SCALEET: A Scalable and Performant Permissionless Blockchain

João Campos
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
joao.costa.campos@tecnico.ulisboa.pt

***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.**

***Index Terms*—Blockchain, Consensus, Byzantine Fault Tolerance, Scalability, Permissionless Setting, Committees, Sharding**

I. INTRODUCTION

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 beyond cryptocurrencies, expanding into different areas such as voting [2], social media [3] and government records [4].

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 [5] and Btcoin-NG [6], 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 [7] 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)* [8], 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 [9], which are the basis for BFT state machine replication [10], 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 over the system [7].

The goal of this paper 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.

To achieve this, we propose a novel blockchain system 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. The experimental evaluation of the prototype of SCALEET shows 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.

II. Related Work

Several practical solutions have been proposed over the years to address the security and scalability issues of blockchains. We structure 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 and techniques chosen by existing systems.

A. 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 are *Sybil Attacks* [7], 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.

1) Proof-of-Work: Proof-of-Work (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 its system, followed later by several other designs [6], [11]–[13], which makes PoW-based membership selection the most widely used and known method. In Bitcoin and its variants, the process of solving a puzzle is called *mining* and it is done by special 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 cryptographic hashes make it impossible to speed up the process. An extension to standard PoW is the one used by J.Yu et al. in RepuCoin, named *Proof-of-Reputation* [5]. 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.

2) Proof-of-Stake: Proof-of-Stake replaces the mining power selection with the 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, namely deposit based [8] where 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 and balance based schemes [14] which are very similar to deposit based solutions, but the stake is calculated based on the node's current account balance.

3) Proof-of-Capacity: In *Proof-of-Capacity*, the node's 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.

4) Trusted Execution Environment Based: A *Trusted Execution Environment* [15] 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. *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.

B. Byzantine Consensus Protocols

The *consensus protocol* 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). Existing solutions can be divided into two major groups: the ones following *Nakamoto's* consensus and the ones following classic *BFT* consensus.

1) Nakamoto's consensus: Introduced in Bitcoin [1], *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 the set of 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.

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, Nakamoto's consensus achieves a planetary scale, but with a transaction throughput that is currently processing up to 7 tps. In contrast, centralized payment systems, like Visa and MasterCard are reporting $1,200$ to $56,000$ tps [16], which is three to four orders of magnitude higher than the one obtained by Bitcoin.

2) 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. 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.

BFT solutions can achieve higher throughput and lower latency than schemes based on Nakamoto's consensus, 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 II-A. Regarding scalability, when the number of participants increases, the proposals rely on optimizing the standard and well-known *Practical Byzantine Fault Tolerance (PBFT)* [17] 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, recent designs use mainly three techniques: - Rearrange the communication topology in a balanced tree [13], [18], 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 [19], 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 [20]. 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 very important in enabling the efficient functioning of different patterns of communication, namely *Collective Signatures* [13], *Threshold Encryption* [21] and *Verifiable random functions* [20].
- **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 [18].
- **Parallelization of transactions**: If instead of one representative committee, several were selected, it would be possible to employ sharding. 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 [18].
- **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 [22]–[24], by providing an efficient way to conduct several aspects of the protocols by leveraging hardware support from the *TEE*.

III. Design

This Section presents the design of our system, laying out the main goals, the system model and the building blocks that make up SCALEET.

A. 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*.

B. Overall assumptions and model

Next, we present the system model. We divide the set of assumptions underlying this model according to the different components of the system.

1) Overall Distributed 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 [13], [18] and relatively easy to lift.

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.

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 Byzantine 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. In that regard, we assume a *mildly era adaptive* adversary [18], which can choose which nodes to corrupt at the start of each 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) 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*, which guarantee the properties of *reliable delivery*, *no duplication* and *no creation* [25]. 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.

C. Membership Management

Our design is based on a separation between membership management and transaction processing, which allows us to use PoW (or any other Sybil-resistant mechanism) on the membership management part, thus removing this expensive component from the performance-sensitive path of transaction processing. We start our explanation of the system design by presenting how the system manages its membership. Once we define a set of abstractions offered by the membership component, it will become easier to design a set of transaction processing protocols that build upon those abstractions.

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. 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.

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.

1) Membership commitment proof submission: The membership commitment proof is the mechanism employed by SCALEET as a Sybil resistance technique, i.e., it uses commitment proofs to narrow down the probabilities of such attacks. The commitment proof mechanism leverages PoW for requiring work from a node 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. When a node wants to submit a new proof, it fetches the hash of the latest block from the membership chain and uses it as seed for PoW. When the solution is found, the node 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 that block is created and appended.

2) 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. For each received proof, a BFT run among the committee members is triggered to check if this proof is valid, i.e., if it took into account the hash of the latest membership block and if it presents the proper difficulty. There are two possible paths: 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. Regarding the latter option, this new block only updates the score of the member that has submitted the proof. Note that, the members with the highest scores at the time of the reconfiguration block will be designated as validators.

3) Membership reconfiguration: In the case where the new block is a reconfiguration block, it 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. In this block the scores are reset and the new constitution of committees is presented. 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.

To provide those properties, our solution leverages the randomness of the membership commitment proof itself. More specifically, we use the hash of the latest membership block in the chain as seed for the reconfiguration, which provides the properties that we require, in particular:

- **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

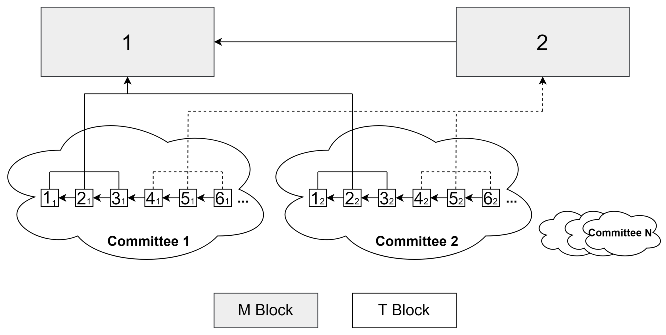

Fig. 1: *Membership blocks* and *transaction blocks* with committees.

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 and all the remaining system nodes.

When a new validator is selected to start at the next epoch, it needs to find a committee that it will join and an existing validator from that committee that it will replace. For picking these, we also leverage the randomness of the membership commitment proof by using it as seed. In order to find the new committee for that member the following operation is made: $H(seed \| PK) \mod c$, where c is the number of committees and PK the public key of the new validator. 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. The process to find the member to leave is identical, 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.

After this reconfiguration 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 node updates its state accordingly.

D. Transaction Validation

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 portion of state. In SCALEET, the state is partitioned by accounts, i.e., each transaction committee manages a set of accounts (addresses). Given this, a special case arises when a transaction is issued from an address another address that is out of scope of the committee that handles the source address. In such cases, the transaction is called an *inter-committee* transaction (and *intra-committee* otherwise). 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. This allows for a simplified protocol to ensure the atomicity of the operation across committees, as detailed subsequently. 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. Lastly, we can move the responsibility of inter-committee transaction processing 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 1 represents both chains. 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. We next detail the protocol for each type of transaction.

1) Intra-committee transactions: 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 responsible committee for each address can be found 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. Recall that the committee identification 0 is assigned to the membership committee. 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.

2) Inter-committee transactions: In the case of inter-committee transactions, the previous protocol does not suffice, since, after appending the block, the SCALEETS (SCALEET cryptocurrency name) 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 introduce an inter-committee protocol, which solves the aforementioned issue in a simple and efficient way. The algorithm can be divided into two stages: the first in the coordinator committee (responsible for the source address), and the second in the target committee (responsible for the destination address).

In the coordinator committee, upon each transaction block confirmation, the committee will collectively assemble a proof in the form of a special transaction for each inter-committee transaction present in the set of transactions of that block.This special transaction will be signed by the members of the source committee, 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. Finally, each of the new special transactions is relayed through the network until it reaches the corresponding target committee. These

special transactions will be eventually received by the target committees and will be handled as normal transactions. The only difference is the confirmations needed to be performed – for a transaction to be valid it just has to have a set of valid signatures, which assert that it was validated correctly at the coordinator committee.
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. Semantically, this behavior is acceptable since, even though the concept of atomicity does not apply, the system will eventually converge to a correct state.

E. 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. 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. Issuing rewards becomes an interesting problem to analyze when each block is validated by multiple validators, which implies the reward to be splitted among a set of members, as is the case in SCALEET. Along this line, SCALEET presents a novel reward assignment mechanism that seeks to only reward the committee members that actually participated in the protocol, and thus avoid rewarding replicas of the BFT consensus protocol that did not contribute to the success of the operation.

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 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.

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. 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. 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.

Finally, we note that the process described in this Section can also be applied to assign transaction fees.

F. BFT Protocol

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 (PBFT)* [17]. 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 [19]. We chose PBFT because it is a well-studied representative of the family of BFT consensus protocol. 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.

IV. Implementation

We implemented a SCALEET prototype in Java 8, consisting of approximately $7,500$ lines of code and, as detailed in the next section, 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 [11], 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. All of this code is publicly available on Github [2].

V. Evaluation

This Section evaluates our prototype. The main high-level goal of our evaluation is to measure the performance and scalability of SCALEET. 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 addition, we will also evaluate some of the system overheads, namely in terms of its 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.

A. Experimental Setup

To 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). 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. Each transaction block contains $2,000$ transactions and in all the remaining experiments, except in Section V-C the percentage of inter-committee transitions is fixed to 50%.

[1]https://spring.io

[2]https://github.com/jpcc1997/blockchain

B. 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.

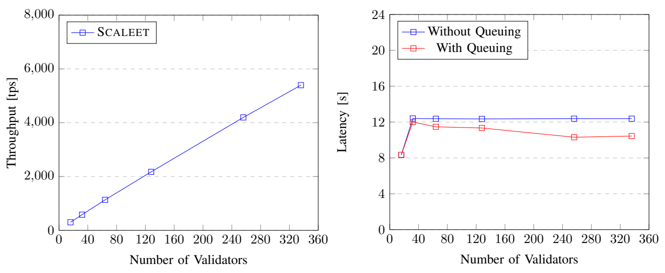

Fig. 2: Throughput and latency when increasing the number of committees.

1) Number of Committees: As we can observe in Figure 2, 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 also observe in Figure 2, 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.

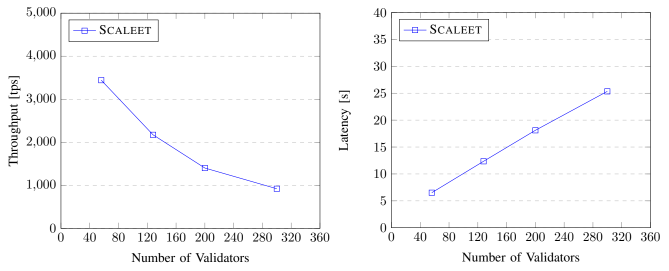

Fig. 3: Throughput and latency when increasing the committees' size.

2) Size of Committees: In this experiment, 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 Figure 3, 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. In each PBFT 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 also in Figure 3 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.

C. Percentage of Inter-Committee Transactions

In this experiment, we vary the percentage of transaction that span more than one committee. As depicted in Figure 4, the results show that 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.

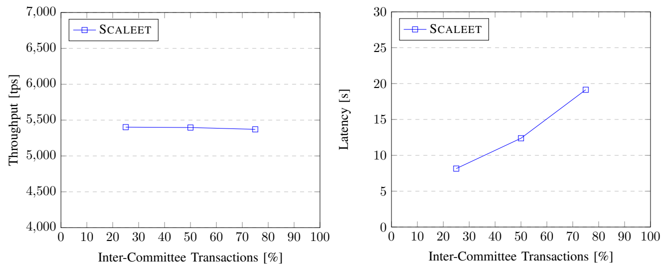

Fig. 4: Throughput and latency when increasing the percentage of inter-committee transactions.

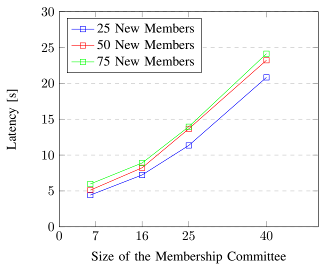

Fig. 5: Latency when increasing the size of the committee of membership, number of committees and size of committees.

Regarding latency, we can also observe in Figure 4 that it increases when the percentage of inter-committee transactions increases. This can be explained by the way that inter-committee transactions are confirmed, namely that they are confirmed twice, first in the source committee and then in the target committee.

D. 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 distribute 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 V-B2. The results presented in Figure 5 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.

E. Resource Consumption

1) 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.

2) 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.

3) Storage Cost: Regarding storage costs, 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.

VI. 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.

VII. Conclusions

This work presents a novel blockchain design called SCALEET. 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.

References

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

[2] Friorik P. Hjalmarsson, Gunnlaugur K. Hreioarsson, Mohammad Hamdaqa, and Gisli Hjalmtysson. Blockchain-Based E-Voting System. In *IEEE International Conference on Cloud Computing, CLOUD*, pages 983–986, 2018.

[3] Steem. https://steem.io.

[4] The 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.

[5] Jiangshan Yu, David Kozhaya, Jeremie Decouchant, and Paulo Esteves-Verissimo. RepuCoin: Your Reputation Is Your Power. *IEEE Transactions on Computers*, 68(8):1225–1237, 2019.

[6] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

[7] John R. Douceur. The sybil attack. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 251–260, 2002.

[8] Jae Kwon. Tendermint : Consensus without mining. 2014.

[9] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[10] Fred B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[11] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151(1):1–32, 2014.

[12] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 17–30, 2016.

[13] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. *25th USENIX Security Symposium (USENIX Security 16).*, pages 279–296, 2016.

[14] Aggelos 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*, 19:1–27, 2017.

[15] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

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

[17] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. *Proceedings of the Symposium on Operating System Design and Implementation*, 99:173–186, 1999.

[18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 19–34, 2018.

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

[20] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP 2017 - Proceedings of the 26th ACM Symposium on Operating Systems Principles*, pages 51–68, 2017.

[21] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. pages 31–42, 10 2016.

[22] Hyperledger sawtooth. Hyperledger Sawtooth. *https://Sawtooth.Hyperledger.Org*, 2019.

[23] Giuliana Veronese, Miguel Correia, Alysson Bessani, Lau Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *Computers, IEEE Transactions on*, 62:16–30, 01 2013.

[24] Rüdiger Kapitza, Johannes Behl, Seyed Vahid Mohammadi, Christian Cachin, Tobias Distler, Simon Kuhnle, Wolfgang Schröder-Preikschat, and Klaus Stengel. CheapBFT: Resource-efficient Byzantine fault tolerance. In *EuroSys'12 - Proceedings of the EuroSys 2012 Conference*, pages 295–308, 2012.

[25] Rachid Guerraoui and André Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27:29–41, 2001.