SCP: A Computationally-Scalable Byzantine Consensus Protocol For Blockchains

Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, Prateek Saxena
School of Computing, National University of Singapore
{loiluu, visweshn, bawejaku, chaodong.zheng, seth.gilbert, prateeks}@comp.nus.edu.sg

***Abstract*—In this paper, we design a new blockchain Byzantine consensus protocol SCP where the throughput scales nearly linearly with the computation: the more computing power available, the more blocks selected per unit time. SCP is also efficient that the number of messages it requires is nearly linear in the network size. The *computational scalability* property offers the flexibility to tune bandwidth consumption by adjusting computational parameters (*e.g.*, proof-of-work difficulty). The key ideas lie in securely establishing identities for network participants, randomly placing them in several committees and running a classical consensus protocol within each committee to propose blocks in *parallel*. We further design a mechanism to allow reaching consensus on blocks without broadcasting actual block data, while still enabling efficient block verification. We prove that our protocol is secure, efficient and applicable to several case studies. We conduct scalability experiments on Amazon EC2 with upto 80 cores, and confirm that SCP matches its theoretical scaling properties.**

I. Introduction

Byzantine agreement is a classical problem, with a wide range of applications in building robust distributed systems. Imagine there are n processors communicating over a distributed network. Up to f of the processors may be malicious or Byzantine [1]. Each processor has a private input bit (0 or 1), and the goal is to ensure that all honest processors agree on the same value, ideally a value that is the input bit for at least one honest processor. Malicious processors can act in arbitrary ways to prevent the honest processors from reaching a valid agreement. Byzantine agreement is a building block for a wide variety of applications, extending to agreement on the state of a larger data structure (via replicated state machine techniques), with applications ranging from self-organizing network overlays and fault-tolerant DHTs to payment ledgers such as those in cryptocurrencies.

Traditional Byzantine agreement protocols are *bandwidth-limited*: in real systems, the limiting factor on performance is bandwidth. Early algorithms had exponential communication complexity [2], [3], with eventual improvements yielding polynomial communication complexity (e.g., [4]). A big step forward was the first claimed practical Byzantine fault-tolerant agreement protocol (PBFT) [1]. Practical byzantine consensus protocols such as PBFT, however, require quadratic number of messages in the number of participants. Thus, the major bottleneck in scaling classical byzantine consensus algorithms is network capacity, since protocols become bandwidth-bound with less a small number of processors.[1]

Recently, we have seen the advent of an entirely different approach that is *computationally-limited* rather than bandwidth-limited: Nakamoto consensus. This "consensus" protocol was introduced in Bitcoin [9], and solves a relaxation of the traditional Byzantine consensus problem (discussed in Section 2), which is sufficiently powerful to be quite useful for many practical applications. A key advantage of Nakamoto consensus is that it uses bandwidth much more efficiently, achieving $O(n)$ communication complexity, with low communication overhead. (This contrasts to the best known classical Byzantine agreement protocols with $O(n \cdot \texttt{polylog}(n))$ communication [5]–[8].) Nakamato consensus trades off computation for bandwidth.

At a high level, the Nakamoto consensus works by randomly selecting one processor per epoch (say 10 minutes) which issues a proposal that everyone adopts—thus requiring only a single broadcast to reach agreement.[2] It uses a proof-of-work mechanism to elect the leader, ensuring a fair choice of leaders. Since its proposal in 2009, Bitcoin and over 250 other cryptocurrencies use Nakamoto consensus as their primary mechanism for ensuring secure distributed payment ledgers (called blockchains). In terms of scale, Bitcoin employs a billion CPUs worth of computational power today (by observable hashrates), and attracted more than one billion dollars worth of investments in 2015 alone—one of the few well-fielded distributed systems of such scale.

Unfortunately, Bitcoin's throughput is not scaling well. The Bitcoin network consumes massive computation power and presently processes up to 7 transactions (TXs) per second. Other centralized fiat payment systems like Paypal or Visa process up to 2000 TXs per second, so the demand from practical applications is three orders of magnitude more [10]. Scalability of the protocol is a raging debate in the Bitcoin community [11]–[14]. While a number of recent proposals and variants suggest changes to specific protocol parameters [11]–[13], [15] (such as inter-block time, size of transaction block size, and so on), these have fundamental scalability limits [16].

[1]Another series of exciting breakthroughs in the communication complexity of Byzantine agreement [5]–[8] has reduced the communication complexity to polylogarithmic bits per node, in theory. (See Section VIII.) The resulting protocols, while elegant, are not simple and appear difficult to adapt to practical settings.

[2]In fact, there may be temporary disagreement if two proposals occur at the same time; eventually, with very high probability, one proposal will be established.

With classical consensus protocols, since we are bandwidth-limited, more participants leads to *worse* performance. Since Nakamoto consensus is computationally limited, one might hope that more participants yield more computation and hence *better* performance. Unfortunately, that is not the case: Nakamoto consensus, by design, does not utilize its computation capacity for scaling its throughput—the protocol ensures that one transaction block generated per time epoch (10 minutes on Bitcoin) on average, irrespective of the total computational power.

This raises a fundamental question — are there (relaxed) Byzantine agreement protocols that scale in throughput linearly with the increase in computational capacity of the network? If such secure *computational scaling* is achievable, some important advantages are immediate. First, doubling the computation power of the distributed network, doubles it throughput (number of agreed values per unit time) without any parameter adjustments. Second, by limiting how much computation power constitutes one "processor", the protocol can tune its bandwidth consumption with flexibility. For instance, the protocol can reduce or increase its bandwidth requirements for consensus, simply by varying its proof-of-work parameters.

Problem & Approach. We design a scalable byzantine consensus protocol suitable for use in blockchain applications, which achieves a sweet spot between classical byzantine consensus and Nakamoto consensus protocols. Specifically, our protocol SCP is designed to have nearly linear (*i.e.*, $O(n/log(n)$ or sublinear) scalability with computation capacity and does not require quadratic number of messages as the network grows. Our protocol tolerates up to $f < n/3$ adaptive byzantine adversaries, where f and n are bounds on the adversarial and total computational power respectively.[3] The protocol can support the same blockchain data structure format (a hash-chain) as Bitcoin; but, for further scalability, we propose a modification that permits better efficiency parameters. Our protocol makes the same assumptions as existing cryptocurrencies about the underlying network layer — that is, the peer-to-peer network enables synchronous broadcasts with low and bounded delay, and the network provably maintains connectivity between honest peers.

The key idea in our approach is to partition (or parallelize) the network into sub-committees, where the number of committees is linear in the total computational power of the network. Each sub-committee runs a classical byzantine consensus protocol to process a separate set of transactions. To do this partition, SCP leverages proof-of-work used in Bitcoin to i) limit sybil identities in the computational network; ii) securely split the computation nodes in the network into sub-committees. Each committee agrees on one block, consisting of a set of valid transactions. A final committee is designated to combine the outputs of sub-committees into an ordered blockchain data structure. The protocols are such that they permit each network participant to identify the committee it belongs to, and which transactions are meant to be processed by that committee independently. The protocol runs once in a time epoch (say of 10 minutes), and protocol outputs are cryptographically committed in a blockchain stored fully replicated, the same way as in Bitcoin.

[3]Here, $1/3$ is an arbitrary constant bounded away from $1/2$, selected as such to yield reasonable constant parameters.

Our goal is to ensure that the protocol offers the same level of security as Nakamoto consensus under the same assumptions, but with a higher throughput. To achieve security equivalent to Nakamoto consensus, several additional considerations emerge in our protocol. First, the protocol must ensure that each sub-committee consists of an honest majority with overwhelming probability. Without this property, the committees may not reach agreement on a consistent output or agree on invalid outputs. We solve this key challenge by developing a protocol to distributively generate a public random coin — a source of randomness that has a weak, bounded bias from the computationally-limited adversary. Our further constructions utilize this weakly-biased random coin and carefully selected size parameters to bound the bias (or adversarial influence) in committee selection and transaction allocation *w.h.p.* Similar to Bitcoin, our random coin generation algorithm makes no assumptions on an external randomness beacon. A second challenge is to ensure that the protocol can be serially composed, ensuring defense against an adaptive adversary which can observe all outputs from previous protocol runs. To ensure this, our protocol randomizes the construction primitives in each run, thereby limiting the adversary's advantage to a negligible quantity. We provide proofs of security of our constructions.

We explain how to build two blockchain applications using our new protocol in Section VI, including a cryptocurrency. As an additional advantage, SCP decouples the consensus step from that of broadcasting all the transaction data immediately. This yields a better bandwidth utilization, since data transmissions for the consensus step are needed between constant size committees only and the size of data is a small header block. In Bitcoin and several other proposals, heavy transaction data of previous blocks needs to be broadcast before the start of epoch. SCP cleanly decouples the broadcast of agreed value (e.g. transactions) from that of the consensus protocol information, enabling lazy broadcasts of the former.

Efficiency & Results. From an efficiency perspective, our protocol parallelizes the network into a sublinear number of committees thus scaling with computation capacity. Within each committee of size c (approximately 400) identities, we design and run a secure consensus protocol of $O(c^3)$ message complexity. Overall, if there are n identities participating in an epoch, this yields a message complexity of $O(nc + c^3)$. A classical BFT protocol without SCP's parallelization would run in $O(n^2)$ or $O(n^3)$. In Bitcoin, n is over 6000 nodes, thus the improvement is about two orders of magnitude in principle.

We run our experiments on Amazon EC2 using the same network implementation as Bitcoin [17] and simulating several experiment scenarios. Our experiments show that Bitcoin does not scale up even if we plug more computational power, as expected. With increase in block size or the block rate (*i.e.*, reducing epoch time), the transaction rate scales up *almost* linearly. However we soon hit other constraints such as bandwidth limitation and block verification time. For instance, when the block rate is 12 seconds, nodes have different views of the blockchain frequently because the underlying network is not efficient enough to broadcast new blocks. As a result, nodes spend much time resolving the conflicts and the transaction rate drops significantly.

Our prototype of SCP scales its throughput sublinearly, when run on our network simulation. With the same network topology, the block rates (blocks per epoch) for X, 2X, 4X, 8X (X is 10 i7-3520M processors) are 1, 2.02, 2.37, 3.17 respectively. Finally, due to SCP's ability to decouple the consensus from block-data broadcasts, the size of data exchanges in the consensus step is drastically smaller, leading to smaller block-propagation delays and overall fairness.

Contributions. This paper makes following contributions.

- We propose a scalable consensus protocol SCP which scales its throughput sublinearly with the computation power of the network. We prove that SCP is secure and efficient.
- We build a blockchain based on SCP, and show two applications: a cryptocurrency and a credential ledger.
- Our experiments on an idealized network simulation on Amazon EC2 cloud instances, ranging up to 160 CPU cores, confirm a sublinear scale up for SCP. In comparison, Bitcoin's present implementation does not show a throughput improvement with increase in computation power.

II. Problem & Challenges

A. *Byzantine Consensus For BlockChains*

The blockchain protocol is a fundamental contribution introduced in Bitcoin, and the underlying basis of over 250 cryptocurrencies. A blockchain is a data structure which stores a time-ordered set of facts (or transactions [4]). The blockchain data structure at any given time instance is identified by a cryptographic digest which provides authenticity. Anyone can verify whether a transaction is in the ordered set and its ordering w.r.t to other committed transactions, given the cryptographic digest. The cryptographic digest is implemented by organizing committed transactions into subsets known as *blocks*. Each block uses a Merkle tree to maintain authenticity of the subset of transactions in it, and Merkle roots of blocks form a cryptographic hash-chain [18].

Let us consider the key problem in maintaining the blockchain data structure in a decentralized network, which we call as the *blockchain problem*. The blockchain problem is to allow an arbitrary large network of processors to agree on the blockchain state (identified by its cryptographic digest), under the assumption that the computation power of malicious processors is bounded. Equivalently, we can consider the network to consist of n independent processors of equal computation power, out of which f processors are byzantine or malicious and can deviate arbitrarily from the protocol. Processors have no inherent identities, nor is there any trusted PKI infrastructure to establish identities for processors. Each processor can choose a set of transactions it wishes to commit to the blockchain; the goal of the protocol is to ensure that the honest processors agree on one committed set at the end of the protocol. The commit set is the set of new blocks to append to the blockchain.

[4]Facts are typically referred to as *transactions*, since they were originally introduced to assert ownership and transfer of Bitcoins between pseudonymous entities. We use the two interchangeably.

In this paper, we are interested in solving the blockchain problem with byzantine adversaries accounting for up to $n/3$ of the processors. A solution to the blockchain problem can be cast into a solution to original byzantine consensus problem under the right assumptions about the network — such a modification to Bitcoin's solution was shown to yield a secure byzantine consensus solution recently [19].

B. *Existing Solutions Do Not Scale (Securely)*

The first challenge is that processors have no inherent identities or external PKI to trust. Each processor can thus simulate many virtual processors, thereby creating a large set of *sybils* in the consensus game. Many classical byzantine consensus solutions assume a limited number of sybil identities controlled by the byzantine adversary; therefore, we cannot directly use such protocols. Bitcoin proposes a solution to the sybil sub-problem by using a proof-of-work. Conceptually, the main idea is to design a cryptographic puzzle which admits no more than k solutions (w.h.p) in a unit of time for each processor. When a processor exhibits a successful solution, it can attach a virtual identity (identified by say a public key). Given that the adversary controls at most f out of the n processors worth of computation power, it can establish $k \cdot f$ virtual identities per unit time. The protocol can thus prescribe a proof-of-work puzzle which admits controllable k to limit sybil identities.

Let us start with such a defense which creates virtual identities based on computation power bounds. The next challenge is to agree in the network of such computational identities (say n'), at most k' $(=k \cdot f)$ of which are byzantine. We discuss why previous practical solutions to byzantine consensus do not yield a scalable solution. Largely, this is because increasing the number of computational identities either makes them insecure or increase their communication costs prohibitively.

Practical byzantine consensus protocols. Since the first paper in 1982 [3], researchers have introduced several models for Byzantine consensus problem (see a survey [20]). For instance, in [21], Fisher, Lynch and Patterson show a famous impossibility result which says that achieving a deterministic consensus in a asynchronous model is impossible even if one party is dishonest. Therefore, many solutions focus on probabilistic algorithms, tolerating a non-zero probability of incorrectness. For a long period of time, most practical byzantine consensus protocol require a quadratic number of messages in the number of network identities [1]. Since the number of messages is quadratic in the number of interacting identities, standard byzantine consensus protocols are only effective for use among a small number of indentions, usually a thousand at most. Such protocols are known to become bandwidth-bound and latency-bound quickly after a few tens of identities are involved. As a result, the protocols cannot create large number of computational identities, since the underlying consensus protocol hit network bandwidth bottlenecks quickly. Protocols with sub-linear and polylog communication costs are an active area of research [8], they are much more complicated and have large constants — their practical and efficient implementations are yet to be demonstrated.

Nakamoto Consensus in Bitcoin. Bitcoin proposes a new protocol called Nakamoto consensus, which drastically reduces communication costs w.r.t to classical solution. In Nakamoto consensus, the protocol adjusts the proof-of-work to ensure that (on average) only one out of the n' identities is able to solve a solution. Thus, in each protocol run, one identity (called the block founder) is elected who broadcasts the proposed block to the network. Clearly, since the broadcast is limited to one block from one founder, the bandwidth utilized is not proportional to the number of computational identities. However, Nakamoto consensus probabilistically elects one founder, independent of the available processors on the network — this design does not scale with increase in the number of processors added to the network. Note that increasing the number of founders elected in each round of Nakamoto consensus makes it insecure — if two blocks are proposed, the network see a "fork" or two possible agreement values. Nakamoto consensus rules eventually resolve the ambiguity in subsequent runs when one of the possible hash-chain becomes longest. A longest blockchain is only guaranteed when a protocol run yields a single block, otherwise the network may always have more than one possibility for agreement.

In summary, existing solutions and their direct application do not yield a scalable and secure blockchain protocol. In this paper, we ask whether a secure protocol exists that scales the number of agreed transactions linearly in n.

C. Problem Definition: Subset Consensus

We propose the *subset-consensus problem*, which captures the essence of agreement over a subset of transactions. Let there be n identity-less processors, f of which are controlled by a byzantine adversary. Each processor i has its integer input $x_i \in \mathbb{Z}_N$, and an output $y_i \in 2^{\mathbb{Z}_N}$ and a externally-specified constraint function $\mathcal{C}\colon\ 2^{\mathbb{Z}_N} \mapsto 0, 1$. We seek a protocol Π run between the processors repeatedly, yielding protocol runs $\Pi_1, \Pi_2, \ldots, \Pi_r$ such that for all choices of x_i in each P_i, if $f < n/3$ then the following conditions hold:

- *Agreement*. There exists an $X \in 2^{\mathbb{Z}_N}$, such that the outputs of all honest processors is X with overwhelming probability $p > (1 - 2^r)$.
- *Validity*. The value X satisfied an externally specified constraints, i.e., $C(X)$ is 1.

Our goal is to scale the size of the set of agreed transactions X per invocation of Π. More specifically, the size of the set should increase sublinearly in n. If so, we say the solution has sublinear scalability.

We point out that the subset consensus problem is a relaxation of the original byzantine consensus problem. The first significant distinction is the definition of "agreement". Here, we allow the honest processors to be in "probabilistic agreement" such that processors agree on a value with some probability greater than p, rather than be in exact agreement. It is desirable to have cryptographically strong level of agreement, so in this paper we consider agreement with overwhelmingly high probability. The second distinction is that the agreed value can be the input of any processor, honest or byzantine. The classical definition requires that the agreed value also be "valid", that is the input of at least one honest processor. In the blockchain problem, validity can be checked externally — each honest processor can check the agreed value to satisfy externally-specified constraints C, and accept a solution only if so. We discuss why such a formulation is useful in practical blockchain applications.

Note that subset consensus protocol captures the essence of scalability, and cleanly separates other concerns in building a full-blockchain. Once the network agrees on the subset of transactions (denoted by x_i) included in the agreed set, it can create a cryptographic digest using a hash-chain as explained. Our problem formulation cleanly separates the consensus problem from that of sending actual transactions, which may be hundreds of bytes rather than integers. However, as we show in Section III, we can agree on a hash-value of transactions using the subset-consensus protocol, and lazily broadcast the transactions themselves separately. This decouples the consensus protocol from the transmission of transactions, allowing the two functions to not have a serial dependency. Our problem formulation does not mandate a specific definition of validity beyond "agreed by all honest parties", as captured by the function $\mathcal{C}$. This has immediate advantages, because we decouple the consensus protocol from checking validity of transactions. Different applications of a blockchain demand different kinds of validation checks. For instance, a blockchain may function simply as an decentralized immutable ledger which requires no checks on the agreed set. A cryptocurrency application, such as Bitcoin, may wish to enforce constraints on eliminating double-spending transactions. We show in our evaluation that we can implement a distributed implementation of C to preserve such double-spending defenses; however, this is specific to the Bitcoin application and need not be baked into the consensus protocol.

Threat Model. We consider a threat model that includes *arbitrary and round-adaptive adversary*. Processors controlled by the byzantine adversary can be arbitrarily malicious, *e.g.*, deviate from the protocol, and/or drop messages from other processors. All malicious processors can collude together. Further, we consider a round-adaptive adversary, which can select which processors to corrupt at the start of each round (or run) Π_i. The adversary has complete access to the outputs of all previous $i - 1$ runs to make its choices. However, once a protocol run begins, the choices of compromised processors are fixed. The processors are assumed to point-to-point communication links between them, and the adversary has full information about the messages transmitted on all links.

Security Assumptions. First, we assume that we know the upper bounds on the true computation power n (say in Giga-hash/sec), and that f is less than $n/3$. Estimating such a bound is feasible from observing network hashrates, as in Bitcoin, with the caveat that adversaries can pretend to control f much lower than they actually do. For this work, we assume such information is externally available. Second, we assume that the total computation power of the byzantine adversaries is still confined to standard cryptographic assumptions of probabilistic polynomial-time adversaries. We assume availability of certain standard cryptographic primitives, described later in our constructions. Third, we make the same assumptions about the underlying network overlay layer as Bitcoin. Explicitly, the network graph between honest processors is connected and the communication channel is synchronous, *i.e.*, once any user broadcasts any message, the rest of the honest nodes will receive it within a negligible and bounded delay (say 5 seconds); we refer to this as a "consistent" broadcast.[5] Note that such timing and connectivity assumptions are implicit in Bitcoin, and attacks on our protocol affecting these assumptions apply to Bitcoin too. In this work, we make assume an "idealized" network layer that is reliable and low-latency broadcast channel. We recognize that these are open challenges for many distributed systems beyond blockchains, especially which deal with dynamic churn [22], [23].

III. SCP DESIGN

In this section, we present the SCP protocol. We will begin with an overview of the protocol, and then proceed to describe each part of the protocol in more detail.

A. Solution overview

The algorithm proceeds in *epochs*, each of which decides on a set of values (published as a cryptographically verifiable digest). In this description, we describe the steps taken during one epoch.

The key idea is to parallelize the available computation power, dividing it into sub-committees. The number of committees grows proportionally to the total computation power in the network. A committee has a small number of members (e.g., $c = 400$), and each committee runs a consensus protocol to internally agree on one value. A final committee called the consensus committee is responsible for combining the values selected by the committees, computing a cryptographic digest and broadcasting it to the whole network.

As the last step in the epoch, the final committee generates a set of shared public random bit strings, all of which have bounded bias. These random strings are used in subsequent epochs as a source of randomness.

In each epoch, processors execute the following 5 steps:

- *Committee Formation.* This step is a local computation at each processor, which reveals to the processor its virtual identity and the identity of the committee that it participates in for that epoch.
- *Committee Overlay Join.* In this step, processors communicate to discover identities of other processors in their committee.
- *Intra-committee Consensus.* In this step, processors run an authenticated Byzantine agreement protocol within their committee to agree on a single value. Each committee sends the selected value, signed, to a designated final committee.
- *Final Consensus Broadcast.* The final committee computes a final value from all the values received from committees, and broadcast the final result.
- *Shared Randomness Generation.* The final committee runs a distributed commit-and-xor scheme to generate a sufficiently unbiased random value. The random value is broadcast to the network.

In Step 1, each processor generates an identity consisting of a public key, an IP address and a PoW solution. The public key and the IP address are selected by the processor; the processor must solve a a proof-of-work (PoW) to generate the final component of the identity. Because PoW requires computation, the number of identities that the malicious processors can create is limited, as they control at most $1/3$ of the total computational power.

In Step 2, processors join their committees, creating an overlay. The overlay is a fully-connected subgraph containing all the committee members (i.e., it is small). A naıve solution is for every processor to broadcast its identity and committee membership to everyone; however, this solution will result in $O(n^2)$ messages, which is not scalable. We provide a simple solution that requires a small number of broadcasts, after which group multicasts within a committee complete the overlay join step.

In Step 3, each committee executes an intra-committee consensus protocol to agree on a value. We ensure that each committee has a majority of honest members, and any (authenticated) Byzantine agreement protocol that requires only a correct majority can be used.

In Step 4, the results from the committees are combined into a final value by taking a set union to generate the final agreed subset. This step is performed by a special designated *final committee*.

In Step 5, the final committee generates a random value which has at most $c/2$ biased bits. This random value is used in the PoW in the next epoch.

Throughout this paper, we use the following notation: n is the total number of identities that we expect to be generated in an epoch, f is the fraction of total computational power that is controlled by malicious users (i.e., the adversary), the size of each committee is c, the number of committees is 2^s, for some constant s. Without loss of generality, s can be picked such that $(1 - f) \cdot n = c \cdot 2^s$.

Efficiency. The protocol agrees on a set of values equal to the number of committees, which scales sublinearly in n. Our protocol requires $O(c)$ broadcasts to the whole network (step 2, 4 and 5). Each such broadcast can be implemented in $O(n)$ message transmissions and has low latency. Step 3, 4 and 5 require at most c^2 multicasts with committees of size c, thus the message transmissions is $O(c^3)$. Therefore, the total number of message transmissions is $O(nc + c^3)$ or roughly $O(n)$ if we consider c to be a small constant (e.g., 400).

[5]Throughout Section III, we discuss how to relax this assumption, allowing messages from malicious users to be delivered inconsistently.

All the committees can communicate internally in parallel, so the overall latency of each step $O(\ell)$. The computation costs have to be carefully setup as they impact security and correctness. Clearly, the PoW must be sufficiently larger than ℓ, say constant times larger, so that network latencies do not raise unmanageable unfairness for processors. We discuss these parameters carefully in our constructions.

Security. In each epoch, for $f = 1/3$ and a sufficiently large c, our protocol guarantees the following properties with high probability:

- S1. Among the first n' identities created, at most $n'/2-1$ are controlled by malicious users, with very high probability (depending on n').
- S2. For each committee, the number of honest processors is at least $c/2+1$ at the end of Step 1.
- S3. All honest committee members will discover all the other honest committee members in Step 2.
- S4. For each committee, Step 3 will yield a consensus value, signed by over half of the identities on the committee.
- S5. Step 4 yields a valid solution to the subset-consensus problem, signed by more than half of the final committee.
- S6. Step 5 will yield a set of random r-bit value with at least $r - c/2$ entropy. Therefore, the random values are sufficiently unbiased to be used for proof-of-work in the next round.

Our main analysis, presented in Section IV, shows that the adversaries cannot create more than $n/2$ identities in step 1 (S1), and cannot establish a committee with more than $c/2$ malicious identities (S2) in an epoch time with non-negligible probability (even if all the malicious users collude). Given that, the remaining security properties S3–S6 above will hold even in a adversarial execution.

There is one more security consideration that merits clarification. The shared randomness generated in Step 5 of an epoch is used as an input to the PoW in the next epoch. There are two potential sources of advantage for the adversary here: (a) the adversary learns the random value before of honest nodes (gaining a headstart on the next PoW); and (b) the adversary can bias the random string. However, our protocol ensure that these two sources give negligible advantage to the adversary in the next epoch. (See, e.g., S6.)

B. Committee Formation

First, each processor creates a set of identities of the form (IP, PK), where PK is a public key. The processor will reveal its identities to the network, keeping its private key secret. Throughout the epoch, this identity can sign messages. As discussed, we ensure that the total number of identities created by malicious processors should not exceed those created by honest processors (S1).

Next, our protocol assigns each identity to a random committee in 2^s, identified by an s-bit committee identity. The committee assignment must be random, even for the malicious users: a probabilistic polynomial-time adversary should not be able to bias its committee assignment with non-negligible probability.

We use a proof-of-work to achieve these goals. Let H be a random oracle that acts as a pre-image resistant hash function. As a "seed" for the proof-of-work, we need a public random string generated at the end of the previous epoch. (This ensures that the proof-of-work was not precomputed.) See the later discussion for how this is generated and verified. Assume, for now, that `epochrandomness` is a public random string generated in the previous epoch.

To join a committee, each processor locally searches for a valid `nonce` that satisfies the following constraint:

$$O = \mathtt{H}(\mathtt{epochRandomness}||IP||PK||\mathtt{nonce}) \leq D.$$

D is a predefined parameter in the network which determines how much work a processor has to do to solves a PoW. For example, D is set to require O to have 50 leading zeros. The last s bits of O specifies which (s-bit) committee id that the processor belongs to.

All processors know `epochRandomness` and choose their identity IP and PK privately. For any choice of `nonce`, H produces a sufficiently long random output. Assuming H generates an r bit string, the probability that a single invocation of H satisfies the constraint for a randomly chosen `nonce` is thus $p = (D+1)/2^r = 2^{-50}$ (for the choice of D involving 50 leading zeros). No efficient adversary can find a `nonce` that satisfies the constraint on the number of leading zeros with non-negligible probability better than p by the cryptographic pre-image resistance assumption. Therefore, each invocation can be seen as a coin flip with probability of yielding success with odds at most p. Once a processor obtains an output that satisfies the constraint, it can use the least significant bits and the `nonce` as its solution. If each processor has the computational power to invoke the hash function k times, the number of successes (i.e., the number of identities) is a binomial distribution $B(k,p)$. Therefore, the expected number of identities created by honest and byzantine processors is $n(1-f)\cdot p$ and $nf \cdot p$ respectively. Applying a Chernoff bound [24] and we can easily obtain S1.

For establishing S2, we need to examine the number of honest and Byzantine identities that map to any given committee. Since H is a random oracle, we can treat the bits in its output as unbiased and random. Therefore, the s bit strings generated in the solution are random, and an identity is mapped to a given committee with probability 2^{-s}. Further, If $n = 2^s c$, then within $O(n \log n)$ solutions to the proof-of-work, all the committees are filled (with high probability).

Byzantine adversaries can choose to not broadcast valid solutions, thereby denying membership in a committee. However, this does not advantage their membership in any other committee. It remains to choose the parameters s, which determines the number of committes, and D, which determines the difficulty. The difficulty should be significantly larger than the broadcast time, so as to limit the unfairness created by network broadcast latency.

C. Committee Overlay Join

Once identities and their committees are established, committee members need a way to establish point-to-point connections with their committee peers. A challenge here is that identities are established through a proof-of-work, which is a probabilistic process that occurs over time: identities are continuously being created at some rate. We need a mechanism to establish the first c members of the committee so that all honest members have the same view of the member set.

One could run any authenticated byzantine fault tolerant protocol here (aBTF) which tolerate upto $n/2$ malicious identities. However, this would yield BFT protocol running over the entire network without any parallelization. Here we need something more efficient.

The first c identities created in the network broadcast their identities to everyone—these identities are automatically elected to form a special *directory* committee. All subsequent identities created will contact the directory members to announce their committee membership. Directory members keep track of the committee membership announcements, and when their list for a committee reaches size c, they multicast it to all members of that committee. (If two announcements arrive at the same time, they are processed in lexicographic order.) Committee members accept any list that is sent by at least $c/2+1$ directories. Notice that there are only $O(nc)$ messages to or from the directories (in expectation), along with an additional $O(c)$ broadcasts to create the directory.

Why is this protocol secure? As with the committees, the directory will have an honest majority, with very high probability (as per Property S2). All the honest identities on the directory will, for each committee, identify the same c members (due to consistent broadcast and a consistent ordering in which the members are processed).

If we weaken the consistent broadcast assumption, then it becomes possible for honest directories to have slightly different sets of c committee members (depending on which messages arrive and in which order). In this case, a closer analysis of the rate at which proof-of-works are solved shows that, with high probability, the sets only differ by a very small amount. We can observe that small differences in the views on committee membership have no impact (as those identities can be just treated as Byzantine).

D. Intra-committee Consensus

Once a committee is established, the protocol to agree on a value can reuse any existing authenticated Byzantine agreement protocol. (e.g., [3], [25].) All members have the public keys of all other committee members, therefore an authenticated BFT protocol are a natural choice. We point out that one can use an asynchronous or synchronous protocol here depending on your network; our implementation uses a simple synchronous protocol for ease of implementation.

Thus, each member of the committee chooses a value to propose, checks its validity (if such a check is necessary), and executes the agreement protocol. If the value produces by the consensus protocol is invalid, then the agreement protocol is repeated.

The selected value is digitally signed by the committee members, i.e., it acquires at least $c/2+1$ signatures from the honest members. Each committee member then sends the signed value to the final committee (using the directory, again, to acquire the list of final committee members). A value that is not the agreed one cannot have $c/2+1$ signatures since there are fewer than $c/2$ malicious identities, and honest identities only sign the selected value. The final committee can verify that a certain value is the selected one by checking that it has at least $c/2+1$ signatures.

E. Final Consensus Broadcast

The next step of the protocol is to merge the results of committees and to create a cryptographic digest (a digital signature) of the final agreed result. A final committee (say, with a fixed s-bit committee id) is designated to perform this step. The merge function is simple: each final committee member validates that the values received from the committees are signed by at least $c/2+1$ members of the proper committee, and takes the ordered set union of all inputs. To ensure that the result is indeed correctly composed from the correct inputs, the final committee run the same intra-committee algorithm described previously. This step obtain a verifiable signature by at least $c/2+1$ members of the final committee, which the entire network can verify upon broadcast.

F. Generating Shared Randomness

In the final step of the protocol, the final committee generates a set of random strings for use in the next epoch. It is critical that this happens at the very end of the epoch, so that the malicious users do not learn the random string too early (and hence cannot start working on the proof-of-work for the next epoch too early).

This phase of the protocol consists of two rounds of communication. In the first round, each member of the final committee chooses a random string u consisting of r random bits and broadcasts a hash $H(u)$ to everyone (i.e., not just to the final committee). This serves as a commitment to the random string. In the second round, each member of the final committee broadcasts a message containing the random string u itself to everyone (i.e., not just to the final committee).[6]

At this point, each user in the system has received at least $c/2+1$ and at most c pairs of commitments and random strings from members of the final committee: the honest members follow the protocol, while the malicious users may choose to withhold either the commitment or the random string. Each user discards any random strings that do not match the commitment.

For the purpose of the next epoch, each user takes an XOR of the set of valid random strings received. In a network with consistent broadcast, everyone receives the same set of strings. In a network without consistent broadcast, users may receive different subsets of the strings. We consider the XOR of *any* subset of size at least $c/2+1$ of the valid random strings sent by the final committee to be a valid shared random string.

[6]In practice, the first round of communication should be long enough to compensate for any variation in when members start the round; these two rounds may be separated by a small gap to ensure that no message from the first round is accepted as valid after any message from the second round has been sent. Each round should be long enough to ensure that all the messages sent by honest members are received.

Recall that these random strings are used as the seed for the proof-of-work in the next epoch. In order to verify that a proof-of-work is valid, the user should attach to the solution of the proof-of-work the set of random strings of size at least $c/2+1$ used to generate the seed. Any other user can then verify that these random strings were sent by the final committee members and match the commitments.

Finally, we will need to argue that the random strings generated from this process have sufficient entropy, i.e., are not too biased by the malicious users. Notice that by choosing which $c/2+1$ of the c valid random strings are used to generate the seed for the proof-of-work, a malicious user can gain some control over the random string. However, as we discuss in Section IV, this reduces the entropy by at most $c/2$ bits, yielding a random string with at least $r-c/2$ bits of entropy—which is sufficient.

IV. Security analysis

In this section, we show how the security properties described in S1 to S6 are achieved in the SCP protocol.

We begin by clarifying several assumptions. First, we assume time is divided into discrete communication *rounds* each of which has fixed length (e.g., five seconds) long enough for users to broadcast information to each other. We assume all honest processors are synchronized: they have the same view during beginning and ending of each round. Second, we assume that the network layer can provide consistent broadcast (i.e., every broadcast message is delivered to every other user).[7]

Definition 1. *In a given epoch, for each committee, we refer to the first c identities that broadcast a valid proof-of-work for this committee as the* members *of this committee. (If more than one proof-of-work is broadcasted during the same round, they are ordered lexicographically, and the smallest one is picked.)*

Notice that, by definition, each committee has exactly c members, and these are exactly the members identified by the directory.

All our claims for an epoch depend on their being sufficient random strings generated in the previous epoch:

Definition 2. *We say that an epoch has* good randomness *if: (1) every user has a publically random string of r bits, verifiably generated in the previous epoch, with at least $r-c/2$ bits of entropy, and (2) no user has access to such a verifiable random string more than two communication rounds prior to the beginning of the epoch.*

We now prove the security properties of SCP. In particular, we start with S1, which states honest identities take a dominate portion in all the generated identities.

Lemma 3 (**S1: Good Majority**)**.** *In every epoch with good randomness, for every integer n': among the first n' identities created, at most $n'/2$ are controlled by the adversary, with probability at least $1-e^{-27n'/160}$,*

[7]In Section III, we discussed in presenting the protocol how to cope with weaker broadcast assumptions; the reliance on consistent broadcast simplifies, but is not necessary.

Proof: The proof-of-work guarantees that if all the users start at the same time, each solution generated has a $2/3$ probability of being held by an honest processor since the honest processors have $2/3$ of the computational power. However, the malicious users may start at most two (communication) rounds early—if T is the expected time for all the users, collectively, to find one proof-of-work, then a loose upper bound shows that each proof-of-work is malicious with probability at most $[(T+2)/T](1/3)$, i.e., it increases their power very slightly proportional to the extra time they have. Assuming $T \geq 10$, this yields the probability of a malicious proof of work to be $2/5$.

As a result, we know for each identity generated, with probability at most $2/5$ it will be taken by the adversary, and with probability at least $3/5$ it will be taken by the honest processors. Now, Let X_i be an indicator random variable which takes value one if the i^{th} identity is generated by an honest processor. Let $X = \sum_{i=1}^{n'} X_i$. We know in expectation, $X = 3n'/5$. Apply a Chernoff bound [24], we know with probability at most $\mathbb{P}(X \leq (1-3/4)(3n'/5)) = \mathbb{P}(X \leq n'/2) \leq \exp(-(3n'/5)(9/16)/2) = \exp(-27n'/160)$, the first n' identities will contain at least $n'/2$ malicious identities. ∎

By a similar argument, we can obtain S2, which states each committee is dominated by honest identities. We also show that the malicious users will not generate too many malicious identities during the epoch. (Recall that even after the committee is full, the malicious users can continue to generate identities which they could use to form a fake committee.)

Lemma 4 (**S2**: Committees Good)**.** *In every epoch with good randomness, for each committee, at least $c/2+1$ committee members will be honest with probability at least $1-e^{-27c/160}$. Moreover, the probability of generating $c/2+1$ malicious identities by the end of the epoch is also exponentially small.*

Proof: Consider a given committee, by an argument similar to the one as in the proof of Lemma 3, we know that each "seat" in the committee is honest with probability at least 3/5 and malicious with probability at least 2/5. Thus by the same Chernoff bound calculation, the probability of a committee containing $c/2+1$ malicious users is at most $\exp(-27c/160)$.

Similarly, notice that the expected time for the malicious users to generate $c/2+1$ identities is (approximately) twice that of the honest users. Assume that T_{honest} is the expected time for the honest users to generate $c/2+1$ identities (and hence a lower bound on the expected time to form a committee) and T_{rest} is the time for the epoch *after* the honest users generate $c/2+1$ identities. As long as $T_{honest} = \Omega(T_{rest})$ (e.g., the committee generation is at least 10 times longer than the remainder of the epoch), then a similar Chernoff bound shows that the probability of the malicious nodes generating $c/2+1$ identities in time $T_{honest}+T_{rest}$ is exponentially small. ∎

(A somewhat tighter bound on the probability can be found numerically evaluating the binomial distribution, which we used when selecting our committee size of $c=400$ to ensure a probability of error of approximately 2^{-40}.)

We now proceed to prove S3, which states for any given committee, all honest members will correctly obtain the (same) member list.

Lemma 5 (**S3**: Consistent Committees)**.** *In every epoch with good randomness, if the directory size is* d*, then for each committee, with probability at least* $1-(e^{-27c/160}$*, all honest committee members will correctly adopt the (same) member list for the committee.*

Proof: The directory consists of the first c identities to be established, and we know from Lemma 3 that at least a majority of the directory is honest with probability at least $1-e^{-27c/160}$. Each honest directory member will adopt the same memership list for the directory (by consistently ordering the membership requests for the committee), and hence will broadcast identical lists. Hence each member on the committee will receive at least $c/2+1$ identical lists from the directory, and hence they will adopt this (unique) list. ■

We now show S4, which argues that a committee correctly decides a single value.

Lemma 6 (**S4**: Consensus)**.** *In every epoch with good randomness, the honest members agree on a unique value with at least* $c/2+1$ *signatures, with probability at least* $1-e^{-27c/116}$.

Proof: Consider a given committee, by Lemma 4, we know that a majority of the committee is correct with probability at least $1-e^{-27c/116}$. The authenticated Byzantine agreement protocol then guarantees agreement, i.e., only one value selected). The protocol is repeated until a valid value is produced, which is then signed by (at least) all the honest members. ■

By a similar argument, we can show S5, which states the final committee can correctly combine values from other committees and broadcast it to the whole network.

Lemma 7 (**S5**: Final Committee)**.** *In every epoch with good randomness, the final committee its honest members will broadcast a combined value (from values from other committees) which has at least* $c/2+1$ *signatures, with probability at least* $1-2^s\cdot(e^{-27c/160}$.

Proof: By Lemma 6, each committee submits a valid value to the final committee with at least $c/2+1$ signatures. Moreover, according to Lemma 4, we know that there cannot be more than $c/2$ malicious members that have solved the proof-of-work for this committee in this epoch. This implies, for each committee, the final committee will only receive one value from it which has at least $c/2+1$ signatures. Therefore, the final committee can correctly combine values from the other committees.

On the other hand, once the honest members of the final committee combine the values and broadcast it, the adversary cannot broadcast another such value to fool the network, as the value is signed by at least $c/2+1$ members (and the final committee indeed has at least $c/2+1$ honest members), yet the adversary can only control at most $c/2$ members in the final committee (due to Lemma 4). ■

Lastly, we show S6, i.e., that the shared randomness generated is sufficient:

Lemma 8 (**S6**: Good Randomness)**.** *In every epoch with good randomness, with probability at least* $1-\cdot(e^{-27c/160}$*: at the end of the epoch every user computes a random string of* r *bits that: (a) contains at least* $r-c/2$ *bits of entropy, and (b) can be verified as validly generated in that epoch. Moreover, no user knows any of the random strings until at least two rounds prior to the end of the epoch. That is, if epoch* e *has good randomness, then epoch* $e+1$ *also has good randomness.*

Proof: By Lemma 4, the final committee has a majority of honest identities with probability at least $1-2^s\cdot(e^{-27c/160})$, and the probability that the malicious users have more than $c/2$ identities having solved the proof-of-work for the final committee is exponentially small.

Thus, each user receives between $c/2+1$ and c commitments and random strings from members of the final committee in the last two rounds of the epoch. The user's random string is generated by `xor`ing these strings.

Since there are at least $c/2+1$ bit strings received, we know that least one of them originated at an honest user, and hence contains r bits of entropy. The remaining bit strings may be generated by the malicious users. However, the malicious users committed to these strings prior to observing the bit strings from the honest users. Hence the only method by which they can reduce the entropy of the `xor` of the strings is by choosing which subset of the c possible strings are included in the subset of size $c/2+1$.

For each bit string that the malicious users can choose to include or not include, the entropy of the final `xor` is reduced by 1 bit, yielding a final bit string with at least $r-c/2$ bits of entropy. (To see this, notice that to fix y bits in the outcome, the adversary needs 2^y possible choices, which requires y different committed bit strings, and $y\leq c/2$.)

Finally, by construction any user can verify that the set of $c/2+1$ bit strings is valid by checking the commitments sent out previously. Similarly, it is immediate that no user knows the random string prior to two rounds before the end of the epoch. ■

With the above lemmas in hand, the correctness of the SCP protocol follows by induction:

Theorem 9. *For every epoch* $i\geq 1$*, with probability at least* $1-i\cdot 2^s\cdot(e^{-27c/160}$*, the following properties hold: (a) the final committee will broadcast only one combined value to the network with at least* $c/2+1$ *signatures and no other combined value will have* $c/2+1$ *signatures; (b) this combined value contains* 2^s *sub-values each of which comes from a committee and is verified by at least one honest processor; and (c) at the end of the epoch, each user has a publically verifiable random bit string of length* r *containing at least* $r-c/2$ *bits of entropy (i.e., the following epoch has good randomness).*

Notice that we have shown, at this point, that each epoch ends with a correct (combined) value selected. Any user that is in the system can, by listening, verify that this combined value is the correct and honest one. Often, it is desirable that the correctness by externally verifiable, e.g., by a user that was not in the system at the time. In Bitcoin, this is achieved by showing that the chain constructed is the longest chain, with

```
Input:
  PreviousTX: ID of previous transaction
  Index: 0
  scriptSig: Sign(PubKey), PubKey

Output:
  Value: 5000000000
  scriptPubKey: %take Signature and PubKey as params
    checkif Hash(PubKey) = Payee's ID,
    checkif Sign(PubKey) is valid
```

Fig. 1: Illustration of a simple transaction in Bitcoin. User's address is computed by hashing their public key. `scriptPubKey` is a script that defines how the payee claims the recieved Bitcoin.

very high probability (i.e., exponentially small probability). In fact, the same property holds here: on average, it will take the malicious users twice as long to generate a signed final value as the honest users. Hence, an honest "chain" will grow twice as fast as a malicious "chain" and hence with very high probability it will be externally verifiable. (For a more detailed discussion of this issue in Bitcoin, see [19]; the argument here is similar.)

V. Implementing SCP in blockchain

We describe our implementation to adapt SCP consensus protocol and build scalable blockchains. However, we first provide brief background about blockchain structure including transactions and the cryptographic digest that we use in SCP.

A. Implementation Background

A transaction (TX) represents an atomic operation that users carry out in the network. Each valid TX updates the state of the blockchain. For example, in Bitcoin, a regular TX sends some amount of Bitcoin from a sender to a receiver, as shown Figure 1. The sender must specify in the `Input` part of the TX where he gets the Bitcoin from and a proof to show that he is a valid owner, *i.e.*, having a valid private key. The sender also specifies in the `Output` part the receiver and on what condition (in `scriptPubKey`) the receiver can spend the money. Users when verify a TX need to check the sender's proof in `scriptSig` by running through the `scriptPubKey` of the `previousTX` that the sender is spending. Currently this computation is done by all users in the Bitcoin network for all TXs included in a new block.

Bitcoin and existing cryptocurrencies use Merkle tree structure to cryptographically commit a set of transactions in a block [26]. A Merkle tree is essentially a binary hash tree. Each leaf node stores some transaction ID (hash of some transaction). Each internal node stores a hash on the concatenation of its children nodes' values. Thus, Merkle tree allows all transactions to e committed in a single hash value stored by the Merkle root. An example of a Merkle tree storing all transactions having IDs from 1 to 8 is illustrated in Figure 2. The Merkle tree allows one to prove the existence of some transaction in a committed tree by a short proof of roughly $log(m)$-size, where m is the number of leaf nodes in the tree. Additionally, a verifier only needs to store the Merkle

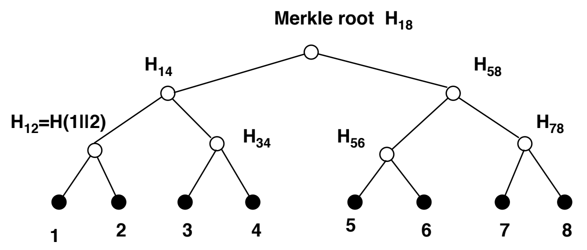

Fig. 2: An example of a Merkle tree which commits all transactions having IDs from 1 to 8.

root hash of the committed tree. For example, one can prove that 1 is included in the Merkle tree in Figure 2 by providing $\mathtt{H}(2), \mathtt{H}_{34}, \mathtt{H}_{58}$. The verifier checks if he/she can reconstruct the committed Merkle root hash to accept the proof.

B. Two-layer blockchain

We devise a novel two-layer blockchain design to implement SCP and improve the transaction rate in the network. Our two-layer blockchain includes two different block types namely consensus block and data block. A data block is proposed by normal committee while a consensus block is proposed by the final designated committee in SCP to include all data blocks.

A data block contains transactions like a normal block in Bitcoin but is not broadcast to the network immediately. On the other hand, a consensus block has only commitments of data blocks with header information, *e.g.*, timestamp, block hash. Further, there is only one consensus block in an epoch and a valid consensus block must be broadcast immediately. As shown in Figure 3, a data block is committed to a consensus block by only two `SHA256` values: a block hash and a Merkle root hash. As discussed in Section III, SCP allows users to trust the validity of all blocks without downloading the data. Thus, our blockchain implementation enjoys the advantage of separating the transactional data and data needed for the consensus formation. We depict the overview of a blockchain which implements SCP's protocol in Figure 4.

Estimate computation capacity in the network. Another important feature when implementing SCP is to estimate the computation capacity in the network and to adjust the number of committees accordingly. This is to keep the average of epoch time constant, *e.g.*, 10 minutes as in Bitcoin. We employ a similar technique as in Bitcoin to adjust the number of data blocks can be introduced per each time blocks. Concretely, we consider a timespan of, say, 100 epochs. If the average time taken by an epoch is longer than the expected time, *e.g.*, 10 minutes, we reduce the number of committees as in the following equation.

$$\text{Next \#. committees} = \text{Current \#. committees} \times \frac{\text{Expected epoch time}}{\text{Average epoch time}} \quad (1)$$

Based on Equation 1, all parties can determines how many bits that they need to consider in their valid PoWs to computer their committee numbers. There is an edge case where the number of committees is already 1 and the expected epoch time is more

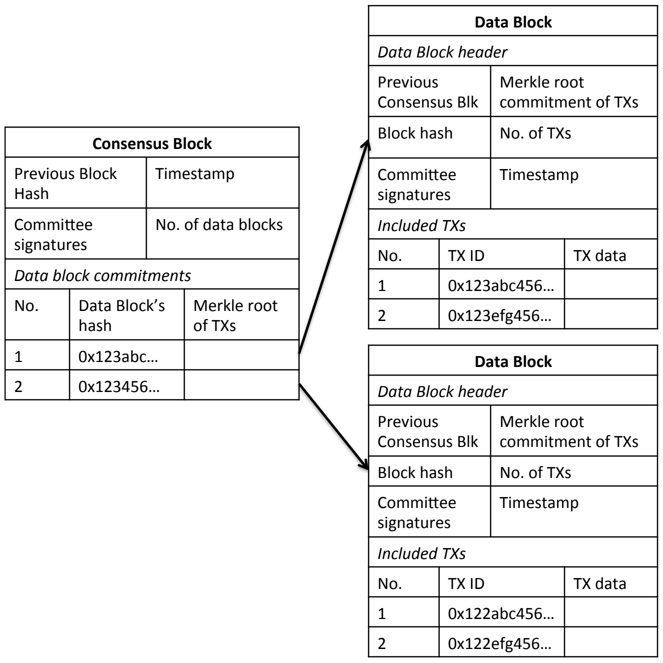

Fig. 3: Block templates of consensus block and data block in SCP. Each data block is committed to a consensus block by their block hash and the Merkle root of a Merkle hash tree over included transactions.

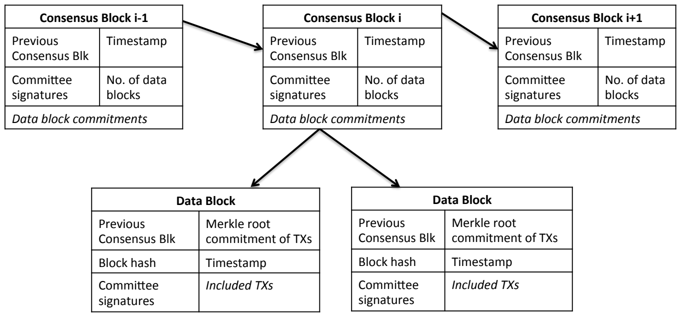

Fig. 4: Blockchain design in SCP. Each block epoch has only one consensus block but multiple data blocks.

than 10 minutes. In this case we fall back to the Nakamoto consensus protocol to adjust the difficulty D of the PoW using a similar formula as in Equation 1.

C. Potential advantages

We discuss about other advantages that SCP provides in addition to achieving our scalability goal.

Verifier's dilemma. The verifier's dilemma happens when verifying a block is expensive [27]. The intuition of the dilemma is that users do not know whether to i) skip the verification of a block to gain more advantage in finding the next ones or ii) to verify and avoid the risk of mining invalid blocks. SCP eliminates this dilemma by requiring users to verify only a small constant amount of information, *i.e.*, the block header, when they receive a block from their peers. It is because users can rely on the fact that the correctness of a block is verified by at least one honest member in some committee. Thus, users can process to the next block race without verifying all data included in all blocks.

Mining and transaction centralization. One serious problem in Bitcoin and most cryptocurrencies is mining centralization in which users with insufficient computational power often join some mining pool to mine together. As a subsequence, the blocks in Bitcoin mostly come from a few popular pools. For example, as of writing, 3 most popular pools account for 55% of computation capacity in the network [8]. Thus, the security of the network relies on how those big pools behave. Further, these pools can decide 55% of transactions going to the network. In SCP, we allow miners of much less computational power to join a committee and vote for a block. Second, the transactions are split uniformly among several data blocks, thus improving the decentralization of the network.

VI. Case Studies

We next describe how to build several blockchain-based applications on top of our two layer blockchain. For each application, we show how one can efficiently implement different validity checkers $\mathcal{C}$ based on SCP's design to allow committee members to check if a block is valid. A validity checker $\mathcal{C}$ takes a block data and previous block headers as input then outputs if the block data is valid. In order to verify a block in SCP, one needs to verify all included transactions. Thus, an atomic operation in $\mathcal{C}$ is to check whether a transaction is valid. A normal transaction is valid if its input i) refers to an output of a previous transaction; ii) includes a valid proof that it can update the account associating with that proof. There may be other application-specific property of a transaction like double spending in cryptocurrency will require an additional check. We illustrate how to efficiently perform all general and specific checks in SCP.

A. Blockchain-based certificate directory

Previous works have proposed several decentralized credential systems based on Bitcoin and cryptocurrency which support attestation about user identity [28], [29]. These systems allow organizations like service providers (*e.g.*, Gmail, Facebook), DNS maintainers to verify whether a user is valid, *e.g.*, has a correct private key according to some account. This design is shown to have many advantages over traditional centralized authentication systems. First, organization can securely authenticate users without a centralized certificate/ password database, thus removing the risks of insider attacker or single point of failure. Second, users can actively manage their credentials without contacting the organization servers. For example, users can add/ revoke their public keys corresponding to some account by directly sending a transaction to the blockchain. The next time the user logs in, the website verifies if the user is valid based on the latest state of the blockchain. The third and most significant advantage is that users and websites can leverage the script feature provided in

[8]https://blockchain.info/pools

a cryptocurrency to define more authentication policies. For example, a group of 4 users sharing an account can dictate a rule that requires at least any 3 users to login and all 4 users to update the corresponding certificates. The systems in [28], [29] also provide high guarantee for user privacy and anonymity.

In this case study, we discuss how one can implement such decentralized credential systems by employing SCP. For simplicity, we introduce a simple global certificate directory namely SCERT that allows websites to store users' certificates in the blockchain. One can draw an analogy between transactions in Bitcoin and SCERT. Specifically, in Bitcoin a transaction represents a coin ownership while in the latter system it represents an ccount ownership. A user U can transfer and update the ownership of an account by creating a new transaction that "spends" some existing one.

Building a validity checker $\mathcal{C}_{cert}$ in SCERT. We illustrate how one can check if a SCERT's transaction T_i is valid given only the transaction data and previous block headers. In SCOIN, we explicitly ask the sender of T_i to provide two proofs in T_i's input: One proves that T_i is using an output of some transaction T_p included in previous blocks, the other proves that the sender is a valid owner of that T_p's output. In order to generate the first proof, T_i's sender provides T_p's data and a Merkle proof that shows T_p's hash was committed in some previous block $\mathcal{B}$. Checking this proof is easy given $\mathcal{B}$'s header, as we discussed in Section V-A. On the other hand, the second proof depends on the semantic defined in T_p's output. For example, this second proof can be as simple as a signature signed by a correct private key. Although the second proof does not have a fixed format, it is verified easily given T_p's output. Thus, using our general validity checker $\mathcal{C}$, committee members can verify the correctness of a block based on the block's data and all previous block headers.

B. SCOIN*: A Scalable Cryptocurrency*

We next discuss how one can build a scalable cryptocurrency SCOIN on top of SCP. Building SCOIN is trickier than SCERT as now there exists the double spending problem. We consider a simple scenario in which Alice has some coin and wants to send to Bob. She does that by creating a transaction with a valid signature as a proof of payment. Using our above general validity checker $\mathcal{C}$, Bob can verify if Alice actually has the money, but Bob cannot check if Alice has not sent the same coin to someone else. A naive approach to check if a transaction T_i is double spending some money is to check all previous blocks to see if some transaction $T_j (i <> j)$ has spent that money. This approach is used in Bitcoin and requires previous block data [9]. In SCP, the validity checker $\mathcal{C}_{coin}$ only has the current block data and previous block headers, thus this naive approach does not work.

1) Avoid double spending transactions: Our approach to prevent double spending is performed in two steps: local double spending detection and global double spending detection. The two steps check if a transaction T_i is double spending within a block, *e.g.*, T_i and T_j are included in the same block, and across blocks, *i.e.*, T_j is included in a different block. We show that our cryptocurrency-specific $\mathcal{C}_{coin}$ can guarantee that all TXs are not locally double spending and provides a ground trust so that TX receivers can perform the global double spending detection. Specifically, Alice can include her global double spending transaction in SCOIN, but Bob can easily detect the problem and decide not to accept Alice's payment. Thus, our approach gets receivers involved in preventing double spending transactions. This is a reasonable approach since double spending transactions affect receivers directly, hence they should be more responsible than the network to check whether a transaction is not spent elsewhere. We next describe how to build such $\mathcal{C}_{coin}$ and how Alice can prove to Bob that her transaction is not spent elsewhere.

[9]Bitcoin implements a mempool data structure to make this search progress more efficient, but the check still cannot be done without previous block data.

```
struct BlockHeader:
    uint256 prevBlockHash
    uint256 txsHash
    uint256 OutMklTree
    uint256 nonce
    uint timestamp
```

Fig. 5: A 164-byte block header of SCOIN. `txsHash` is the Merkle hash root of a Merkle tree storing all Transactions included in the block while `OutMklTree` is the second hash root of a Merkle tree committing all sorted pTxOuts, *i.e.*, outputs of previous transactions which are being spent in this block.

For simplicity, we assume that a transaction in SCOIN includes only one transaction input (TxIn) and one transaction output (TxOut). The TxIn indicates a previous TxOut (pTxOut) that it is going to spend the money from and a prove that the transaction creator is eligible to spending it.

Constructing a cryptocurrency-specific checker $\mathcal{C}_{coin}$. We recall that $\mathcal{C}_{coin}$ has two main functions namely checking if a transaction is double spending within a block and building a platform for detecting global double spending transactions. We introduce an additional Merkle tree namely `OutMklTree` to commit all previous transaction outputs (pTxOuts) which are being spent in a block. The pTXOuts stored in leaf nodes are sorted in an ascending order if we travel from left to right. Our new block header is illustrated in Figure 5. Typically, our cryptocurrency-specific $\mathcal{C}$ will mark a block as valid if: i) all transactions are marked valid by a general check; ii) there is no pTxOut appearing twice in our `OutMklTree` — this is to prevent double spending transaction within a block; iii) pTxOuts are in ascending order and the Merkle tree which stores them is correctly built. Checking all these conditions is thus easily done, given current block data and previous block headers. We next explains how Bob can rely on the correctness of $\mathcal{C}$ to detect if a transaction is globally double spending.

Detecting global double spending. In SCOIN, a transaction is included in a block does not guarantee that it is valid. However, it still shows that the senders, *e.g.*, Alice, try to spend the money from a particular pTxOut specified in the TxIn of that transaction. Thus, Bob can accept Alice's transaction if only pTxOut has not been spent in any previous blocks. SCOIN allows Bob to do that by using previous block headers to check

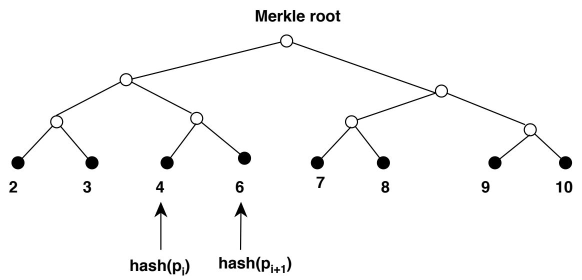

Fig. 6: Alice proves that a pTxOuts p which has $\texttt{SHA2}(p) = 5$ does not get included in a committed Merkle tree. Alice provides Merkle paths of p_i, p_{i+1} that $\texttt{SHA2}(p_i) = 4$ and $\texttt{SHA2}(p_{i+1}) = 6$ to Bob. Bob can verify if p_i, p_{i+1} are committed and consecutive in the pTxOuts tree to conclude if p is not included in a block.

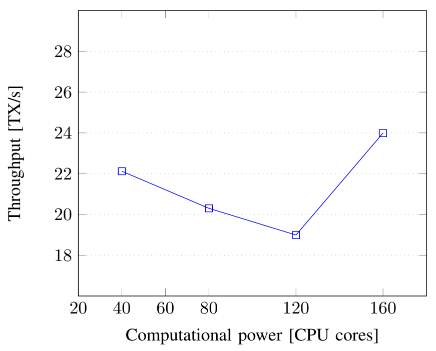

Fig. 7: The throughput in Bitcoin does not increase if we add more computational power to the network. Epoch time: 1 minute.

a proof from Alice. Specifically, Alice has to generate a proof which demonstrates that her spending pTxOut has not been included in any block. We recall that a SCOIN block has a Merkle tree which commits all pTxOuts that are being spent, and these pTxOuts are sorted. Thus, Alice can generate a proof of size $\texttt{log}(m)$ to show that her pTxOut is not included in a specific block, given m is the number of transactions in that block. Bob can verify the correctness of Alice's proof based on the `OutMklTree` stored in the block header. For example, if her pTxOut p is indeed not in a block, Alice can find two consecutive pTxOuts p_i and p_{i+1} in the second Merkle tree which satisfy

$$\texttt{SHA2}(p_i) < \texttt{SHA2}(p) < \texttt{SHA2}(p_{i+1}) \tag{2}$$

where `SHA2` is the function that compute the ID of a transaction output. Alice then computes the Merkle paths of p_i, p_{i+1} and send them to Bob. To verify, Bob recalculates two Merkle paths of the pTxOuts given by Alice. If the two are adjacent in the committed Merkle tree and condition (2) holds, Bob can assure that there is no transaction in the block spending p. We illustrate an example of this proof in Figure 6. Similarly, a proof which shows p is not included in any previous blocks is of size $k\texttt{log}(m)$ with k is the number of blocks in the blockchain.

VII. EVALUATION

We implemented a prototype of SCP and provide empirical evaluation on the scalability of SCP and Bitcoin. We will discuss about our experiment setup and the properties that we seek to evaluate SCP.

A. *Bitcoin's Scalability Limits*

Experiment setup. Our experiments are run over Amazon EC2 platform. We launch several instances with high network performance from different geographical regions. We feed enough transactions to the network in every experiment [10]. We run the first set of experiments using the same network implementation in Bitcoin [17] to show that Bitcoin and Nakamoto protocol do not scale up if computationally power or bandwidth in the network increases. We also show that reparameterization helps improve the throughput in Bitcoin, but the network will soon hit some constraints like bandwidth limit, block validation cost.

We run the first set of experiments with various computation capacities in the network, *e.g.*, 20 CPU cores, 40 CPU cores and 80 CPU cores. Since the difficulty adjustment for the PoW happens after a certain number of blocks, *e.g.*, 2016 blocks in Bitcoin, we measure the throughput after the first adjustment happens and the network becomes more stable. To speed up our experiments, we set the epoch time as 1 minute and the difficulty adjustment happens after every 50 blocks. We report our results in Figure 7. Our experiments show that, because of the self-adjustment mechanism in Bitcoin, the block rate remains almost constant regardless of how much computational power is in the network. Thus, we see that the transaction rates are fluctuating around 22 TX/s when the network has 40 CPU cores (22.12 TX/s), 80 CPU cores (20.32 Tx/s) or 160 CPU cores (23.99 TX/s). Therefore, Bitcoin cannot scale up linearly in the computation capacity of the network.

Reparameterization in Bitcoin. One simple way to scale up Bitcoin is by reparameteraization, *i.e.*, changing some parameters like epoch time, block size. We experimentally show that Bitcoin scales up somewhat linearly with the adjustment of these parameters up to a point after which the agreement begins to fail. Particularly, we change the epoch time of Bitcoin protocol and see how much the improvement on the transaction throughput is. We also feed as many transactions as possible to the network and run several Bitcoin blockchain of 12 seconds, 1 minute, 2 minutes, 4 minutes and 10 minutes. The underlying network parameters like bandwidth, computational power remains constants throughout the experiments at 80 CPU cores. Our results in Figure 8 show that the more the epoch time reduces, the higher the TX rate is. For example, the TX rate are 3.29 and 7.33 TXs per second at epoch times of 10 minutes and 4 minutes respectively. However, as the epoch time gets smaller, the gap between the TX rate and the expected TX rate is wider. For instance, at epoch time of 1 minute, we get a TX rate of 20.32 TXs per second instead

[10]We guarantee that the mempool size in Bitcoin is always larger than the block size

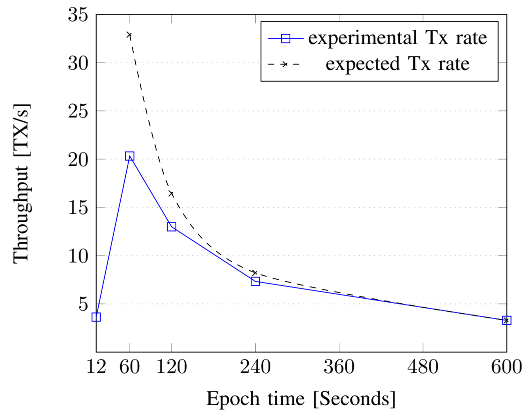

Fig. 8: Bitcoin can scale up by adjusting the protocol parameters like epoch time. The expected TX rate for a network with particular epoch time of X seconds is computed as (experimental rate for 10-mins)$\times(600/X)$

of 32.9 *e.g.*, 10 times the TX rate in the 10-minute epoch blockchain. Further, at epoch time of 12 seconds, the network hits the bandwidth constraint when nodes cannot broadcast new blocks fast enough. Thus we observe a significantly low TX rate (3.63 TX/s) since the network spends much more time reaching consensus on the blockchain stain.

B. Scalability of SCP

We implement a prototype of SCP by modifying the existing Bitcoin client code [17]. For the sake of simulation, we fix our committee size at $c = 50$ and our epoch time is at 9 minutes to ensure that there is a sufficiently large gap between two consecutive members in a committee. We also simulate an authenticated BFT protocol in [25] which has a message complexity of $O(c^2)$ and each message has the size of a block hash. In order to simulate the consistent broadcast, we run our SCP experiments in a local network to prevent the deterministic failure in EC2 network.

The goal of our experiments is to show that SCP can scale up computationally and the bandwidth incurred in the network does not increase quadratically. We start with a fixed amount of computational power (10 CPU cores) and double it several times to 20, 40 and 80 CPU cores. We quantify the block rate (block per epoch time) in each setup. We report our results in Figure 9. We see that our experimental throughput closely matches the expected throughput when the computational power increases.

We further show the optimism of our design in separating data needed for reaching consensus and actual transactional data. We provide a breakdown of how measure how much data needs to be transfered immediately and how much data is transfered by a node. We report our numbers in Figure 10. We see that SCP's bandwidth for reaching consensus is independent of the bandwidth in the data plane, *i.e.*, data block.

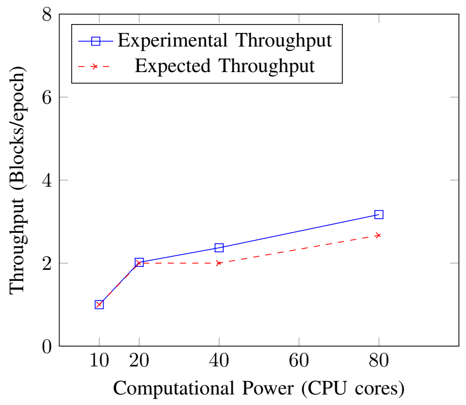

Fig. 9: SCP can scale up nearly linearly in the computation capacity of the network without quadratically increasing the bandwidth.

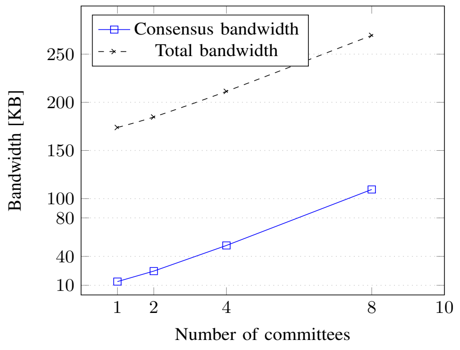

Fig. 10: Breakdown of bandwidth cost for a node in SCP. Our broadcast mechanism reduces the bandwidth broadcast to the network significantly and makes it independent of the block size.

VIII. Related work

A. Blockchain Scalability Solutions

Building a scalable Bitcoin is an active problem in the Bitcoin community. There have been several proposals from both academia and industry. However, There are two significant differences from SCP to others that i) only SCP allows the throughput to scale up sublinearly with the computation capacity; ii) SCP efficiently separates the bandwidth of consensus layer and that of data layer. We discuss all proposed solutions below.

GHOST, introduced by Sompolinsky *et al.* [30], pushes more transactions to the network by modifying the rule to accept the main valid blockchain. At every fork, GHOST considers the branch which has the most amount of work as the valid chain. Thus GHOST accepts not only the earliest block at each epoch, but also other blocks which are found later, *e.g.*, "orphaned blocks". GHOST succeeds at allowing block parallelism in the network, but it does not separate the bandwidth data plane and the consensus plane. Thus, all data of those orphaned blocks needs to be broadcast to the network and puts more burden on the network bandwidth when reaching agreement. Further, GHOST does not allow the network to scale up with the computational power.

A concurrent work namely Bitcoin-NG proposed recently by Eyal *et al*. also improves the transaction rate by introducing more blocks within an epoch [14]. Similar to SCP, Bitcoin-NG also introduces two different block types, namely key blocks and microblocks. However, these blocks have different roles. The key blocks are for leader elections while the microblocks only contain transactions proposed by the leader of that epoch. Similar to GHOST, all Bitcoin-NG blocks in one epoch still need to be broadcast to users immediately, thus creating a serial dependency between the data blocks. Moreover, Bitcoin-NG does not allow a system to grow linearly with the computational capacity of the network. On the contrary, SCP completely decouples the data block and the consensus block in addition to the sublinear scale up with the computational power.

Recently Poon *et al*. introduced lightning-network to improve the transaction throughput in Bitcoin network by having an offchain micro-payment channel between two parties [31]. Their technique allows both parties to send multiple and instant transactions between them more efficiently, with only a few transactions that will be included in the blockchain. However, micro-payment channel, although improves the transaction throughput significantly, is only an application running on top of Bitcoin and still relies on the underlying protocol of Bitcoin. Thus, lightning-network does not solves the core problem of the Nakamoto consensus protocol. First, micro-payment channels only make sense for sending transactions with small values. The need of a scalable cryptocurrency network will show up when more transactions of higher values occur in the network.

Buterin *et al*. also aims to address the scalability problem in blockchain with sampling and challenging techniques [32]. Similar to SCP, the paper proposes to split network participants into several sub-committees in the network to distribute computational and verification cost. However, Buterin *et al*. randomly samples verifier to verify the correctness of others' updates, and allow someone to challenge others' verification results if they ever detect an invalid update. Their solution is somewhat complicated and does not provide security proofs.

Sidechains is a protocol that allows users to move coins between different blockchains [33]. It is widely understood that Sidechains does not solves the scalability problem in Bitcoin. Specifically, Sidechains does not makes Bitcoin scalable or reduces any costs in Bitcoin. Sidechains is only but a method to easily setup experimental blockchain without requiring a currency. For example, with Sidechains, users can transfer their Bitcoin to a new blockchain which provides a different security guarantee. Thus, applications enabled by Sidechains do not enjoy the same security guarantee as the applications that run on the Bitcoin blockchain. On the other hand, SCP allows scaling up computationally without degrading any security property of Bitcoin.

B. *Scalable Consensus Protocol*

There have been significant efforts devoted to developing scalable communication-efficient consensus protocols. The idea of dividing the users into committees (as we do in this paper) is prevalent in the existing literature; first introduced by Bracha [34].

If the users are honest, but crash prone, there exists an optimal algorithm with $\Theta(n)$ communication complexity based on the idea of universe reduction, i.e., choosing a small committee to manage the process [35].

If the users are malicious, it is much more difficult to achieve good communication complexity. For many years, the best known protocols had exponential communication complexity [2], [3]. A key improvement was made by Srikanth and Toueg [4], who developed an efficient algorithm with polynomial communication complexity.

While the preceding algorithms generally assumed a synchronous network, there was also significant work on consensus in asynchronous and partially synchronous networks. In a seminal paper, Castro and Liskov [1] implemented a replicated state machine based on Byzantine agreement that is often described as the first practical Byzantine fault-tolerant system. It led to a floor of work on Byzantine agreement, with many attempts to improve the efficiency and trade-off different aspects of the performance (e.g., [36]–[39]).

Despite these significant efforts, these protocols remained bandwidth limited, typically requiring $\Theta(n^2)$ messages (or more). Over the last several years, there has been an exciting breakthrough [5]–[8], reducing the communication complexity of agreement to $O(n \cdot \texttt{polylog}(n))$ for a system with n players. The basic idea is to first solve *almost everywhere* agreement, convincing most of the users to agree on a single value. Then, a secondary *almost-everywhere-to-everywhere* protocol is used to spread the information to the remaining laggards.

The basic idea (for almost everywhere agreement) is to assign the users to committees, and organize the committees into a tree. They assign the users to committees in such a way as to ensure that almost all the committees have a majority of honest users (using an adapted version of Feige's leader election algorithm [40]). A leader is elected at the root of the tree, and then information is propagated down the tree to everyone. Later, to cope with an adaptive adversary, secret sharing and additional information hiding techniques are needed [41]. However, these papers do not discuss the issues of splitting processors securely using computational power assumption.

The algorithm proposed in this paper is not directly comparable to these newer communication-efficient protocols: SCP is simpler and potentially more communication efficient, but it relies on a stronger underlying network model. Its key advantage is in using computational power to tune the parallelization of processors, yet detaining security on bounded computational assumption.

IX. Acknowledgment

We thank Dawn Song, Elaine Shi, Andrew Miller and Xiao Liu for useful discussions and feedback on the early version of the paper. This work is supported by the Ministry of Education, Singapore under under grant R-252-000-560-112 and MOE ARC-2 grant MOE2014-T2-1-157. All opinions expressed in this work are solely those of the authors.

References

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999.

[2] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[3] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[4] Sam Toueg, Kenneth J. Perry, and T. K. Srikanth. Fast distributed agreement (preliminary version). In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, pages 87–101. ACM, 1985.

[5] V. King, J. Saia, V. Sanwalani, and E. Vee. Towards secure and scalable computation in peer-to-peer networks. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 87–98, 2006.

[6] Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In *Proceedings of the 23rd International Conference on Distributed Computing*, pages 464–478. Springer-Verlag, 2009.

[7] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In MarcosK. Aguilera, Haifeng Yu, Nitin H. Vaidya, Vikram Srinivasan, and RomitRoy Choudhury, editors, *Distributed Computing and Networking*, volume 6522 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2011.

[8] Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, pages 57–64. ACM, 2013.

[9] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.

[10] Bitcoin wiki. Scalability. https://en.bitcoin.it/wiki/Scalability, 2015.

[11] Jeff Garzik. Making decentralized economic policy. http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf, 2015.

[12] Gavin Andresen. Bitcoin improvement proposal 101. https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki, 2015.

[13] Jeff Garzik. Bitcoin improvement proposal 102. https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki, 2015.

[14] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert van Renesse. Bitcoin-ng: A scalable blockchain protocol. http://arxiv.org/abs/1510.02037, 2015.

[15] Ethereum Foundation. Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2014.

[16] Anonymous. Bitcoin scaling. Anonymous, 2015.

[17] Bitcoin client. https://github.com/bitcoin/bitcoin.

[18] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[19] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765, 2014. http://eprint.iacr.org/.

[20] Valerie King and Jared Saia. Scalable byzantine computation. *SIGACT News*, 41(3):89–104, sep 2010.

[21] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[22] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proceedings of the 4th International Conference on Peer-to-Peer Systems*, IPTPS'05, pages 13–23, Berlin, Heidelberg, 2005. Springer-Verlag.

[23] Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Self-healing deterministic expanders. *CoRR*, abs/1206.1522, 2012.

[24] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[25] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[26] Wikipedia. Merkle tree. https://en.wikipedia.org/wiki/Merkle_tree.

[27] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. Cryptology ePrint Archive, Report 2015/702, 2015. http://eprint.iacr.org/.

[28] Sophia Yakoubov Conner Fromknecht, Dragos Velicanu. A decentralized public key infrastructure with identity retention. Cryptology ePrint Archive, Report 2014/803, 2014. http://eprint.iacr.org/.

[29] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. Cryptology ePrint Archive, Report 2013/622, 2013. http://eprint.iacr.org/.

[30] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881, 2013. http://eprint.iacr.org/.

[31] Thaddeus Dryja Joseph Poon. The bitcoin lightning network: Scalable off-chain instant payments. http://lightning.network/lightning-network-paper.pdf, 2015.

[32] Buteri Vitalik, Wood Gavin, Zamfir Vlad, Coleman Jeff, Wampler-Doty Matthew, and Cohn John. Notes on scalable blockchain protocols (version 0.3.2). https://github.com/vbuterin/scalability_paper/raw/master/scalability.pdf, 2015.

[33] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timon, , and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf, 2014.

[34] Gabriel Bracha. An $O(\log n)$ expected rounds randomized byzantine generals protocol. *J. ACM*, 34:910–920, October 1987.

[35] Seth Gilbert and Dariusz R. Kowalski. Distributed agreement with optimal communication complexity. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 965–977. Society for Industrial and Applied Mathematics, 2010.

[36] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, October 2006.

[37] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 153–168. USENIX Association, 2009.

[38] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4):7:1–7:39, January 2010.

[39] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Trans. Comput. Syst.*, 32(4):12:1–12:45, January 2015.

[40] U. Feige. Noncryptographic selection protocols. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 142–152, 1999.

[41] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58:18:1–18:24, July 2011.