where the node’s voting power is proportional to the amount of currency they have locked.

Balance based: Balance based implementations are very similar to deposit based solutions, but they calculate the stake using the node’s current account balance.
The most recognized implementations are *Proof-of-Coin-Age* and *Randomized PoS*.
Proof-of-Coin-Age represents for *PoS* the same that *Proof-of-Reputation* represents for PoW: a way of preventing a quick gain of voting power. To prevent this, systems like PPCoin [30] make use of coin age in the formula to calculate the miner’s voting power, where the currency held by the node is multiplied by the period of time that the node has held that currency.
In previous variants, the chosen group of nodes can be formerly determined by an adversary, which can lead to *Denial of Service* targeted attacks and hence to *liveness* degradation. Algorand [31] uses *Randomized PoS* and coped with such vulnerability by randomly choosing a committee to run each step of its protocol by using a cryptographic sortition algorithm. Their cryptographic sortition, based on *Verifiable Random Functions (VRF)*, is an algorithm for choosing a random subset of users according to per-user weights; i.e., given a set of weights w_i and the weight of all users W, the probability that a user i is selected is proportional to w_i/W. *Verifiable Random Functions* are pseudo-random functions that provide publicly verifiable proofs of its outputs’ correctness. Given a certain input x, a node can use its private key PrK to compute a value $y = F_{PrK}(x)$ and a proof $Proof_{PrK}(x)$. Any other node that knows the corresponding public key, can use the proof to check that the value y was indeed computed correctly (i.e., computed with a certain input) and yet, this information does not expose or facilitate the finding of the private key. Therefore, the members of the committee are chosen in a private and non-interactive way, where only the nodes who are selected for the next consensus step know that they were selected, while other nodes just learn the membership selection result afterwards.

Delegated PoS: Initially suggested in BitShares [32], in *Delegated PoS*, stakeholders vote on nodes to become *delegates*, which are the group of nodes that will be responsible for performing the consensus. These votes have a weight that is proportional to the amount of currency held by the voting node. At any time, if a node is found guilty of not following the protocol, the voting nodes can reallocate their votes.

Proof-of-Burn: In *Proof-of-Burn* [33], nodes that want to be selected have to burn, or in other words, destroy, some of its currency, by sending it to a verifiable unspendable address. By burning some quantity of currency, the node shows its dedication to the network.

In spite of mitigating the mining drawbacks of PoW, *PoS* continues to be prone to a number of inherent weaknesses. The main drawback of *PoS* is that the probability of the richest stakeholders being capable of having a significant control over the system becomes very critical. To tackle the aforementioned issue, some implementations, like the previous described *Algorand's VRF*, also use randomness together with pure *PoS*. However, this issue still has huge impact on node selection due to the high potential of stake centralization.

3.1.3 Proof-of-Capacity (PoC)

In *Proof-of-Capacity*, node selection is weighted by their capacity to allocate a non-trivial amount of disk space, which acts as a proof that a node is valid. Thus, to increase the probability of being selected, a node may increase the amount of storage it allocates to the blockchain system, which entails an assumed cost. The allocated space, besides serving as a manifestation of commitment, also assists the blockchain itself by allowing the blockchain information to be stored and shared among these nodes.

Therefore, *PoC* gives a new purpose to the hardware used by PoW, avoiding the waste of computation resources. However, *PoC* is also vulnerable to centralization due to the possibility of participants outsourcing storage to an external provider. To withstand this problem, recent implementations like PermaCoin [34] requires fast random access to memory. This counter measure directly increases the bandwidth latency in case of outsourced storage, which decreases the miner's probability of being chosen.

3.1.4 Trusted Execution Environment Based

A *Trusted Execution Environment* is a secure and isolated environment of a main processor in which it is guaranteed that code and data loaded to it are protected with respect to confidentiality and integrity. In other words, it guarantees that code that is executed inside it will honestly follow the pre-defined specification. Examples of popular TEE providers are ARM and Intel with platforms like ARM TrustZone [35] and Intel SGX [36], respectively.

TEE-based membership selection relies on hardware to prove validity and engagement with the network. This validity through hardware possession is called *Proof-of-Ownership*. Members are selected based on the output of a random function that is run inside its *TEE*. Therefore, rather than using hardware to leverage the probability of being selected by wasting CPU or storage, *TEE*-based ones place trust on hardware to randomly determine whether a node is selected. If a node wants to increase its probability of being chosen, it should purchase more chips with *TEE* capabilities, since this membership selection is proportional to the number of *TEE* that a node controls. One prominent aspect of *TEE* chips is that each of them is uniquely identified (e.g., unique ID), which prevents a node from spoofing other chips' IDs and allows the network to permanently exclude one misbehaving node from the network based on its ID.

In addition to leveraging these hardware-based IDs, existing solutions also take advantage of other features of *TEEs*, namely those solutions based on *Proof-of-Luck* and *Proof-of-Elapsed-Time*.

3.1.4.A Proof-of-Luck

In *Proof-of-Luck*, initially proposed by Milutinovic et al. [37], each node runs a random number generator function inside of its *TEE*, which could be considered a lucky value. The values of each potential participant will be compared against each other and the luckiest ones will be chosen.

3.1.4.B Proof-of-Elapsed-Time

An alternative method was suggested by Intel and later implemented in HyperLedger Sawtooth project [38], *Proof-of-Elapsed-Time*. Each node will, at the same time, request a wait time from their enclave, which is also known as a trusted function inside the chip. Afterwards, each node will sleep until its received time elapses. The node(s) that wake up first, will be elected as leader(s).

The major weakness of *TEE*-based solutions is the inherent trust that is placed in the hardware. Thus, breaking a single piece of trusted hardware allows an adversary to always be selected. Even if the chips are correctly developed and manufactured, there are known attacks [39] that can target and exploit existing *TEEs* anyway. Moreover, moving the trust to the hardware also implies trusting the vendors that supply the hardware, which can lead to monopolization.

3.2 Byzantine Consensus Protocols

The *consensus protocol*, as we discussed in Section 2.7, is the component that is responsible for deciding the next block to be appended to the chain among a competing set of proposals. *Byzantine* consensus is a subject that has been studied for decades and, yet, there is still an ongoing effort for techniques that can scale it (to a large extent, driven by the application of this problem to blockchains). One of the recent reasons for this search is its direct application to blockchain systems, which have recently become a large object of studies.

Existing solutions can be divided into two major groups: the ones following *Nakamoto's* consensus and the ones following classic *BFT* consensus. This Section follows that division, explaining the differences between the two approaches.

3.2.0.A Nakamoto's Consensus

Introduced in Bitcoin [1] by Satoshi Nakamoto, *Nakamoto's* consensus protocol was the first consensus to be applied in permissionless blockchains. The main idea behind it is called the *longest chain* rule, which is used to solve transient forks that may happen. This rule states that the chosen chain must be the longest one, since it is believed to have the most "work" performed and hence complying with the principle that the strict majority of the network is honest.

The concept of *Nakamoto's* consensus is commonly used interchangeably with PoW, because they were proposed and are typically coupled together, but they do not mean the same. The latter is a way of choosing what are the nodes that can append blocks at a given height. Therefore, if PoW were used alone, there would be no mechanisms for deciding between blocks in case of concurrent proposals. In fact, *Nakamoto's* consensus can be paired with other selecting membership algorithms.

Nakamoto's consensus only operates correctly under a synchronous network, since it requires all nodes to learn about the latest blocks in order to correctly extend the main chain. As such, it only provides a probabilistic consensus when the expected time bounds are exceeded. This can affect either termination or agreement as there is a possibility of unseen chains after a certain amount of time has elapsed. Therefore, it never reaches consensus finality, i.e. a point in time when one can be sure that consensus has been achieved. All that can be done is to estimate the probability that a block is in the final chain. As a result, *Nakamoto's* consensus only guarantees eventual consistency, i.e., during periods where there are no blocks being proposed, all nodes will eventually be aware of the updated chain. Otherwise, during periods where the chain experiences transient forks, consistency could not be met.

3.2.0.B Classic Byzantine Fault Tolerance Consensus

An attractive alternative to *Nakamoto's* consensus are classic *BFT* protocols, which provide strong consistency guarantees. Strong consistency offers two major benefits to blockchain systems that are intrinsic related with each other. First, it ensures near-instantaneous block finality, which leads to the second advantage, which is that clients do not need to wait for extend periods of time to be sure that a submitted transaction was indeed committed. Consequently, blockchains based on classic BFT protocols are being increasingly adopted.

Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) [23] is a well-known BFT protocol, which is considered the first practical *BFT* replication solution in the partial synchrony setting, that achieves consensus through leader based communication.

In PBFT, at each moment, there is a participant configuration that is called a view. In a certain view, there is a primary node that coordinates the system and the remaining ones are called backups. Views are numbered consecutively and its primary node is the node whose $id = current\_view\_number \mod n$. The replicas change their view when they suspect that the current primary has failed.

A new value is agreed in each round of the protocol. Each round is composed by three phases: *pre-prepare*, *prepare* and *commit*. The first phase starts when a primary assigns a sequence number to the request and broadcast them to all the other nodes. From here, in each phase, for a node to proceed to the next phase it has to receive $2/3$ votes from other nodes matching the previous phase. The first two phases ensure total order of the messages in the same view whereas the last two phases ensure that commits are totally ordered across views.

The view change algorithm plays a key role in the system, since it provides *liveness* by allowing the system to make progress when the primary is suspected to have failed. It is based on timeouts. If a client did not receive any response to the request that it had sent after a given period, it broadcasts the request to all the replicas. If a replica receives a request from the client, it starts a timer waiting for that request from the primary. If it expires, the replica starts a view change that will eventually be followed by other replicas.

FastBFT

FastBFT [40] belongs to a group of BFT protocols [38, 41, 42], that uses *TEEs* as an approach for achieving more scalable *Byzantine* consensus while only requiring $2f + 1$ nodes in the network.

The key idea is a novel message aggregation technique that does not require either the typical $\mathcal{O}(n^2)$ messages or any public key operations (e.g., multisignatures), thus incurring considerably both lower communication and computation overhead. Instead, it is used secret sharing with the aid of *TEEs*. To implement secret sharing, an extra phase called *pre-processing* is needed to set up the secret shares. In this phase, the current consensus leader generates a secret which is bound to the current request. Then, the primary creates a set of shares of this secret and divides them among the remaining nodes. This will all come together in the *commit phase*, where every node will reveal its share to the primary, that will successfully reconstruct the secret, if it receives enough valid shares. Finally, this secret is then multicast by the leader to all the nodes that can verify it. The use of *TEE* ensures that none of the nodes can impersonate any other. Moreover, *FastBFT* also integrates other optimizations, such as *optimistic execution* and a *tree-like* pattern of communication.

HoneyBadgerBFT

HoneybadgerBFT [43] uses a novel *Asynchronous Common Subset (ACS)* reduction by using *threshold encryption* and an efficient reliable broadcast with erasure codes. A *Threshold cryptosystem* is a cryptosystem that leverages the security of the information by encrypting it and distributing it among a set of fault tolerant nodes. The difference is at time of decryption where, in order to be successfully done, several parties should cooperate with its share. A *Threshold encryption* scheme is used to counter the possibility of an adversary choosing which nodes' proposals are being included and hence the ability to censor transactions to its liking. With a *Threshold encryption* scheme, the agreement phase is being run on cipher texts, thus the adversary can not figure out which node had proposed which transactions.

Algorand

Algorand [31] uses a novel *Byzantine Agreement* protocol (*BA**) to efficiently reach consensus among the users based on Verifiable Random Functions. *Liveness* and *safety* are guaranteed under strong synchrony and weak synchrony assumptions, respectively.

The members of the consensus committee are chosen in a random, private and non-interactive way, in which an adversary does not known which user to target until that user starts participating in *BA** as described in 3.1.2. This idea of a secret crypto-based member selection has been explored in many systems, for instance, in Gosig [13]. Nevertheless, it can happen that an adversary targets a committee member once that member sends a message in *BA**. This is mitigated by requiring a member to speak just once. This way, after the member sends one message, he becomes irrelevant. This is achieved by avoiding any private state in each user except its private key, and, therefore, users are equally capable of participating.

In all communications between nodes, a gossip protocol is used. Each user assembles in a block the transactions that have reached it. Next, Algorand will initiate a round starting with a block proposal protocol, which includes (1) creating a committee using the weighted sortition algorithm and (2) each committee member proposing a block. After that, each node waits for a time period to receive the blocks and only keeps the highest priority block (this priority is calculated in a deterministic way). Finally, each node who receives some block will initialize *BA** to reach majority consensus and commit a block.

The *BA** algorithm cycles through these steps and, in each iteration, sortition is used to create a new committee. The execution of *BA** consists of two phases. The first phase is called *reduction* and the second goes by the name of *binary agreement*.

- **Reduction**: First, each committee member votes for a block, again based on priority. Second, they vote on a block that received at least T votes (which is a value established with the expected number of nodes elected in sortition) in the last step or in a default empty block if no block received enough votes.
- **Binary Agreement**: In this phase consensus is reached on one of two blocks: the one passed by the reduction or the empty block.

ByzCoin

ByzCoin [28] improves upon *PBFT* by replacing *MACs* with digital signatures and by employing *collective signing* and a different pattern communication, namely tree communication.

The protocol can be described in an incremental manner, starting with an inefficient and closed-membership *PBFT* algorithm. Then the following refinements were done to the initial naive implementation:

- Replace *MACs* with digital signatures: This first refinement tackles the *MAC* authentication implying an all to all message exchange in *PBFT*, which has communication complexity of $\mathcal{O}(n^2)$ as already stated. Therefore, the usage of digital signatures enables the use of better communication patterns once these messages become third-party verifiable, which will be useful in the following improvement.
- Use collective signing and tree communication in *PBFT*: This step makes use of a protocol, *CoSi* [44], for scalable collective signing, which enables an authority or a leader to request some statements to be publicly validated and co-signed by a decentralized group of witnesses. This protocol combines *Schnorr* multi-signatures [45] with communication trees. Each run of the protocol yields a collective signature that has size and verification cost similar to an individual one. *CoSi* does not implement consensus or BFT by itself. Instead, it can be used as a primitive by the BFT leader to collect and aggregate (pre-)prepare and commit messages during *PBFT* rounds and consecutively leading to communication complexity of $\mathcal{O}(\log n)$ and typical signature verification complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. To implement *PBFT*, two rounds of *CoSi* are used: one for pre-prepare and prepare phases and another one to the commit phase.
- Make the consensus group non-static and decouple transactions confirmation from identity managment (including leader election): The last improvement is based on an idea that had already

been proposed by another blockchain protocol called Bitcoin-ng [7]. In particular, this is the idea of having two types of blocks: *microblocks* and *keyblocks*. However, Bitcoin-NG just uses a single chain to append both block types, whereas Byzcoin introduced a second chain for the *keyblocks*.

Keyblocks are used to manage Byzcoin's open-membership. They are generated through *PoW* as in Bitcoin and they are collectively signed by the current consensus group. A miner that manages to successfully mine a block is rewarded with what they call a consensus group share. This share serves as a proof of membership that gives the miner the opportunity to join the consensus group for the next round. There is a virtual fixed-size sliding window mechanism that goes through this *keyblock* chain and constitutes the total number of consensus participants. Each time a *keyblock* is mined, the window advances. The nodes that are inside the current window are the ones that will participate in the next consensus round. *Microblocks* are the traditional transaction blocks that are appended using the algorithm described above. As such, they are appended in a strong consistency model. Each *microblock* contains, in addition to what the normal blocks have, a hash of the leader's *keyblock* to identify the "era" that it belongs to.

This way the protocol permits that many *microblocks* are appended in some round in a total order, and *PoW* is only used in the second chain to enable open-membership. Therefore, the impact of having open-membership is less perceptible compared to previous solutions, like Bitcoin [1].

Motor

Motor [46] is a novel consensus algorithm that is built on top of Byzcoin's consensus [28] and addresses some of its shortcomings that kept it from being suitable for deployment in a open and adversarial network. In particular, the authors identified the following issues in Byzcoin: (1) the two-round *Schnorr* multisignature scheme used is susceptible to simple DoS attacks by a single malicious node; (2) the tree communication pattern employed in order to boost scalability is fragile under strong adversaries; (3) the leader election mechanism is predictable and cannot deal with a round-adaptive adversary.

The first modification in Byzcoin is removing the DoS susceptible two-round CoSi and instead use BLS signatures [47], named BLS-CoSi. BLS signatures are secure with high probability, and can be aggregated into compact multi-signatures. Due to their non-interactive and deterministic nature, there is no need for the quorum to be the same between phases as long as there exist $2f+1$ honest nodes per phase.

The second modification in Motor is the employment of a suitable communication pattern to avoid the leader bottleneck present in classical BFT protocols, while also proposing a communication pattern that preserves *safety* and *liveness*. In detail, it employs a rotating-subleader communication pattern, which can be seen as a hybrid form of the robust start communication pattern and the scalable tree communication pattern that ByzCoin uses, a star of stars. The leader divides the nodes into random groups, where, in which one, a sub leader exists. Each sub leader forms a star with the remaining nodes of its group. When a sub leader crashes or does not forward messages to its group, the leader can choose a new sub leader and retry. In this protocol, since the leader knows who to blame, the maximum number of retries it needs to perform in order to find an honest per-group sub leader is constant (which is not the case in a pure tree pattern, where it would need to retry an exponential number of times).

The third and last modification is related to the view change protocol, in order for it to handle mildly adaptive adversaries who can predict the next f leaders and hence compromise them. To achieve this, Motor uses again the strength of BLS signatures in terms of their deterministic, unpredictable and unbiasable nature and, at the time of the previous view change, uses BLS signatures to randomly generate a number that hence will randomly schedule a new leader for the next round.

Omniledger

Designed by the same authors of Byzcoin [28], Omniledger [12], seeks to tackle the scalability issue by improving basically every axis mentioned in the beginning of the Section. The core idea is based on sharding and parallel transactions. The blockchain state is partitioned into shards, which are divided across multiple sets of nodes using the *UTXO* model.

For membership selection, Omniledger is based on the idea of Byzcoin's second chain. Every time that the chosen set of nodes needs to be partitioned across committees (e.g., after a pre-defined amount of time), a distributed randomness generator protocol called RandHound [48] is used to randomly generate a number, which will serve as a seed for sharding securely. This way, adversaries cannot predict, *a priori*, the configuration of any shard. This seed is also used to decide which are the nodes that leave in each committee.

For handling cross-shard commits under a *Byzantine* context, the authors adapted the two-phase-commit (lock and unlock) protocol from databases to create what they called *AtomicX*. It builds on the idea that each shard is collectively honest, does not crash infinitely, since each of them runs ByzCoin BFT internally. *AtomicX* is client-driven in order to keep the shards' logic simple and make any direct shard-to-shard communication unnecessary.

To handle parallel commits, Omniledger used a *block-based directed acyclic graph* as a data structure, where every block can have multiple parents, to detect dependencies between transactions. This way the consensus leader can enforce that each pending block includes only non-conflicting transactions.

Omniledger also introduced the idea of truncating the transaction history by adopting a checkpointing process similar to the one described in PBFT [23] called state blocks. In this scheme, at the end of each epoch, the leader of each shard stores the current set of committed transactions in an ordered Merkle tree and puts the Merkle tree's root hash in the header of a new state block. The consensus protocol is then run against this block and if it is accepted, it becomes the genesis block of the next epoch.

Finally, they introduced low latency transactions, where the opportunity to decrease the overall security of the transaction by trading security for performance (lower latency) is given to the client. In this model, each transaction is first processed by an optimistic group, smaller than normal ones and consecutively more likely of being overpowered by an adversary, that produces optimistically validated blocks. Following this, a re-validation is done by core validators. When core validators detect an inconsistency, the respective optimistically validated transaction is excluded and the validators who signed this block are identified and punished.

3.3 Discussion

This Chapter surveyed several *state-of-the-art systems*, with a specific focus on what were the techniques used to obtain good scalability in a open membership setting without neglecting security. At a high level, we can draw the following main insights from this analysis.

On the one hand, solutions based on *Nakamoto's* consensus exhibit high latency and low throughput, but can scale well with the number of nodes in an open Internet environment, since they only require nodes to choose the longest chain locally to reach an agreement. As an example, when paired with PoW, just like what happens with Bitcoin [1], *Nakamoto's* consensus achieves a planetary scale, but with a transaction throughput that is currently processing up to 7 Transactions Per Second (tps). In contrast, centralized payment systems, like Visa and MasterCard are reporting 1,200 to 56,000 *tps* [49], which is three to four orders of magnitude higher than the one obtained by Bitcoin.

On the other hand, BFT solutions can achieve higher throughput and lower latency, which is in conformity with current centralized payment systems. However, they can have scalability problems when considered in a permissionless setting. Firstly, the set of eligible participants that collaborate in blockchain systems is not fixed, nor predefined. Therefore, the set of replicas and quorum size that is required for agreement is not constant. Another main factor that negatively affects scalability is the typical quadratic number of messages. The first concern is generally solved by applying one of the membership selection techniques presented in Section 3.1. Regarding scalability, when the number of participants increases, the proposals rely on optimizing the standard PBFT on a well-defined set of axes:

- **Communication Topology**: Improving the pattern of communication and distributing the communication load as evenly as possible makes bottlenecks less likely to appear. To address this, the systems we presented use mainly three techniques:

  - Rearrange the communication topology in a balanced tree [28], where the leader is at the root position and non leaf nodes forward messages to their children. The reply process is initiated by the leaf nodes in a bottom-up approach. However, this improved topology can cause the loss of *liveness*, since an adversary might control one or more internal nodes, making the nodes below the overpowered ones (subtrees) biased. For circumventing this issue, upon detecting a faulty node, these systems fall-back to the typical communication pattern (flat communication). Another approach is taken by Motor [46], which uses a star of stars to bound the number of retries upon the detection of a faulty node.
  - Relieve the communication effort taken by the leader by disseminating the messages using gossip [31]. The leader initiates the sending process by randomly choosing some of its neighbors to receive the message. At each hop, each node does the same. However, since this technique makes use of randomness, it is not guaranteed that, when the nodes stop the gossip process, all of the nodes needed to receive the message (i.e., all correct ones) had received it. Thus, this technique only provides probabilistic guarantees.
  - Another probabilistic approach, which was heavily inspired by gossip mechanisms, is the use of leaderless communication. The idea behind it is similar to leader based gossip communication, but instead of the protocol being guided by a leader, it is assumed that, at a given round, the correct nodes have already converged to the same correct value.

- **Cryptographic Primitives**: Cryptographic Primitives play a key role on scaling BFT and can be, as we have observed, very important in enabling the efficient functioning of different patterns of communication. A set of primitives was previously presented along with their direct application to blockchain systems: *Collective Signatures* [28], *Threshold Encryption* [43] and *Verifiable Random Functions* [31].

- **Representative Committees**: Another key idea to scale consensus is the use of a representative committee. By choosing a representative committee, we avoid saturating the network with extra and unnecessary messages [12].

- **Parallelization of Transactions**: If instead of one representative committee, several were selected, it would be possible to employ sharding. As we discussed, sharding is an approach in which the overall system state is partitioned across committees, and therefore this leverages the possibility of parallelizing transactions within and across shards [12].

- **Trusted Hardware Components**: Finally, another idea is to move some costs from the critical path of the protocol to a trusted hardware. This idea can both be used in membership selection by providing an unforgeable uniqueness within the network to counter *Sybil attacks* and in the consensus protocol as used in [38,41,42], by providing an efficient way to conduct several aspects of the protocols by leveraging hardware support from the *TEE*.

4 SCALEET

Contents

This Chapter introduces SCALEET, a novel blockchain design that shows how classical BFT can be securely and efficiently used in a permissionless setting. Section 4.1 presents the fundamental goals that drive the system's design. Section 4.2.1 states the assumptions made and the system's model. Finally, in Section 4.3 the architecture and design of SCALEET are thoroughly detailed.

4.1 Design Goals

The goal of SCALEET is to show that we can incorporate classical consensus algorithms in the design of a secure permissionless blockchain that can output similar performance to classical centralized payment systems, such as Visa, while still supporting existing techniques to contain the power that a single entity may have over the system, namely PoW and PoS, and also ensuring that these techniques do not significantly interfere with the scalability of the system. Therefore, SCALEET aims to present the necessary building blocks to strike a good balance between *scalability*, *open membership* and *security*, and achieves this by bringing together the following characteristics in its system design:

- **Permissionless setting**: SCALEET stands on the permissionless setting where every node can create an address and freely join the network.
- **High throughput**: SCALEET'S throughput is expected to increase linearly in the number of nodes that perform the key transaction execution steps.
- **Low client-perceived latency**: SCALEET'S client-perceived latency should be low, namely orders of magnitude lower than in existing PoW schemes.
- **Separate Sybil resistance from transaction processing**: In a permissionless setting, as we already discussed, having a Sybil resistance technique is an imperative step for the proper functioning of the network. However, in contrast to what happens in Bitcoin, in which PoW is used as a Sybil resistant technique during the transactions verification, SCALEET should be built on the idea introduced by Bitcoin-NG [7], which decouples this verification from the membership management by using two parallel chains.
- **Nonexistence of forks**: SCALEET should implement the idea of *Deterministic Finality*, which means that once a block, and consecutively a transaction, is appended to the blockchain it can be considered final and will not be removed by any means. When coupling this requirement with the previous point, the possibility of the occurrence of forks has to be taken into account both in the main chain and in the second chain that is responsible for the membership management. SCALEET should not allow forks even in such case.

- **Simple reasoning**: SCALEET should take as a principle that its design should be as simple as possible, so that it is easy to reason about the correctness of the system.
- **Modularity**: SCALEET should allow to easily change key aspects of the protocol, such as the BFT algorithm that is being used or the way that a node has to show its commitment to the network.

4.2 Overall assumptions and model

Modeling a system is a crucial step when designing one, since it states what are the chosen assumptions, the system's characteristics and in what circumstances they still hold. This Section goes through the model where our system fits in.

4.2.1 System Model

A SCALEET deployment is composed of n nodes that can confirm transactions, called *validators*, and by m *relay* nodes that forward transactions and learn the state of the blockchain from the validators. There are also c *committees* which can be seen as a decision group that agree on a set of commands by running a BFT consensus-based state machine replication algorithm. All the validators belong, in a given moment, to a single committee. Each node is assigned a public/private key pair which is denoted by (PK, PK^{-1}). Therefore, a node i has been set up with the pair (PK_i, PK_i^{-1}) and is uniquely identified by PK_i.

To enable us to focus on the essential design features and ease the evaluation of the initial prototype, we made some simplifying assumptions regarding how system parameters vary throughout the system lifespan, namely that the number of committees, the size of each committee and the interval between committee reconfiguration is fixed throughout the execution of the system. These assumptions are common [12, 28] and relatively easy to lift.

4.2.2 Cryptography Model

SCALEET rests on the common cryptographic assumptions, namely that every adversary is computationally bounded and that cryptographic primitives such as public-key signatures and hash functions are computationally secure.

4.2.3 Threat Model

SCALEET considers a *Byzantine adversary*, where Byazantine-faulty nodes can not only simply crash, but also behave arbitrarily and collude with other Byzantine nodes to attack the system, i.e., they could arbitrarily delay, insert, drop, re-order and forge messages. All the non-faulty nodes are considered honest and do not deviate from the protocol. Our system design includes groups of cs validators, where we assume the presence of at most f Byzantines faults, where $cs = 3f + 1$. Furthermore, we need to classify the adaptivity of the adversary, which is the ability of the adversary to corrupt nodes dynamically based on information that he learns during the system's lifespan. Regarding such classification, there are two main groups:

- **Static**: the adversary has to decide in advance, even before the system starts, which nodes he wants to corrupt.
- **Adaptive**: the adversary can decide dynamically, during the system's lifespan, which nodes he wants to corrupt, and this way take advantage of the information that he had already learned. With regard to the time that takes between the adversary's decision to corrupt a set of nodes and the corruption itself, there are two types of classification: *strongly adaptive* and *mildly adaptive*. In the *strongly adaptive* setting, the corruption is instantaneous, i.e., there is no delay between the decision and the corruption. On the other hand, in the *mildly adaptive* setting, a certain delay between the events is considered. Such delay varies from system to system.

As we will discuss, the system lifespan is divided in eras, which are the time between membership reconfigurations. Given this concept, it will be guaranteed that these reconfigurations are unpredictable and unbiased events, which leads SCALEET to assume a *mildly era adaptive* adversary [12], which can choose which nodes to corrupt at the start of each era, e.g., the delay in the time of one era. This type of adversary is aware of all the decisions made in all the previous eras and can take actions from that information. However, once the era begins the set of actions are fixed.

4.2.4 Network Model

Regarding the underlying network, SCALEET assumes that all the honest nodes are well connected and can send and receive messages to other honest nodes. More precisely, we assume that honest nodes are connected through *perfect links*, sometimes also called *reliable links*. Such *perfect links* are characterized by three properties:

- **Reliable delivery**: If a correct node n sends a message m to a correct node n_i, then n_i eventually delivers m.
- **No duplication**: No message is delivered by a node more than once.
- **No creation**: If some process node n delivers a message m with sender n_i, then m was previously sent to n by process n_i.

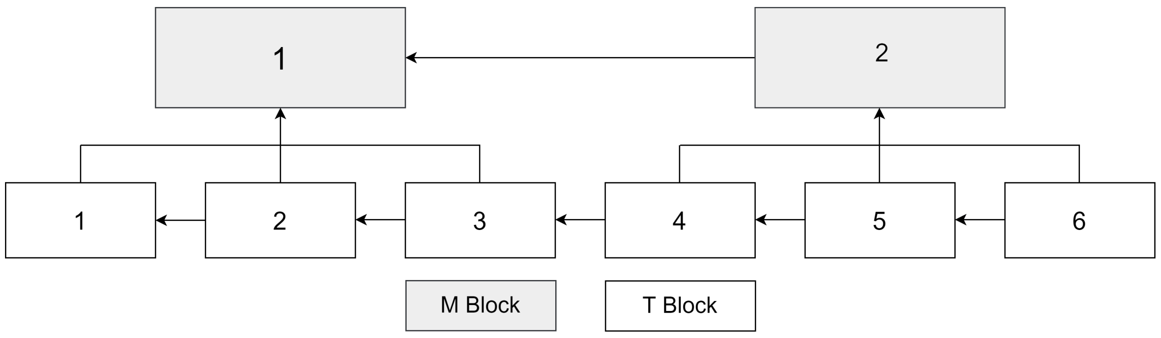

Figure 4.1: *Membership blocks* and *Transaction blocks*.

Moreover, our protocol can keep safety under an *asynchronous* network, and as a way to circumvent the *FLP* impossibility, we guarantee liveness if the network has *partial synchrony*, which is a common assumption.

4.3 Architecture and Design of SCALEET

The high level architecture of SCALEET is a combination of several building blocks, which are constructed on top of some of the techniques identified in Section 3.3. These techniques are carefully combined together in a novel way to achieve a more scalable blockchain. The system is relatively compound, since it has various dependencies between its components, hence we will start by giving a general overview of the system architecture and then we will zoom into the modules and their details.

4.3.1 SCALEET Overview

Membership Decoupling

One of the key design decisions in SCALEET is to shift the focus of how techniques such as PoW and PoS are employed, from their previous use as a consensus mechanism to exclusively being used as a Sybil resistance technique. In addition, as previously stated, SCALEET aims to prevent those techniques from directly impacting the performance of the system. One of the design choices, firstly introduced in Bitcoin-NG [7] and later used in Byzcoin [28], that enables SCALEET to achieve that, is the decoupling of membership management and transaction validation. For that, SCALEET makes use of two different kinds of blocks: *transaction blocks* and *membership blocks*. As we explained earlier in Section 3.2.0.B, *transaction blocks* represent the set of transactions that the system is processing, whereas *membership blocks* enable managing the group membership eligible to participate in each consensus era. Thus, the

Table 4.1: UTXO and Balance model pros and cons.

Model	Pros	Cons
UTXO	**Scalability**: multiple UTXOs can be processed at the same time, which ease parallel processing **Privacy**: more difficult to associate transactions issued by the same party	**Stateless**: more difficult to get assets owned by a party **Great costs**: have more storage costs as the number of UTXOs grow and increased computation cost to validate
Balance	**Simplicity**: easier to understand and highly thigh with traditional notion of balance **Efficiency**: only needs to check the current balance of the sender in order to validate a transaction	**Privacy**: encourage reuse of addresses, which generally degrades privacy **Increased difficulty to parallel transactions**: transactions affecting the same account must be performed one after another

relation between *membership blocks* and *transaction blocks* is one-to-many, i.e., there will exist more than one *transaction block* associated with each *membership block*. In particular, in our design, there are two block chains, the membership chain and the transaction chain, where each *transaction block* has a link for both the previous *transaction block* and the last *membership block*, which represents the current membership that will vouch for the validity of the transaction, as depicted in Figure 4.1.

Representative Committees

A key building block is the use of *representative committees*. The idea is to assign a special role to a subset of nodes that are part of the system membership. These nodes are randomly selected to participate in one of several committees for a given era. Each of these committees will then run a BFT consensus algorithm each time they need to agree on something, e.g., on a valid block to be appended to the chain. For a given era, one of these committees is called *membership committee* and is responsible for managing the membership chain, whereas the others are called *transaction committees*, and will be in charge of the transaction chain.

Sharding

Another important building block that allows for achieving good scalability is the partitioning of the state of the blockchain across the committees, i.e., *sharding*. However, the form of sharding that we employ represents a significant departure when compared to Omniledger [12]. While Omniledger partitions the state randomly by taking into account the first bits of the hash of each UTXO, we will partition it by accounts. Thus, each committee will accommodate a set of specific accounts, e.g., chosen according to their addresses. For this, SCALEET chose to abandon the UTXO model in favor of the account/balance

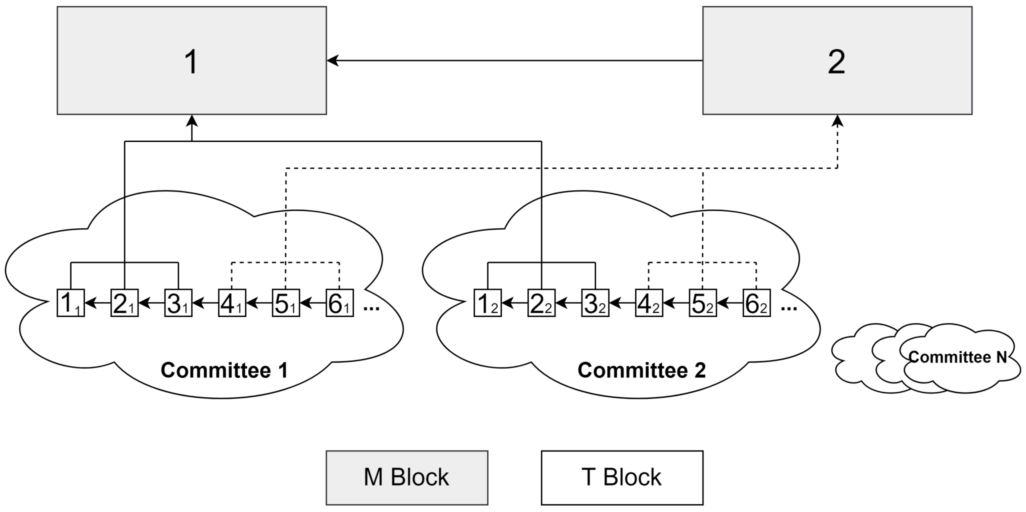

Figure 4.2: *Membership blocks* and *transaction blocks* redesigned with committees.

model. In the latter model, assets are represented as balances within accounts, which is what happens with traditional bank accounts. Transactions are interpreted as events in the blockchain state machine which leads to a transition from a set of account balances to a new set of values, based on the previous blockchain state. Both models have trade-offs, which are summarized in Table 4.1. In our design, we decided for the account model mainly because of its efficiency. We identified as its main drawback the greater difficulty to process transactions in parallel, which we have not yet addressed in our current prototype, and only discuss how to add this feature in Section 7.2. Finally, we need to take into account the possibility of double spending attacks, namely to avoid the same transaction to be issued more than once. To counteract this attack, we adopt the same strategy as Ethereum [26], namely by incrementing a nonce tied to each account each time a transaction for that spends funds from that account is issued.

Note that, by doing such partitioning, three important design features and optimizations are enabled. First, each possible inter-committee transaction can span, at most, two shards: the shard that is responsible for the address of the sender and the shard responsible for the address of the receiver. Second, this brings in the ability to apply clustering algorithms, to try to co-locate accounts (i.e., into the same shards) that have issued more transactions between each other, avoiding inter-committee transactions as much as possible. Lastly, we can move the responsibility of inter-committee transactions processing from the client to the committee in charge of the address of the sender, without the need for a distributed two-phase-commit, since the validation of the transaction can be done in the originating shard, as will also become apparent later. As a result of the use of committees and sharding, Figure 4.1 can be redrawn to Figure 4.2, since different committees work on different data. The membership chain is learned by all the nodes in the system and managed by the membership committee and then there are multiple transaction chains that are only learned by the members of the committee that manage each chain.

Committee Reconfiguration

As mentioned in Section 4.2.1, there are two types of nodes: *relay* nodes and *validators*. We call validators to the nodes that, at a given moment, belong to a committee and are therefore allowed to validate blocks. All the remaining ones are called relay nodes because their job is to relay requests to the validators. As we will explain subsequently, the system lifespan is divided into *eras*, which corresponds to the time spent between committee reconfigurations. Moreover, inside an era, the time is measured in *rounds*, which correspond to an instance of a BFT consensus algorithm (and are already referred to in the BFT literature as a protocol round). In contrast to eras, rounds do not need to be synchronized across committees and the total number of rounds per era does not have to be fixed.

At the beginning of each *era*, a set of relay nodes is chosen to become the validators, replacing the previous set of validators, and picked in way that ensures that these nodes are randomly distributed across the committees. Consequently, any node can be chosen as a validator, provided that it is able to prove its commitment to the network. Furthermore, this proof should be made available to the system in the era before the one that it wants to be part of. Thus, formally, if a relay node i wants to participate in an era e, it needs to register for it in era $e-1$. The membership committee is in charge of receiving all these commitment proofs, validate them and append them to the membership chain if they are valid. This way, during an era, while the set of transaction committees is confirming transactions, the membership committee is receiving the proofs of commitments from the relay nodes nodes that desire to join the protocol by becoming part of a committee in the next era.

4.3.2 Membership Management

Next, we focus on the design of the membership chain, and more generally how the system manages its membership. Firstly, it is important to clarify what is meant by membership. Given the system's permissionless nature, any node can freely join and leave the network; however, they initially join the network as relay nodes. In order to be designated as validators and, consequently, distributed among the committees, relay nodes have to show engagement towards the network and register for the next era. At each moment in the system execution, the membership corresponds to all the designated validators at the time.

The system's lifespan is divided into eras, which span the time between reconfigurations. The system bootstraps with a given set of predefined validators, and from then on this set will change in every reconfiguration. These validators are also assigned to one of several committees of different types, namely a single membership committee and a set of transaction committees. Transaction committees are responsible for receiving transactions, validating them and appending them as a part of transaction blocks to their local transaction chains. In turn, the membership committee is responsible for receiving membership commitment proofs from relay nodes, validating them and appending them to a globally shared chain, called the membership chain. Therefore, during each era and until the next reconfiguration, SCALEET performs two parallel jobs: handling registrations for the next era issued by relay nodes (performed by the membership committee) and validating transactions (performed by the transaction committees). By doing such jobs separately and in parallel, SCALEET achieves one of its main goals, which is decoupling membership management from transaction validation. This way, the performance of transaction validation is not affected by the Sybil resistance technique that is employed.

Before explaining in detail the steps that compose the membership management mechanism, we describe the various components of the membership block:

- **Block index**: The height of the block in the chain.
- **Era counter**: Sequence number identifying each era.
- **Reconfiguration seed**: Property that is used only when the current block is a reconfiguration block. This corresponds to a random number that serves as a seed for taking random and unbiased decisions.
- **Membership commitment proof**: Proof of commitment of a relay node.
- **Committees**: Current constitution of all committees.
- **Registered members score**: Current score of each relay node that has sent, at least, one membership proof in the current era. These scores will dictate the nodes chosen to be designated as validators in the next era. At the beginning of each era, all members start with a score of zero. Each proof submitted by a member will increase the score of that member. The top scored members will then become validators in the next era. In other words, the more commitment a node puts into the system, the higher the probability of being chosen as a validator. More details will be given in section 4.3.2.B
- **Previous block hash**: Hash of the previous block.
- **Hash**: Hash of the current block, covering all of the previously mentioned fields.
- **Prepare and commit signatures**: List of signatures from the BFT consensus run that allowed this block to be appended to the chain. We will discuss their usage later in Section 4.3.5.

The membership management can then be divided into three main phases: the submission of membership commitment proofs, the creation of membership blocks, and the reconfiguration of committees. The first two happen in parallel throughout each era, while the latter happens between eras.

Each era has a fixed size in terms of membership blocks, in which each one represents a new membership proof received in that era. Therefore, in our design, it is the rate of membership block production, or in other words, the rate of membership proof submission that controls how long an era takes. After a given number of blocks is appended to the chain, a membership reconfiguration occurs. Thus, each time a node aims to be a validator in the next era, it has to build at least one membership proof and submit it to the network, which will relay this proof until it lands on the membership committee. Recall that a node that wants to participate in an era e, needs to provide the proof in the era $e-1$. Each time a proof arrives at the membership committee, it will trigger a BFT run to build and append a new membership block.

4.3.2.A Membership Commitment Proof Submission

The membership commitment proof is the mechanism employed by SCALEET as a Sybil resistance technique. As usually happens in computer security, one can never ensure that something is completely secure. Instead, one should ensure that the cost of attacking is higher than the value of what it is protecting. Given this general principle, SCALEET uses commitment proofs to narrow down the probabilities of attacks on the system membership, namely Sybil attacks. The commitment proof mechanism leverages PoW to provide such cost by requiring work during the era previous to the one it wants to participate in. Per era, a relay node can issue as many proofs as he desires, with the prospect of increasing the probability of being chosen. By principle, the more valid work a node produces, the higher the probability of being selected. (This will be discussed in more detail in Section 4.3.2.C when we discuss the reconfiguration phase.)

The membership commitment proof is comprised of:

- **PoW seed**: Seed used as input to the PoW.
- **Proof**: Set of bytes that prove the PoW was completed successfully with some predefined difficulty d.
- **Public key (address)**: PK of the member that has done the PoW.
- **Hash**: Hash of the commitment proof, covering the previously mentioned fields.
- **Signature**: Signature of the hash, computed with the PK^{-1} of the node.

As depicted in Figure 4.3, when a node wants to submit a new proof, it fetches the hash of the last block from the membership chain and uses it as seed for PoW. Then it starts the PoW proof finding, which succeeds when it finds a random set of bytes that, when concatenated with the previously fetched hash and computed its hash, satisfies a given criteria. This criteria is called the PoW difficulty and it is often expressed as the difficulty of the aforementioned hash value being lower than a predefined value. This difficulty is the way that SCALEET uses to control the time of an era, since the longer each membership proof takes to be found, the longer each era is, since the length of an era is defined

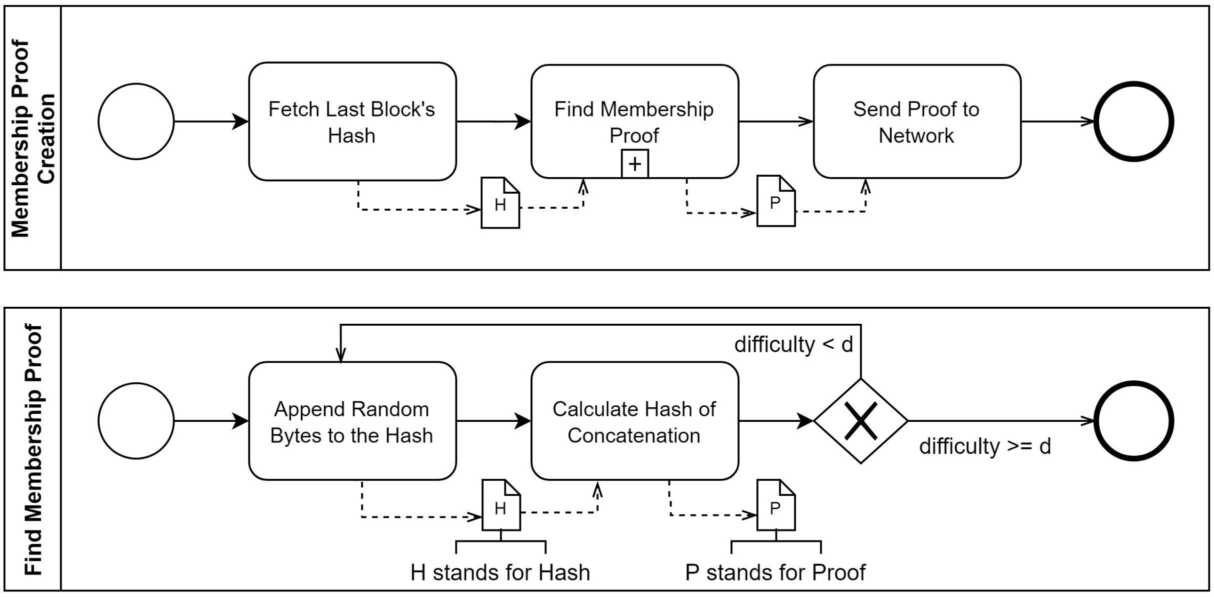

Figure 4.3: Membership proof creation and submission

as a fixed number of membership blocks. When the solution is found, the node packages it with the aforementioned information and sends it to the network.

Note that a node cannot predict the PoW seed and, consequently, cannot perform work ahead of time. This is because the PoW seed is the hash of the latest membership block appended to the chain, and therefore, in addition to being random and unpredictable given the properties of hash functions, it only becomes known from the moment the block is created and appended.

4.3.2.B Creation of Membership Blocks

When the membership commitment proof is created and sent to the network, it is relayed until it reaches the membership committee. Upon receiving the proof, the committee is responsible for validating it, packaging it into a block and appending it to the membership chain. As mentioned, for each proof that is received, a BFT run among the committee members is triggered to check if this proof is valid. The BFT protocol itself will be discussed later in Section 4.3.5, since the same protocol is used for both confirming transactions and confirming membership commitment proofs, and consecutively generating a transaction block or a membership block, respectively. In our prototype, we employ a leader-driven BFT protocol, so the following explanation takes this into consideration, but it can be easily adapted to any other BFT protocol. Therefore, what matters is to define what is a valid membership block, or in other words, the confirmations done by each committee member on that block.

So, once a proof is received, the committee leader will create and propose a new membership block, which will be validated and agreed among the committee members. The steps for validating the block taken by the remaining members will be the same taken by the leader when creating the block. The mechanisms to detect a malicious leader will be discussed when the BFT protocol is described in Section 4.3.5.

The first confirmation is made on the proof itself: it will be confirmed if the PoW seed is the hash of the latest membership block and checked if the proof has the predefined difficulty for that era. In our prototype, the difficulty is constant but can be easily modified to increase or decrease as necessary. To check the proof's difficulty, one just has to take the PoW seed, concatenate it with the proof, calculate its hash and confirm that the hash value is below the predefined difficulty. This highlights an important advantage of PoW: it is very hard to calculate the proof and extremely costless to validate it.

From here, there are two options: either this new block still belongs to the current era or it is a reconfiguration block. The actions taken in case of a reconfiguration block are explained in the next section. We therefore focus on the case when the new block is still part of the current era, in which case, following the template previously presented, its fields will be the following:

- **Block index**: The latest block index incremented by one.
- **Era counter**: The same era number as the one present in the latest block since this is not a reconfiguration block.
- **Reconfiguration seed**: This field is not used in non-reconfiguration blocks.
- **Membership commitment proof**: The membership commitment proof that triggered and justifies this membership block.
- **Committees**: The constitution of the committees is the same as the latest membership block since this is not a reconfiguration block.
- **Registered members score**: These scores are the mechanism that SCALEET employs to control how many proofs each member has submitted during the current era. The score table of this new block will be the same as the one from the latest block with the exception of the score of the member to whom the membership commitment proof present in this block belongs to. The score added to that member depends on the function applied to the previous score of the member. The function used in our prototype is simply incrementing the previous score of the member when a new proof is received. As we will see in the next section, the members with the highest scores at the time of the reconfiguration block will be designated as validators. This function can be as complex as the designer wants, in order to be as realistic as possible. For example, one could incorporate the idea already discussed in Section 3.1 behind Repucoin [6], by making the function also depend on the valid proofs that the node had submitted during previous eras, i.e., the

reputation of the node. That way, we could benefit nodes that had produced valid proofs during long periods and avoid sheer mining power attacks.

4.3.2.C Secure and Unbiased Reconfiguration

This Section describes the path taken when the membership committee appends the last membership block of the current era. This block is called a reconfiguration block and represents a marker between eras: from this block on, a new era begins. New validators for the following era will be chosen from the ones registered in the current era and will be randomly distributed across the committees, replacing some of the old validators.

Firstly, a set of the registered members have to be chosen to become validators. As explained in the previous section when the scores were described, the new validators will be the top scorers when the reconfiguration block is reached. Both the distribution of the new validators across the committees and the old validators that are replaced need to be random, unbiased, unpredictable and to provide third-party verifiability in order to make it impossible for a node to choose an assignment that fits his agenda better. If such properties were not held, one could control the assignment and increase the probabilities of overpowering a committee with multiple entities. In addition to the membership proofs, this mechanism can also be seen as extra protection against Sybil attacks, because it narrows down the probabilities of multiple entities of an attacker being assigned to the same committee, even if that attacker had managed to have multiple validators in the same era.

Again, let us see how the committee leader creates the new block and, consequently, the confirmations made during the BFT run. Basically, it is the committee leader that will propose the new constitution of the committees but will be able to prove that the new assignments were done satisfying the previously mentioned desired properties. Multiple proposals exist for a distributed-randomness generator [48, 50] but all of them would add a great cost to the reconfiguration phase or even degrade liveness as is the case in Omniledger [12]. Our solution is simple, scalable and does not imply extra steps of communication apart from the ones that the BFT run already incur, while maintaining the desired security properties. In particular, following the same line of thought of the previous section when describing the path taken in the case of a middle era block, the configuration block proposed would be the following:

- **Block index**: the latest block index incremented by one.
- **Era counter**: the latest block era incremented by one, since this block marks the beginning of a new era.
- **Reconfiguration seed**: proof that will enable all third-parties to validate the random assignments of new validators.

- **Committees**: the new constitution of the committees taking into account the new validators and the ones that were removed.
- **Registered members score**: the scores are reset.

Next, we will show how we can get to the value of the reconfiguration proof and the new constitution of the committees. First, recall that the top scorers will be designated as validators (and that the number of new validators in each era is constant). Now, all of these new validators have to be randomly assigned to the committees replacing a set of old ones. In order to achieve that goal, our solution leverages the randomness of the membership commitment proof and works as follow:

1) Find a random, unpredictable and third-party verifiable seed: it is important to find a seed that is random, unpredictable and third-party verifiable that can serve as seed to the reconfiguration process. For this purpose, we used the hash of the latest block in the chain. It is easy to demonstrate that this hash satisfies the aforementioned properties:

- **Random**: One of the characteristics of hash functions is their pseudo random nature.
- **Unpredictable**: Since hash functions are deterministic, we have to guarantee that the input used is unpredictable. In our case, the input used for the hash function is the set of properties of the block being hashed, which include a membership commitment proof that originates that block. This previous proof cannot be easily predicted, since it is a result of a PoW run and furthermore it would be difficult for the leader to somehow influence this by ignoring some recently completed proof, since the BFT consensus protocols include mechanisms, namely a leader change after a timeout, to prevent a situation where a pending operation does not execute for a long period of time.
- **Third-party verifiable**: This hash is public and shared among all the committee members that need to verify the new committees' constitution (in fact, by all the remaining system nodes). Therefore, any node can reproduce the steps made by the leader to reach the new constitution and validate it.

2) Find a committee for each new validator and replace old validators: now that there is a random number that can be used as seed, we used this number to assign each new elected validator to a committee, either to a transaction committee or to the membership committee. To this end, for each new validator, the following steps are executed:

- **Find a committee**: To find the new committee, first we hash the concatenation of the values of the seed and the public key of the new validator. Then, we apply to the result in the previous operation the modulo operator by c, i.e., $H(seed \parallel PK) \mod c$, where c is the number of committees. The final result is an integer between 0 and $c-1$ that corresponds to the identification of the committee to which the new validator will be assigned.

- **Choose an old validator to leave**: In a given committee, the operation to determine the node to be replaced is very similar to the one that had found the committee, i.e., instead of computing a value modulo c, we use modulo cc, where cc is the number of committee members of that committee. The result is an integer between 0 and $cc-1$ and it represents the index of the validator that will be removed.

- **Find new leaders for each committee**: Finally, the leader of each committee is also swapped. To this end, the operation $seed \mod c$ is applied for each committee and the result is the index of the leader for that committee.

After the new block has been validated by the membership committee members, it is relayed across the network. Upon receiving, validating and appending the new block to the membership chain, each member is faced with some options and has to update its state accordingly:

- **If it was a validator**, then either:
  - it will remain as a validator and just has to update the state of the membership, or
  - it will no longer be a validator, and it can clean up its state regarding the transaction chain that it held.

- **If it was a relay node**, then either:
  - it will remain as a relay node and, therefore, does not need to update its state, or
  - it is indicated as a new validator, and, in addition to updating the membership, it has to fetch the last transaction block of the transaction chain maintained by the committee it was assigned to. Note that the bootstrap time is practically insignificant because it only has to fetch one block to start participating in the BFT runs. This is because, as we will see, each transaction block has a snapshot of the balances of each participant and can be validated without the need of the previous blocks, since it holds sufficient valid signatures made by the previous membership

4.3.3 Transaction Validation

In the last Section, we described how the membership management works and specifically what is the role of the committee of membership on this process. Now that we have already detailed how the membership evolves throughout time and the techniques used to maintain the system on the *permissionless* setting, it is time to reveal how the transactions are validated and subsequently appended in a form of a transaction block to a transaction chain.

As we already explained, the state of the transaction chain is sharded across a set of multiple transaction committees, which in fact means that there are multiple transaction chains where each one accommodates a set of accounts. Each transaction committee manages all the accounts, e.g., addresses, that are part of the chain it maintains. Given this, a special case arises when a transaction is issued from a certain address to an address that is out of scope of the committee that handles the source address, i.e., when the committee that manages the source address is different from the one that manages the destination address. In such cases, the transaction is called an *inter-committee* transaction (and *intra-committee* otherwise). The rest of this Section is divided into two subsections: in the first subsection we will describe the protocol assuming that all the transactions are intra-committee, and in the second subsection we will extend the protocol to handle inter-committee transactions.

First, it is important to understand what are the main internals of a transaction and transaction blocks. Starting with the transactions:

- **Amount**: This is the amount of SCALEETS to be exchanged from one account to another. The currency in SCALEET is called SCALEETS.
- **Source address**: Address from where the SCALEETS are charged.
- **Destination address**: Address where the SCALEETS are credited.
- **Account nonce**: Nonce to prevent double spend attacks.
- **Hash**: Hash of the transaction, covering the previously mentioned fields.
- **Signature**: Signature of the hash, computed with the PK^{-1} corresponding to the source address. This ensures that the transaction was in fact issued by the holder of the source address.

Then, regarding transaction blocks:

- **Block index**: The height of the block in the chain.
- **Previous block hash**: The hash of the previous block.
- **Parent block hash**: The hash of the latest membership reconfiguration block, which made public the current membership.

- **Set of transactions**: The set of transactions that take effect after this block is appended.
- **Balances after block**: The balance of each account after the block is appended.
- **Account nonces after block**: The latest account nonce used for each account after the block is appended.
- **Hash**: Hash of the transaction block, covering the previously mentioned fields.
- **Prepare and commit signatures**: List of signatures from the BFT consensus run that allowed this block to be appended to the chain. Again, we discuss their usage subsequently in Section 4.3.5.

4.3.3.A Intra-Committee Transactions

Note that the following protocol is executed by all the transaction committees in parallel, but we will focus the explanation by focusing on one of these committees. Every time a given transaction is issued, the network has the job to relay it until it reaches the committee that is responsible for the source address. The way that each node has to confirm what is the committee responsible for a given address is by doing the operation: $1 + (H(sourceaddress) \mod tc)$. This operation outputs a committee identification between 1 and tc, where tc is the number of transaction committees. Note that the committee identification 0 is assigned to the membership committee, which explains why we add one to the result of the mod operator.

When the committee leader receives a request to process a new transaction, it buffers the request until it reaches a parameter corresponding to the batch size, i.e., the predefined number of transactions that each block will accommodate, it creates a new transaction block with that set of transactions and triggers a BFT run with this block. After the block is validated by the committee members, the block becomes final and it is appended to the committee's transaction chain. Since both the source and destination addresses belong to the same committee, there is no need to have communication with any other committee.

For a transaction to be considered valid it has to satisfy some requirements:

- The signature has to be valid as a way of proving that the issuer for the transaction is the entity that manages the source account.
- The amount of SCALEETS exchanged has to be positive.
- The nonce present in the transaction has to be the following one for that account, i.e., this transaction is the following one to be applied to the source address.

4.3.3.B Inter-Committee Transactions

If the destination address does not belong to the same committee as the source address, the previous protocol does not suffice, since, after appending the block, the SCALEETS will be debited from the source account, but the committee itself does not have the required information to credit the destination account. To bridge this gap, we introduced an inter-committee protocol, which solves the aforementioned issue in a simple and efficient way. First, recall that as a consequence of the way SCALEET shards the state, an inter-committee transaction always involves two committees, the one responsible for the source account and the one responsible for the destination account.

The inter committee protocol is achieved by extending the one described in the previous Section. From here, we will use the term coordinator committee when referring to the committee responsible for the source address and target committee to identify the one responsible for the destination address. At a high level, the protocol works as follows: after appending a block that contains at least one inter committee transaction, the coordinator committee will send a proof in a form of a new transaction to the target committees (note that inside a block transactions that have different target committees can exist), which proves that the SCALEETS were indeed debited from the sources addresses and can now be credit to the destination addresses. These proofs will be treated as transactions and will be validated and packaged inside transaction blocks on the target committees. Next, the protocol will be described in more detail by explaining first the steps on the coordinator committee and then on the target committee of a given transaction:

Steps on the coordinator committee

At the time of each transaction block proposal, the leader will also create and propose a new special transaction for each inter-committee transaction present in the block it is proposing. Each of these new special transactions will represent a transaction from the coordinator committee to the target committee of the original transaction. These transactions will be signed by the members of this committee (coordinator committee) in order for the target committees to be sure that the SCALEETS were indeed debited from the source accounts in the coordinator committee, and can be credited into the accounts on the corresponding target committee. These signatures are the aforementioned proofs and can be seen as an attestation proving that the original transaction was applied in the coordinator committee.

Figure 4.4 depicts an example of an inter committee transaction and its corresponding special transaction. On the left side, there is a transaction issued from account $address1$ to account $address2$. Account $address1$ is managed by committee 1 and account $address2$ by the committee 2. On the right side of the image, there is the transaction created and proposed by the leader of the committee 1 that,

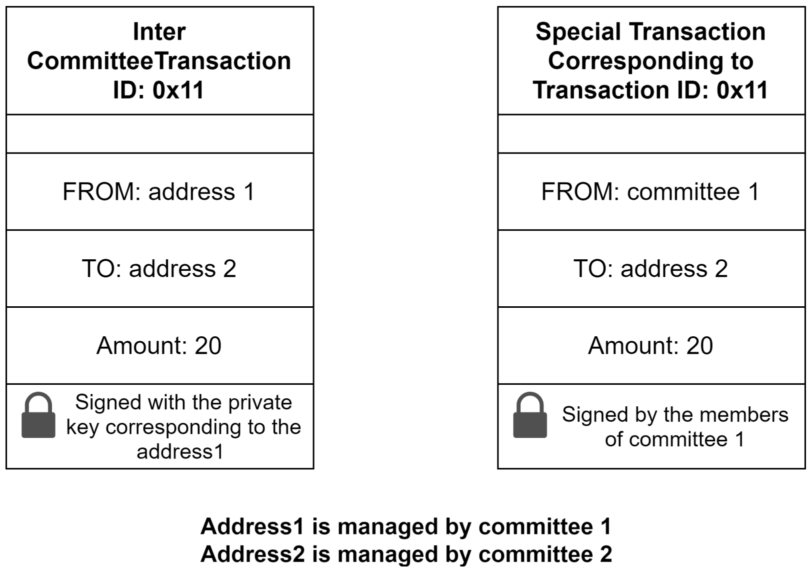
Inter CommitteeTransaction ID: 0x11
FROM: address 1
TO: address 2
Amount: 20
Signed with the private key corresponding to the address1
Special Transaction Corresponding to Transaction ID: 0x11
FROM: committee 1
TO: address 2
Amount: 20
Signed by the members of committee 1
Address1 is managed by committee 1
Address2 is managed by committee 2

Figure 4.4: Example of a inter-committee transaction and its corresponding special transaction.

after being signed by committee 1, will be sent to committee 2 in order to prove that the SCALEETS were already debited from account $address1$ and can now be credited on account $address2$.

Therefore, when each committee member is validating transactions, it will sign each of the new special transactions created and proposed by the leader. These signatures ensure that, given the state of the committee member that emits it, that this transaction can happen. Note that all the requirements for a transaction to be valid can be checked in the coordinator committee. This proof is simply a signature on the new transaction hash. When each BFT run ends, each of the new inter committee transactions is signed by at least $2f+1$ committee members, which is sufficient. Note that just $f+1$ signatures were needed to ensure correctness, but the BFT protocol itself needs at least $2f+1$ signatures in each phase to progress, so we also collect the corresponding proof of each member which totals the $2f+1$ proofs.

Finally, each of the new special transactions is relayed throughout the network until it reaches the corresponding target committee.

Steps on the target committee

These special transactions will be eventually received by the target committees and will be handled as normal transactions. The only difference is at the time of confirming these transactions: for a transaction to be valid it just has to have $2f+1$ signatures, which assert that it was validated correctly at the coordinator committee. As already stated, these signatures ensure that the coordinator committee can be seen as a unique and correct "node". To avoid double spending in inter-committee transactions, the nonces are again used but now regarding the coordinator committee, i.e., each committee keeps a counter for every other committee, ensuring that each inter committee transaction arriving from that committee is applied sequentially and only once. After being confirmed, these transactions will just credit the SCALEETS on the destination account, since the balance of the source account was already updated on the coordinator committee.

To sum up and by merging all the possibilities described, when appending a validated block to the transaction chain and subsequently updating the account balances with the transactions in that block, a committee can face one of four options:

- **Intra committee transaction**: The SCALEETS are debited from the source account and credited in the destination account.
- **Inter committee transaction**: The SCALEETS are just debited from the source account.
- **Special transaction signed by a set of validators**: In this case, there are two options:
  - **On the coordinator committee for that transaction**: There will exist one of these transactions for each inter committee transaction in the block. These transactions do not have an impact on the coordinator committee, since they represent a proof that the SCALEETS were actually debited from the source account and can now be credited to the destination account on the target committee.
  - **On the target committee of that transaction**: This transaction was sent by other committee, i.e., the coordinator committee of this transaction. The SCALEETS were already debited from the source account on the coordinator committee, therefore, only the destination account on this committee need to be updated.

Regarding the correctness of inter-committee transactions, we need to consider what happens if the leader of a coordinator committee of a given transaction does not forward the corresponding agreed special transaction to the target committee. To address this issue, the leader is also forced to forward the special transactions to the members of its committee with the risk of being considered faulty otherwise. The committee members after producing the proof for a given special transaction, start a timer for that transaction. If such transaction is not received from the leader until that timer expires, the leader is considered faulty by that member, which will eventually start a leader change. Note that by enforcing each special transaction to be forwarded to at least $f+1$ members of the coordinator committee, at least one member is correct and will relay this transaction to the target committee. Note that even if the same transaction is submitted twice to the target committee, the nonce that it contains prevents it from being executed twice.

This protocol can be categorized as a lazy update protocol since, for a given inter-committee transaction, there is a moment when the SCALEETS were already debited from the source account and not credited to the destination account, i.e., it is not atomic. In SCALEET, this does not represent a problem since, from the moment the original transaction is validated in the coordinator committee, i.e, the SCALEETS are debited from the source account, the matching special transaction is on its way to the target committee and will ensure that these funds will eventually be credited in the destination account. This way, we avoid freezing the BFT run on the coordinator committee each time an inter-committee transaction is validated, which implies that this incurs in a very small overhead on the performance of the system. On the whole, even though the concept of atomicity does not apply, the system will eventually converge to a correct state since if the funds are debited from one account will be for sure eventually credited to the destination account. Moreover, as we well see, given the performance of the system the interval between debit and credit in such cases is short.

4.3.4 Reward Assignment

Until now, we have been assuming that all the transactions are issued by clients. However, this is not the case when we consider reward transactions. Reward transactions are used to reward validators that participate and are engaged with the network by validating both transaction blocks and membership blocks. These rewards can be seen as an incentive given to validators in order to promote their commitment to the system and make them continue to behave honestly towards the network. In reward transactions, there are no source accounts: the currency is created and assigned to nodes, and therefore this is also the mechanism used by blockchains to generate more currency throughout time. There are several proposals for assigning rewards, e.g., in bitcoin a node that mines a block inserts, in that same block, a reward transaction whose destination is its own account.

Issuing rewards becomes an interesting problem to analyze when each block is validated by multiple validators, as is the case in SCALEET. In this scenario, there are basically two options:

- **Give rewards to every validator of the committee per each block that is validated**: This solution, in spite of being simple to implement, is far from being a fair approach. This is because it would be encouraging floppiness and non-participation since, after a node is designated as validator, it would receive a reward for each block validated by the committee it belongs to, even if it is not participating in any of these validations.

- **Give rewards only to the validators of the committee that participate in each block validation**: In contrast, this approach has as the main advantage encouraging the proactivity of every validator, since their only possibility of being rewarded is by participating in block validation. However, this type of approach is more difficult to implement, since it is necessary to prove what were the nodes that participate in each block validation. This option was the one we chose to implement on SCALEET.

We therefore have to address the challenge of identifying who has participated in a block validation. Moreover, even if we know the nodes that had participated in a given block validation, this information is only acquired a posteriori, which disallows the possibility of proposing the reward transactions along with the block proposal to which those rewards refer to. As a consequence, the reward transactions corresponding to a given block validation are only going to be validated, and subsequently assigned to the validators, after the block they refer to is validated and appended to the chain.

The way that SCALEET addresses this challenge is by having each committee appending one extra block per era, called a reward block, as the last block of that era. This block will contain all the reward transactions (one transaction per validator) corresponding to all block's validation during the era. The reward of each block validation is equally divided across all the validators that have contributed to that validation, which results in a given share per validator. The total amount of each reward transaction is calculated by summing up all the shares earned by a given validator during the era. Note that the goal of the reward block is to agree on the reward transactions to be created and not applying the transaction directly since the destination accounts of reward transactions can be managed by other committees. Therefore, after this block is agreed and appended, the transactions present in it are relayed to their corresponding target committees, where they will be handled as normal transactions, as happens with special transactions created by the coordinator committee in the case of inter committee transactions.

Next, we need a way to correctly infer the set of nodes that deserve a reward in each era. It is clear that we could not let the committee leader choose the members that receive rewards (or, for that matter, any other committee member on its own). However, since most BFT protocols are leader-driven, we can therefore allow the leader to propose a set of rewards, provided that this proposal is vouched for by a sufficient number of other replicas of the BFT protocol. Thus, we have to find a way to allow this decision to be verified by the members of the committee. To this end, for each validated block in an era, each committee member will create a proof, called reward proof, where it packages the first $2f+1$ signatures it receives for each phase of the BFT protocol, then sign it, and send it to the leader. At the end of an era, the leader has, for each validated block, at least $2f+1$ proofs ($3f+1$ minus f possible faulty or slow nodes), where each proof contains at least $2f+1$ signatures. The rewarded members for each block will then be the members whose signatures are present in at least $f+1$ proofs. Finally, after the leader calculates the reward for each member using this method, it can create and propose a reward block with that information and with the received proofs.

Note that the leader can still choose what are the $2f+1$ proofs that it uses for the reward distribution, however, its choice will highly depend on the proofs it receives. Since proofs are unforgeable, one may wonder why we only assign rewards to members that appear in at least $f+1$ proofs: this happens because the signature by itself does not prove that the node had participated on a given block participation, because the signature can be done at any time, even after the block validation process has ended. So, given that we can have at most f failures on a committee, $f+1$ is the required quorum to ensure that the signatures present in a given proof were indeed received by the member that correctly issued it at the time of the block validation.

Furthermore, since it is the membership chain that controls the era changes, we require the membership committee to notify all the committees to produce the reward block, whenever a reconfiguration initiates. Upon hearing about the reconfiguration, all the members of each committee are notified in order to detect if the leader does not trigger the BFT run for the reward block. The membership change described in the reconfiguration block only takes effect on each committee after that committee has appended the reward block for the previous era. In contrast, at the level of the membership committee, the membership change takes effect immediately after the reconfiguration block has been appended, since there are no reward blocks appended by the membership committee.

Finally, the process described in this Section can also be applied to assign transaction fees to the validators by including them in the reward/fee block. Transaction fees were not implemented in the prototype because they portray little relevance to the scope of this work and its implementation could be easily added to a final deployed system.

4.3.5 BFT Protocol Running on Committees

As we explained in the previous Sections, each committee uses a BFT algorithm to make decisions, namely to agree whether a block is valid or not. Specifically, we have implemented in SCALEET a well-known BFT algorithm: *Practical Byzantine Fault Tolerance*, or PBFT [23]. We presented an overview of this protocol in Section 3.2.0.B. Essentially, it starts with a block proposal from the committee leader and ends with that block being considered valid or not. If it is considered valid, a set of valid signatures that prove that the block was validated is appended to the final protocol message, making this validation possible to be verified by anyone in the future.

Our choice of using PBFT was not because of its performance or scalability, since there are other protocols that appeared as a follow-up to PBFT improving it in that regard [46]. We chose PBFT because it is a well-studied representative of the family of BFT consensus protocols. In fact, SCALEET is developed in a way that it is very easy to change the algorithm used in favor of others with better performance or scalability, since all the other components of the design do not depend on the BFT algorithm. This way, and even in a permissionless setting (unlike prior work), we can leverage the advantages of BFT consensus protocols and choose the most appropriate protocol for a given deployment.

4.3.6 Concluding Remarks

In Section 3.3 we discussed the state-of-the-art blockchain systems and subsequently their techniques to achieve both a good scalability and performance, and we concluded that traditional BFT solutions could reach higher throughput and lower latency when compared to the ones under the Nakamoto consensus paradigm. However, such solutions exposed two major issues: non compatibility with a permissionless setting and scalability issues. The main goal of SCALEET was to try to achieve the best of both worlds, by coming up with a design with high levels of modularity where it is possible to plug any traditional BFT algorithm into the system, allowing us to leverage its higher performance features in a permissionless environment. Furthermore, by allowing different BFT protocols to be used as a drop-in replacement, SCALEET allows for an easy integration of newer and more performant BFT algorithms as they are being developed.

Another important feature of SCALEET is the guarantee of deterministic finality, both in the membership and transaction chains. Previous systems with two chains [12, 28] relegate PoW to a second plane by moving it from the transaction validation pipeline to the second chain management. In spite of improving the throughput by removing the PoW from the critical path (transaction validation), they have inserted liveness issues on the system by allowing forks on the second chain. SCALEET moves one step further by completely modularizing the use of PoW. This is achieved by using PoW features purely as a Sybil Resistant mechanism, while the membership management is also managed with the aid of a BFT consensus-based algorithm. Note that PoW is not used continuously in our system, having its use restricted for creating membership commitment proofs when a given node wants to become a validator. Therefore, it can be seen as a parameterized “knob” for the membership management, allowing one to choose its desired trade-off between more security or more performance: at the limit, one can force all validators to be removed from the membership, and subsequently replaced by new validators, at the reconfiguration phase, meaning that every validator for one given era has to have had produced work during the era before (for creating the membership proof).

SCALEET leverages sharding by accounts, which, as we discussed, allows two main optimizations, namely the possibility of applying clustering on accounts and the fact that an inter committee transaction can span only two committees. Still regarding inter committee transactions, SCALEET introduces a “lazy” inter transaction commit protocol that although it is not atomic, ensures the same transaction principles as others systems (no currency creation, no double spending and the guarantee that when a given amount it is debited from one source account, the same amount is always eventually credited on the destination account), while enabling higher levels of performance.

SCALEET also proposes a fair and novel approach for assigning rewards, ensuring that rewarded members had indeed produced valid block confirmations.

5

Implementation

Contents

This Chapter presents our prototype implementation of SCALEET. It starts in Section 5.1 by giving an introductory overview of SCALEET'S implementation and then dives into more details about its main modules and their relation in the following Sections.

5.1 Implementation Overview

We implemented a SCALEET prototype in Java 8, consisting of approximately $7,500$ lines of code and, as detailed in the next chapter, a baseline implementation equivalent to Bitcoin was also implemented, consisting of approximately $3,000$ lines of code. The SCALEET prototype was developed from scratch, since the set of modifications needed to be performed to build SCALEET on top of other known blockchain systems, such as Bitcoin [1] or Ethereum [26], would be substantial as they represent a significant departure from their design. This would complicate a direct experimental comparison to these systems.

Each node runs as a *Spring* application[1] and can be seen as a web service that provides a set of publicly accessible services. The various system nodes communicate and access each other's services through REST with the use of the HTTP stack of suitable actions, mainly with *GET* and *POST* requests. An IP book is currently provided to each node, listing the IP address and port number for each node's public key. In a real deployment, each node could sign this information and gossip it throughout the network. Regarding the communication pattern, there are two cases: (1) inside each committee, it follows an all-to-all pattern, and (2) for other protocol aspects, each node connects to five random random other nodes and forwards the message to each of them.

We also developed a wallet application in Python, which allows a user to manage its addresses and, subsequently its SCALEETS by issuing commands to the network.

All of this code is publicly available on Github [2].

The remainder of this chapter describes the main modules that comprise the SCALEET prototype, as depicted in Figure 5.1.

5.2 Util Module

The utility module comprises a set of functions and primitives that are used across all the other modules. The ones worth mentioning are the Hash, public key signature primitives and database management. Hashes were implemented using a *SHA-256* instance. Regarding the digital signatures primitives, they were implemented using a *SHA-256-RSA* instance and they provide the possibility to create a new key pair, sign messages and verify signatures. Finally, as a relational database management system we chose *SQLite*.

[1] https://spring.io
[2] https://github.com/jpcc1997/blockchain

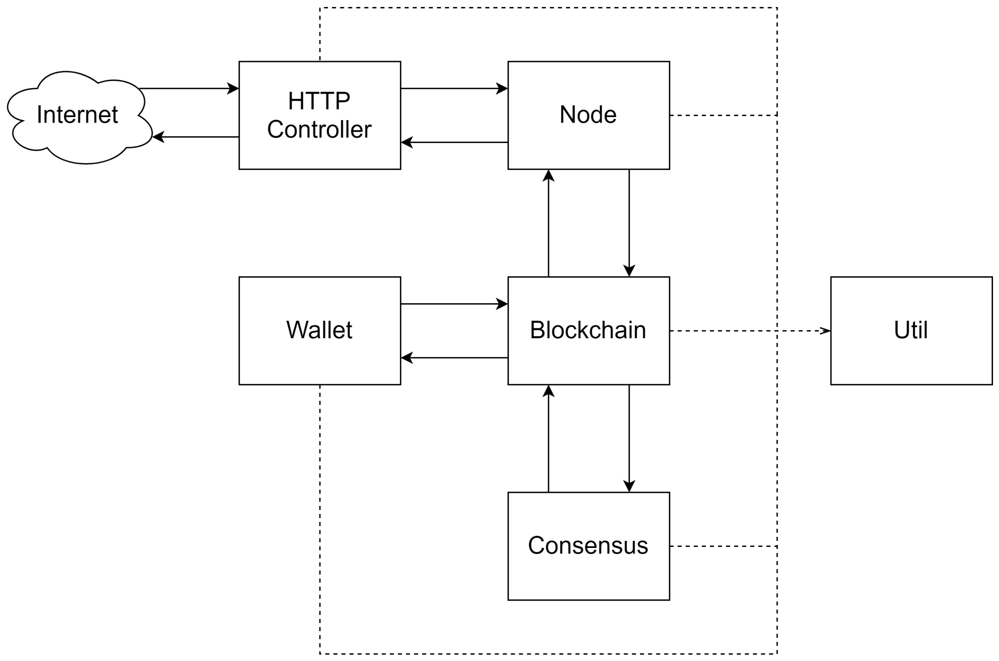

Figure 5.1: SCALEET architecture modules.

5.3 HTTP Controller Module

The HTTP controller module is both the entry and exit point of a node. On the one hand, it exposes a set of services to the internet, which can be accessed with GET and POST requests. On the other hand, all the messages from a given node exit through this module by accessing services provided by another node's HTTP controller module. It is also in this module where a given *Data Object*, *DTO*, is constructed in case of being an outgoing message or where the information is extracted from a received DTO, in case of being an incoming message.

5.4 Node Module

The node module encapsulates all the information regarding the peers for a given node and manages all the actions needed to be done in order to allow the chosen communication pattern to be put in place. It is also in this module that we confirm from whom each message is arriving by verifying signatures before the message is pushed to the HTTP controller.

5.5 Blockchain Module

The blockchain module is the backbone of the SCALEET prototype and can be seen as an orchestrator, in which all the functionalities exposed by a node are controlled. The local blockchain state is managed by this module, namely membership blocks, transaction blocks, unconfirmed transactions, account balances and nonces.

5.6 Wallet Module

The wallet module provides all the services regarding wallets and accounts, namely creating new wallets and new accounts for a given user.

5.7 Consensus Module

The consensus module encapsulates the implementation of the chosen BFT algorithm, which in our implementation is PBFT [23]. As previously stated, as long as the BFT algorithm exposes the interface for starting a new decision either on a transaction or membership block and outputs a decision with the corresponding proofs, this can be replaced by any other BFT protocol.

6 Evaluation

Contents

This Chapter presents and describes the experiments that were carried out in order to evaluate our prototype of the proposed solution. As previously stated, since our prototype was developed from scratch and given the complexity and size of blockchain systems, comparing our system directly to widely deployed systems Bitcoin [1], Ethereum [26] or others would not be a fair comparison, since the other systems are completely different code bases that are the result of many years of effort made by thousand of contributors.

Therefore, in order to bridge this gap, a baseline prototype embodying the main design characteristics to Bitcoin was also developed, starting from the code base of SCALEET. This way, since SCALEET can be seen as a product of applying the proposed optimizations to the baseline solution, a comparative evaluation can be considered as apples-to-apples.

This Chapter starts by stating the goals of the evaluation in Section 6.1 and by detailing the experimental settings used in Section 6.2. Then, in Sections 6.3 and 6.4, it describes and discusses the outcomes of a series of experiments in the baseline and SCALEET prototypes, respectively. Finally, in Section 6.5, it highlights the main key takeways from the evaluation.

6.1 Goals

The main high-level goal of our evaluation is to measure the performance and scalability of SCALEET when compared to the baseline solution.

In this context, scalability can be measured by varying the number of nodes, whereas performance is measured by the throughput (in *tps*) and latency.

In more detail, the evaluation should attempt to answer the following questions:

- What is the latency that SCALEET can achieve for validating transactions?
- What is the throughput that SCALEET can achieve (measured in tps)?
- How do the previous metrics vary as the number of nodes grows?
- What is SCALEET'S resource consumption (CPU, storage and bandwidth)?

Furthermore, it is important to understand how the answer to these questions varies as we change other parameters, like the block length, number of committees, size of committees and the percentage of inter-committee transactions.

Finally, we also want to evaluate how the previous metrics are affected by events outside the common case, namely a system reconfiguration.

6.2 Experimental Setup

To answer the previous questions and gauge the practicality of our design, the prototype was deployed on our local cluster at *INESC-ID/Instituto Superior Técnico*. The cluster consists of *21* machines, each of which has the following specifications: Intel(R) Xeon(R) CPU E5506 @ 2.13GHz CPU with 8 cores, 40GB of RAM and up to 1 Gbps of network throughput. As means of having more control over the processes and measure the performance of SCALEET with as many nodes as possible, a Virtual Machine (VM) with *16* docker containers was deployed in each machine, totaling a maximum of *336* nodes when using the entire capacity of the cluster (each docker container is a node).

To effectively control the deployment and management of servers during our evaluation, we assigned a coordinator role to a separate machine, which uses automation tools, namely Ansible [1] and Vagrant [2]. These tools enable controlling not only their bootstrap and shutdown before and after each evaluation run, but also issues all the needed operations during the system's lifespan throughout the wallet application also developed, such as issuing transactions.

All the experiments and results presented throughout this Section are the average of the measurements taken by all the nodes through 3 runs and based on the results of the first $1,000$ blocks generated.

6.3 Baseline Experimental Results

First, we will present and discuss the results of the baseline prototype, with the goal of gauging the performance and scalability of the baseline we implemented in absolute terms. The experiments were performed varying the number of miners with the following values: 16, 32, 64, 128, 256 and 336. Regarding the block size, i.e., the number of transactions it accommodates, it is fixed at $2,000$ transactions per block, which is the average number of transactions per block in Bitcoin. The block size is *per se* an object of study and discussion, with arguments either for or against increasing or decreasing it [51]. Thus, we consider this an orthogonal discussion that is beyond the scope of our work, which justifies why that number is fixed.

6.3.1 Throughput

In this experiment, we evaluate the throughput of the baseline prototype. The throughput is measured by dividing the number of transactions validated, and subsequently inserted in blocks, by the time of the experiment. Moreover, in order to measure the maximum throughput correctly, the clients need to saturate the system with a large number of transactions from the moment the experiment starts, so that

[1] https://www.ansible.com
[2] https://www.vagrantup.com

the maximum throughput is reached. In our experiments, we achieve this by conveying to the miners all the transactions that will be run before the measurement starts, so that these nodes are not handling incoming transactions while blocks are being validated.

Given that our cluster resources were limited, we could not have a large set of miners, since this would imply many miners on the same machine competing for CPU. In other words, it would be impossible to launch 16 nodes per machine as we did, and allow each one of them to mine concurrently, since it substantially increase the mining time due to contention on the CPU resources of the machine. Thus, in order to mimic the value that, on average, a block takes to be mined in Bitcoin network, the difficulty of each PoW is set to one, which causes this puzzle to be solved instantaneously and then, before making this new mined block public to the rest of the network, the node randomly waits between 9 and 11 minutes. This value is justified by the fact that, for security reasons [1], in Bitcoin, the difficulty to mine each new block is constantly updated in order to take on average 10 minutes.

The results in Figure 6.1 show that the number of transactions added to the chain per second is steadily around 3.4, even if we increase the number of miners. This is aligned with what we expected, since the on-chain transaction processing capacity of the network is limited by the average block mining time of 10 minutes. Furthermore, as the number of miners increases, the probability of concurrent proposals (and consequently forks) appearing also increases, which could lead to a drop in throughput. This is one of the reasons that leads to dynamically adjusting the difficulty of mining a block in Bitcoin. In our baseline prototype, we do not update the difficulty, but the number of miners is also not sufficient to exhibit such impact on throughput.

Overall, the results show that, in spite of this baseline being capable of accommodating an increasing number of miners without impacting the throughput, it outputs an absolute throughput value that is considerably low, especially when compared to other transaction processing systems such as relational databases. In other words, our baseline prototype shows good scalability, but at a low absolute performance, in line with existing systems.

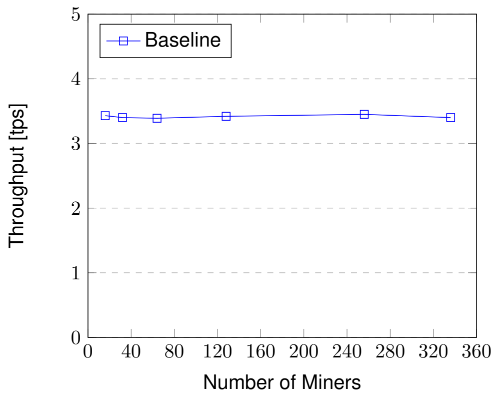

Figure 6.1: Baseline's throughput with increasing number of miners.

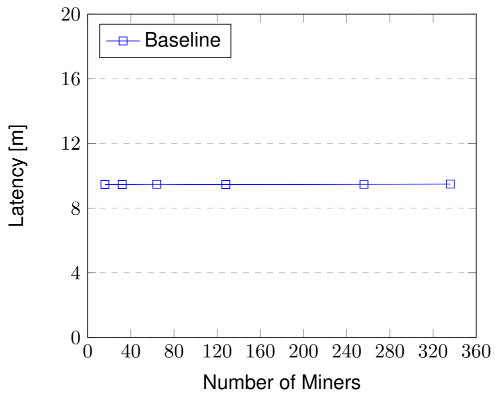

Figure 6.2: Baseline's latency with increasing number of miners.

6.3.2 Latency

In this experiment, we evaluate the client perceived latency of the baseline prototype. The latency of a transaction is the time elapsed between a given user issuing it and receiving the confirmation that it was inserted in a given block. In contrast to the throughput measurement, latency is measured when the system is not loaded with transactions, to avoid measuring any queuing effects.

As we can observe in Figure 6.2, the latency measured is almost constant when increasing the number of miners in the network, at around 9.48 minutes, which happens due to the same reasons presented for throughput. As we already discussed, in Bitcoin and, correspondingly, in our baseline prototype, the idea of finality is probabilistic. Therefore, in order for a transaction to be considered validated, the block that contains it has to be followed by a set of new blocks. In other words, Bitcoin confirmations for a given transaction represent the number of blocks in the blockchain that have been accepted and appended by the network, starting from the block that includes that transaction. The bigger the number of confirmations, the higher the confidence that this transaction will not be reversed. In this experiment, to avoid the discussion around the right number of confirmations to wait, we consider a transaction to be validated from the moment the block it belongs to is appended. A generalization of this metric would be to wait for the time required for appending the next b blocks, where b is a parameter defining the number of confirmations, and corresponding degree of confidence we want.

6.3.3 Resource Consumption

6.3.3.A CPU Usage

The CPU cost of running the baseline prototype is extremely high due to the use of PoW, normally within the interval of $90-100$% per machine. As we explained, to allow for running many instances on the same machine, we set the difficulty of each PoW to one, which allows this puzzle to be solved instantaneously, and then mimic the remaining time with a $sleep()$ function.

6.3.3.B Bandwidth

Regarding bandwidth, each node uses about 1 Mbps with a block size of 2.8 MBytes (containing $2,000$ transactions each). This bandwidth is computed as the total amount of data sent, divided by the duration of the experiment. Note that, in the case of our baseline prototype, the communication cost is not impacted by the number of nodes/miners in the network, since each node only sends data to a fixed set of neighbours.

6.3.3.C Storage Cost

Regarding storage costs, the largest fraction is taken by the blocks themselves: around 2.8 MBytes each. The unconfirmed transactions that a given node is aware of can also represent a significant cost of storage, namely around 0.0014 MBytes per transaction. The remaining cost is shared among the neighbors information and other auxiliary structures, and it is negligible when compared with the confirmed and unconfirmed transaction cost.

6.4 SCALEET Experimental Results

Finally, we present the evaluation results regarding the prototype of SCALEET. In this set of experiments, the setup concerning the number of blocks and the number of validators will remain the same when compared to the evaluation of the baseline. However, the structure of this Section is different from the previous one, since SCALEET has more parameters that can influence both the throughput and latency of the system. In particular, we will analyze how these metrics are influenced when we change the number of validators, number of committees, size of committees, and percentage of inter-committee transactions. To avoid confusion with the membership committee, we clarify that the number of committees presented in the following tables and plots refers to the number of transaction committees. With the exception of Section 6.4.2, where we will vary the percentage of inter-committee transactions to observe the corresponding impact, and the experiments that only have one transaction committee, in all the remaining experiments this percentage is fixed to 50%. We will also observe the impact of membership reconfigurations. Lastly, the resource consumption of the system is evaluated.

We will not evaluate the behavior of SCALEET in the presence of malicious nodes. Given the way SCALEET works, studying the impact of malicious nodes in a given committee is basically the same as studying its impact on the BFT algorithm that we employ. The presence of malicious nodes in PBFT is recognized by the community as having a major impacting factor on its performance, and some solutions have been proposed to mitigate this impact [52]. Moreover, as we explained early, SCALEET has a high level of modularity, and consequently the BFT algorithm can be easily swapped by another algorithm that provides better performance under attack.

6.4.1 Number of Validators

Evaluating SCALEET when the the number of validators increases can be done either by increasing the number of committees or the size of each committee. Therefore, this analysis is split into those two alternatives.

Table 6.1: SCALEET'S throughput and latency when increasing both the number of validators and number of committees.

Number of Validators	16	32	64	128	256	336
Number of Committees	1	2	4	8	16	21
Size of Committees	16	16	16	16	16	16
Throughput [tps]	299	580	1132	2174	4196	5396
Latency [s]	8.34	12.40	12.37	12.35	12.38	12.38
Latency with queue [s]	8.34	12.01	11.47	11.34	10.31	10.44

6.4.1.A Number of Committees

For this experiment, we used the values depicted in Table 6.1 for the number of validators, number of committees and committee size, respectively. Each column represents one experiment.

As we can observe in Table 6.1 and Figure 6.3, the throughput increases almost linearly with the number of committees. These results were expected considering that, in our solution, increasing the number of committees does not increase much the work done by each committee and therefore the throughput is expected to double when the number of committees is doubled. That said, given the way SCALEET shards the state, when one increases the number of committees, the number inter-committee transactions will naturally grow since there are fewer accounts per committee. This might raise the issue that, by increasing the number of committees, the communication performed by each committee also increases; however, as we stated in the beginning of this Section, we enforce the percentage of inter-committee transactions to stay the same. Moreover, as we will see, the presence of more inter-committee transactions does not affect throughput.

Regarding latency, as we can observe in Table 6.1 and Figure 6.4, it stays practically constant when increasing the number of validators and committees. This is related to the way that we measured latency, since, as we stated earlier, we issue transactions in batches (with the exact size of each block) only after the previous block was appended. After this event, we repeat this test and measure the latency with some transactions in the queue. This time, we can observe a small decrease in the latency, most likely due to the increase of sharding. In other words, if we increase the number of committees, each committee will hold fewer accounts, and therefore the number of transactions in the queue for each committee will decrease, and so will the time for them to be validated (latency).

Note that the reason for the first latency measurement to be slightly lower when compared with the following ones is the absence of inter-committee transactions when only one transaction committee is used.

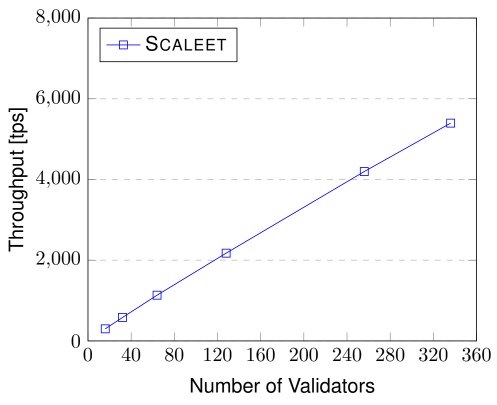

Figure 6.3: SCALEET'S throughput when increasing both the number of validators and number of committees, omitting the increasing of the number of committees.

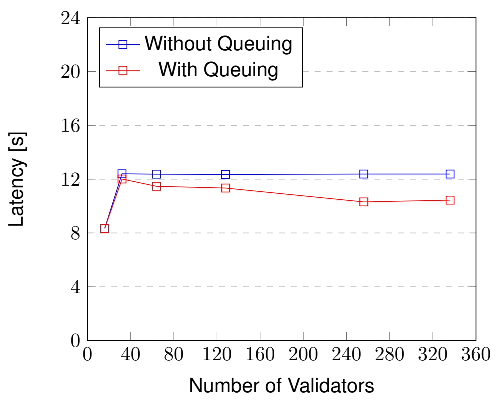

Figure 6.4: SCALEET'S latency with and without queuing, when increasing both the number of validators and number of committees, omitting the increasing of the number of committees.

6.4.1.B Size of Committees

For this experiment, we used the values shown in Table 6.2 for the number of validators, number of committees and committee size. Again, each column represents one experiment. As we can see, we fixed the number of committees while increasing both the number of validators and the size of each committee. Moreover, note that the number of tolerated faults increases in each experiment with the size of each committee, since the protocol requires that $n = 3f + 1$, where n is the size of each committee. In other words, if we have a committee of size $n = 7, 16, 25, 40$ we can have f possible faulty nodes with $f = 2, 5, 8, 13$, respectively.

As we can observe in Table 6.2 and Figure 6.5, the latency increases when we increase the size of each committee. This is related with the number of messages needed to be exchanged between the members in each phase of PBFT, specifically the number of correct messages needed to be received to proceed in each phase. As we discussed in Section 3.2.0.B, in each phase of the protocol each node needs to wait for $2f + 1$ messages from correct nodes; therefore, if f increases, the latency is also expected to increase. Another relevant aspect worth mentioning, although barely noticeable, is the progressive decrease of the slope of the line that joins each pair of consecutive dots. Our explanation for this effect is that it reflects the progressive decrease, as f goes up, between the number messages needed to be received to complete the operation and the total of nodes, i.e., the proportion between $2f + 1$ and $3f + 1$, which has a monotone decreasing convergence towards $2/3$ as f increases.

The decrease in throughput shown in Figure 6.6 is a consequence of what we previously explained: the time to agree in each new block is higher and since we do not have concurrent block confirmations (given that the primary replica of PBFT serializes the request execution) the throughput decreases.

Table 6.2: SCALEET'S throughput and latency when increasing both the number of validators and the size of committees.

Number of Validators	56	128	200	300
Number of Committees	8	8	8	8
Size of Committees	7	16	25	40
Throughput [tps]	3443	2174	1402	921
Latency [s]	6.51	12.35	18.11	25.35

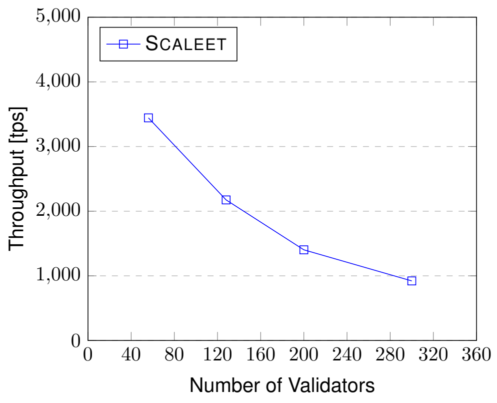

Figure 6.5: SCALEET'S throughput when increasing both the number of validators and the size of committees, omitting the increasing of the size of committees.

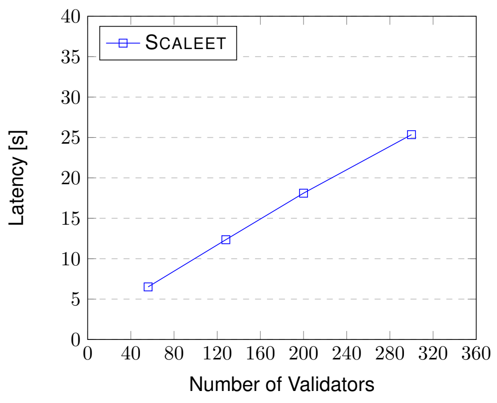

Figure 6.6: SCALEET'S latency when increasing both the number of validators and the size of committees, omitting the increasing of the size of committees.

6.4.2 Percentage of Inter-Committee Transactions

For this experiment, we used the values shown in Table 6.3 for the number of validators, number of committees and committee size. Again, each column represents one experiment. We used a fixed configuration regarding the number of validators, number of committees and committee size – 336, 21, and 16, respectively – and vary the percentage of inter-committee transactions in order to observe their impact on the system.

In this experiment, as depicted in Table 6.3 and in Figure 6.7, the throughput has only a small degradation. The throughput was expected to remain almost constant since the protocol that handles the communication between committees when validating inter-committee transactions is lazy and therefore not on the critical path, i.e., it does not use lock-and-confirm techniques. This small degradation can then be explained by the extra work on computing digital signatures, which need to be performed by the members of the source committees in order to make these transactions verifiable by the target committees.

Regarding latency, we can observe in Table 6.3 and in Figure 6.8 that it increases when the percentage of inter-committee transactions increases. This can be explained by the way that inter-committee

Table 6.3: SCALEET'S throughput and latency when increasing the percentage of inter-committee transactions.

Number of Validators	336	336	336
Number of Committees	21	21	21
Size of Committees	16	16	16
Inter-Committee Transactions [%]	25	50	75
Throughput [tps]	5401	5396	5341
Latency [s]	8.15	12.38	19.14

transactions are confirmed, namely that they are confirmed twice, first in the source committee and then in the target committee. In the former, we check whether the transaction can be performed, and, in the latter, we validate the signatures produced by the source committee on these transactions. Therefore, and roughly speaking, the latency of an inter-committee transaction is expected to be around twice the latency of intra-committee transactions, since inter-committee transactions comprise confirming and appending two blocks, one in the source committee and the other in the target committee.

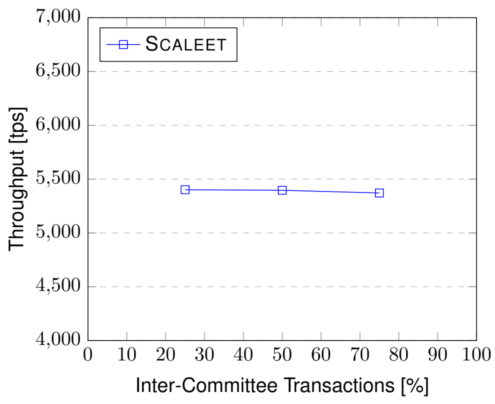

Figure 6.7: SCALEET'S throughput when increasing the percentage of inter-committee transactions, omitting the number of validators, number of committees and size of committees.

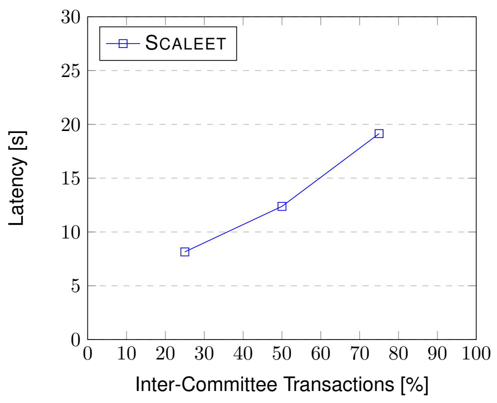

Figure 6.8: SCALEET'S latency when increasing the percentage of inter-committee transactions, omitting the number of validators, number of committees and size of committees.

6.4.3 Impact of Reconfiguration

In this experiment, we measure the cost of moving from an epoch e to an epoch $e+1$. First, recall that this process is not on the critical path, i.e., it occurs concurrently with the epoch e, and therefore the latency and throughput on the transaction committees are not affected. In any case, this experiment measures how long this process takes.

The two factors that can influence the cost of this transition are the number of new members to dis-

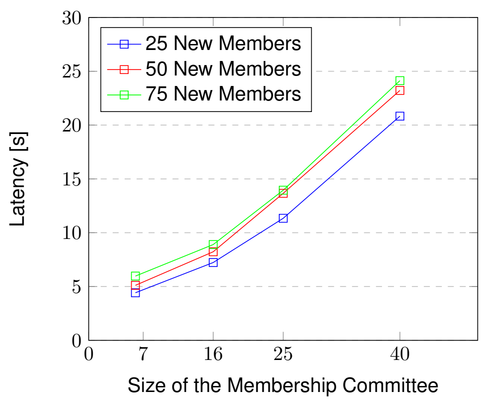

Figure 6.9: SCALEET'S latency when increasing the size of the committee of membership, number of committees and size of committees.

tribute across the committees and the size of the membership committee. The former happens because the higher the number of new members, the higher the number of calculations and signatures performed, and the latter because of the same reasons presented when we discussed the effects of increasing the committee size in Section 6.4.1.B. The results presented in Figure 6.9 support the previous arguments, namely since they show an increase of the latency both when the size of the membership committee increases and the number of new members increases.

6.4.4 Resource Consumption

6.4.4.A CPU Usage

The CPU cost of running the SCALEET prototype is modest, normally within the interval of $30 - 40\%$ per machine when running 16 nodes on it .

6.4.4.B Bandwidth

The size of each block is highly dependent on the number of transactions and the number of addresses created in the network (recall that a snapshot of the balances and nonces for each account is present in each block). Regarding the transactions, we fixed this number to $2,000$ transactions per block. In terms of the number of accounts present in each block, each account has an overhead of 0.00077 MBytes. Considering that our setup has $10,000$ accounts, this implies that a block would have 10.5 MBytes.

With a block size of 10.5 MBytes and considering committees of size 40 (the maximum we tested), each node uses at most 29.11 Mbps. This bandwidth is computed as the total amount of data sent, divided by the duration of the experiment.

6.4.4.C Storage Cost

Regarding storage costs, and similarly to what occurs with the baseline prototype, the largest fraction is taken by the blocks themselves: in the example of the previous paragraph, this is around 10.5 MBytes each. The unconfirmed transactions that a given node is aware of can also represent a significant cost in terms of storage, namely around 0.0014 MBytes per transaction. The remaining costs are split across the neighbor information and other auxiliary structures, and they are negligible when compared with the confirmed and unconfirmed transaction cost.

6.5 Concluding Remarks

Comparing the evaluation results of the baseline prototype and SCALEET, one can conclude that SCALEET outperforms the baseline in terms of both throughput and latency, with differences that, for the former, are on the order of thousands of tps, and for the latter on the order of a few seconds. Regarding resource consumption, SCALEET shows low CPU usage but it is outperformed concerning bandwidth and storage costs.

More specifically, regarding the evaluation of SCALEET, it is important to highlight the following key points:

- When increasing the number of validators, we may be faced with a choice between improving security or improving performance. On the one hand, in order to improve security, one may choose to accompany the increase in the number of validators with an increase of the size of each committee. This way, it can accommodate more failures. On the other hand, one may prefer to improve the overall performance and increase the number of committees instead.
- The percentage of inter-committee transactions barely affects the throughput of the system. In contrast, the latency is highly affected since it requires each inter-committee transaction to be included in two blocks.
- Regarding resource consumption, the biggest drawback identified is the unbounded growth of the block size, which leads to a large impact in both bandwidth and storage costs. This uncapped growth is related to the fact that the block size directly depends on the number of existing accounts in the network. As a countermeasure, it is important to clarify that instead of having a snapshot of the balances and accounts nonces in every transaction block, we could have it only present in the first transaction block of each era. This way, all the other remaining blocks of the era would be capped in size. This way, we would need to perform more calculations at the time of validating each block, since we would need to reconstruct the state from the first era block, which could also impact performance. Another key factor that contributes for the high bandwidth usage that

was observed is the quadratic number of messages exchanged by the committee members when executing PBFT. As already discussed, given the modularity of the design of SCALEET, one can plug any other BFT algorithm to tackle this issue.

7 Conclusion

Contents

7.1 Conclusions

Given the recent success and increasing use of blockchain systems, there are several recent proposals that employ different techniques and designs that seek to scale blockchains as much as possible, which is the main motivation of this work. However, we have not yet seen a systematic treatment of the scalability and robustness of blockchains by separately handling each aspect of their design having these characteristics in mind.

This work, in addition to giving an initial overview of blockchain technology, analyzed the state-of-the-art regarding the emerging directions towards a scalable blockchain, highlighting their respective advantages and drawbacks. Through this analysis we identified a set of techniques for scaling blockchains, which can be categorized into a well-defined set of axes, such as: the communication topology and pattern (e.g., trees and gossip), novel cryptographic primitives (e.g., threshold and collective signatures), trusted hardware environment (e.g., Intel SXG), representative committees and the parallelization of transactions (e.g., sharding).

Following this line, this document presents a novel blockchain design, SCALEET, where some of the previously identified techniques that were scattered among other solutions are revisited, rearranged and packed together into one design. Moreover, SCALEET stands out by presenting high levels of modularity, thus enabling an easy integration of newer and more performant BFT algorithms that can be used as a drop-in replacement. Furthermore, we still enable the use of PoW, while also avoiding its main negative aspects, namely by ensuring the absence of forks in both chains. SCALEET also presents a novel inter-committee transaction protocol that relaxes atomicity in order to boost performance, while offering the same transaction properties as other systems. Finally, SCALEET also introduces a novel and fairer approach for assigning rewards.

Finally, the evaluation shows that SCALEET can scale-out linearly in the number of validators, enabling a throughput in the order of thousands of tps and confirm transactions in the magnitude of seconds.

7.2 Future Work

SCALEET showed promising results; however, it is still an initial prototype that can be improved in many aspects. We highlight as important directions for the future the following points:

- Incorporate parallel transaction validation inside each committee. This would require the nonces of consecutive transactions from the same account to not be incremental but, e.g., random. This way we could counter double spending and enable parallel processing.
- Apply clustering algorithms periodically to approximate “highly” connected accounts into the same committee.

- Apply techniques to guarantee anti-censorship at the time of new proposals in each committee.
- Reason about techniques to punish and exclude misbehaving nodes and incorporate them into the design.

Bibliography

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” *Journal for General Philosophy of Science*, 2008.

[2] C. INSIGHTS, “Banking Is Only The Beginning: 30 Big Industries Blockchain Could Transform,” https://www.cbinsights.com/research/industries-disrupted-blockchain/, 2017.

[3] F. P. Hjalmarsson, G. K. Hreioarsson, M. Hamdaqa, and G. Hjalmtysson, “Blockchain-Based E-Voting System,” in *IEEE International Conference on Cloud Computing, CLOUD*, 2018, pp. 983–986.

[4] “Steem,” https://steem.io.

[5] T. Economist, “Governments may be big backers of the blockchain,” https://www.economist.com/business/2017/06/01/governments-may-be-big-backers-of-the-blockchain, 2017.

[6] J. Yu, D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo, “RepuCoin: Your Reputation Is Your Power,” *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1225–1237, 2019.

[7] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, “Bitcoin-ng: A scalable blockchain protocol,” in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 45–59.

[8] J. R. Douceur, “The sybil attack,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002, pp. 251–260.

[9] J. Kwon, “Tendermint : Consensus without mining,” 2014.

[10] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[11] F. B. Schneider, “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial,” *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.

[12] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *Proceedings - IEEE Symposium on Security and Privacy*, 2018, pp. 19–34.

[13] P. Li, G. Wang, X. Chen, and W. Xu, "Gosig: Scalable byzantine consensus on adversarial wide area network for blockchains," 2018.

[14] M. Hearn, "Corda: A distributed ledger," *Whitepaper*, 2016.

[15] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering*, vol. 3, no. 2, pp. 125–143, 1977.

[16] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[17] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[18] L. Lamport, "The Part-Time Parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[19] B. M. Oki and B. H. Liskov, "Viewstamped replication: A new primary copy method to support highly-available distributed systems," in *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 1988, pp. 8–17.

[20] L. Lamport et al., "In Search of an Understandable Consensus Algorithm," *Atc '14*, vol. 22, no. 2, pp. 305–320, 2014.

[21] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2011, pp. 245–256.

[22] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM (JACM)*, 1980.

[23] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proceedings of the Symposium on Operating System Design and Implementation*, vol. 99, pp. 173–186, 1999.

[24] "Bitcoin Cash," https://www.bitcoincash.org, 2017.

[25] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1993, pp. 139–147.

[26] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 1, pp. 1–32, 2014.

[27] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2016, pp. 17–30.

[28] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," *25th USENIX Security Symposium (USENIX Security 16).*, pp. 279–296, 2016.

[29] A. Miller, A. Kosba, J. Katz, and E. Shi, "Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2015, pp. 680–691.

[30] A. Kiayias, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, vol. 19, pp. 1–27, 2017.

[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *SOSP 2017 - Proceedings of the 26th ACM Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[32] Bitshares, "Delegated Proof-of-Stake Consensus," 2017.

[33] J. Frankenfield, "Proof of Burn," 2018.

[34] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proceedings - IEEE Symposium on Security and Privacy*, 2014, pp. 475–490.

[35] ARM, "ARM Security Technology. Building a Secure System using TrustZone Technology," *ARM white paper*, 2009.

[36] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[37] M. Milutinovic, W. He, H. Wu, and M. Kanwal, "Proof of Luck: An efficient blockchain consensus protocol," in *SysTEX 2016 - 1st Workshop on System Software for Trusted Execution, colocated with ACM/IFIP/USENIX Middleware 2016*, 2016, pp. 1–6.

[38] H. sawtooth, "Hyperledger Sawtooth," *https://Sawtooth.Hyperledger.Org*, 2019.

[39] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems," 01 2020.

[40] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 139–151, 2019.

[41] G. Veronese, M. Correia, A. Bessani, L. Lung, and P. Veríssimo, "Efficient byzantine fault-tolerance," *Computers, IEEE Transactions on*, vol. 62, pp. 16–30, 01 2013.

[42] R. Kapitza, J. Behl, S. V. Mohammadi, C. Cachin, T. Distler, S. Kuhnle, W. Schröder-Preikschat, and K. Stengel, "CheapBFT: Resource-efficient Byzantine fault tolerance," in *EuroSys'12 - Proceedings of the EuroSys 2012 Conference*, 2012, pp. 295–308.

[43] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," 10 2016, pp. 31–42.

[44] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," 2015.

[45] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr multi-signatures with applications to Bitcoin," *Designs, Codes, and Cryptography*, vol. 87, no. 9, pp. 2139–2164, 2019.

[46] E. Kokoris-Kogias, "Robust and scalable consensus for sharded distributed ledgers," Cryptology ePrint Archive, Report 2019/676, 2019.

[47] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology – ASIACRYPT 2018*, T. Peyrin and S. Galbraith, Eds. Cham: Springer International Publishing, 2018, pp. 435–464.

[48] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable Bias-Resistant Distributed Randomness," in *Proceedings - IEEE Symposium on Security and Privacy*, 2017, pp. 444–460.

[49] Bitcoin, "Bitcoin Scalability," https://en.bitcoin.it/wiki/Scalability, 2019.

[50] J. Bonneau, S. Goldfeder, and J. Clark, "On Bitcoin as a public randomness source," *IACR Cryptology ePrint Archive*, 2015.

[51] "Blcok Size Debate," https://en.bitcoin.it/wiki/Block_size_limit_controversy, 2019.

[52] D. Leung, Y. Gilad, S. Gorbunov, L. Reyzin, and N. Zeldovich, "Aardvark: A Concurrent Authenticated Dictionary with Short Proofs," *IACR Cryptology ePrint Archive*, 2020.