A Secure Sharding Protocol For Open Blockchains

Loi Luu
National University of Singapore
loiluu@comp.nus.edu.sg

Viswesh Narayanan
National University of Singapore
visweshn@comp.nus.edu.sg

Chaodong Zheng
National University of Singapore
chaodong.zheng@comp.nus.edu.sg

Kunal Baweja
National University of Singapore
bawejaku@comp.nus.edu.sg

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

Prateek Saxena
National University of Singapore
prateeks@comp.nus.edu.sg

ABSTRACT

Cryptocurrencies, such as Bitcoin and 250 similar alt-coins, embody at their core a blockchain protocol — a mechanism for a distributed network of computational nodes to periodically agree on a set of new transactions. Designing a secure blockchain protocol relies on an open challenge in security, that of designing a *highly-scalable agreement* protocol open to manipulation by byzantine or arbitrarily malicious nodes. Bitcoin's blockchain agreement protocol exhibits security, but does not scale: it processes 3–7 transactions per second at present, irrespective of the available computation capacity at hand.

In this paper, we propose a new distributed agreement protocol for permission-less blockchains called ELASTICO. ELASTICO scales transaction rates almost linearly with available computation for mining: the more the computation power in the network, the higher the number of transaction blocks selected per unit time. ELASTICO is efficient in its network messages and tolerates byzantine adversaries of up to one-fourth of the total computational power. Technically, ELASTICO uniformly partitions or parallelizes the mining network (securely) into smaller committees, each of which processes a disjoint set of transactions (or "shards"). While sharding is common in non-byzantine settings, ELASTICO is the first candidate for a secure sharding protocol with presence of byzantine adversaries. Our scalability experiments on Amazon EC2 with up to $1,600$ nodes confirm ELASTICO's theoretical scaling properties.

1. INTRODUCTION

A blockchain is an append-only distributed database that stores a time-ordered set of facts, also known as transactions. Transactions are grouped into batches or "blocks" and form a cryptographic hash-chain, hence the name blockchain. In 2009, Bitcoin introduced the first blockchain protocol called Nakamoto consensus which underlies over 250 cryptocurrencies [1]. The blockchain protocol maintains the distributed database in a decentralized network, thus aiming to solve what we call as the *blockchain agreement* problem. Conceptually, the problem is to allow an arbitrary large network of several processors to agree on the blockchain state (identified by its cryptographic digest), under the assumption that the fraction of malicious processors is bounded by f $(0 \leq f < 1)$. Processors have no inherent identities, nor is there any trusted PKI infrastructure to establish identities for processors. Each processor can choose a set (*e.g.*, block) of transactions it wishes to commit to the blockchain; the goal of the protocol is to ensure that all honest processors agree on one set of transactions at the end of the protocol. The commit set is appended as a new block to the blockchain.

At a high level, the blockchain protocol in Bitcoin randomly selects one processor per epoch (say 10 minutes) which issues a proposal that everyone adopts, thus requiring only a single broadcast to reach agreement [1]. There may be temporary disagreement if two proposals occur at the same time; eventually, with very high probability, one proposal will be established by picking the longest blockchain. Nakamoto consensus uses a proof-of-work (PoW) mechanism to probabilistically elect the leader, ensuring a fair choice of leaders. In terms of scale, Bitcoin employs billions of CPUs worth of computational power today (by observable hashrates [2]), and is one of the largest completely decentralized systems of such scale.

Unfortunately, Bitcoin's transaction throughput does not scale well. The Bitcoin network consumes massive computational power and presently processes up to 7 transactions per second [3]. Other centralized fiat payment processing systems, like MasterCard or Visa are reported to processing $1,200$ to $56,000$ transactions per second [4, 5]. The demand from practical applications is 3 to 4 orders of magnitude higher. Modification to scale up existing protocol is a raging debate in the Bitcoin community [6–9]. Recent work shows that these proposals have fundamental scalability limits [10].

On the other hand, solutions which use classical Byzantine consensus protocols [11–14] do not work in an open environment like cryptocurrencies because of two fundamental challenges. First, many of these papers assume that the network nodes have pre-established identities or public-key infrastructure in place, which does not exist in open environments like Bitcoin. Second, practical byzantine consensus protocols such as PBFT [13] require at least a quadratic number of messages in the number of participants, thus they are bandwidth-limited — more network identities leads to *worse* performance. Network bandwidth limits the transaction throughputs for a network of even a few hundred nodes severely. This raises a fundamental question — are there any blockchain protocols that scale throughput linearly with the increase in the size of the network?

Problem & Approach. Our goal is to seek a protocol for the open, permissionless network wherein participating processors have no pre-established identities, and where the transaction throughput scales. We provide a new blockchain protocol called ELASTICO, which achieves a sweet spot between classical byzantine consensus

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978389

and Nakamoto consensus protocols. The key idea in our approach is to partition the network into smaller committees, each of which processes a disjoint set of transactions (or a "shard"). Specifically, the number of committees grows near linearly in the total computational power of the network. Each committee has a reasonably small number of members so they can run a classical byzantine consensus protocol to decide their agreed set of transactions in parallel. Sharding protocols are commonly used in distributed databases and in cloud infrastructure, wherein certain network infrastructure can be trusted (*e.g.*, see the commonly used two-phase commit protocol) or where the goal is to tolerate crash (non-byzantine) failures [15–17]. Note that several commercial and open-source blockchains do not target the permissionless (open) setting, and as a result, promise to scale by relying on trusted infrastructure [18–20] or by using federated identities [21, 22] (see Section 6). To our knowledge, we provide the first sharding protocol for permissionless blockchains tolerating a constant fraction of byzantine network nodes. This is a well-recognized open problem [10]. Our protocol makes the same assumptions as those implied in Bitcoin and other cryptocurrencies, and we provide security proofs for key invariants in our protocol.

Results. Without loss of generality, we assume that the network contains n processors which have equivalent computational power. ELASTICO exhibits almost linear scalability with computation capacity and does not require quadratic number of messages as the network grows. ELASTICO tolerates up to $f < n/4$ adaptive byzantine adversaries, where f and n are bounds on the adversarial and total computational power respectively. [1] 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.

From an efficiency perspective, our protocol shards the network into an almost linear number of committees that scales with computation capacity. Within each committee of size c (a few hundred) identities, we run a secure consensus protocol which has message complexity of $O(c^2)$ (best case) to $O(c^3)$ (worst case). Overall, this yields a message complexity of at most $O(nc^3)$, where messages are of constant size.

We implement ELASTICO based on the most popular client for Bitcoin [23]. Our implementation adds roughly $5,000$ C++ LoCs on top of Bitcoin. The throughput of our prototype scales near linearly with respect to available computation *i.e.*, $O(n/\log\log(n))$, when runs on our network simulation. With the same network implementation as in Bitcoin, the scale up (blocks per epoch) for 100, 200, 400, 800 and $1,600$ nodes with equal computational power [2] are as theoretical expectation, namely 1, 1.89, 3.61, 6.98 and 13.5 times respectively. Finally, ELASTICO's clean-slate design decouples the consensus from block-data broadcasts, hence the bandwidth spent by each node remains almost constant, regardless of the size of the network. Our simulations are necessarily on a smaller scale than Bitcoin; however, if we project our results to a full deployment to a network of Bitcoin's scale, we can expect a scale up of $10,000$ in the number of agreed values per epoch. This agreement throughput is 4 orders of magnitude larger than Bitcoin's.

Contributions. We claim the following contributions.

- To our knowledge, ELASTICO is the first secure candidate for a sharding protocol for open blockchains that tolerates byzantine adversaries. ELASTICO increases the blockchain's transaction throughput almost linearly with the computational power of the network.
- Our experiments on an idealized network simulation on Amazon EC2, ranging up to $1,600$ network nodes, confirm a near linear scalability for ELASTICO.

[1] Here, $1/4$ is an arbitrary constant bounded away from $1/3$, selected as such to yield reasonable constant parameters.
[2] each node is one Amazon EC2 vCPU

2. PROBLEM & CHALLENGES

2.1 Problem Definition

We formalize the problem of designing a secure sharding protocol for blockchains as follows. Let there be n identity-less processors having the same computational power, a fraction f of which are controlled by a byzantine adversary. The network accepts transactions per block, a transaction i in block j is represented by an integer $x_i^j \in \mathbb{Z}_N$. All processors have access to an externally-specified constraint function $\mathcal{C} : \mathbb{Z}_N \mapsto \{0, 1\}$ to determine the validity of each transaction. We seek a protocol Π run between the processors which outputs a set X which contains k separate "shards" or subsets $X_i = \{x_i^j\}$ $(1 \leq j \leq |X_i|)$ such that the following conditions hold:

- *Agreement*. Honest processors agree on X with a probability of at least $1 - 2^{-\lambda}$, for a given security parameter λ.
- *Validity*. The agreed shard X satisfies the specified constraint function $\mathcal{C}$, *i.e.*, $\forall i \in \{1..k\}, \forall x_i^j \in X_i, \mathcal{C}(x_i^j) = 1$.
- *Scalability*. The value of k grows near linearly with the size of the network (or n).
- *Efficiency*. The computation and bandwidth used per processor stays constant regardless of n and k.

Our goal is to split the network into multiple committees, each processes a separate set of transactions (*e.g.*, X_i) called a *shard*. The number of shards (k) grows near linearly on the size of the network. The *efficiency* property represents the sharding advantage, where the cost is localized within a committee. Once the network agrees on the set X, it can create a cryptographic digest of X and form a hash-chain with previous agreed sets in the previous runs of Π. This serves as a distributed ledger of facts or transactions.

We point out that the agreement property in our problem is a relaxation of the original byzantine consensus problem [11, 12]. 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 high probability, rather than be in exact agreement. 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 the inputs of honest processors. In the blockchain problem, validity can be checked externally — each honest processor can check if the agreed value satisfies an externally-specified constraint $\mathcal{C}$, and accept a solution only if so.

Remark. Notice that this problem does not directly guarantee a double spending check (a problem in cryptocurrency [1]), but implementing such a check is possible given the agreement on the transaction set which satisfies the constraints specified in $\mathcal{C}$. In Appendix 10.2, we describe how one might build a new cryptocurrency like Bitcoin based on ELASTICO with all validity checks.

Threat Model. We consider the threat model of a *static, 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 run Π. 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 can setup point-to-point communication links between them, and the adversary has full information about the messages transmitted on all links.

Security Assumptions. We make two assumptions about the underlying network overlay layer as in Bitcoin. Explicitly, (a) the network graph between honest processors is connected and (b) the communication channel between honest processors is synchronous, *i.e.*, once an honest user broadcasts any message, other honest processors will receive it within a known bounded delay of δ_t seconds. Note that such timing and connectivity assumptions are implicit and necessary even in Bitcoin; otherwise, byzantine nodes can delay blocks significantly (simulating a gain in computation power) or worse — a fraction of the network can be "eclipsed" by the adversary. Attacks targeting our assumptions will apply to Bitcoin too. However, such assumptions can be achieved with the right design of the underlying network topology [24, 25] — an orthogonal problem of active research. On the other hand, we do not make any assumption about a secure broadcast channel or a PKI system or access to external source of randomness. That means, in our threat model, the malicious processors can drop or modify messages, send different messages to honest processors. We show in Section 4 that the most that an adversary can do is to delay our consensus process.

Further, we assume that we know the upper bounds on the true computation power n (say in Gigahash/sec), and that f is less than $1/4$. 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 (just as in Bitcoin today). For this work, we assume such information is externally available. We further assume that nodes are reliable during protocol runs, and failed nodes are counted in the f malicious fraction. Second, we assume that the total computation power of the byzantine adversaries is still confined to standard cryptographic assumptions of probabilistic polynomial-time adversaries. Lastly, we assume there exists a random oracle $\mathtt{H} : \{0,1\}^* \mapsto \{0,1\}^\gamma$ which outputs γ random bits given any input string.

2.2 Challenges

Sharding in a permission-less blockchain with the presence of byzantine adversary is a well-recognized open problem [10] due to many challenges. First, processors have no inherent identities or external PKI to trust. A malicious processor can thus simulate many virtual processors, thereby creating a large set of *sybils* [26, 27]. Thus, the protocol must prescribe some mechanism to allow processors to establish their identities, yet limiting the number of sybil identities created by malicious processors.

Once identities are established, the next challenge is to run a sharding protocol among the identities with a fraction f of them are byzantine. Our goal is to *uniformly* split all identities into several committees with a condition that each committee has the majority as honest with high probability. Such a protocol becomes straight-forward if one assumes a shared random coin to do the sharding properly [15–17]. However, achieving a good randomness in a distributed network is a known hard problem. The best solution to date can only tolerate up to $1/6$ fraction of malicious, with excessive message complexity [28]. Our protocol makes no such assumption.

Third, we must ensure that an adaptive adversary observing all the protocol runs, does not gain significant advantage in biasing its operations or creating sybil identities. Our protocol must tolerate variable rate of identity creation and inconsistency in views of committee members (*i.e.*, they may not agree on who are in the same committee) because of both byzantine failures and network delays in real networks (as in our threat model).

3. ELASTICO DESIGN

In this section, we present ELASTICO. For the rest of the paper, unless otherwise stated, if some probability p is negligible, it means it happens with probability at most $\mathcal{O}(1/2^\lambda)$ for some security parameter λ. Similarly, if some event happens with high probability (*w.h.p*), it happens with probability of at least $1 - \mathcal{O}(1/2^\lambda)$. If some event happens with non-negligible probability, it happens with probability greater than $\mathcal{O}(1/2^\lambda)$.

3.1 Solution Overview

The algorithm proceeds in *epochs*, each of which decides on a set of values $X = \bigcup_{i=1}^{2^s} X_i$ where 2^s is the number of subsets X_i. In this description, we describe the steps taken during one epoch.

The key idea is to automatically parallelize the available computation power, dividing it into several smaller committees, each processes a disjoint set of transactions (or shards). The number of committees grows proportionally to the total computation power in the network. All committees, each of which has a small constant number c of members, run a classical byzantine consensus protocol internally to agree on one value. A final committee called the consensus committee is responsible for combining the shards selected by other 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, which have a bounded bias. These random strings are used in the subsequent epoch as a source of randomness—ensuring that the adversary cannot use observations from previous epoch to simulate any non-negligible gain in computational power.

In each epoch, processors execute the following 5 steps:

1. *Identity Establishment and Committee Formation.* Each processor locally generates an identity consisting of a public key, an IP address and a proof-of-work (PoW) solution [1]. The processor must solve a PoW puzzle which has publicly verifiable solutions to generate the final component of the identity. A PoW solution also allows others to verify and accept the identity of a processor. Because solving PoW requires computation, the number of identities that the malicious processors can create is limited by the fraction of malicious computational power f. Each processor will then be assigned to a committee corresponding to its established identity.

2. *Overlay Setup for Committees.* In this step, processors communicate to discover identities of other processors in their committee. The overlay of a committee is a fully-connected subgraph containing all the committee members. A naïve solution is for every processor to broadcast its identity and committee membership to everyone; however, this solution will result in $\mathcal{O}(n^2)$ messages, which is not scalable. We provide a simple solution that requires a small number of broadcasts, *i.e.*, $\mathcal{O}(nc)$, after which identities in same committees can quickly identify each other.

3. *Intra-committee Consensus.* Processors run a standard byzantine agreement protocol (*e.g.*, PBFT [13]) within their committee to agree on a single set of transactions (or a shard).

There exist simple solutions to guarantee that all committees propose disjoint shards, *e.g.*, each committee works on a separate shard of transactions based on their committee ID. Each committee sends the selected shard, signed by enough members (*i.e.*, $c/2+1$), to a designated final committee.

4. *Final Consensus Broadcast.* The final committee computes a final value from all the values received from other committees. To do so, members in final committee run a byzantine consensus protocol to agree on the final result and broadcast it to the network.

5. *Epoch Randomness Generation.* The final committee runs a distributed commit-and-xor scheme to generate an exponential biased, but bounded, set of random values. The random values are broadcast to the network and used in the PoW in the next epoch.

Parameters. Throughout this paper, we use the following notation: n is the total number of identities that we expect to be generated in an epoch, $f = 1/4$ is the fraction of computational power controlled by malicious users, 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 $n = c \cdot 2^s$. Thus, ELASTICO scales up almost linearly because the expected number of PoW solutions to have each committee has at least c members is $\mathcal{O}(n \log s)$ (or $\mathcal{O}(n \log \log (n/c))$). For a concrete analysis, we refer readers to Appendix 10.1. Note that picking smaller s leads to lower latency and bandwidth consumption, which allows one to tune the network consumption. In addition, the size of committee c is determined by the security parameter λ and the expected network delay δ_t.

Efficiency. Our protocol requires $\mathcal{O}(c)$ broadcasts to the whole network (steps 2, 4 and 5). Each such broadcast can be implemented in $\mathcal{O}(n)$ message transmissions. Steps 3, 4 and 5 require at most c round of c^2 multicasts for each committee of size c. Therefore, the total number of messages is $\mathcal{O}(nc + nc^3)$ or roughly $\mathcal{O}(n)$ if we consider c to be a small constant compared to n.

Security. In each epoch, for $f = 1/4$, our protocol guarantees the following security properties S1–S5, the proofs of which are presented in Section 4.

- S1. Given a security parameter λ, there exists n_0 such that $\forall n' \geq n_0$, among the first n' identities created, at most $1/3$ are malicious *w.h.p.* The gap between f and $1/3$ accounts for the variance in the number of PoW solutions found by the adversary. Our committee size c is then chosen as based on the value of n_0 (*e.g.*, $c \geq n_0$) such that every committee has at most a fraction $1/3$ of malicious identities.

- S2. After Step 2, all committee members have their own view of at least c members in the committee. There may be discrepancies between two views due to network latency and byzantine behaviors. This discrepancy, however, is bounded by $c/3$ *w.h.p* and all honest members have identities of other honest members in their views. Further, the number of unique identities in all views is bounded by $3c/2$, of which at most $1/3$ fraction are malicious *w.h.p.*

- S3. For each committee, Step 3 yields a consensus on the set of transactions X_i proposed by members in the committee. The chosen X_i is signed by at least $c/2+1$ of the identities on the committee. This ensures at least one honest member has verified and agreed on the value.

- S4. Step 4 yields a valid set $X = \bigcup_{i=1}^{2^s} X_i$ which combines all proposed sets X_i from other committees. X is also signed by at least $c/2+1$ of the members in the final committee.

- S5. Step 5 will yield a set of random r-bit values with sufficient randomness. Explicitly, the attacker can predict the random value, given r is large enough, with a negligible probability in λ.

Note that we select $f = 1/4$ in order to achieve a practical value of committee size. Theoretically, ELASTICO can work with any f less than $1/3$ by increasing the committee size c accordingly to f. The $1/3$ bound is because we need to run a consensus protocol (*e.g.*, PBFT [13]) at every committee in Step 3, which can tolerate at most $1/3$ fraction of malicious committee members.

3.2 Identity Setup and Committee Formation

First, each processor locally chooses its own identity of the form $(\texttt{IP}, \texttt{PK})$, which are public key and IP address respectively for the authenticated communication later. In order for the network to accept an identity, the processor must find a PoW solution corresponding to its chosen identity. As a "seed" for the PoW, we need a public random string `epochRandomness` generated at the end of the previous epoch to ensure that the PoW was not precomputed. We discuss how this is generated and verified in Section 3.6. Assume, for now, that `epochRandomness` is a public random string generated in the previous epoch. Specifically, each processor locally searches for a valid `nonce` that satisfies the following constraint:

$$O = \texttt{H}(\texttt{epochRandomness}||\texttt{IP}||\texttt{PK}||\texttt{nonce}) \leq 2^{\gamma - D}.$$

D is a predefined parameter in the network which determines how much work a processor has to do to solves a PoW [3]. Note that one can use other mechanisms like Proof of Stake [29], Proof of Space [30,31] instead of PoW to establish identities for processors.

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 PoW to achieve these goals. Specifically, the last s bits of O specifies which (s-bit) committee id that the processor belongs to. Each committee will process a separate set of values (*e.g.*, a shard) based on this s-bit ID.

All processors know `epochRandomness` and choose their identity IP and PK privately. For any choice of `nonce`, `H` produces a γ-bit random output. The probability that a single invocation of `H` satisfies the constraint for a randomly chosen `nonce` is thus $p = 2^{-D}$. No efficient adversary can find a `nonce` that satisfies the constraint with non-negligible probability better than p by the cryptographic pre-image resistance assumption. We later prove in Lemma 2 (Section 4) that among the first n' identities, at most $1/3 \cdot n$ of identities are created by byzantine processors *w.h.p.*

For establishing S1, 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 on average it requires $\mathcal{O}(n \log s)$ PoW solutions to have each of 2^s committees has at least c members (see Appendix 10.1). This is why the scale up factor is almost linear.

[3] For example, $D = 20$ means O has at least 20 leading zeros.

Byzantine adversaries can choose not to 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 committees, and D, which determines the difficulty of the PoW. These are discussed in Section 5 with our experiments.

3.3 Overlay Setup for Committees

Once identities and their committees are established, committee members need a way to establish point-to-point connections with their committee peers. A naıve solution is to ask every processor to broadcast its identity and committee membership to everyone; however, this solution will result in $\mathcal{O}(n^2)$ messages, which is not scalable. Another challenge is that identities are established through a PoW, which is a probabilistic process that occurs over time: new identities are continuously being created at some rate. Ideally, 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 byzantine agreement protocol here which tolerates up to $1/3$ fraction of malicious identities. However, this would yield BFT protocol running over the entire network without any parallelization (*e.g.*, $\mathcal{O}(n^3)$ cost in the worst case). Here we show something more efficient which has $\mathcal{O}(nc)$ message complexity.

To reduce the number of broadcast messages, we have a special committee of size c to serve as a set of "directories." All identities can find their committee peers by contacting the directory committee and then set up point-to-point links. Further, we allow committee members (including directory members) to have different views of the member set, a challenge that most previous BFT protocols do not face. Our protocol can tolerate this discrepancy and show that i) all honest members have others' identities in their view; ii) the difference is bounded by $c/2$ as in S2. Our algorithm to setup the overlay for committee is depicted in Algorithm 1.

More specifically, the directory committee is simply a committee of the first c identities. During step one, if a processor finds a valid solution for PoW, and it has not seen c identities, then the processor will broadcast this identity to the whole network. On the other hand, during step one, whenever a processor finds a valid solution for PoW for an identity, the processor will inform all directory members (Line 17). Each processor informs only the first c directories that it sees in the network. Note that each processor can have its own view of who are the first c directories.

In this way, directory members keep track of the committee membership announcements. Once each committee contains at least c identities each, directory members multicast the list of committee members to each committee member (Line 33). To reduce the number of messages, a non-directory committee member only receives the list of members in its own committee. Notice that directory members do not have to agree on the same set of members for a committee. Each directory member can decide independently which members are in a given committee.

For committee members, each will receive at least $2c/3$ lists of c committee members (from at least $2c/3$ honest directories — due to S2). A malicious directory may not send any list to committee members. Worse, malicious directories may send a bad member list to committee members (*i.e.*, a list which favors malicious identities). To prevent that, a committee member takes the union of all the identities that it receives to create a view of at least c committee members (including itself) (Line 19). It is still possible that committee members have different member sets. We analyze both the sources and show that the discrepancy is bounded by at most $c/2$ members in Lemma 3, which proves S2.

Algorithm 1 Algorithm to form the directory committee and setup overlay for other committees.

Input: c: committee size; $k = 2^s$: number of committees
Output: Every committee member receives at least c members of its committee

```
1: procedure FormCommittee                          ▷ Done by everyone
2:     curDirectories ← empty
3:     myPoW ← empty
4:     while True do
5:         w ← ReceivePoW()                         ▷ Receive a PoW solution
6:         if len(curDirectories) < c then
7:             curDirectories.append(w)
8:         end if
9:         w ← SolvePoW()                           ▷ Find a PoW solution
10:        if len(curDirectories) < c then
11:            curDirectories.append(w)
12:            Execute RunAsDirectory()
13:            BroadcastToNetwork(w)
14:        else
15:            myPoW.append(w)
16:            Send(curDirectories, w)
17:        end if
18:        if lViews ← GetViewsFromDirectories() then
19:            v ← union(lViews)
20:            return v                             ▷ Move to next step in the protocol
21:        end if
22:    end while
23: end procedure
24: procedure RunAsDirectory                        ▷ Done by directories
25:    while True do
26:        w ← ReceivePoW()                         ▷ Accept all PoWs in last round
27:        i ← GetCommitteeNo(w)
28:        if len(commList[i]) < c then
29:            commList[i].append(w)
30:        end if
31:        if ∀i ≤ k, len(commList[i]) ≥ c then
32:            for i = 0, i < k, i ← i + 1 do
33:                MulticastCommittee(i)            ▷ Send commList[i] to members of committee i
34:            end for
35:            return True
36:        end if
37:    end while
38: end procedure
```

Overall, security follows from two arguments. First, the most that a malicious directory can do is to not send honest identities and favor malicious identities in a committee. It is because malicious directories cannot create new identities or change committee assignment of new identities due to the PoW. Further, committee members decide who are in their committees based on all views received from directories. As per S1, among the c directories, at least $2c/3$ are honest. Thus honest members in a committee will know all other honest members.

3.4 Intra-committee Consensus

Once a committee is established, the protocol to agree on a set of transactions can reuse any existing authenticated byzantine agreement protocol [32, 33]. As shown in S3, the union of all views will have at most $3c/2$ members, and at most $1/3$ of them are malicious. In the worst case, we can assume that each committee has $3c/2$ members, of which at most $1/3$ of them are malicious. Since each honest member knows all other honest members (due to S2), a member can treat all identities which are not in its view as malicious. Hence, existing byzantine agreement protocols will work securely in our setting. For example, we can use the PolyByz algorithm [32], or PBFT [13], or the more recent SybilSensus algorithm [33].

Once an agreement is reached, the selected transaction set is signed by at least $c/2+1$ signatures to guarantee some honest member has verified and accepted the value. Each committee member then sends the signed value along with the signatures to the final committee (using the directory, again, to acquire the list of final committee members). The final committee can verify that a certain value is the selected one by checking that it has sufficient signatures. Note that this final value can be a small cryptographic digest (*e.g.*, a Merkle hash root) representing the set X_i of values x_i^j (or transactions) that the committee agrees on.

3.5 Final Consensus Broadcast

The next step of the protocol is to merge the agreed values of committees and to create a cryptographic digest (a digital signature) of the final agreed result. A final committee (which includes all members 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 runs the same intra-committee algorithm described previously. This step obtains a verifiable signature by at least $c/2+1$ members of the final committee, which the entire network can verify upon broadcast.

3.6 Generating Epoch Randomness

In the final step of the protocol, the final committee (or *consensus committee*) also generates a set of random strings for use in the next epoch. For the first epoch, a previous approach can be used to generate the `epochRandomness` [28]. However, this solution tolerates at most $1/6$ fraction of malicious members and only works for a small network since it requires excessive message complexity.

Our solution for this step consists of two main phases. In the first phase, each member of the final committee chooses a r-bit random string R_i and sends a hash $\mathtt{H}(R_i)$ to everyone in the committee. The final committee then runs an interactive consistency protocol to agree on a single set of hash values $\mathbb{S}$ [11]. The final committee members will broadcast $\mathbb{S}$, along with the final set of X in Section 3.5 to everyone in the network. This set $\mathbb{S}$ contains at least $2c/3$ hash values and serves as a commitment to the random strings. This first phase can be done with the previous step in Section 3.5, but for simplicity, we describe it separately here.

In the second phase, each member of the final committee broadcasts a message containing the random string R_i itself to everyone (*i.e.*, not just to the final committee). This phase starts only *after* the agreement of $\mathbb{S}$ is done, *i.e.*, having $2c/3$ signatures on $\mathbb{S}$. This is to guarantee that honest members release their commitments only after they are sure that the committee has agreed on $\mathbb{S}$ and the adversary cannot change its commitment.

At this point, each user in the system has received at least $2c/3$ and at most $3c/2$ pairs of R_i and $\mathtt{H}(R_i)$ from members of the final committee: the honest members follow the protocol, while the malicious users may choose to not release their commitments. Users discard any random strings R_i that do not match the commitments $\mathtt{H}(R_i)$.

For the purpose of the next epoch, each user takes an XOR of any $c/2+1$ random strings R_i that it receives. Note that users may select different R_i. We consider the XOR of *any* subset of $c/2+1$ valid random strings sent by the final committee to be a valid `epochRandomness`. Recall that these random strings are used as the seed for the PoW in the next epoch. In order for others to verify that a PoW is valid, the user should attach to their PoW solution the set of $c/2+1$ strings R_i used to generate the seed. Any other user can then verify if these $c/2+1$ random strings match the commitments in $\mathbb{S}$.

Step	Potential problems	State-ment	Lemma
1	P1. PoW variance - Too many malicious identities - Bias the committee distribution	S1	Lem. 2
2	P2. Inconsistency between views - Adversaries withhold identities - Network latency	S2	Lem. 3
3, 4	P3. Cannot reach consensus	S3, S4	Lem. 4
5	P4. Bias the random strings	S5	Lem. 6

Table 1: Sources of byzantine advantage in each step of ELASTICO and the corresponding supporting security statements and lemmas which resolve them.

A formal proof which shows the strings generated from this process are sufficiently random is presented in Section 4. Here we provide an intuition why this works. Any combination of $c/2+1$ strings R_i must share at least one R_i from an honest member, and that value is not known to the adversaries before everyone has agreed on the set of commitments $\mathbb{S}$. Thus, the value R produced by XOR-ing $c/2+1$ strings R_i is perfectly random. The adversary, however, can choose which $c/2+1$ values he uses to bias R to be in a set $\mathbb{S}_a$ of his choice. We show that this probability is bounded by $1/2^{r-\lambda-c+\log(c)/2}$. Thus, if we pick r large enough (*e.g.*, $r > 2\lambda + c - \log(c)/2$), the advantage of the adversary is negligible in λ.

4. SECURITY ANALYSIS

In this section, we provide security analysis for how ELASTICO prevents potential threats and works securely. We also discuss how byzantine adversary gain no significant advantage.

We begin by clarifying several assumptions. First, we assume that the network is partially synchronous: any message can reach the destination with a maximum delay of δ_t, thus network protocol can be assumed to happen in rounds, each lasts for δ_t time. All our claims for an epoch depend on there being sufficiently random strings generated in the previous epoch.

Definition 1 (Good randomness). *We say that an epoch has a good randomness if:*

(i) every user has a publicly random string of r bits, verifiably generated in the previous epoch;

(ii) no user has access to such a verifiable random string more than δ_t prior to the beginning of the epoch;

(iii) malicious users can bias the randomness with negligible probability.

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

Lemma 2 (S1: Good Majority). *In every epoch with good randomness, for every sufficiently large integer $n' \geq n_0$: among the first n' identities created, at most $n'/3 - 1$ are controlled by the adversary w.h.p.*

Proof. If all the users start at the same time, each solution generated has a probability $1 - f = 3/4$ of being 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 $X = \sum_{i=1}^{n'} X_i$. Then, X follows a binomial distribution.

Thus we have:

$$\Pr\left[X \leq 2n'/3\right] = \sum_{k=0}^{\lceil 2n'/3 \rceil} \Pr[X = k]$$
$$= \sum_{k=0}^{\lceil 2n'/3 \rceil} \binom{n'}{k} f^{n'-k}(1-f)^k$$

This probability decreases exponentially in n'. Given a security parameter λ, we can find n_0 such that $\Pr\left[X \leq 2n'/3\right] \leq 2^{-\lambda}, \forall n' \geq n_0$. □

The committee size c is at least n_0 to guarantee that the fraction of malicious members in a committee is bounded by $1/3$, with regard to the security parameter λ. The value of n_0 depends on the security parameter λ. For example, if $\lambda = 20$, or the probability that something bad happens is once every 1 million epochs, we have $n_0 \approx 600$.

Lemma 3 (**S2**: Good views with bounded inconsistency)**.** *In every epoch with good randomness, if the directory size is c, then for each committee, with high probability, we guarantee the following properties.*

(i) Each member has their own view of who are in the committee. Two views of two honest members differ by at most $1/3$ of the committee size;

(ii) All honest members have identities of other honest members in their views;

(iii) The total number of unique identities in all views is at most $3c/2$ of which less than $1/3$ fraction are malicious.

Proof sketch. The inconsistency between views of member set is because of two reasons: network latency and byzantine behaviors. We calculate the difference in views caused by each source.

Since the network delay and the probability in finding a PoW solution, *e.g.*, it is hard to tell who are the last committee members. Since honest directories accept all valid PoWs in the last round (before the committee gets filled by at least c members), all honest committee members will have other honest identities in their view. This leaves the malicious behaviors the main source of the discrepancy.

Malicious processors can decide to withhold their identities and only publish them in some particular time to cause the maximum discrepancy between views of member set of honest members. However, honest members always share the same view of honest identities, thus the maximum discrepancy that an adversary can cause is the number of identities that he/she can create. As per Lemma 2, this amount is always less than $1/3$ of the total number of committee members *w.h.p.*

We next prove that the maximum number of identities in all views is $3c/2$ and less than $c/2$ identities are malicious. An adversary can wait until right before the last identity in a committee is found to publish all his/her identities. According to Lemma 2, the maximum number of malicious identities can be created before honest processors find all c identities is $c/2$. Thus, the total number of identities is bounded by $3c/2$ and at most $c/2$ identities are malicious *w.h.p.* □

We now show S3, S4 which argue that a committee (including the final committee and other commitees) correctly decides a single value (or a set of transactions X_i).

Lemma 4 (**S3, S4**: Consensus)**.** *In every epoch with good randomness, the honest members agree on a unique set X_i with at least $c/2 + 1$ signatures, with high probability.*

Proof. Although committee members have different views of member set, we can tolerate the inconsistency by considering a broader committee of size up to $3c/2$. As we have established above, the fraction of malicious identities is bounded by $1/3$ in any view, and honest members have other honest members in their view. Thus, any Byzantine consensus protocol (*e.g.*, POLYBYZ, PBFT) which can tolerate up to $1/3$ malicious fraction can guarantee agreement in the committee, *i.e.*, only one value selected.

Since our network is synchronous, the liveness property is always achieved in these consensus protocols, *i.e.*, the committee always reaches consensus in polynomial time. The committee then collects enough number of signatures (*e.g.*, $c/2 + 1$) to guarantee that at least one honest member has signed the chosen value. □

Lastly, we show Lemma 5 and S5, *i.e.*, the shared randomness generated is sufficiently random. Specifically, we derive the following definition from the standard definitions of bias [28].

Definition 1. Let $F : \{R_0, R_1, .., R_c\} \mapsto \{0,1\}^r$, computed over the set of r-bit input strings. For all choices of set $S \subseteq \{0,1\}^r$, let $E_F(S)$ be the expected number of R values that the adversary can generate such that $R \in S$. Assuming the adversary controls some of the inputs, the bias $\beta(F)$ is defined as:

$$\beta(F) = \log\left(\max_{S \subseteq \{0,1\}^r} \max\left\{\frac{E_F(S)}{E(S)}, \frac{E(S)}{E_F(S)}\right\}\right),$$

in which $E(S) = |S|/2^r$ — the value of $E_F(S)$ if the adversary does not control any inputs.

Lemma 5 (Bounded Bias to Randomness)**.** *Given a set of $c > 3$ random r-bit strings of which at most $\lfloor c/2 \rfloor$ strings are known and generated by the adversary, the computation of $R = \oplus R_i$ has a bias bounded by $c - \log(c)/2$.*

Proof. We define $F : \{R_0, R_1, .., R_c\} \mapsto \{0,1\}^r$ as a function which selects $c/2 + 1$ strings R_i from c strings and compute $R = \oplus R_i$. The computation of R can be decomposed into $F = F_h \oplus F_d$, in which F_d is computed over the set of inputs controlled by the adversary, and F_h is computed over the set of inputs R_h from honest users which the adversary does not know.

The bias of F_d, by definition of bias is $c - \log(c)/2$. This follows from the fact that the adversary can pick $\binom{c}{c/2}$ combinations of $c/2$ strings R_i to decide R_d. This number has an upper bound of $2^{c-\log(c)/2}$ computed based on Stirling's approximation. Thus, we have

$$\beta(F_d) < c - \log(c)/2.$$

On the other hand, R_h is not generated by the adversary, so bias of F_h is 0 because $E_{F_h}(S) = E(S)$. By composition, the total bias is $c - \log(c)/2$.

□

Lemma 6 (**S5**: Good Randomness)**.** *In every epoch with good randomness, with high probability, at the end of the epoch every user computes a random string of r bits that:*

(i) can be predicted with a negligible probability;

(ii) can be verified as validly generated in that epoch.

Moreover, no user knows any of the random strings until at least one round 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 3, there are at least $2c/3$ honest members in the final committee *w.h.p.* Thus, each user in the network receives a set

$\mathbb{R}$ having from $2c/3$ to $3c/2$ random strings from members in the final committee in the last round of the epoch. The user's random string R is generated by XOR–ing any $c/2+1$ strings in $\mathbb{R}$.

Since there are at most $c/2$ malicious members in the final committee, at least one random string R_i used by malicious users originated at an honest user. Further, this R_i is not known by the malicious users before the final committee has agreed on the set $\mathbb{S}$. Thus, the r-bit string R generated by XOR–ing any $c/2+1$ is perfectly random. However, the attacker can still pick any of his $c/2$ values R_i to bias his random string R. As per Lemma 5, the attacker's bias to R is bounded by $c - \log(c)/2$. Thus if we pick r large enough, *e.g.*, $r > \mathcal{O}(\lambda) + c - \log(c)/2$, the probability that the attacker can guess R is negligible.

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 ELASTICO protocol follows from the following facts. We omit a theorem and its formal proof, which is by induction and invokes Lemma 2–Lemma 6 on different cases.

Claim 7. *For every epoch $i \geq 1$, with high probability, the following properties hold:*

(i) 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;

(ii) 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

(iii) at the end of the epoch, each user has a publicly verifiable random bit string of length r which has sufficient randomness (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 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 [34]; the argument here is similar.)

5. IMPLEMENTATION & EVALUATION

We implement ELASTICO and empirically evaluate the scalability of ELASTICO and previous solutions. The goals of our evaluation are twofold. We first measure the scalability and efficiency of ELASTICO when the network size increases. We aim to establish that the performance of ELASTICO matches its theoretical analysis. The second goal is to compare ELASTICO to other related consensus protocols including Bitcoin [1], Bitcoin-NG [9] and PBFT [13].

5.1 Implementation

We implemented all components of ELASTICO based on the most popular Bitcoin client version v0.12.1 [23]. Our implementation is in C++ and has roughly $4,000$ LoC above that of Bitcoin's code base. We choose PBFT [13] as the consensus protocol for committees in ELASTICO. Since there was no well-maintained open source implementation of PBFT when we started, we built a full-fledged implementation of our own, and the implementation may be of independent interest to the community. We plan to release ELASTICO for public use.

Recall that once the committee formation is complete, a normal committee runs one instance of PBFT protocol to agree on a data block of size 1 MB. The final committee (or consensus committee) runs two consecutive instances of PBFT protocol to agree on one data block and the final block which aggregates all data blocks from other committees. Thus, nodes in the consensus committee bear more cost than in normal committees.

Network implementation. We reuse the network implementation of Bitcoin for our peer-to-peer layer in the overall network. The communications within committee are handled separately, in which nodes in a committee form their own peer-to-peer overlay network. Messages passed within a committee are not sent to nodes in other committees. The choice of using peer-to-peer instead of point-to-point communications within a committee is because the latter requires much more resource for a node to open a socket connection to each of 100 committee nodes.

Rate-limited mining. We artificially limit the mining rate to allow testing ELASTICO with large scale experiments on a the public EC2 infrastructure (which is expensive for our scale of testing). In order to control the exact fraction of malicious computational power, we limit the number of hash operations (*i.e.*,SHA2) that a node (*i.e.*, a processor) can perform to 1 operation per second. Each hash output has a probability $1/600$ of finding a valid PoW solution. Thus, on average it takes a processor 600 seconds to establish its identity. Note that we still faithfully simulate the mining process in Bitcoin, since we use the same implementation for mining.

5.2 ELASTICO's Scalability

Experimental Setup. We run several experiments with different settings on Amazon EC2 to measure the scalability of ELASTICO. We vary the number of nodes in the network from 100 to $1,600$, using up to 800 `c4.large` EC2 instances in two different regions including Oregon and California. Each EC2 instance is shared by two nodes, has 2 Amazon vCPUs and 3.75 GB of memory. Thus, increasing the number of nodes in the network will increase the computation power accordingly. We fix our committee size at $c = 100$, to limit the performance overheads due to PBFT as we discuss in Section 5.3.

Scalability of ELASTICO. We start with a network of 100 nodes then double the network size 4 times, raising to $1,600$ nodes in the last setting. We quantify the number of blocks created in each epoch, time to reach consensus, number of PoWs required to fill all committees. The results are plotted in Figure 1. All numbers are averaged after 10 epochs.

Figure 1 shows that ELASTICO scales up the block throughput almost linear to the size of the network. The number of blocks per epoch increases linearly (from 1 to 16) as we increase more nodes to the network (100 to $1,600$ accordingly). However, the epoch time is longer (*e.g.*, 600 seconds in 100 nodes to 711 seconds in $1,600$ nodes) since it requires relatively more time to find enough PoWs to fill up all committees when the number of committees increases. In addition, we observe that the time to reach consensus on blocks once the committee formation is complete remains almost constant, regardless of the network size. For example, the latencies to reach consensus are 103 and 110 seconds when the network sizes are 400 and 800 nodes respectively. In summary, our experi-

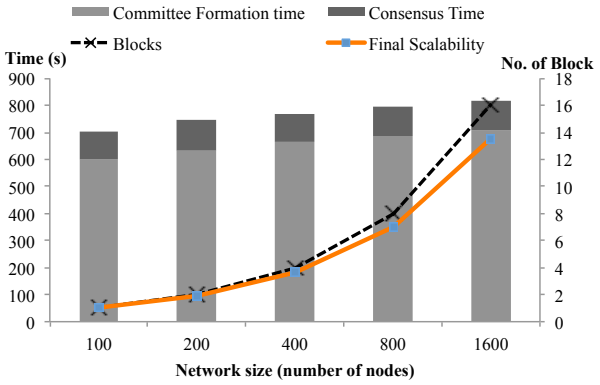

Figure 1: ELASTICO scales up the throughput nearly linearly in the computation capacity of the network.

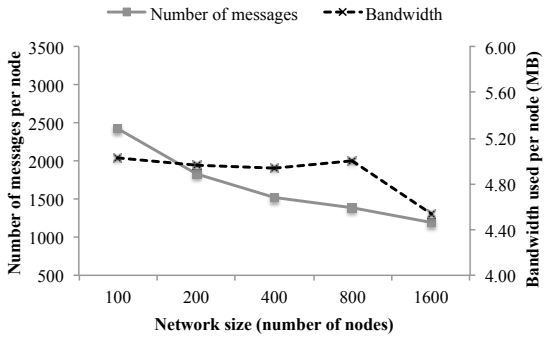

Figure 2: Cost per node in ELASTICO stays almost constant regardless of network size.

ments confirm the expected scalability of ELASTICO's transaction throughput.

Efficiency of ELASTICO. We next evaluate the efficiency property (defined in Section 2) of our protocol. Figure 2 depicts the number of messages sent/received and the bandwidth consumed at each node for different network sizes for reaching consensus once the committees are formed. The number of messages per node reduces as more nodes join the network, while the bandwidth consumed at each node fluctuates around 5 MB per node.

The decrease in the number of messages exchanged when there are more nodes in the network is because the extra messages required for running the second instance of PBFT in the final committee are amortized. Thus, we see the reduction in the number of messages exchanged per node when the network size increases from 100 nodes (2, 416 messages) to 200 nodes (1, 821 messages). The bandwidth used at each node, on the other hand, remains roughly unchanged, *e.g.*, 4.93 MB (400 nodes) and 5.01 MB (800 nodes) per node. This is because the communication costs within committees are localized. Thus even when more nodes join the network, existing nodes does not have to bear more costs.

Extrapolation to Bitcoin's scale. We extrapolate the scale up that ELASTICO can achieve in a network of current Bitcoin's scale, if we assume our scalability holds in real network. As of February 2016, the hash rate of Bitcoin network is 1.2×10^9 GHash/s. Thus, we assume that the network consists of $n = 1,000,000$ equivalent processors, each of which can perform 1.2×10^3 GHash/s. We consider a setting where the committee size is 100, thus the number of committees is $10,000$. Each committee agrees on a single block,

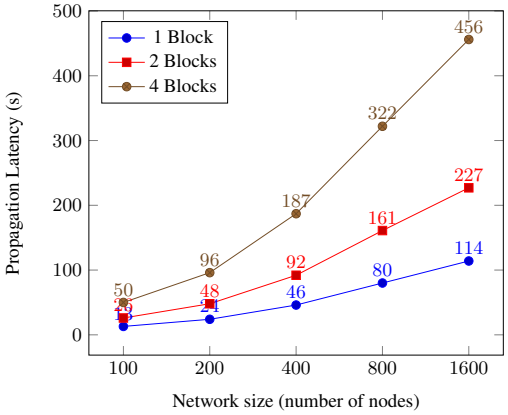

Figure 3: Latency of Bitcoin-NG with different number of micro blocks

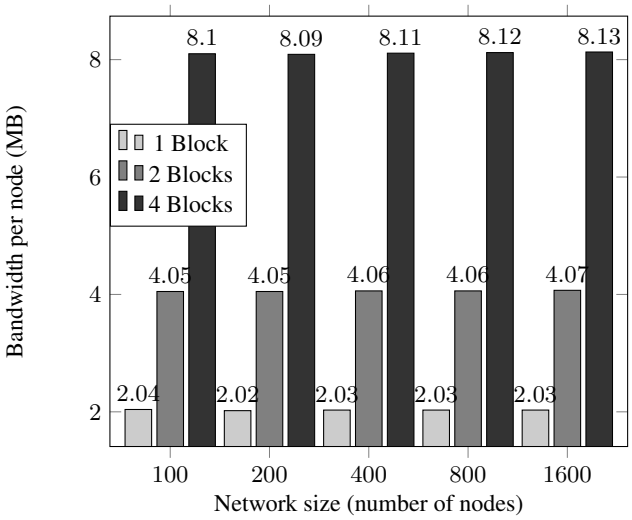

Figure 4: Bandwidth consumption of Bitcoin-NG with different number of micro blocks and network sizes

therefore the number of blocks per epoch that the network can agree on is $10,000$. State differently, ELASTICO scale up the agreement by 4 orders of magnitude when deployed in a network of current Bitcoin's scale.

5.3 Comparison to Related Systems

We show how ELASTICO outperforms existing protocols in many applications where the verification check of a block does not require to scan through the entire history of the blockchain, but only the block itself. A prominent example of such applications is an append-only database for certificate directory, information registration platform and so on. In these applications, verifying a data block can be done independently disregard to the current state of the blockchain, thus ELASTICO nodes do not have to download data blocks from other committees. We show that only ELASTICO achieves the *efficiency* property defined in Section 2. On the contrary, other protocols like Bitcoin, Bitcoin-NG [9] and PBFT require much more network bandwidth and/ or local computation at each node when processing more transactions.

Bitcoin. Recent work has shown the scalability limit of Bitcoin [10]. We also run our own set of experiments and get consistent results with the results in [10]. Due to the space constraint, we do not include this set of experiments and the results here.

Bitcoin-NG. Our next set of experiments compare the scalability of ELASTICO and a recent scalability proposal namely Bitcoin-NG [9]. Bitcoin-NG slightly modifies Nakamoto consensus protocol to propose more blocks per epoch. At a high level idea, Bitcoin-NG also probabilistically selects a leader, and the leader can propose several micro blocks (*i.e.*, data blocks) per epoch. The leader of the next epoch can decide which data block of previous epoch he/ she wants to build on top.

We measure the bandwidth consumption per node of Bitcoin-NG in different settings where the numbers of nodes are 100, 200, 400, 800, $1,600$. We also quantify the latency to broadcast different number of micro blocks. In terms of experimental setup, nodes are distributed equally in two Amazon regions namely Oregon and California. Each node randomly connects with 4 other nodes to form a peer-to-peer network. We collect the source code of Bitcoin-NG from the authors through our private communication. Figure 3 and Figure 4 report our results.

Figure 3 shows the latency of Bitcoin-NG for different number of micro blocks. Bitcoin-NG performs well (*i.e.*, low latency) when the network size is small (100 nodes, 4 blocks, 50 seconds). Its performance worsens when the network size grows larger ($1,600$ nodes, 4 blocks, 456 seconds). Since Bitcoin-NG is a variant of Nakamoto consensus, nodes have to broadcast all blocks to the whole network since they are needed for the consensus protocol of the next epoch. Thus, increasing the block throughput entails a longer overall latency, especially when the network grows. ELASTICO, on the other hand, scales up the throughput as more nodes join the network without expanding the system latency, as we establish in Section 5.2. ELASTICO does that by decoupling messages needed for consensus from the final data broadcast.

In terms of bandwidth consumption, the bandwidth used at each node in Bitcoin-NG increases linearly with the throughput, *e.g.*, 2.04 and 4.05 MB per node when there are 1 and 2 data blocks respectively. This is because Bitcoin-NG needs to broadcast all blocks to the network for the next consensus step, thus more blocks, more bandwidth used. On the other hand, in ELASTICO, only the block headers are broadcast to the whole network since ELASTICO decouples data broadcast from consensus metadata. Therefore the bandwidth used at each node for the consensus step remains roughly the same, regardless of the throughput, *e.g.*, 4.93 and 5.01 MB when there are 4 and 8 blocks per epoch respectively.

Network constraint in PBFT. We next measure the scalability of traditional byzantine consensus protocols. The selected candidate is PBFT [13] with our own implementation. Similar to the experiments with Bitcoin-NG, we report the number of messages exchanged per node, the bandwidth consumed at each node and the latency to reach consensus for various network sizes. The results of our experiments are in Figure 5 and Figure 6. Note that all communications are done via a peer-to-peer network, with each node connects to 4 other peers randomly.

We observe that the bandwidth cost and the total number of messages exchanged at each node increase linearly with the size of the network. For example, a node sends/ receives 970 and $1,930$ messages when the network is of sizes 80 and 160 respectively. Similarly, the latency grows quadratically as we introduce more nodes. For instance, when network size increases from 40 to 80 nodes (2 times), the latency is 6 times longer (*e.g.*, from 3 seconds to 18 seconds). This linear increase in cost and quadratic increase in latency render PBFT inefficient even if the network has only a few hundreds nodes. In fact, our experiment when the network has 320 nodes did not terminate after running for 1 hour. We remark that in our experiments, there is no faulty (malicious) nodes. Thus, the cost will increase if we introduce faulty nodes in our experiments.

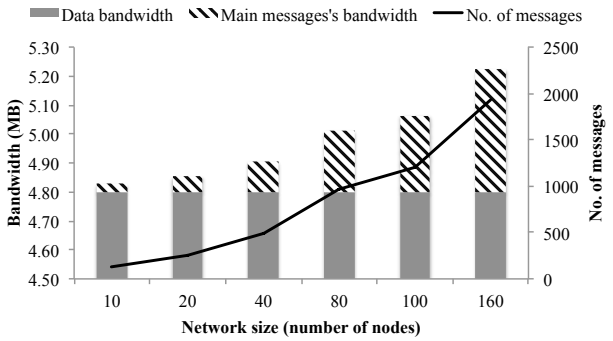

Figure 5: Cost per node in PBFT increases linearly with the size of the network.

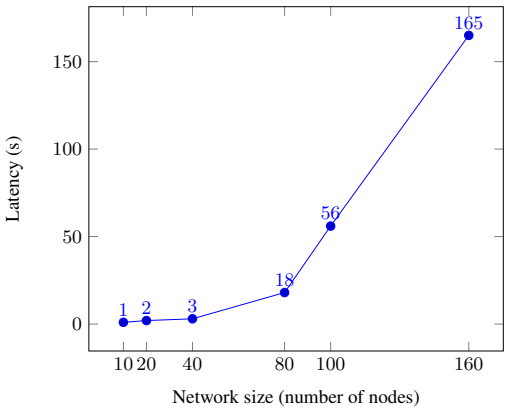

Figure 6: Latency to reach consensus in PBFT with different network sizes

6. RELATED WORK

We compare our solution to existing solutions for blockchain scalability in Table 2. The detailed discussions are below.

6.1 Centralized Sharding Protocols

ELASTICO is related to other sharding protocols in distributed databases, *e.g.*, Google's Spanner [15], Scatter [16], RSCoin [39]. However, these sharding protocols consider a different model which does not handle byzantine failures, make assumptions of PKI systems and a trusted infrastructure and access to external random seed. For example, RSCoin only works for centralized cryptocurrencies where there is a central point of trust (central bank). Such protocols are inapplicable to deploy in a byzantine environment like blockchains. In fact, sharding is a well-recognized open problem in byzantine environment [10]. In this work, we explain all the challenges and propose the first such sharding solution in the partially synchronous setting. We have established that ELASTICO is secure and cost-efficient even with byzantine adversaries, allowing the transaction throughput to scale up almost linearly with the network computation capacity.

6.2 Blockchain Scalability Solutions

Building a scalable blockchain is an active problem in the Bitcoin and cryptocurrency community. There have been several proposals from both academia and industry.

The first approach is to push more blocks to the blockchain, *e.g.*, GHOST [40], Bitcoin-NG [9]. GHOST modifies the rule to accept the main valid blockchain to accept not only the earliest block at each epoch, but also other blocks which are found later, *e.g.*, "or-

	BFT	Nakamoto	Quorum-BFT	Two-phase commit	ELASTICO
Candidate	Tendermint [35] IBM Blockchain [36] Chain OS [19] DigitalAsset [37]	Bitcoin [1] Ethereum [38] BitcoinNG [9] IntelLedger [18]	Ripple [22] Stellar [21]	Spanner [15] RSCoin [39] Databases	This work
Decentralized	Yes ✓	Yes ✓	No	No	Yes ✓
Identity-less	No	Yes ✓	No	No	Yes ✓
Bandwidth (per node)	$O(n^2)$	Constant ✓	$O(n)$	Constant ✓	Constant ✓
Scalability	No	No	No	Yes ✓	Yes ✓

Table 2: Comparison between ELASTICO and existing blockchain protocols in academia and industry. ELASTICO is the first solution which can scale up the throughput when the network size increases in a byzantine and decentralized environment.

phaned blocks". On the other hand, Bitcoin-NG allows each leader in an epoch to propose more blocks to the network. Both GHOST and Bitcoin-NG succeed at allowing block parallelism in the network, but they do not localize the verification of transactions as in ELASTICO. Thus, more transactions in the network, more local computation is required at each node and delay the consensus process as pointed out in previous work [41].

A different approach for increasing Bitcoin's transaction throughput is to allow transactions to take place outside of the main Bitcoin blockchain. Two prominent examples are lightning-network [42] and Sidechains [43]. Lightning-network creates offchain micropayment channels between users. Thus users can send multiple and instant transactions more efficiently, with only a few transactions included in the blockchain. Sidechains takes a different approach to allow users to move coins to different blockchain, thus allowing Bitcoin transactions to happen elsewhere. It is widely understood that Sidechains do not solve the scalability problem in Bitcoin [44]. Both the techniques, although improve the transaction throughput significantly, are applications running on top of Bitcoin thus still rely on the scalability of the underlying protocol. It is worth noting that applications enabled by Sidechains do not enjoy the same security guarantee provided by Bitcoin, and micro-payment channels only work for a few applications. ELASTICO, however, allows scaling up the underlying blockchain protocol without degrading any security property.

Buterin *et al.* also address the scalability problem in blockchain with sampling and challenging techniques [45]. Similar to ELASTICO, the paper's approach is to use sharding. However, the protocol "randomly" samples verifiers to verify others' updates, and allows users to challenge others' verification results if they ever detect an invalid update. The solution relies on a random seed, for which the paper does not provide any security analysis. Further, the paper does not consider byzantine adversaries but rational ones in a "cryptoeconomic" threat model, which is different from the threat model that we consider in this paper.

Recent non-peer-reviewed proposals including Stellar [21], Ripple [22], and Tendermint [35] claim to support high transaction rate, but either have weaker threat models or are not as scalable as ELASTICO. Specifically, Tendermint assumes all identities are known before the protocol starts, thus is not applicable in decentralized environments like cryptocurrencies. Besides, Tendermint is essentially a variant of PBFT [4], which has its own scalability limitation if the network size grows as we discussed in Section 2. Plus, the network nodes in Ripple and Stellar are permissioned, hence it faces no challenges of establishing identities. For instance, identities in Stellar need financial agreements or reputations of others to form their "slices" (or committees). In Elastico, these have to be chosen randomly based on computational assumptions.

[4] http://tendermint.com/posts/tendermint-vs-pbft/

6.3 Prior Byzantine Consensus Protocols

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 [46].

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 [47].

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 [11, 12]. A key improvement was made by Srikanth *et al.* [14], 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 *et al.* [13] implemented a replicated state machine based on Byzantine agreement, often described as the first practical BFT 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., [48–51]).

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 [52–55], 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. To achieve almost everywhere agreement, they assign the users to committees, and organize the committees into a tree with a guarantee that almost all the committees have a majority of honest users (using an adapted version of Feige's leader election algorithm [56]). 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 [57]. However, the protocols are complex and practical implementations are yet to be demonstrated.

ELASTICO is not directly comparable to these newer communication efficient protocols: ELASTICO is simpler and works in open networks like cryptocurrencies where identities are unknown. Its key advantage is in using computational power to tune the parallelization of network, yet detaining security on bounded computational assumption.

ELASTICO is related to other protocols which use proof of work for processors to establish their identities [58–61]. The main difference here is that ELASTICO is a sharding protocol, and establishing identities is just the first step of the 5 major steps in the protocol.

7. CONCLUSION

We present ELASTICO, the first candidate for a secure sharding protocol for permissionless blockchains. At its core, ELASTICO scales up the agreement throughput near linearly with the computational power of the network and tolerates byzantine adversaries which controls up to one-forth computation capacity, in the partially synchronous network. It offers promising scalability in experiments and suggest strong usability in next-generation cryptocurrencies.

8. ACKNOWLEDGMENT

We thank Dawn Song, Elaine Shi, Christian Cachin, Andrew Miller, Jason Teutsch, Shweta Shinde, Shruti Tople, Alex Zikai Wen, Hung Dang, Xiao Liu and Vincent Gramoli for useful discussions and feedback on the early version of the paper.

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2014NCR-NCR001-21) and administered by the National Cybersecurity R&D Directorate. All opinions expressed in this work are solely those of the authors.

9. REFERENCES

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

[2] Blockchain stats. Bitcoin statistics. https://blockchain.info/stats, 2012.

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

[4] Mastercard. Mastercard's transaction per second. http://newsroom.mastercard.com/2012/11/26/mastercard-sees-black-friday-performance-up-26-percent/, 2016.

[5] Visa. Visa's transactions per second. https://usa.visa.com/content_library/modal/benefits-accepting-visa.html, 2016.

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

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

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

[9] 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.

[10] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gun Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains (a position paper). *Workshop on Bitcoin and Blockchain Research*, 2016.

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

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

[13] 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.

[14] 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.

[15] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, aug 2013.

[16] Lisa Glendenning, Ivan Beschastnikh, Arvind Krishnamurthy, and Thomas Anderson. Scalable consistency in scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 15–28, New York, NY, USA, 2011. ACM.

[17] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *Proceedings of the Conference on Innovative Data system Research (CIDR)*, pages 223–234, 2011.

[18] Intel. Intel distributed ledger. http://intelledger.github.io/, 2016.

[19] Chain Inc. Chain open standard: A secure blockchain protocol for high-scale financial networks. http://chain.com/os/, 2016.

[20] Rhett Creighton. Domus tower blockchain. http://domustower.com/domus-tower-blockchain-latest.pdf, 2016.

[21] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. April 2015.

[22] Arthur Britto David Schwartz, Noah Youngs. The ripple protocol consensus algorithm. *Ripple Labs Inc.*, 2014.

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

[24] Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly dynamic distributed computing with byzantine failures. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, 2013.

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

[26] John R. Douceur. The sybil attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[27] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: Analysis & defenses. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, IPSN '04, pages 259–268, New York, NY, USA, 2004. ACM.

[28] Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. *Theor. Comput. Sci.*, 410(6-7):453–466, feb 2009.

[29] Bitcoin wiki. Proof of stake. https://en.bitcoin.it/wiki/Proof_of_Stake, 2015.

[30] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. Cryptology ePrint Archive, Report 2013/796, 2013. http://eprint.iacr.org/.

[31] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the

essence. Cryptology ePrint Archive, Report 2013/805, 2013. http://eprint.iacr.org/.

[32] Nancy Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[33] Seth Gilbert, Calvin Newport, and Chaodong Zheng. Who are you? secure identities in ad hoc networks. In *Distributed Computing*, pages 227–242. Springer, 2014.

[34] 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/.

[35] Jae Kwon. Tendermint: Consensus without mining.

[36] IBM. Ibm blockchain. http://www.ibm.com/blockchain/, 2016.

[37] Digital Asset Holdings. Digital asset. https://digitalasset.com/, 2016.

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

[39] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. Cryptology ePrint Archive, Report 2015/502, 2015. http://eprint.iacr.org/.

[40] 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/.

[41] 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/.

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

[43] 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.

[44] Pieter Wuille. Would sidechains help bitcoin scale? 2015.

[45] 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.

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

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *Distributed Computing and Networking*, volume 6522 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2011.

[55] 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.

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

[57] 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.

[58] Jonathan Katz, Andrew Miller, and Elaine Shi. Pseudonymous broadcast and secure computation from cryptographic puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. http://eprint.iacr.org/2014/857.

[59] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 13:1–13:10, New York, NY, USA, 2016. ACM.

[60] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged byzantine impostors. *Department of Computer Science, Yale University, New Haven, CT, Tech. Rep*, 2005.

[61] Marcin Andrychowicz and Stefan Dziembowski. Distributed cryptography based on the proofs of work. Cryptology ePrint Archive, Report 2014/796, 2014. http://eprint.iacr.org/2014/796.

[62] Wikipedia. Coupon collector's problem. https://en.wikipedia.org/wiki/Coupon_collector%27s_problem.

[63] Donald J. Newman. The double dixie cup problem. *The American Mathematical Monthly*, 67(1):58–61, 1960.

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

[65] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[66] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, February 2012.

10. APPENDIX

10.1 The Scalability of ELASTICO

We now explain why ELASTICO achieves the desired asymptotic $\mathcal{O}(n/\log\log n)$ scalability by calculating the expected number of PoW solutions that n processors have to generate such that each of 2^s committees has c members. Our problems is equivalent to the extended coupon collector problem [62] where there are 2^s types of coupon in some urn and we want to calculate the number of draws to collect c copies of each coupon. It is shown in [63] that the expected number of PoW solutions E is

$$\begin{aligned} E &= 2^s \log 2^s + (c-1)2^s \log\log 2^s + \mathcal{O}(2^s) \\ &= s \cdot \frac{n}{c} + n \log s - \frac{n}{c} \log s + \mathcal{O}(n/c) \end{aligned}$$

If $c < s$, ELASTICO can achieve $\mathcal{O}(n/\log n)$ scalability. On the other hand, if $c > s$, we have

$$s \cdot \frac{n}{c} < n,$$

or $E = \mathcal{O}(n \log s)$ asymptotically given that $s < c$. Since $n = c \cdot 2^s$, we have $E = \mathcal{O}(n \log\log n)$, hence the $\mathcal{O}(n/\log\log n)$ scalability. The second scenario is more likely to happen in practice because c must be of a few hundred to provide a reasonable security guarantee.

In addition, [63] also shows that if c is large enough, E is asymptotically $\mathcal{O}(c \cdot 2^s)$. That means, ELASTICO can achieve linear scalability if the committee size is chosen correctly. By using Chernoff bound, we prove in Lemma 8 that $c > 24 \cdot \ln(2^s)$ gives us such scalability.

Lemma 8 (Balls to Bins)**.** *We throw balls uniformly at random into a collection of X bins. When $c > 24 \cdot \ln X$, the expected number of balls to ensure each bin to have at least c balls is $\mathcal{O}(cX)$*

Proof. Let us consider the scenario of throwing cX balls uniformly at random into those X bins. The expected number of balls in each bin is c. Let's fix a particular bin of interest and we consider x_i be a random variable that equals 1 if ball i lands in that bin, and 0 otherwise. Let $C = \sum x_i$. Notice that the expected value $E(C) = c$, and all x_i are independent. Using Chernoff bound, we have:

$$\Pr\left[C < (1-\delta)E(C)\right] \leq e^{-E(C)\delta^2/2}$$

Fix $\delta = 1/2$, and we get:

$$\Pr\left[C < c/2\right] \leq e^{-c/8}$$

Given $c > 24 \cdot \ln X$, we get:

$$\Pr\left[C < c/2\right] \leq 1/X^3.$$

Recall we are looking at all one particular bin. Now we consider all X bins. Take a union bound, and we get the probability that *any* bin has less than $c/2$ balls is at most $1/X^2$. This also means that with probability $1/X^2$, we throw cX balls more into X bin. Doing the same calculation for the new cX balls, we need to throw cX more balls with probability $1/X^2$.

Thus the expected number of balls we have to throw to ensure each bin to have at least $c/2$ balls is:

$$cX + cX/X^2 + cX/X^4 + cX/X^6 + ... \leq 2cX.$$

Therefore, the number of balls to throw guarantee each bin to have at least c balls is $\mathcal{O}(cX)$. □

10.2 Verification Checks in ELASTICO

Different blockchain applications may require different verification checks on the transactions (*e.g.*, [64]). In the presence of sharding, supporting arbitrary checks within local committees is not possible, as is well-known in sharded databases [65, 66]. In a cryptocurrency, the most important checks are ensuring that the payer signs the coin transfer and that the transaction is not double-spending. We next discuss how one can efficiently implement such verification check for transactions in a hypothetical cryptocurrency built on top of ELASTICO.

In Bitcoin and popular cryptocurrencies, a regular transaction sends some amount of coins from a sender to a recipient. Each transaction has two parts, namely an `Input` and an `Output`. The `Input` specifies the source of the coin and a proof (by digital signatures) which shows the sender is a valid coin owner. The `Output` names the recipient; later on the recipient can use this `Output` as the source of the coin in his/her new transaction). Checking if a transaction is double-spending entails checking if it uses some coin which has been spent in a previous transaction. A naive implementation of such check would require one to scan through the entire history of the blockchain. In Bitcoin, to avoid such costs, each node maintains a local database containing all "unspent transaction outputs" (or UTXO for short) to check if an `Output` has been spent or not. Bitcoin nodes frequently update the UTXO database after each new block based on the set of transactions included in the block. Thus, if a transaction is finalized in the Bitcoin blockchain, it is guaranteed to be valid.

In ELASTICO, our transaction verification is similar, *i.e.*, the node also maintain a local database of UTXOs and update such database after each epoch when they download data blocks from committees. To avoid the scenario that two committees process the same transaction, each committee in ELASTICO processes a separate list of transactions which have some specific range of `Input`. For example, if there are 4 committees, they will handle transactions having `Inputs`' IDs (*i.e.*, the hashes of the `Inputs`) with different prefixes of $00, 01, 10, 11$ respectively. As a result, within an epoch, all shards will include disjoint transaction inputs (thus disjoint transaction sets).

The verification in ELASTICO is superior to other solutions like GHOST [40], Bitcoin- NG [9] in many aspects. First, ELASTICO nodes do not need to verify data blocks from other committees. Instead, they only need to check if a block is committed in the consensus block (from the final committee) to decide if the block is valid. It is because the committee corresponding to that shard already verifies its block, and the final committee has verified if a block is agreed by a committee before committing the block to the final (consensus) block. On the other hand, in other solutions, nodes have to verify all transactions included in the blockchain, thus higher throughput will require more local computation from nodes. Further, increasing the amount of verification check will slow down the block broadcast, and further pose a security threat in the consensus protocol as pointed out by previous work [41].

Thus, ELASTICO scales up the throughput as the network allows, without requiring more local computation for verification at each node or affecting the consensus security. Note that ELASTICO still permits bandwidth efficiency in its consensus step for a cryptocurrency application, but has to broadcast all the data blocks to all nodes (as done in all solutions today). However, in applications where the consistency checks can themselves be sharded and checked locally, there would be no need to broadcast the data blocks to the network.