RapidChain: Scaling Blockchain via Full Sharding

Mahdi Zamani
Visa Research
Palo Alto, CA

Mahnush Movahedi[†]
Dfinity
Palo Alto, CA

Mariana Raykova
Yale University
New Haven, CT

Abstract

A major approach to overcoming the performance and scalability limitations of current blockchain protocols is to use *sharding* which is to split the overheads of processing transactions among multiple, smaller groups of nodes. These groups work in parallel to maximize performance while requiring significantly smaller communication, computation, and storage per node, allowing the system to scale to large networks. However, existing sharding-based blockchain protocols still require a linear amount of communication (in the number of participants) per transaction, and hence, attain only partially the potential benefits of sharding. We show that this introduces a major bottleneck to the throughput and latency of these protocols. Aside from the limited scalability, these protocols achieve weak security guarantees due to either a small fault resiliency (*e.g.*, 1/8 and 1/4) or high failure probability, or they rely on strong assumptions (*e.g.*, trusted setup) that limit their applicability to mainstream payment systems.

We propose RapidChain, the first sharding-based public blockchain protocol that is resilient to Byzantine faults from up to a 1/3 fraction of its participants, and achieves complete sharding of the communication, computation, and storage overhead of processing transactions without assuming any trusted setup. RapidChain employs an optimal intra-committee consensus algorithm that can achieve very high throughputs via block pipelining, a novel gossiping protocol for large blocks, and a provably-secure reconfiguration mechanism to ensure robustness. Using an efficient cross-shard transaction verification technique, our protocol avoids gossiping transactions to the entire network. Our empirical evaluations suggest that RapidChain can process (and confirm) more than 7,300 tx/sec with an expected confirmation latency of roughly 8.7 seconds in a network of 4,000 nodes with an overwhelming time-to-failure of more than 4,500 years.

1 Introduction

Our global financial system is highly centralized making it resistant to change, vulnerable to failures and attacks, and inaccessible to billions of people in need of basic financial tools [63, 28]. On the other hand, decentralization poses new challenges of ensuring a consistent view among a group of mutually-distrusting participants. The permissionless mode of operation, which allows open membership and entails constant churn (*i.e.*, join/leave) of the participants of the decentralized system, further complicates this task. Furthermore, any agile financial system, including a decentralized one, should be able to adequately serve realistic market loads. This implies that it should scale easily to a large number of participants, and it should handle a high throughput of transactions with relatively low delays in making their outputs available. Achieving these properties together should also not require significant resources from each of the participants since otherwise, it runs contrary to the idea of constructing a tool easily accessible to anyone.

Existing solutions currently either fail to solve the above challenges or make security/performance trade-offs that, unfortunately, make them no longer truly-decentralized solutions. In particular, traditional Byzantine consensus mechanisms such as [44, 20, 17] can only work in a closed membership setting, where the set of participants is fixed and their identities are known to everyone via a trusted third party. If used in an open setting, these protocols can be easily compromised using *Sybil attacks* [25], where the adversary repeatedly rejoins malicious parties with fresh identities to gain significant influence on the protocol outcome. Moreover, most traditional schemes assume a *static* adversary who can select the set of corrupt parties only at the start of the protocol. Existing protocols that are

[*]Email: *mzamani@visa.com* | *mahenush@dfinity.org* | *mariana.raykova@yale.edu*
[†]This work was done in part while this author was affiliated with Yale University.

secure against an *adaptive* adversary such as [12, 18, 38] either scale poorly with the number of participants or are inefficient.

Most cryptocurrencies such as Bitcoin [54] and Ethereum [16] maintain a distributed transaction ledger called the *blockchain* over a large peer-to-peer (P2P) network, where every node maintains an updated, full copy of the entire ledger via a Byzantine consensus protocol, dubbed as the *Nakamoto consensus*. Unlike traditional consensus mechanisms, the Nakamoto consensus allows new participants to join the protocol using a *proof-of-work (PoW)* process [27], where a node demonstrates that it has done a certain amount of work by presenting a solution to a computational puzzle. The use of PoW not only allows the consensus protocol to impede Sybil attacks by limiting the rate of malicious participants joining the system, but also provides a lottery mechanism through which a random leader is elected in every round to initiate the consensus process.

Unfortunately, it is now well known that Bitcoin's PoW-based consensus comes with serious drawbacks such as low transaction throughput, high latency, poor energy efficiency [46], and mining-pool centralization [34, 1]. Moreover, the protocol cannot scale out its transaction processing capacity with the number of participants joining the protocol [47, 42]. Another major scalability issue of Bitcoin is that every party needs to initially download the entire blockchain from the network to independently verify all transactions. The size of the blockchain is currently about 150 GB and has nearly doubled in the past year [2]. One can expect a larger growth in the size of blockchains that are updated via higher-throughput consensus protocols than that of Bitcoin.

Recently, several protocols have been proposed to mitigate the performance and scalability issues of Bitcoin's blockchain [23, 29, 41, 52, 56, 47, 32, 4, 42] using hybrid architectures that combine the open-membership nature of Bitcoin with traditional Byzantine fault tolerance [57, 19]. While most of these protocols can reportedly improve the throughput and latency of Bitcoin, all of them still require the often-overlooked assumption of a trusted setup to generate an unpredictable initial common randomness in the form of a common genesis block to bootstrap the blockchain. Similar to Bitcoin, these protocols essentially describe how one can ensure agreement on new blocks given an initial agreement on some genesis block. Such an assumption plays a crucial role in achieving consistency among nodes in these protocols, and if compromised, can easily affect the security of the entire consensus protocol, casting a major contradiction to the decentralized nature of cryptocurrencies.

In addition to being partially decentralized, most of these solutions have either large per-node storage requirements [23, 29, 41, 52, 56, 47, 4], low fault resiliency [47, 42], incomplete specifications [23, 41], or other security issues [41, 47, 42] (see Section 2.2 for more details). Furthermore, all previous protocols require every participant in the consensus protocol to broadcast a message to the entire network to either submit their consensus votes [29, 52], verify transactions [41, 47, 42], and/or update every node's local blockchain replica [47, 56, 32, 4].

While the large overhead of such a broadcast for every participant is usually reduced from a linear number of messages (with respect to the number of participants) to nearly a constant using a peer-to-peer gossiping protocol [36], the relatively high latency of such a "gossip-to-all" invocation (*e.g.*, 12.6 seconds per block on average [24]) increases the overall latency of the consensus protocol significantly (*e.g.*, the gossip-to-all latency roughly quadruples the consensus time in [42]). Moreover, due to the very high transaction throughput of most scalable blockchain protocols (*e.g.*, about 3,500 tx/sec in [42]), the bandwidth usage of each node becomes very large (*e.g.*, at least 45 Mbps in [42] – see Section 5 for more details), if all transactions are gossiped to the entire network.

1.1 Our Contributions

We propose RapidChain, a Byzantine-resilient public blockchain protocol that improves upon the scalability and security limitations of previous work in several ways. At a high level, RapidChain partitions the set of nodes into multiple smaller groups of nodes called *committees* that operate in parallel on disjoint blocks of transactions and maintain disjoint ledgers. Such a partitioning of operations and/or data among multiple groups of nodes is often referred to as *sharding* [21] and has been recently studied in the context of blockchain protocols [47, 42]. By enabling parallelization of the consensus work and storage, sharding-based consensus can scale the throughput of the system proportional to the number of committees, unlike the basic Nakamoto consensus.

Let n denote the number of participants in the protocol at any given time, and $m \ll n$ denote the size of each committee. RapidChain creates $k = n/m$ committees each of size $m = c \log n$ nodes, where c is a constant depending only on the security parameter (in practice, c is roughly 20). In summary, RapidChain provides the following novelties:

- **Sublinear Communication.** RapidChain is the first sharding-based blockchain protocol that requires only a sublinear (*i.e.*, $o(n)$) number of bits exchanged in the network per transaction. In contrast, all previous work

Protocol	# Nodes	Resiliency	Complexity[1]	Throughput	Latency	Storage[2]	Shard Size	Time to Fail
Elastico [47]	$n = 1{,}600$	$t < n/4$	$\Omega(m^2/b{+}n)$	40 tx/sec	800 sec	1x	$m = 100$	1 hour
OmniLedger [42]	$n = 1{,}800$	$t < n/4$	$\Omega(m^2/b{+}n)$	500 tx/sec	14 sec	1/3x	$m = 600$	230 years
OmniLedger [42]	$n = 1{,}800$	$t < n/4$	$\Omega(m^2/b{+}n)$	3,500 tx/sec	63 sec	1/3x	$m = 600$	230 years
RapidChain	$n = 1{,}800$	$t < n/3$	$O(m^2/b{+}m\log n)$	4,220 tx/sec	8.5 sec	1/9x	$m = 200$	1,950 years
RapidChain	$n = \mathbf{4{,}000}$	$t < n/3$	$O(m^2/b{+}m\log n)$	**7,380 tx/sec**	**8.7 sec**	**1/16x**	$m = \mathbf{250}$	**4,580 years**

Table 1: Comparison of RapidChain with state-of-the-art sharding blockchain protocols (b is the block size)

incur an $\Omega(n)$ communication overhead per transaction (see Table 1).

- **Higher Resiliency.** RapidChain is the first sharding-based blockchain protocol that can tolerate corruptions from less than a 1/3 fraction of its nodes (rather than 1/4) while exceeding the throughput and latency of previous work (*e.f.*, [47, 42]).

- **Rapid Committee Consensus.** Building on [5, 60], we reduce the communication overhead and latency of P2P consensus on large blocks gossiped in each committee by roughly 3-10 times compared to previous solutions [47, 32, 4, 42].

- **Secure Reconfiguration.** RapidChain builds on the Cuckoo rule [8, 62] to provably protect against a slowly-adaptive Byzantine adversary. This is an important property missing in previous sharding-based protocols [47, 42]. RapidChain also allows new nodes to join the protocol in a seamless way without any interruptions or delays in the protocol execution.

- **Fast Cross-Shard Verification.** We introduce a novel technique for partitioning the blockchain such that each node is required to store only a $1/k$ fraction of the entire blockchain. To verify cross-shard transactions, RapidChain's committees discover each other via an efficient routing mechanism inspired by Kademlia [48] that incurs only a logarithmic (in number of committees) latency and storage. In contrast, the committee discovery in existing solutions [47, 42] requires several "gossip-to-all" invocations.

- **Decentralized Bootstrapping.** RapidChain operates in the permissionless setting that allows open membership, but unlike most previous work [29, 41, 56, 47, 42, 4], does not assume the existence of an initial common randomness, usually in the form of a common genesis block. While common solutions for generating such a block require exchanging $\Omega(n^2)$ messages, RapidChain can bootstrap itself with only $O(n\sqrt{n})$ messages without assuming any initial randomness.

We also implement a prototype of RapidChain to evaluate its performance and compare it with the state-of-the-art sharding-based protocols. Table 1 shows a high-level comparison between the results. In this table, we assume 512 B/tx, one-day long epochs, 100 ms network latency for all links, and 20 Mbps bandwidth for all nodes in all three protocols. The choices of 1,600 and 1,800 nodes for [47] and [42] respectively is based on the maximum network sizes reported in these work. Unfortunately, the time-to-failure of the protocol of [47] decreases rapidly for larger network sizes. For [42], we expect larger network sizes will, at best, only slightly increase the throughput due to the large committee sizes (*i.e.*, m) required.

The latency numbers reported in Table 1 refer to block (or transaction) confirmation times which is the delay from the time that a node proposes a block to the network until it can be confirmed by all honest nodes as a valid transaction. We refer the reader to Section 2.2 and Section 5 for details of our evaluation and comparison with previous work.

1.2 Overview of RapidChain

[1]Total message complexity of consensus per transaction (see Section 6.6).

[2]Reduction in the amount of storage required for each participant after the same number of transactions are processed (and confirmed) by the network.

RapidChain proceeds in fixed time periods called *epochs*. In the first epoch, a one-time bootstrapping protocol (described in Section 4.6) is executed that allows the participants to agree on a committee of $m = O(\log n)$ nodes in a constant number of rounds. Assuming $t < n/3$ nodes are controlled by a slowly-adaptive Byzantine adversary, the committee-election protocol samples a committee from the set of all nodes in a way that the fraction of corrupt nodes in the sampled set is bounded by $1/2$ with high probability. This committee, which we refer to as the *reference committee* and denote it by C_R, is responsible for driving periodic reconfiguration events between epochs. In the following, we describe an overview of the RapidChain protocol in more details.

Epoch Randomness. At the end of every epoch i, C_R generates a fresh randomness, r_{i+1}, referred to as the *epoch randomness* for epoch $i + 1$. This randomness is used by the protocol to (1) sample a set of 1/2-resilient committees, referred to as the *sharding committees*, at the end of the first epoch, (2) allow every participating node to obtain a fresh identity in every epoch, and (3) reconfigure the existing committees to prevent adversarial takeover after nodes join and leave the system at the beginning of every epoch or node corruptions happening at the end of every epoch.

Peer Discovery and Inter-Committee Routing. The nodes belonging to the same sharding committee discover each other via a peer-discovery algorithm. Each sharding committee is responsible for maintaining a disjoint transaction ledger known as a *shard*, which is stored as a blockchain by every member of the committee. Each transaction tx is submitted by a user of the system to a small number of arbitrary RapidChain nodes who route tx, via an *inter-committee routing protocol*, to a committee responsible for storing tx. We refer to this committee as the *output committee* for tx and denote it by C_{out}. This committee is selected deterministically by hashing the ID of tx to a number corresponding to C_{out}. Inspired by Kademlia [48], the verifying committee in RapidChain communicates with only a logarithmic number of other committees to discover the ones that store the related transactions.

Cross-Shard Verification. The members of C_{out} batch several transactions into a large block (about 2 MB), and then, append it to their own ledger. Before the block can be appended, the committee has to verify the validity of every transaction in the block. In Bitcoin, such a verification usually depends on other (input) transactions that record some previously-unspent money being spent by the new transaction. Since transactions are stored into disjoint ledgers, each stored by a different committee, the members of C_{out} need to communicate with the corresponding *input committees* to ensure the input transactions exist in their shards.

Intra-Committee Consensus. Once all of the transactions in the block are verified, the members of C_{out} participate in an *intra-committee consensus protocol* to append the block to their shard. The consensus protocol proceeds as follows. First, the members of C_{out} choose a local leader using the current epoch randomness. Second, the leader sends the block to all the members of C_{out} using a fast gossiping protocol that we build based on the information dispersal algorithm (IDA) of Alon *et al.* [6, 5] for large blocks.

Third, to ensure the members of C_{out} agree on the same block, they participate in a Byzantine consensus protocol that we construct based on the synchronous protocol of Ren *et al.* [60]. This protocol allows RapidChain to obtain an intra-committee consensus with the optimal resiliency of $1/2$, and thus, achieve a total resiliency of $1/3$ with small committees. While the protocol of Ren *et al.* requires exchanging $O(m^2\ell)$ bits to broadcast a message of length ℓ to m parties, our intra-committee consensus protocol requires $O(m^2 h \log m + m\ell)$ bits, where h is the length of a hash that depends only on the security parameter.

Protocol Reconfiguration. A consensus decision in RapidChain is made on either a block of transactions or on a *reconfiguration block*. A reconfiguration block is generated periodically at the end of a *reconfiguration phase* that is executed at the end of every epoch by the members of C_R to establish two pieces of information: (1) a fresh epoch randomness, and (2) a new list of participants and their committee memberships. The reconfiguration phase allows RapidChain to re-organize its committees in response to a slowly-adaptive adversary [56] that can commit join-leave attacks [25] or corrupt nodes at the end of every epoch. Such an adversary is allowed to corrupt honest nodes (and hence, take over committees) only at the end of epochs, *i.e.*, the set of committees is fixed during each epoch.

Since re-electing all committees incurs a large communication overhead on the network, RapidChain performs only a small reconfiguration protocol built on the Cuckoo rule [8, 62] at the end of each epoch. Based on this strategy, only a constant number of nodes are moved between committees while provably guaranteeing security as long as at most a constant number of nodes (with respect to n) join/leave or are corrupted in each epoch.

During the reconfiguration protocol happening at the end of the i-th epoch, C_R generates a fresh randomness, r_{i+1}, for the next epoch and sends r_{i+1} to all committees. The fresh randomness not only allows the protocol to move a certain number of nodes between committees in an unpredictable manner, thus hindering malicious committee takeovers, but also allows creation of fresh computational puzzles for nodes who want to participate in the next epoch (*i.e.*, epoch $i+1$).

Any node that wishes to participate in epoch $i+1$ (including a node that has already participated in previous epochs) has to establish an *identity* (*i.e.*, a public key) by solving a fresh PoW puzzle that is randomized with r_{i+1}. The node has to submit a valid PoW solution to C_R before a "cutoff time" which is roughly 10 minutes after r_{i+1} is revealed by C_R during the reconfiguration phase. Once the cutoff time has passed, the members of C_R verify each solution and, if accepted, add the corresponding node's identity to the list of valid participants for epoch $i+1$. Next, C_R members run the intra-committee consensus protocol to agree on and record the identity list within C_R's ledger in a reconfiguration block that also includes r_{i+1} and the new committee memberships. This block is sent to all committees using the inter-committee routing protocol (see Protocol 1).

Further Remarks. Note that nodes are allowed to reuse their identities (*i.e.*, public keys) across epochs as long as each of them solves a fresh puzzle per epoch for a PoW that is tied to its identity and the latest epoch randomness. Also, note that the churn on C_R is handled in exactly the same way as it is handled in other committees: r_{i+1} generated by the C_R members in epoch i determines the new set of C_R members for epoch $i+1$. Finally, the difficulty of PoW puzzles used for establishing identities is fixed for all nodes throughout the protocol and is chosen in such a way that each node can only solve one puzzle during each 10-minute period, assuming without loss of generality, that each node has exactly one unit of computational power (see Section 3 for more details).

Paper Organization. In Section 2, we review related work and present a background on previous work that RapidChain builds on. In Section 3, we state our network and threat models and define the general problem we aim to solve. We present our protocol design in Section 4. We formally analyze the security and performance of RapidChain in Section 6. Finally, we describe our implementation and evaluation results in Section 5 and conclude in Section 7.

2 Background and Related Work

We review two categories of blockchain consensus protocols: *committee-based* and *sharding-based* protocols. We refer the reader to [9] for a complete survey of previous blockchain consensus protocols. Next, we review recent progress on synchronous Byzantine consensus and information dispersal algorithms that RapidChain builds on.

2.1 Committee-Based Consensus

The notion of committees in the context of consensus protocols was first introduced by Bracha [13] to reduce the round complexity of Byzantine agreement, which was later improved in, *e.g.*, [55, 61]. The idea of using committees for scaling the communication and computation overhead of Byzantine agreement dates back to the work of King *et al.* [39] and their follow-up work [38], which allow Byzantine agreement in fully-connected networks with only a sublinear per-node overhead, *w.r.t.* the number of participants. Unfortunately, both work provide only theoretical results and cannot be directly used in the public blockchain setting (*i.e.*, an open-membership peer-to-peer network).

Decker *et al.* propose the first committee-based consensus protocol, called PeerCensus, in the public blockchain model. They propose to use PBFT [20] inside a committee to approve transactions. Unfortunately, PeerCensus does not clearly mention how a committee is formed and maintained to ensure honest majority in the committee throughout the protocol. Hybrid Consensus [56] proposes to periodically select a committee that runs a Byzantine consensus protocol assuming a slowly-adaptive adversary that can only corrupt honest nodes in certain periods of time. ByzCoin [41] proposes to use a multi-signature protocol inside a committee to improve transaction throughput. Unfortunately, ByzCoin's specification is incomplete and the protocol is known to be vulnerable to Byzantine faults [56, 4, 9].

Algorand [32] proposes a committee-based consensus protocol called BA$\star$ that uses a verifiable random function (VRF) [51] to randomly select committee members, weighted by their account balances (*i.e.*, stakes), in a private and non-interactive way. Therefore, the adversary does not know which node to target until it participates in the BA$\star$ protocol with other committee members. Algorand replaces committee members with new members in every step of BA$\star$ to avoid targeted attacks on the committee members by a fully-adaptive adversary. Unfortunately, the randomness used in each VRF invocation (*i.e.*, the VRF seed) can be biased by the adversary; the protocol proposes a look-back mechanism to ensure strong synchrony and hence unbiased seeds, which unfortunately, results in a problematic situation known as the "nothing at stake" problem [32]. To solve the biased coin problem, Dfinity[33] propose a new VRF protocol based on non-interactive threshold signature scheme with uniqueness property.

Assuming a trusted genesis block, Solida [4] elects nodes onto a committee using their solutions to PoWs puzzles that are revealed in every round via $2t+1$ committee member signatures to avoid pre-computation (and withholding) attacks. To fill every slot in the ledger, a reconfigurable Byzantine consensus protocol is used, where a consensus decision is made on either a batch of transactions or a reconfiguration event. The latter records membership change in the committee and allows replacing at most one member in every event by ranking candidates by their PoW solutions. The protocol allows the winning candidate to lead the reconfiguration consensus itself avoiding corrupt internal leaders to intentionally delay the reconfiguration events in order to buy time for other corrupt nodes in the PoW process.

2.2 Sharding-Based Consensus

Unlike Bitcoin, a sharding-based blockchain protocol can increase its transaction processing power with the number of participants joining the network by allowing multiple committees of nodes process incoming transactions in parallel. Thus, the total number of transaction processed in each consensus round by the entire protocol is multiplied by the number of committees. While there are multiple exciting, parallel work on sharding-based blockchain protocols such as [64, 65], we only study results that focus on handling sharding in the Bitcoin transaction model.

2.2.1 RSCoin

Danezis and Meiklejohn [22] propose RSCoin, a sharding-based technique to make centrally-banked cryptocurrencies scalable. While RSCoin describes an interesting approach to combine a centralized monetary supply with a distributed network to introduce transparency and pseudonymity to the traditional banking system, its blockchain protocol is not decentralized as it relies on a trusted source of randomness for sharding of validator nodes (called mintettes) and auditing of transactions. Moreover, RSCoin relies on a two-phase commit protocol executed within each shard which, unfortunately, is not Byzantine fault tolerant and can result in double-spending attacks by a colluding adversary.

2.2.2 Elastico

Luu *et al.* [47] propose Elastico, the first sharding-based consensus protocol for public blockchains. In every consensus epoch, each participant solves a PoW puzzle based on an epoch randomness obtained from the last state of the blockchain. The PoW's least-significant bits are used to determine the committees which coordinate with each other to process transactions.

While Elastico can improve the throughput and latency of Bitcoin by several orders of magnitude, it still has several drawbacks: (1) Elastico requires all parties to re-establish their identities (*i.e.*, solve PoWs) and re-build all committees in "every" epoch. Aside from a relatively large communication overhead, this incurs a significant latency that scales linearly with the network size as the protocol requires more time to solve enough PoWs to fill up all committees. (2) In practice, Elastico requires a small committee size (about 100 parties) to limit the overhead of running PBFT in each committee. Unfortunately, this increases the failure probability of the protocol significantly and, using a simple analysis (see [42]), this probability can be as high as 0.97 after only six epochs, rendering the protocol completely insecure in practice.

(3) The randomness used in each epoch of Elastico can be biased by an adversary, and hence, compromise the committee selection process and even allow malicious nodes to precompute PoW puzzles. (4) Elastico requires a trusted setup for generating an initial common randomness that is revealed to all parties at the same time. (5) While Elastico allows each party to only verify a subset of transactions, it still has to broadcast all blocks to all parties and requires every party to store the entire ledger. (6) Finally, Elastico can only tolerate up to a $1/4$ fraction faulty parties even with a high failure probability. Elastico requires this low resiliency bound to allow practical committee sizes.

2.2.3 OmniLedger

In a more recent work, Kokoris-Kogias *et al.* [42] propose OmniLedger, a sharding-based distributed ledger protocol that attempts to fix some of the issues of Elastico. Assuming a slowly-adaptive adversary that can corrupt up to a

1/4 fraction of the nodes at the beginning of each epoch, the protocol runs a global reconfiguration protocol at every epoch (about once a day) to allow new participants to join the protocol.

The protocol generates identities and assigns participants to committees using a slow identity blockchain protocol that assumes synchronous channels. A fresh randomness is generated in each epoch using a bias-resistant random generation protocol that relies on a verifiable random function (VRF) [51] for unpredictable leader election in a way similar to the lottery algorithm of Algorand [50]. The consensus protocol assumes partially-synchronous channels to achieve fast consensus using a variant of ByzCoin [41], where the epoch randomness is further used to divide a committee into smaller groups. The ByzCoin's design is known to have several security/performance issues [56, 4], notably that it falls back to all-to-all communication in the Byzantine setting. Unfortunately, due to incomplete (and changing) specification of the new scheme, it is unclear how the new scheme used in OmniLedger can address these issues.

Furthermore, there are several challenges that OmniLedger leaves unsolved: (1) Similar to Elastico, OmniLedger can only tolerate $t < n/4$ corruptions. In fact, the protocol can only achieve low latency (less than 10 seconds) when $t < n/8$. (2) OmniLedger's consensus protocol requires $O(n)$ per-node communication as each committee has to gossip multiple messages to all n nodes for each block of transaction. (3) OmniLedger requires a trusted setup to generate an initial unpredictable configuration to "seed" the VRF in the first epoch. Trivial algorithms for generating such a common seed require $\Omega(n^2)$ bits of communication. (4) OmniLedger requires the user to participate actively in cross-shard transactions which is often a strong assumption for typically light-weight users. (5) Finally, OmniLedger seems vulnerable to denial-of-service (DoS) attacks by a malicious user who can lock arbitrary transactions leveraging the atomic cross-shard protocol.

When $t < n/4$, OmniLedger can achieve a high throughput (*i.e.*, more than 500 tx/sec) only when an optimistic trust-but-verify approach is used to trade-off between throughput and transaction confirmation latency. In this approach, a set of optimistic validators process transactions quickly providing provisional commitments that are later verified by a set of core validators. While such an approach seems useful for special scenarios such as micropayments to quickly process low-stake small transactions, it can be considered as a high-risk approach in regular payments, especially due to the lack of financial liability mechanisms in today's decentralized systems. Nevertheless, any blockchain protocol (including Bitcoin's) has a transaction confirmation latency that has to be considered in practice to limit the transaction risk.

2.3 Synchronous Consensus

The widely-used Byzantine consensus protocol of Castro and Liskov [20], known as PBFT, can tolerate up to $t < n/3$ corrupt nodes in the authenticated setting (*i.e.*, using digital signatures) with asynchronous communication channels. While asynchronous Byzantine consensus requires $t < n/3$ even with digital signatures [15], synchronous consensus can be solved with $t < n/2$ using digital signatures. Recently, Ren *et al.* [60] propose an expected constant-round algorithm for Byzantine consensus in a synchronous, authenticated communication network, where up to $t < n/2$ nodes can be corrupt. While the best known previous result, due to Katz and Koo [37], requires 24 rounds of communication in expectation, the protocol of Ren *et al.* requires only 8 rounds in expectation.

Assuming a random leader-election protocol exists, the protocol of Ren *et al.* runs in iterations with a new unique leader in every iteration. If the leader is honest, then the consensus is guaranteed in that iteration. Otherwise, the Byzantine leader can prevent progress but cannot violate safety, meaning that some honest nodes might not terminate at the end of the iteration but all honest nodes who terminate in that iteration will output the same value, called the *safe value*. If at least one node can show to the new leader that has decided on a safe value, then the new leader proposes the same value in the next iteration. Otherwise, the new leader proposes a new value.

2.4 Information Dispersal Algorithms

Rabin [58] introduces the notion of information dispersal algorithms (IDA) that can split a message (or file) into multiple chunks in such a way that a subset of them will be sufficient to reconstruct the message. This is achieved using erasure codes [11] as a particular case of error-correcting codes (ECC) allowing some of the chunks to be missing but not modified. Krawczyk [43] extends this to tolerate corrupted (*i.e.*, altered) chunks by computing a fingerprint for each chunk and storing the vector of fingerprints using ECC. Alon *et al.* [6, 5] describe a more-efficient IDA mechanism by computing a Merkle hash tree [49] over encoded chunks in order to verify whether each of the received chunks is corrupted.

In RapidChain, we build on the IDA of Alon *et al.* [5] to perform efficient gossips on large blocks within each committee. Once an ECC-encoded message is dispersed in the network via IDA, honest nodes agree on the root of the Merkle tree using the intra-committee consensus protocol to ensure consistency. Using the corresponding authentication path in the Merkle tree sent by the sender, recipients can verify the integrity of all chunks and use a decoding mechanism to recover the message (see Section 4.2 for more details).

3 Model and Problem Definition

Network Model. Consider a peer-to-peer network with n nodes who establish identities (*i.e.*, public/private keys) through a Sybil-resistant identity generation mechanism such as that of [7], which requires every node to solve a computationally-hard puzzle on their locally-generated identities (*i.e.*, public keys) verified by all other (honest) nodes without the assumption of a trusted randomness beacon. Without loss of generality and similar to most hybrid blockchain protocols [23, 56, 47, 42], we assume all participants in our consensus protocol have equivalent computational resources.

We assume all messages sent in the network are authenticated with the sender's private key. The messages are propagated through a synchronous gossip protocol [36] that guarantees a message sent by an honest node will be delivered to all honest nodes within a known fixed time, Δ, but the order of these messages is not necessarily preserved. This is the standard synchronous model adopted by most public blockchain protocols [47, 32, 42, 4]. We require synchronous communication only during our intra-committee consensus. In other parts of our protocol, we assume partially-synchronous channels [20] between nodes with exponentially-increasing time-outs (similar to [42]) to minimize latency and achieve responsiveness.

Threat Model. We assume nodes may disconnect from the network during an epoch or between two epochs due to any reason such as internal failure or network jitter. We also consider a probabilistic polynomial-time Byzantine adversary who corrupts $t < n/3$ of the nodes at any time. The corrupt nodes not only may collude with each other but also can deviate from the protocol in any arbitrary manner, *e.g.*, by sending invalid or inconsistent messages, or remaining silent. Similar to most committee-based protocols [23, 41, 56, 42, 4], we assume the adversary is *slowly adaptive*, meaning that it is allowed to select the set of corrupt nodes at the beginning of the protocol and/or between each epoch but cannot change this set within the epoch.

At the end of each epoch, the adversary is allowed to corrupt a constant (and small) number of uncorrupted nodes while maintaining their identities. In addition, the adversary can run a join-leave attack [25, 8], where it rejoins a constant (and small) number of corrupt nodes using fresh identities in order to compromise one or more committees. However, at any moment, at least a 2/3 fraction of the computational resources belong to uncorrupted participants that are online (*i.e.*, respond within the network time bound). Finally, we do not rely on any public-key infrastructure or any secure broadcast channel, but we assume the existence of a cryptographic hash function, which we model as a random oracle for our security analysis.

Problem Definition. We assume a set of transactions are sent to our protocol by a set of *users* that are external to the protocol. Similar to Bitcoin [54], a transaction consists of a set of inputs and outputs that reference other transactions, and a signature generated by their issuer to certify its validity. The set of transactions is divided into k disjoint *blocks*. Let $x_{i,j}$ represent the j-th transaction in the i-th block. All nodes have access to an external function g that, given any transaction, outputs 0 or 1 indicating whether the transaction is invalid or not respectively, *e.g.*, the sum of all outputs of a transaction is equal to the sum of its inputs. The protocol Π outputs a set X containing k disjoint subsets or *shards* $X_i = \{x_{i,j}\}$, for every $j \in \{1..|X_i|\}$ such that the following conditions hold:

- *Agreement:* For every $i \in \{1..k\}$, $\Omega(\log n)$ honest nodes agree on X_i with a high probability of at least $1 - 2^{-\lambda}$, where λ is the security parameter.
- *Validity:* For every $i \in \{1..k\}$ and $j \in \{1..|X_i|\}$, $g(x_{i,j}) = 1$.
- *Scalability:* k grows linearly with n.
- *Efficiency:* The per-node communication and computation complexity is $o(n)$ and the per-node storage complexity is $o(s)$, where s is the total number of transactions.

4 Our Protocol

In this section, we present RapidChain in detail. We start by defining notations and terms used in the rest of the paper.

Notation and Terminology. Let n denote the total number of nodes and $t < n/3$ denote the total number of corrupt nodes. We say an event occurs *with high probability* meaning that it occurs with probability $1 - O(1/2^{\lambda})$, where λ is the security parameter. We refer to any set of $m = o(n)$ nodes as a *committee* if at least an $f < 1/2$ fraction of its members belongs to honest nodes. Let node P be a member of a group C. We refer to other members of C as the *neighbors* of P in C. When we say a committee runs a protocol, we mean all honest members of the committee participate in an execution of the protocol. Let C_1, C_2 be two committees. When we say C_1 sends a message M to C_2, we mean every honest member of C_1 sends M to *every* member of C_2 who he knows. Since each member of C_2 may receive different messages due to malicious behavior, it chooses the message with a frequency of at least $1/2 + 1$.

4.1 Design Components

RapidChain consists of three main components: Bootstrap, Consensus, and Reconfiguration. The protocol starts with Bootstrap and then proceeds in epochs, where each epoch consists of multiple iterations of Consensus followed by a Reconfiguration phase. We now explain each component in more details.

Bootstrapping. The initial set of participants start RapidChain by running a *committee election* protocol, where all nodes agree on a group of $O(\sqrt{n})$ nodes which we refer to as the *root group*. The group is responsible for generating and distributing a sequence of random bits that are used to establish a reference committee of size $O(\log n)$. Next, the reference committee creates k committees $\{C_1, ..., C_k\}$ each of size $O(\log n)$. The bootstrap phase runs only once at the start RapidChain.

Consensus. Once members of each committee are done with the epoch reconfiguration, they wait for external users to submit their transactions. Each user sends its transactions to a subset of nodes (found via a P2P discovery protocol) who batch and forward the transactions to the corresponding committee responsible for processing them. The committee runs an intra-committee consensus protocol to approve the transaction and add it to its ledger.

Reconfiguration. Reconfiguration allows new nodes to establish identities and join the existing committees while ensuring all the committees maintain their 1/2 resiliency. In Section 4.5, we describe how to achieve this goal using the Cuckoo rule [62] without regenerating all committees.

In the following, we first describe our Consensus component in Section 4.2 assuming a set of committees exists. Then, we describe how cross-shard transactions can be verified in Section 4.3, and how committees can communicate with each other via an inter-committee routing protocol in Section 4.4. Next, we describe the Reconfiguration component in Section 4.5, and finally, finish this section by describing how to bootstrap the committees in Section 4.6.

4.2 Consensus in Committees

Our intra-committee consensus protocol has two main building blocks: (1) A gossiping protocol to propagate the messages (such as transactions and blocks) within a committee; (2) A synchronous consensus protocol to agree on the header of the block.

4.2.1 Gossiping Large Messages

Inspired by the IDA protocol of [5], we refer to our gossip protocol for large messages as *IDA-Gossip*. Let M denote the message to be gossiped to d neighbors, ϕ denote the fraction of corrupt neighbors, and κ denote the number of chunks of the large message. First, the sender divides M into $(1 - \phi)\kappa$-equal sized chunks $M_1, M_2, \ldots, M_{(1-\phi)\kappa}$ and applies an erasure code scheme (*e.g.*, Reed-Solomon erasure codes [59]) to create an additional $\phi\kappa$ parity chunk to obtain $M_1, M_2, \ldots, M_{\kappa}$. Now, if the sender is honest, the original message can be reconstructed from any set of $(1 - \phi)\kappa$ chunks.

Next, the source node computes a Merkle tree with leaves $M_1, \ldots M_{\kappa}$. The source gossips M_i and its Merkle proof, for all $1 \leq i \leq \kappa$, by sending a unique set of κ/d chunks (assuming κ is divisible by d) to each of its neighbors. Then, they gossip the chunks to their neighbors and so on. Each node verifies the message it receives using the

Merkle tree information and root. Once a node receives $(1-\phi)\kappa$ valid chunks, it reconstructs the message M, *e.g.*, using the decoding algorithm of Berlekamp and Welch [10].

Our *IDA-Gossip* protocol is not a reliable broadcast protocol as it cannot prevent equivocation by the sender. Nevertheless, IDA-Gossip requires much less communication and is faster than reliable broadcast protocols (such as [14]) to propagate large blocks of transactions (about 2 MB in RapidChain). To achieve consistency, we will later run a consensus protocol only on the root of the Merkle tree after gossiping the block.

Improvement via Sparsification. Let $h(M_i)$ denote the hash of M_i, and $h(M_i, M_j)$ denote the digest stored at the first common parent of M_i and M_j. Let $\mathsf{Sib}(M_i)$ denote all sibling nodes of the nodes on the path from the root to M_i. Thus, $\{h(M_i), h(M_1, M_\kappa), \mathsf{Sib}(M_i)\}$ is the Merkle proof to verify the validity of M_i. We further optimize the IDA-Gossip based on the observation that if the source sends $\mathsf{Sib}(M_i)$ to P_i for every i, the Merkle hashes near the root of the tree are sent to almost all the nodes. For example, half of the nodes receive $h(M_1, M_{\kappa/2})$. Instead, for any intermediate digest $h(M_i, M_j)$, it is sufficient to send it to a smaller subset of the nodes.

In RapidChain, the source chooses a random subset of size $d/(1-\phi)$ of its neighbors and only sends the digest to the node in that subset. We refer to this process as *sparsification.* As a result, a node may receive a message from the source that does not contain all of the digests needed to verify the validity of the gossiped message. Therefore, the node may not be able to validate the message immediately. However, since at least one honest node receives each intermediate digest, it will forward the digest. This guarantees that all the node will have all the correct intermediate digests.

In Section 6.2, we show that if an honest node starts the IDA-Gossip protocol for message M in a committee, all honest nodes in that committee will receive M correctly with high probability. We also show that, using sparsification, it suffices to send each intermediate digest to a number of node sublinear in the depth of the tree. This guarantees that all nodes can verify the message with high probability.

4.2.2 Remarks on Synchronous Consensus

In RapidChain, we use a variant of the synchronous consensus protocol of Ren *et al.* [60] to achieve optimal resiliency of $f < 1/2$ in committees and hence, allow smaller committee sizes with higher total resiliency of 1/3 (than previous work [47, 42]).

Unlike asynchronous protocols such as PBFT [20], the protocol of Ren *et al.* [60] (similar to most other synchronous protocol) is not *responsive* [56] meaning that it commits to messages at a fixed rate (usually denoted by Δ) and thus, its speed is independent of the actual delay of the network. Most committee-based protocols (such as [4]) run a PBFT-based intra-committee consensus protocol, and hence, are responsive within epochs. However, this often comes at a big cost that almost always results in significantly poor throughput and latency, hindering responsiveness anyway. Since asynchronous consensus requires $t < n/3$, one needs to assume a total resiliency of roughly 1/5 or less to achieve similar committee size and failure probability when sampling a committee with 1/3 resiliency (see Figure 7). Unfortunately, increasing the total resiliency (*e.g.*, to 1/4) will dramatically increase the committee size (*e.g.*, 3-4x larger) making intra-committee consensus significantly inefficient.

In RapidChain, we use our synchronous consensus protocol to agree only on a digest of the block being proposed by one of the committee members. As a result, the rest of our protocol can be run over partially-synchronous channels with optimistic timeouts to achieve responsiveness (similar to [42]). In addition, since the synchronous consensus protocol is run among only a small number of nodes (about 250 nodes), and the size of the message to agree is small (roughly 80 bytes), the latency of each round of communication is also small in practice (about 500 ms – see Figure 4–left) resulting in a small Δ (about 600 ms) and a small epoch latency.

To better alleviate the responsiveness issue of synchronous consensus, RapidChain runs a pre-scheduled consensus among committee members about every week to agree on a new Δ so that the system adjusts its consensus speed with the latest average delay of the network. While this does not completely solve the responsiveness problem, it can make the protocol responsive to long-term, more robust changes of the network as technology advances.

Another challenge in using a synchronous consensus protocol happens in the cross-shard transaction scenario, where a malicious leader can deceive the input committee with a transaction that has been accepted by some but not all honest members in the output committee. This can happen because, unlike asynchronous consensus protocols such as PBFT [20] that proceed in an event-driven manner, synchronous consensus protocols proceed in fixed rounds, and hence, some honest nodes may terminate before others with a “safe value” that yet needs to be accepted by all honest nodes in future iterations before a transaction can be considered as committed.

4.2.3 Protocol Details

At each iteration i, each committee picks a leader randomly using the epoch randomness. The leader is responsible for driving the consensus protocol. First, the leader gathers all the transactions it has received (from users or other committees) in a block B_i. The leader gossips the block using IDA-gossip and creates the block header H_i that contains the iteration number as well as the root of the Merkle tree from IDA-Gossip. Next, the leader initiates consensus protocol on H_i. Before describing the consensus protocol, we remark that all the messages that the leader or other nodes send during the consensus is signed by their public key and thus the sender of the message and its integrity is verified.

Our consensus protocol consists of four synchronous rounds. First, the leader gossips a messages containing H_i and a tag in the header of the message that the leader sets it to *propose*. Second, all other nodes in the network echo the headers they received from the leader, *i.e.*, they gossip H_i again with the tag *echo*. This step ensures that all the honest nodes will see all versions of the header that other honest nodes received in the first round. Thus, if the leader equivocates and gossips more than one version of the message, it will be noticed by the honest nodes. In the third round, if an honest node receives more than one version of the header for iteration i, it knows that the leader is corrupt and will gossip H'_i with the tag *pending*, where H'_i contains a null Merkle root and iteration number i.

Finally, in the last round, if an honest node receives $mf+1$ echoes of the same and the only header H_i for iteration i, it accepts H_i and gossips H_i with the tag *accept* along with all the $mf + 1$ echoes of H_i. The $mf + 1$ echoes serve as the proof of why the node accepts H_i. Clearly, it is impossible for any node to create this proof if the leader has not gossiped H_i to at least one honest node. If an honest node accepts a header, then all other honest nodes either accept the same header or they reject any header from the leader. In the above scenario, if the leader is corrupt, then some honest nodes reject the header and tag it as *pending*.

Definition 1 (Pending Block). *A block is pending at iteration i if it is proposed by a leader at some iteration j before i, while there are honest nodes that have not accepted the block header at iteration i.*

Since less than $m/2$ of the committee members are corrupt, the leader will be corrupt with a probability less than $1/2$. Thus, to ensure a block header gets accepted, two leaders have to propose it in expectation. One way to deal with this issue is to ask the leader of the next iteration to propose the same block again if it is still pending. This, however, reduces the throughput by roughly half.

4.2.4 Improving Performance via Pipelining

RapidChain allows a new leader to propose a new block while re-proposing the headers of the pending blocks. This allows RapidChain to pipeline its consensus iterations, maximizing its throughput. Since the consensus is happening during multiple iterations, we let nodes count votes for the header proposed in each iteration to determine if a block is *pending* or *accepted*. The votes can be permanent or temporary relative to the current iteration. If a node gossips an *accept* for header H_j at any iteration $i \geq j$, its permanent vote for the header of iteration j is H_j. If the node sends two *accept*s for two different headers H_j and H'_j, then the honest nodes will ignore the vote.

If a node sends an *echo* for H_j at any iteration $i \geq j$, its temporary vote is H_j in iteration i. To accept a header, a node requires at least $mf + 1$ votes (permanent or temporary for the current iteration). If a node accepts a header, it will not gossip more headers since all nodes already know its vote. This will protect honest nodes against denial-of-service attacks by corrupt leaders attempting to force them echo a large number of non-pending blocks.

It is left to describe how the leader proposes a header for a pending block even if some honest nodes might have already accepted a value for it. A new proposal is *safe* if it does not conflict with any accepted value with a correct proof, if there is any. Thus, at iteration i, for all pending block headers, the leader proposes a *safe* value. For a new block, any value is considered safe while for a pending block of previous iterations, the value is safe if and only if it has a correct proof of at least $mf + 1$ votes (permanent or temporary from iteration $i - 1$). If there is no value with enough votes, then any value is safe. In Section 6.3, we prove that our consensus protocol achieves safety and liveness in a committee with honest majority.

4.3 Cross-Shard Transactions

In this section, we describe a mechanism by which RapidChain reduces the communication, computation, and storage requirement of each node by dividing the blockchain into partitions each stored by one of the committees. While

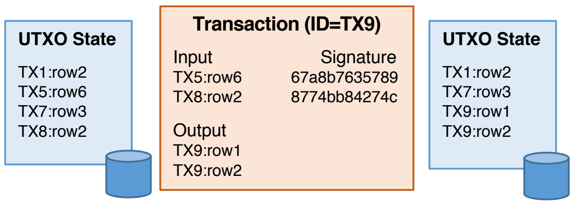


Figure 1: UTXO states before and after a transaction

sharding the blockchain can reduce the storage overhead of the blockchain, it makes the verification of transactions challenging, because the inputs and outputs of each transaction might reside in multiple committees.

Similar to Bitcoin, each transaction in RapidChain has a unique identity, a list of inputs (depicted by their identities), and a list of outputs that is shown by the transaction ID and their row number (see Figure 1). All inputs to a transaction must be *unspent transaction outputs (UTXOs)* which are unused coins from previous transactions. The outputs of the transaction are new coins generated for the recipients of the exchanged money. After receiving a transaction, the nodes verify if a transaction is valid by checking (1) if the input is unspent; and (2) if the sum of outputs is less than the sum of the inputs. The nodes add the valid transaction to the next block they are accepting. RapidChain partitions the transactions based on their transaction ID among the committees which will be responsible for storing the transaction outputs in their UTXO databases. Each committee only stores transactions that have the committee ID as their prefix in their IDs.

Let tx denote the transaction sent by the user. In the verification process, multiple committees may be involved to ensure all the input UTXOs to tx are valid. We refer to the committee that stores tx and its possible UTXOs as the *output committee*, and denote it by C_{out}. We refer to the committees that store the input UTXOs to tx as the *input committees*, and denoted them by $C_{\text{in}}^{(1)}, \ldots, C_{\text{in}}^{(N)}$.

To verify the input UTXOs, OmniLedger [42] proposes that the user obtain a proof-of-acceptance from every input committee and submit the proof to the output committee for validation. If each input committee commits to tx (and marks the corresponding input UTXO as "spent") independently from other input committees, then tx may be committed partially, *i.e.*, some of its inputs UTXOs are spent while the others are not. To avoid this situation and ensure transaction atomicity, OmniLedger takes a two-phase approach, where each input committee first locks the corresponding input UTXO(s) and issues a proof-of-acceptance, if the UTXO is valid. The user collects responses from all input committees and issues an “unlock to commit”.

While this allows the output committee to verify tx independently, the transaction has to be gossiped to the entire network and one proof needs to be generated for every transaction, incurring a large communication overhead. Another drawback of this scheme is that it depends on the user to retrieve the proof which puts extra burden on typically lightweight user nodes.

In RapidChain, the user does not attach any proof to tx. Instead, we let the user communicate with any committee who routes tx to the output committee via the inter-committee routing protocol. Without loss of generality, we assume tx has two inputs I_1, I_2 and one output O. If I_1, I_2 belong to different committees other than C_{out}, then the leader of C_{out}, creates three new transactions: For $i \in \{1,2\}$, tx_i with input I_i and output I'_i, where $|I'_i| = |I_i|$ (*i.e.*, the same amounts) and I'_i belongs to C_{out}. tx_3 with inputs I'_1 and I'_2 and output O. The leader sends tx_i to C_{in}^i via the inter-committee routing protocol, and C_{in}^i adds tx_i to its ledger. If tx_i is successful, C_{in}^i sends I'_i to C_{out}. Finally, C_{out} adds tx_3 to its ledger.

Batching Verification Requests. At each round, the output committee combines the transactions that use UTXOs belonging to the same input committee into batches and sends a single UTXO request to the input committee. The input committee checks the validity of each UTXO and sends the result of the batch to the output committee. Since multiple UTXO requests are batched into the same request, a result can be generated for multiple requests at the input committee.

4.4 Inter-Committee Routing

RapidChain requires a routing scheme that enables the users and committee leaders to quickly locate to which committees they should send their transactions.

Strawman Scheme. One approach is to require every node to store the network information of all committee mem-

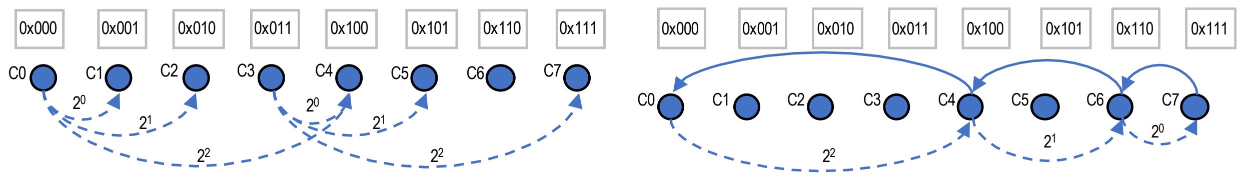


Figure 2: (Left) Each committee in RapidChain maintains a routing table containing $\log n$ other committees. (Right) Committee C_0 wants to locate committee C_7 (via C_4 and C_6) responsible for transactions with prefix $0x111$.

bers in the network. This allows every node to quickly locate the IP addresses of members of any committee in constant time. Then, nodes can create a connection to members of the target committee and gossip among them. However, this requires every node to store the network information about all committee members that compromise privacy and simplify the denial of service attack. Moreover, in practice each node should connect to large number of nodes during his life time which cannot scale for thousands of nodes in the network.

A different solution is to have a dedicated committee (*e.g.*, the reference committee) to be responsible for transaction routing. Every user will obtain network information from reference committee. This approach offers efficient routing, which takes only one communication round. However, reference committee becomes a centralized hub of the network that needs to handle a large amount of communication and thus will be a likely bottleneck.

Routing Overlay Network. To construct the routing protocol in RapidChain, we use ideas from the design of the routing algorithm in Kademlia [48]. In Kademlia, each node in the system is assigned an identifier and there is a metric of distance between identifiers (for example, the Hamming distance of the identifiers). A node stores information about all nodes which are within a logarithmic distance. When a node wants to send a message to another node in the system it identifies the node among its neighbors (which it stores locally) that is closest to the destination node's Kademlia ID (or KID) and it asks it to run recursively the discovery mechanism. This enables node discovery and message routing in $\log n$ steps. We refer the reader to Section 2 of [48] for more details about the Kademlia routing protocol.

We employ the Kademlia routing mechanism in RapidChain at the level of committee-to-committee communication. Specifically, each RapidChain committee maintains a routing table of $\log n$ records which point to $\log n$ different committees which are distance 2^i for $0 \leq i \leq \log n - 1$ away (see Figure 2 for an example). More specifically, each node stores information about all members of its committee as well as about $\log\log(n)$ nodes in each of the $\log n$ closest committees to its own committee. Each committee-to-committee message is implemented by having all nodes in the sender committee send the message to all nodes they know in the receiver committee. Each node who receives a message invokes the IDA-gossip protocol to sent the message to all other members of its committee.

When a user wants to submit a transaction, it sends the transaction to any arbitrary RapidChain node who will forward it to the corresponding committee via the Kademlia routing mechanism. We present in Figure 2 an example of the routing protocol initiated by committee C_0 to request information for committee C_7.

4.5 Committee Reconfiguration

Protocol 1 presents our reconfiguration protocol. In the following, we describe the core techniques used in this protocol.

4.5.1 Offline PoW

RapidChain relies on PoW only to protect against Sybil attacks by requiring every node who wants to join or stay in the protocol to solve a PoW puzzle. In each epoch, a fresh puzzle is generated based on the epoch randomness so that the adversary cannot precompute the solutions ahead of the time to compromise the committees. In RapidChain, all nodes solve a PoW offline without making the protocol stop and wait for the solution. Thus, the expensive PoW calculations are performed off the critical latency path.

Since the adversary is bounded to a $1/3$ fraction of the total computation power during each epoch, the fraction of total adversarial nodes is strictly less than $n/3$. In RapidChain, the reference committee (C_R) is responsible to check the PoW solutions of all nodes. At the start of each epoch, C_R agrees on a *reference block* consisting of the list

Protocol 1 Epoch Reconfiguration

1. **Random generation during epoch** $i - 1$
   (a) The reference committee (C_R) runs the DRG protocol to generate a random string r_i for the next epoch.
   (b) Members of C_R reveal r_i at the end of epoch $i - 1$.
2. **Join during epoch** i
   (a) Invariant: All committees at the start of round i receive the random string r_i from C_R.
   (b) New nodes locally choose a public key PK and contact a random committee C to request a PoW puzzle.
   (c) C sends the r_i for the current epoch along with a timestamp and 16 random nodes in C_R to P.
   (d) All the nodes who wish to participate in the next epoch find x such that $O = \mathsf{H}(\texttt{timestamp}||\mathsf{PK}||r_i||x) \leq 2^{\gamma-d}$ and sends x to C_r.
   (e) C_r confirms the solution if it received it before the end of the epoch i.
3. **Cuckoo exchange at round** $i + 1$
   (a) Invariant: All members of C_R participate in the DRG protocol during epoch i and have the value r_{i+1}.
   (b) Invariant: During the epoch i, all members of C_R receive all the confirmed transactions for the active nodes of round $i + 1$.
   (c) Members of C_r will create the list of all active nodes for round $i + 1$ and also create A, the set of active committees, and I, the set of inactive committees.
   (d) C_R uses r_{i+1} to assign a committees in A for each new node.
   (e) For each committee C, C_R evicts a constant number of nodes in C uniformly at random using r_{i+1} as the seed.
   (f) For all the evicted nodes, C_R chooses a committee uniformly at random using r_{i+1} as the seed and assigns the node to the committee.
   (g) C_R adds r_i and the new list of all the members and their committees and add it as the first block of the epoch to the C_R's chain.
   (h) C_R gossips the first block to all the committees in the system using the inter-committee routing protocol.

of all active nodes for that epoch as well as their assigned committees. C_R also informs other committees by sending the reference block to all other committees.

4.5.2 Epoch Randomness Generation

In each epoch, the members of the reference committee run a *distributed random generation (DRG)* protocol to agree on an unbiased random value. C_R includes the randomness in the reference block so other committees can randomize their epochs. RapidChain uses a well-known technique based on the verifiable secret sharing (VSS) of Feldman [30] to generate unbiased randomness within the reference committee.

Let $\mathbb{F}_p$ denote a finite field of prime order p, m denote the size of the reference committee, and r denote the randomness for the current epoch to be generated by the protocol. For all $i \in [m]$, node i chooses $\rho_i \in \mathbb{F}_p$ uniformly at random and VSS-shares it to all other node. Next, for all $j \in [m]$, let $\rho_{1j}, ..., \rho_{mj}$ be the shares node j receives from the previous step. Node j computes its share of r by calculating $\sum_{l=1}^{m} \rho_{lj}$. Finally, nodes exchange their shares of r and reconstruct the result using the Lagrange interpolation technique [35]. The random generation protocol consists of two phases: sharing and reconstruction. The sharing phase is more expensive in practice but is executed in advance before the start of the epoch.

Any new node who wishes to join the system can contact any node in any committees at any time and request the randomness of this epoch as a fresh PoW puzzle. The nodes who solve the puzzle will send a transaction with their solution and public key to the reference committee. If the solution is received by the reference committee before the end of the current epoch, the solution is accepted and the reference committee adds the node to the list of active nodes for the next epoch.

4.5.3 Committee Reconfiguration

Partitioning the nodes into committees for scalability introduces a new challenge when dealing with churn. Corrupt nodes could strategically leave and rejoin the network, so that eventually they can take over one of the committees and break the security guarantees of the protocol. Moreover, the adversary can actively corrupt a constant number of uncorrupted nodes in every epoch even if no nodes join/rejoin.

One approach to prevent this attack is to re-elect all committees periodically faster than the adversary's ability to generate churn. However, there are two drawbacks to this solution. First, re-generating all of the committees is very expensive due to the large overhead of the bootstrapping protocol (see Section 5). Second, maintaining a separate ledger for each committee is challenging when several committee members may be replaced in every epoch.

To handle this problem, RapidChain uses a modified version of the Cuckoo rule [8, 62], which we refer to as the *bounded Cuckoo rule*, to re-organize only a subset of committee members during the reconfiguration event at the beginning of each epoch. This modification is to ensure that committees are balanced with respect to their sizes as nodes join or leave the network. In the following, we first describe the basic Cuckoo rule algorithm and then, proceed to the bounded cuckoo rule.

Cuckoo Rule. To map the nodes to committees, we first map each node to a random position in $[0, 1)$ using a hash function. Then, the range $[0, 1)$ is partitioned into k regions of size k/n, and a committee is defined as the group of nodes that are assigned to $O(\log n)$ regions, for some constants k. Awerbuch and Scheideler [8] propose the Cuckoo rule as a technique to ensure the set of committees created in the range $[0, 1)$ remain robust to join-leave attacks. Based on this rule, when a node wants to join the network, it is placed at a random position $x \in [0, 1)$, while all nodes in a constant-sized interval surrounding x are moved (or *cuckoo*'ed) to new random positions in $(0, 1]$. Awerbuch and Scheideler prove that given $\epsilon < 1/2 - 1/k$ in a steady state, all regions of size $O(\log n)/n$ have $O(\log n)$ nodes (*i.e.*, they are balanced) of which less than 1/3 are faulty (*i.e.*, they are correct), with high probability, for any polynomial number of rounds.

A node is called *new* while it is in the committee where it was assigned when it joined the system. At any time after that, we call it an *old* node even if it changes its committee. We define the age of a k-region as the amount of time that has passed after a new node is placed in that k-region. We define the age of a committee as the sum of the ages of its k-regions.

Bounded Cuckoo Rule. At the start of each epoch, once the set of active nodes who remain in the protocol for the new epoch is defined, the reference committee, C_R, defines the set of the largest $m/2$ committees (who have more active members) as the *active committee set*, which we denote by A. We refer to the remaining $m/2$ committees with smaller sizes as the *inactive committee set*, denoted by I. Active committees accept new nodes that have joined the network in the previous epoch, as new members of the committee. However, inactive committees only accept the members who were part of the network before, to join them. Both active and inactive committees fulfill any other responsibilities they have in the protocol (such as consensus on blocks and routing transaction) indifferently. For each new node, the reference committee, C_R, chooses a random committee C_a from the set A and adds the new node to C_a. Next, C_R evicts (cuckoos) a constant number of members from every committee (including C_a) and assigns them to other committees chosen uniformly at random from I.

In Section 6.5, we show two invariant properties which are maintained for each committee during the reconfiguration protocol: At any moment, the committees are *balanced* and *honesty*. The first property ensures a bounded deviation in the sizes of the committees. The second property ensures that each committee maintains its honest majority.

4.5.4 Node Initialization and Storage

Once a node joins a committee, it needs to download and store the state of the new committee. We refer to this as the *node initialization* process. Moreover, as transactions are processed by the committee, new data has to be stored by the committee members to ensure future transactions can be verified. While RapidChain shards the global ledger into smaller ones each maintained by one committee, the initialization and storage overhead can be large and problematic in practice due to the high throughput of the system. One can employ a ledger pruning/checkpointing mechanism, such as those described in [45, 42], to significantly reduce this overhead. For example, a large percentage of storage is usually used to store transactions that are already spent.

In Bitcoin, new nodes download and verify the entire history of transactions in order to find/verify the longest (*i.e.*, the most difficult) chain[1]. In contrast, RapidChain is a BFT-based consensus protocol, where the blockchain maintained by each committee grows based on member votes rather than the longest chain principle [31]. Therefore, a new RapidChain node initially downloads only the set of unspent transactions (*i.e.*, UTXOs) from a sufficient number of committee members in order to verify future transactions. To ensure the integrity of the UTXO set received by the new node, the members of each committee in RapidChain record the hash of the current UTXO set in every block added to the committee's blockchain.

4.6 Decentralized Bootstrapping

Inspired by [40], we first construct a deterministic random graph called the *sampler graph* which allows sampling a number of groups such that the distribution of corrupt nodes in the majority of the groups is within a δ fraction of the number of corrupt nodes in the initial set. At the bootstrapping phase of RapidChain, a sampler graph is created locally by every participant of the bootstrapping protocol using a hard-coded seed and the initial network size which is known to all nodes since we assume the initial set of nodes have already established identities.

Sampler Graph. Let $G(L, R)$ be a random bipartite graph, where the degree of every node in R is $d_R = O(\sqrt{n})$. For each node in R, its neighbors in L are selected independently and uniformly at random without replacement. The vertices in L represent the network nodes and the vertices in R represent the groups. A node is a member of a group if they are connected in graph G. Let $T \subseteq L$ be the largest coalition of faulty nodes and $S \subseteq R$ be any subset of groups. Let $\mathcal{E}(T, S)$ denote the event that every group in S has more than a $\frac{|T|}{|L|} + \delta$ fraction of its edges incident to nodes in T. Intuitively, $\mathcal{E}$ captures the event that all groups in S are "bad", *i.e.*, more than a $\frac{|T|}{|L|} + \delta$ fraction of their parties are faulty.

In Section 6.4, we prove that the probability of $\mathcal{E}(T, S)$ is less than $2e^{(|L|+|R|)\ln 2 - \delta^2 d_R |S|/2}$. We choose practical values for $|R|$ and d_R such that the failure probability of our bootstrap phase, *i.e.*, the probability of $\mathcal{E}(T, S)$, is minimized. For example, for 4,000 nodes (*i.e.*, $|L|$ = 4,000), we set $d_R = 828$ (*i.e.*, a group size of 828 nodes) and $|R| = 642$ to get a failure probability of $2^{-26.36}$. In Section 6.1, we use this probability to bound the probability that each epoch of RapidChain fails.

Once the groups of nodes are formed using the sampler graph, they participate in a randomized election procedure. Before describing the procedure, we describe how a group of nodes can agree on an unbiased random number in a decentralized fashion.

Subgroup Election. During the election, members of each group run the DRG protocol to generate a random string s and use it to elect the parties associated with the next level groups: Each node with identification ID computes $h = H(s||ID)$ and will announces itself elected if $h <= 2^{256-e}$, where H is a hash function modeled as a random oracle. All nodes sign the (ID, s) of the e elected nodes who have the smallest h and gossip their signatures in the group as a proof of election for the elected node. In practice, we set $e = 2$.

Subgroup Peer Discovery. After each subgroup election, all nodes must learn the identities of the elected nodes from each group. The elected nodes will gossip this information and a proof, that consists of $d_R/2$ signature on (ID, s) from different members of the group, to all the nodes. If more than e nodes from the group correctly announce they got elected, the group is dishonest and all honest parties will not accept any message from any elected members of that group.

Committee Formation. The result of executing the above election protocol is a group with honest majority whom we call *root group*. Root group selects the members of the first shard, *reference shard*. The reference committee partitions the set of all nodes at random into sharding committees which are guaranteed to have 1/2 honest nodes, and which store the shards as discussed in Section 4.3.

Election Network. The election network is constructed by chaining ℓ sampler graphs $\{G(L_1, R_1), ..., G(L_\ell, R_\ell)\}$ together. All sampler graphs definitions are included in the protocol specification. Initially, the n nodes are in L_1. Based on the edges in the graph, every node is assigned to a set of groups in R_1. Then, each group runs a *subgroup election protocol* (described below) to elect a random subset of its members. The elected members will then serve as the nodes in L_2 of $G(L_2, R_2)$. This process continues to the last sampler graph $G(L_\ell, R_\ell)$ at which point only a single

[1]Bitcoin nodes can, in fact, verify the longest chain by only downloading the sequence of block headers via a method called simplified payment verification described by Nakamoto [54].

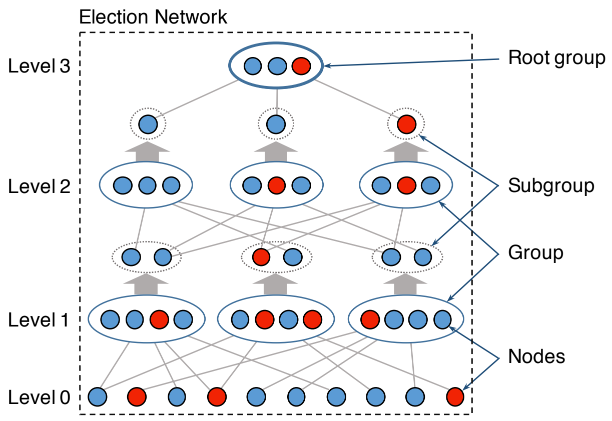


Figure 3: The election network

group is formed. We call the last group, the *leader group* and we construct the election network such that the leader group has honest majority with high probability.

To construct the election network, we set

$$|L_i| = \left\lceil |L_{i-1}|^{\alpha_i+\beta_i\gamma_i} \right\rceil, |R_i| = \lceil |L_i|^{\alpha_i} \rceil, \tag{1}$$

where $|L_1| = n, |R_1| = n^{\alpha_i}$, $0 < \alpha_i, \beta_i, \gamma_i < 1$, and $i = \{2, ..., \ell\}$. It can be easily shown that for some constant ℓ, $|R_\ell| = 1$. From Equation 3, we can bound the error probability for every level i of the election network denoted by p_i, where

$$p_i \leq 2e^{|L_i|+|R_i|-\delta^2 d_{R_i}|S_i|/2}. \tag{2}$$

In Section 6.4, we discuss how the parameters α, β and γ are chosen to instantiate such an election graph and present a novel analysis that allows us to obtain better bounds for their sizes.

5 Evaluation

Experimental Setup. We implement a prototype of RapidChain in Go[1] to evaluate its performance and compare it with previous work. We simulate networks of up to 4,000 nodes by oversubscribing a set of 32 machines each running up to 125 RapidChain instances. Each machine has a 64-core Intel Xeon Phi 7210 @ 1.3GHz processor and a 10-Gbps communication link. To simulate geographically-distributed nodes, we consider a latency of 100 ms for every message and a bandwidth of 20 Mbps for each node. Similar to Bitcoin Core, we assume each node in the global P2P network can accept up to 8 outgoing connections and up to 125 incoming connections. The global P2P overlay is only used during our bootstrapping phase. During consensus epoch, nodes communicate through much smaller P2P overlays created within every committee, where each node accepts up to 16 outgoing connections and up to 125 incoming connections.

Unless otherwise mentioned, all numbers reported in this section refer to the expected behavior of the system when less than half of all nodes are corrupted. In particular, in our implementation of the intra-consensus protocol of Section 4.2, the leader gossips two different messages in the same iteration with probability 0.49. Also, in our inter-committee routing protocol, 49% of the nodes do not participate in the gossip protocol (*i.e.*, remain silent).

To obtain synchronous rounds for our intra-committee consensus, we set Δ (see Section 3 for definition) conservatively to 600 ms based on the maximum time to gossip an 80-byte digest to all nodes in a P2P network of 250 nodes (our largest committee size) as shown in Figure 4 (left). Recall that synchronous rounds are required only during the consensus protocol of Ren *et al.* [60] to agree on a hash of the block resulting in messages of up to 80 bytes size including signatures and control bits.

[1]https://golang.org

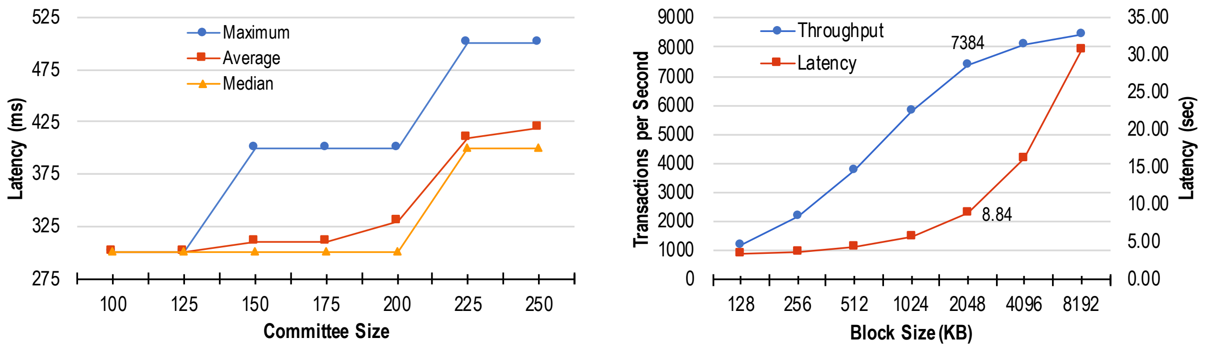


Figure 4: Latency of gossiping an 80-byte message for different committee sizes (left); Impact of block size on throughput and latency (right)

We assume each block of transaction consist of 4,096 transactions, where each transaction consists of 512 bytes resulting in a block size of 2 MB. To implement our IDA-based gossiping protocol to gossip 2-MB blocks within committees, we split each block into 128 chunks and use the Jerasure library [3] to encode messages using erasure codes based on Reed-Solomon codes [59] with the decoding algorithm of Berlekamp and Welch [10].

Choice of Block Size. To determine a reasonable block size, we measure the throughput and latency of RapidChain with various block sizes between 512 KB and 8,192 KB for our target network size of 4,000 nodes. As shown in Figure 4 (right), larger block sizes generally result in higher throughput but also in higher confirmation latency. To obtain a latency of less than 10 seconds common in most mainstream payment systems while obtaining the highest possible throughput, we set our block size to 2,048 KB, which results in a throughput of more than 7,000 tx/sec and a latency of roughly 8.7 seconds.

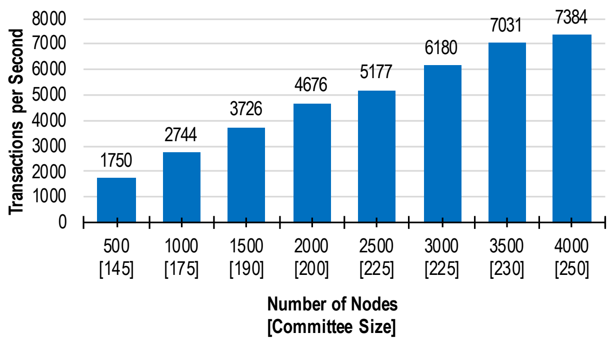
Throughput Scalability. To evaluate the impact of sharding, we measure the number of transactions processed per second by RapidChain as we increase the network size from 500 nodes to 4,000 nodes for variable committee sizes such that the failure probability of each epoch remains smaller than $2 \cdot 10^{-6}$ (*i.e.*, protocol fails after more than 1,300 years). For our target network size of 4,000, we consider a committee size of 250 which results in an epoch failure probability of less than $6 \cdot 10^{-7}$ (*i.e.*, time-to-failure of more than 4,580 years). As shown in Figure 5 (left), doubling the network size increases the capacity of RapidChain by 1.5-1.7 times. This is an important measure of how well the system can scale up its processing capacity with its main resource, *i.e.*, the number of nodes.

We also evaluate the impact of our pipelining technique for intra-committee consensus (as described in Section 4.2) by comparing the throughput with pipelining (*i.e.*, 7,384 tx/sec) with the throughput without pipelining (*i.e.*, 5,287 tx/sec) for $n = 4,000$, showing an improvement of about 1.4x.

Transaction Latency. We measure the latency of processing a transaction in RapidChain using two metrics: *confirmation latency* and *user-perceived latency*. The former measures the delay between the time that a transaction is included in a block by a consensus participant until the block is added to a ledger and its inclusion can be confirmed by any (honest) participant. In contrast, user-perceived latency measures the delay between the time that a user sends a transaction, tx, to the network until the time that tx can be confirmed by any (honest) node in the system.

Figure 5 (right) shows both latency values measured for various network sizes. While the client-perceived latency is roughly 8 times more than the confirmation latency, both latencies remain about the same for networks larger than 1,000 nodes. In comparison, Elastico [47] and OmniLedger [42] report confirmation latencies of roughly 800 seconds and 63 seconds for network sizes of 1,600 and 1,800 nodes respectively.

Reconfiguration Latency. Figure 6 (left) shows the latency overhead of epoch reconfiguration which, similar to [42], happens once a day. We measure this latency in three different scenarios, where 1, 5, or 10 nodes join RapidChain for various network sizes and variable committee sizes (as in Figure 5 (left)). The reconfiguration latency measured in Figure 6 (left) includes the delay of three tasks that have to be done sequentially during any reconfiguration event: (1) Generating epoch randomness by the reference committee; (2) Consensus on a new configuration block proposed by the reference committee; and (3) Assigning new nodes to existing committee and redistributing a certain number of the existing members in the affected committees. For example, in our target network of $4,000$



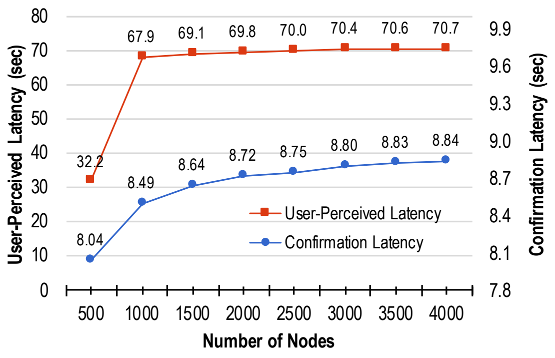


Figure 5: Throughput scalability of RapidChain (left); Transaction latency (right)

nodes, out of roughly 372 seconds for the event, the first task takes about 4.2 seconds (1%), the second task takes 71 seconds (19%), and the third task takes 297 seconds (80%). For other network sizes, roughly the same percentages are measured.

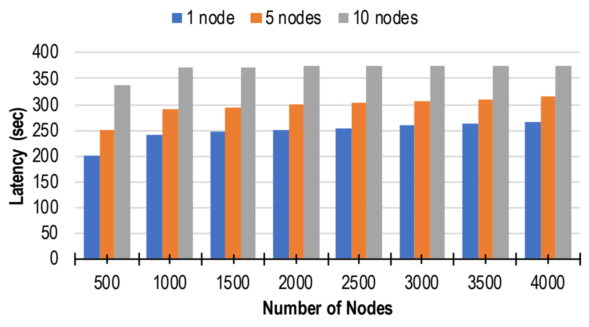
As shown in Figure 6 (left), the reconfiguration latency increases roughly 1.5 times if 10 nodes join the system rather than one node. This is because when more nodes join the system, more nodes are cuckooed (*i.e.*, redistributed) among other committees consequently. Since the churn of different nodes in different committees happen in parallel, the latency does not increase significantly with more joins. Moreover, the network size impacts the reconfiguration latency only slightly because churn mostly affects the committees involved in the reconfiguration process. In contrast, Elastico [47] cannot handle churn in an incremental manner and requires re-initialization of all committees. For a network of 1,800 nodes, epoch transition in OmniLedger [42] takes more than 1,000 seconds while it takes less than 380 second for RapidChain. In practice, OmniLegder's epoch transition takes more than 3 hours since the distributed random generation protocol used has to be repeated at least 10 times to succeed with high probability. Finally, it is unclear how this latency will be affected by the number of nodes joining (and hence redistributing node between committees) in OmniLedger.

Impact of Cross-Shard Batching. One of the important features of RapidChain is that it allows batching cross-shard verification requests in order to limit the amount of inter-committee communications to verify transactions. This is especially crucial when the number of shards is large because, as we show in Section 6.7, in our target network size of 4,000 nodes with 16 committees, roughly 99.98% of all transactions are expected to be cross-shard, meaning that at least one of every transaction's input UTXOs is expected to be located in a shard other than the one that will store the transaction itself. Since transactions are assigned to committees based on their randomly-generated IDs, transactions are expected to be distributed uniformly among committees. As a result, the size of a batch of cross-shard transactions for each committee for processing every block of size 2 MB is expected to be equal to 2 MB/16 = 128 KB. Figure 6 (right) shows the impact of batching cross-shard verifications on the throughput of RapidChain for various network sizes.

Storage Overhead. We measure the amount of data stored by each node after 1,250 blocks (about 5 million transactions) are processed by RapidChain. To compare with previous work, we estimate the storage required by each node in Elastico and OmniLedger based on their reported throughput and number of shards for similar network sizes as shown in Table 2.

Protocol	Network Size	Storage
Elastico [47]	1,600 nodes	2,400 MB (estimated)
OmniLedger [42]	1,800 nodes	750 MB (estimated)
RapidChain	1,800 nodes	267 MB
RapidChain	4,000 nodes	154 MB

Table 2: Storage required per node after processing 5 M transactions without ledger pruning



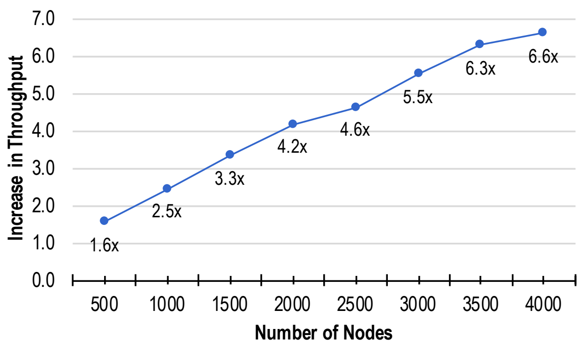


Figure 6: Reconfiguration latency when 1, 5, or 10 nodes join each committee (left); Impact of batching cross-shard verifications (right)

Overhead of Bootstrapping. We measure the overheads of our bootstrapping protocol to setup committees for the first time in two different experiments with 500 and 4,000 nodes. The measured latencies are 2.7 hours and 18.5 hours for each experiment respectively. Each participant in these two experiments consumes a bandwidth of roughly 29.8 GB and 86.5 GB respectively. Although these latency and bandwidth overheads are substantial, we note that the bootstrapping protocol is executed only once, and therefore, its overhead can be amortized over several epochs. Elastico and OmniLedger assume a trusted setup for generating an initial randomness, and therefore, do not report any measurements for such a setup.

6 Security and Performance Analysis

6.1 Epoch Security

We use the hypergeometric distribution to calculate the failure probability of each epoch. The cumulative hypergeometric distribution function allows us to calculate the probability of obtaining no less than x corrupt nodes when randomly selecting a committee of m nodes without replacement from a population of n nodes containing at most t corrupt nodes. Let X denote the random variable corresponding to the number of corrupt nodes in the sampled committee. The failure probability for one committee is at most

$$\Pr\left[X \geq \lfloor m/2 \rfloor\right] = \sum_{x=\lfloor m/2 \rfloor}^{m} \frac{\binom{t}{x}\binom{n-t}{m-x}}{\binom{n}{m}}.$$

Note that one can sample a committee with or without replacement from the total population of nodes. If the sampling is done with replacement (*i.e.*, committees can overlap), then the failure probability for one committee can be calculated from the cumulative binomial distribution function,

$$\Pr\left[X \geq \lfloor m/2 \rfloor\right] = \sum_{x=0}^{m} \binom{m}{x} f^{x}(1-f)^{m-x},$$

which calculates the probability that no less than x nodes are corrupt in a committee of n nodes sampled from an *infinite* pool of nodes, where the probability of each node being corrupt is $f = t/n$. If the sampling is done without replacement (as in RapidChain), then the binomial distribution can still be used to approximate (and bound) the failure probability for one committee. However, when the committee size gets larger relative to the population size, the hypergeometric distribution yields a better approximation (*e.g.*, roughly 3x smaller failure probability for $n = 2,000, m = 200, t < n/3$).

Unlike the binomial distribution, the hypergeometric distribution depends directly on the total population size (*i.e.*, n). Since n can change over time in an open-membership network, the failure probability might be affected

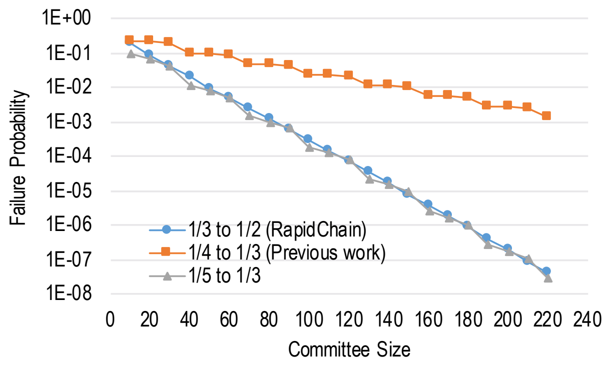


Figure 7: Log-scale plot of the probability of failure to sample one committee from a population of 2,000 nodes in RapidChain and previous work [47, 42] using the hypergeometric distribution.

consequently. To maintain the desired failure probability, each committee in RapidChain runs a consensus in predetermined intervals, *e.g.*, once a week, to agree on a new committee size, based on which, the committee will accept more nodes to join the committee in future epochs.

Figure 7 shows the probability of failure calculated using the hypergeometric distribution to sample a committee (with various sizes) from a population of 2,000 nodes for two scenarios: (1) $n/3$ total resiliency and $n/2$ committee resiliency (as in RapidChain); and (2) $n/4$ total resiliency and $n/3$ committee resiliency (as in previous work [47, 42]). As shown in the figure, the failure probability decreases much faster with the committee size in the RapidChain scenario.

To bound the failure probability of each epoch, we calculate the union bound over $k = n/m$ committees, where each can fail with the probability $p_{\text{committee}}$ calculated previously. In the first epoch, the committee election procedure (from the bootstrapping protocol) can fail with probability $p_{\text{bootstrap}} \leq 2^{-26.36}$. The random generation protocol executed by the reference committee at the beginning of each epoch is guaranteed to generate an unbiased coin with probability one, and the consensus protocol executed by each committee is guaranteed to terminate with probability one. By setting $n = 4,000, m = 250$, and $t < n/3$, we have $p_{\text{committee}} < 3.7 \cdot 10^{-8}$. Therefore, the failure probability of each epoch is

$$p_{\text{epoch}} < p_{\text{bootstrap}} + k \cdot p_{\text{committee}} < 6 \cdot 10^{-7}.$$

Ideally, we would hope that the probability that the adversary taking t faction of blocks in the epoch, and an honest miner takes $1 - t$ fraction of the blocks. However, it is not the case for committee-based sharding protocols such as RapidChain. The increase in the adversary's effective power comes from the fact that the upper-bound on the fraction of adversarial ids will increase inside each shard comparing to the whole system. Thus, we define and calculate the effective power of the adversary.

Definition 2. *Adversarial effective power. The ratio of the blocks that is created by the adversary and is added to the chain to the total number of blocks.*

Theorem 1. *The effective power of the adversary is* $1/2$*; i.e., the adversary can create half of the blocks in the system.*

The proof of this lemma follows from the choices of the target failure probability and committee sizes which we discussed in this section.

6.2 Gossiping Guarantees

RapidChain creates a well-connected random graph for gossiping between nodes on a committee. This guarantees that a message gossiped by any honest node will eventually reach all other honest nodes and the gossiping delay, Δ, can be bounded. In the following, we show that splitting a message into chunks, as described in Section 4.2, does not violate the above gossiping guarantee with high probability.

Lemma 1. *The probability that a message is not delivered to all honest nodes after it is gossiped using IDA-Gossip without sparsification is at most* 0.1.

Proof. Let d denote the degree of every committee member, *i.e.*, the number of neighbors that are chosen uniformly at random. Using the hypergeometric distribution, we find a threshold ζ such that the probability of having more than ζ fraction corrupt neighbors is at most 0.1. For example, for $m = 200$ and $f < 1/2$, we set $\zeta = 0.63$. The sender splits the message into κ chunks and gives a unique set of size κ/d chunks to each of its neighbors so they can gossip the chunks on its behalf. Thus, at the end of the gossiping protocol, each honest node will receive at least $(1-\zeta)\kappa$ correct chunks. Finally, an error-correction coding scheme can be used at the receiver to correct up to a ζ fraction corrupted chunks and reconstruct the message. □

Next, we show that the process of sparsification is not going to change the correctness and security of the gossiping protocol and it increases the probability of failure slightly.

Lemma 2. *Assume the gossiper sparisfies all the Merkle tree nodes up to some level i. The probability that the message is not received correctly after the gossiping of a big message with specification with parameter s is at most $0.1 + 2^{-(s-i-1)}$, where s is the size of the subset of nodes whom gossiper sends each sparsified node and can be set based on the the desired failure probability (as a function of the importance of the message).*

Proof. The reconstruction stage fails if there is a node in the tree for which the hash at that node is distributed to only corrupt nodes. We will call a tree node *sparsified* if the hash $T(M_i, M_j)$ at the node is not sent along with all of the leaves that require that hash for verification of the block. We will sparisfy all nodes up to some level i. The sender can calculate s, which is the size of the subset of nodes whom he sends each sparsified node to guarantee that with probability at least 2^{-c}, the node is sent to at least one honest node. Let $l(x)$ count the number of leaf nodes in the sub-tree rooted at node x, and $u(x)$ count the number of corrupt nodes in the sub-tree rooted at node x.

If a node is distributed to s nodes at random, the probability that only corrupt nodes receive the node is at most f^s. Therefore, taking the union bound over all $2^{i+1} - 1$ nodes and by setting $f < 1/2$,

$$(2^{i+1} - 1)f^s < 2^{i+1} f^s < 2^{i+1-s}.$$

□

The difference between sparsification and non-sparsification is that by sparsification, the gossiper decrease his chance of a successful gossip slightly but in return puts less communication burden on the nodes and network. Since the gossip of the blocks are crucial to the system, we do not use sparsification for them. However, users can use sparsification for their large transactions if the transaction is not time-sensitive. In case the transaction fails to be gossiped correctly due to sparsification, the user can re-send the Merkle tree nodes later which will happen with small probability.

6.3 Security of Intra-Committee Consensus

Recall that honest nodes always collectively control at least $n - t$ of all identities at any given epoch. We first prove safety assuming all the ids are fixed during one epoch and the epoch randomness is unbiased.

Theorem 2. *The protocol achieves safety if the committee has no more than $f < 1/2$ fraction of corrupt nodes.*

Proof. We prove safety for a specific block header proposed by the leader at iteration i. Suppose node P is the first honest node to accept a header H_i for i. Thus, it echoes H_i before accepting it at some iteration $j \geq i$. If another honest replica accepts H_i', there must be a valid accept message with a proof certificate on H_i' at some block iteration $j' \geq i$. However, in all the rounds of iteration i or all iterations after i, no leader can construct a safe proposal for a different header other than H_i since he cannot obtain enough votes from honest nodes on a value that is not safe and thus create a $mf + 1$ proof. Finlay, note that due to correctness of the IDA-Gossiping, it is impossible to finalized a Merkle root of a block that is associates with two different blocks. □

We define liveness in our setting as finality for one block, *i.e.*, the protocol achieves liveness if it can accepts all the proposed blocks within a bounded time.

Theorem 3. *The protocol achieves liveness if the committee has less than mf corrupt nodes.*

Proof. First note that all the messages in the system are unforgeable using digital signatures. We first show that all the honest nodes will accept all the pending blocks, or they have accepted them with the safe value before, as soon as the leader for the current block height is honest. Since, the leaders are chosen randomly and the randomness is unbiased, each committee will have an honest leader every two rounds in expectation. The honest leader will send valid proposals for all the pending blocks to all nodes that is safe to propose and has a valid proof. Thus, all honest replicas have either have already accepted the same value or they will accept it since it is safe (see the theorem for safety). Thus, all honest nodes will vote for the proposed header for all the pending blocks, subsequently the will receive $mf + 1$ votes for them since we have $mf + 1$ honest nodes. Therefore, all honest nodes have finalized the block at the end of this iteration. □

Security of Cross-Shard Verification. Without loss of generality, we assume tx has two inputs I_1, I_2 and one output O. If the user provides valid inputs, both input committees successfully verify their transactions and send the new UTXOs to the output committee. Thus, tx_3 will be successful. If both inputs are invalid, both input committees will not send I'_i, and as a result tx_3 will not be accepted. If I_1 is valid but I_2 is invalid, then C^1_{in} successfully transfers I'_1 but C^2_{in} will not send I'_2. Thus, tx_3 will not be accepted in C_{out}. However, I'_1 is a valid UTXO in the output committee and the user can spend it later. The output committee can send the resulting UTXO to the user.

6.4 Security of Bootstrapping

Theorem 4. *Suppose there are n nodes and a constant fraction,* $1/3$ *of these nodes are corrupt. At the end of the RapidChain bootstrapping protocol from Section 4.6, almost all (99 percentile) of the uncorrupted nodes agree on the random string* r_0 *and consequently on all the sharding committees with constant probability greater than 0.*

Proof. In Lemmas 3 and 4, we present analysis for the the probability of error in the bootstrapping protocol. The first lemma follows from previous work but we provide it from completeness. Then, we present a tighter analysis for the protocol in the second lemma that improves the concrete parameters we need to use for our implementation which give over 20% for large number of parties. □

Original Analysis. Recall that L represents the set of parties and R represents the set of groups selected based on the edges incident to the node from L.

Lemma 3. *In a sampler graph with a random assignment of honest parties and adversarial assignment of dishonest parties, the probability that all the formed groups in any subset of size* $|S|$ *being corrupted is less than* $2e^{(|L|+|R|-\delta^2 d_R|S|/2}$.

Consider two fixed subsets of nodes $T \subseteq L$ and $S \subseteq R$. Let N denote the number of edges between T and S. One can imagine T represents the largest coalition of faulty parties and S represents any subset of groups. Let $\mathcal{E}(T,S)$ denote the event that every node in S has more than a $\frac{|T|}{|L|} + \delta$ fraction of its edges incident to nodes in T. The number of edges incident to S is equal to $d_R|S|$. By linearity of expectation, $\text{E}[N] = \frac{|T|}{|L|} d_R|S|$. Suppose that we add the edges to S one-by-one. Let $\{X_i\}$ denote a sequence of random variables such that $X_i = 1$ if and only if the i-th edge is incident to T. The sequence $\{Z_i = \text{E}[N|X_0, ..., X_i]\}$ defines a Doob martingale [26, Chapter 5], where $Z_0 = \text{E}[N]$. If any of the trials is altered, N changes by at most one. For some positive δ, the failure event $\mathcal{E}$ happens whenever

$$N > \left(\frac{|T|}{|L|} + \delta\right) d_R|S| = \text{E}[N] + \delta d_R|S|.$$

By Azuma's inequality,

$$\Pr\left(\mathcal{E}(T,S)\right) = \Pr\left(N - \text{E}[N] > \delta d_R|S|\right) \leq 2e^{-\delta^2 d_R|S|/2}.$$

By union bound over all possible subsets T and S,

$$\Pr\left(\bigcup_{T\subseteq L, S\subseteq R}\right) \leq 2^{|L|}2^{|R|}2e^{-\delta^2 d_R|S|/2} \leq 2e^{(|L|+|R|-\delta^2 d_R|S|/2}. \quad (3)$$

Tighter Analysis. We now obtain a tighter bound on the failure probability.

Lemma 4. *In a sampler graph with a random assignment of honest parties and adversarial assignment of dishonest parties, the probability that all the formed groups in any subset of size* $|R'|$ *being corrupted is less than* $\beta 2^{|L|}2^{|R|}(e^{-\mu_b}(e\mu_b/x)^x + e^{-\mu_g}(e\mu_g/(2x))^{2x})^{|R'|}$.

Proof. The sampler graph selection process can be seen as the classic balls-and-bins process: $|L|d_L$ balls (parties) are thrown independently and uniformly at random into $|R|$ bins (groups). Without loss of generality we can assume we first through all dishonest parties (bad balls) then all the honest parties (good balls).

For a fix committee C, let X_g be a random variable representing the maximum number of dishonest parties assigned to C, X_b be a random variable representing the minimum number of honest parties assigned to C, μ_g and μ_b be the expected number of honest and dishonest parties per group respectively.

It is well known that the distribution of the number of (good/bad) balls in a bin is approximately Poisson with mean $\mu_b = d_L|L|/4$ and $\mu_g = 3d_L|L|/4$ [53, Chapter 5]. let $\tilde{X}$ and be the Poisson random variable approximating X. We have $\mu = \mathrm{E}[X] = \mathrm{E}[\tilde{X}]$. We use the following Chernoff bounds from [53, Chapter 5] for Poisson random variables:

$$\Pr(\tilde{X} \geq x) \leq e^{-\mu}(e\mu/x)^x, \text{ when } x > \mu, \tag{4}$$

$$\Pr(\tilde{X} \leq x) \leq e^{-\mu}(e\mu/x)^x, \text{ when } x < \mu. \tag{5}$$

We consider a group to be good if $X_g > x$ and $X_b < x$. This is to make sure a good group has honest majority. Note that this definition is an under-estimation and we do not count some of the good groups. Based on this definition, a group is bad if $X_g \leq x$ or $X_b \geq x$.

The the probability that a fixed committee being bad is:

$$e^{-\mu_b}(e\mu_b/x)^x + e^{-\mu_g}(e\mu_g/x)^x. \tag{6}$$

Now, consider a subset of groups of size $|R'|$, the probability that all of them being bad is, $(e^{-\mu_b}(e\mu_b/x)^x + e^{-\mu_g}(e\mu_g/x)^x)^{|R'|}$.

Since the adversary can choose the bad parties and the bad groups, we use union bound over all the possible adversarial choices to find the probability that all the groups in any subset of size $|R'|$ being corrupted:

$$\Pr\left(\bigcup_{T\subseteq L, S\subseteq R}\right) \leq 2^{|L|}2^{|R|}(e^{-\mu_b}(e\mu_b/x)^x + e^{-\mu_g}(e\mu_g/x)^x)^{|R'|}. \tag{7}$$

Tighter Bound. Not all the choices of the adversary gives him the same probability of success. Thus, we can consider strategies that are strictly worst than another strategy and remove them from the union bound since it is not beneficial for the adversary to choose such strategies. We consider the following random process:

1. All good parties are assigned randomly to groups.
2. The adversary assign α fraction of all its bad parties to the groups such that the assignment corrupts maximum number of groups.
3. The adversary assigns remaining $1 - \alpha$ bad parties such that each party assigned to at least one good group.

We claim that any strategy who does not follow the previous process *i.e.* it assigns a bad party to all bad committees at step (3) is a strictly worst since assigning bad parties to the groups that are already bad will not increase the chance of adversary to corrupt a new group.

Similar to the previous analysis, we can calculate the probability that a set of size $|R'|$ has only bad committees in it after throwing all good parties and α fraction of the bad parties that is,

$$(e^{-\mu_{\alpha b}}(e\mu_{\alpha b}/x)^x + e^{-\mu_g}(e\mu_g/x)^x)^{|R'|}. \tag{8}$$

Now, we can calculate the fraction of strategies that the adversary ignores due to the step three rule,

$$\beta = \frac{|R'|!(|R'| - \log n)!}{|R|!(|R| - \log n)}. \tag{9}$$

Thus, in our union bound, we can ignore this fraction,

$$\Pr\left(\bigcup_{T\subseteq L, S\subseteq R}\right) \leq \beta 2^{|L|}2^{|R|}(e^{-\mu_b}(e\mu_b/x)^x + e^{-\mu_g}(e\mu_g/x)^x)^{|R'|}.$$

□

Protocol	**ID Generation**	**Bootstrap**	**Consensus**	**Storage per Node**
Elastico [47]	$O(n^2)$	$\Omega(n^2)$	$O(m^2/b + n)$	$O(\|B\|)$
OmniLedger [42]	$O(n^2)$	$\Omega(n^2)$	$\Omega(m^2/b + n)$	$O(m \cdot \|B\|/n)$
RapidChain	$O(n^2)$	$O(n\sqrt{n})$	$O(m^2/b + m \log n)$	$O(m \cdot \|B\|/n)$

Table 3: Complexities of previous sharding-based blockchain protocols

6.5 Security of Reconfiguration

We now prove that the reconfiguration protocol that we use maintains the balancing and honesty properties of the committees. The proof is an extension of the proof in [8]. We first define two conditions for a committee.

Definition 3 (Honesty)**.** *A committee satisfies the honesty condition if the fraction of corrupt nodes in the committee is strictly less than* $1/2$*.*

Definition 4 (Balancing)**.** *A committee satisfies the balancing condition if the number of nodes in the committee is bounded by* $O(\log n)$*.*

In the following, we let nodes join and leave one-by-one in each round. Moreover, we assume that at any time during the protocol, the fraction of corrupt nodes to honest nodes is ϵ. We also assume the protocol starts from a stable state with n nodes partitioned into m committees which satisfies the balancing and honesty conditions. Recall that we defined the set of active committees as the $m/2$ committees with highest number of nodes in them.

Lemma 5. *For any fixed* active *committee C and at any time, the age of any active committee C is within* $(1 \pm \delta)\frac{nc \log n}{2k}$*, with high probability.*

Proof. We define y_i as the age of k-region called R_i and $Y = \sum_{i=1}^{i=c\log n} y_i$ as the age of C. At any point during the protocol, half of the committees are active so we choose the region for the new node from half of the k regions. Thus, $Pr[y_i = t] = \frac{2k}{n}(1 - \frac{2k}{n})^{t-1}$ is geometrically distributed with probability $\frac{2k}{n}$. Thus, $E[y_i] = \frac{n}{2k}$ and $E[Y] = \frac{nc}{2k} c \log n$. It is easy to show that Y is concentrated around $E[Y]$, meaning that Y is between $(1 \pm \delta)E[Y]$ and we omit the proof here. □

Lemma 6. *Any k-region in active committees has age at most* $\lambda(n/2k)\log n$*.*

Proof. The probability that a k-region R_i is evicted at any round is $2k/n$ since at any round we have $m/2$ active committees and as a result half of the k-regions will accept a new join. Conditioned on the event that the committee does not get inactive during this time, we can assume this probability is independent of other rounds. Note that this condition considers the worst case scenario since otherwise the committee gets inactive during this time. Hence, the probability that R_i has age at least $\lambda(n/2k)\log n$ is $(1 - 2k/n)^{\lambda(n/k)\log n} \leq e^{-2k/n\lambda(n/2k)\log n} = n^{-\lambda}$. □

Lemma 7. *Any fixed node v in an active committee, it gets replaced at most* $(1 + \delta)\lambda \log n$ *times within* $\lambda(n/2k)\log n$ *rounds.*

Proof. We prove this lemma conditioned on the fact that the node is placed in an active committee with probability $1/2$, *i.e.*, half of the committees get to be active after one node joins them. This condition is considers the worst case scenario in which we assume that half of the inactive committees have numbers of nodes very close to being active in the next round. Let the indicator random variable $z_t = 1$ if node p is replaced in t, otherwise it is 0. $Pr[z_t = 1] = 1/2\frac{2k}{n}$ since at any time we randomly choose a region to evict from all active regions. Let $Z = \sum_{t=0}^{t=\lambda(n/2k)\log n} z_t$. We can compute $E[Z] = \lambda(n/2k)\left(\frac{k}{n}\right)\log n = 1/2\lambda \log n$. Using the Chernoff bound, we can show that $Z < (1 + \delta)E[Z]$ with high probability. □

We state the following lemma from [8] (Lemma 2.8), which we use directly since our construction will not change the fraction of honest and corrupt nodes in any way from their construction.

Lemma 8. *Let t be the maximum number of corrupt nodes at any point in the system. At any time, a fixed committee has within $(1 - t/n)(1 \pm \delta)c \log nk/2$ old honest and $\frac{t}{n-t}(1 \pm \delta)c \log nk/2$ old corrupt nodes with high probability.*

The balancing and honesty properties follow but we omit the details here due to space limitations.

Proof. The proof is similar to the proof of Lemma 2.8 from [8] with different parameters based on Lemmas 5, 6, and 7. □

Theorem 5. *At any time during the protocol, all committees satisfy the balancing and honesty conditions.*

Proof. To prove the theorem, it is enough to prove the balancing and honesty properties for any committee. First note that the number of new nodes in each committee is at most $c \log n$. We also calculated the number of old nodes in Lemma 8.

- *Balancing:* The maximum number of nodes in each committee is $c \log n + c/2(1 + \delta) \left(3 - \frac{t}{n} + \frac{t}{n-t}k\right) \log n$ and the minimum load is $c/2(1 - \delta) \log n$.
- *Honesty.* Choosing k such that $\frac{t}{n-t} < 1 - 1/k$, any committee has $(1 - t/n)(1 - \delta)c \log nk/2$ honest and $\frac{t}{n-t}(1 + \delta)c \log nk/2$ corrupt nodes with high probability. Note that this values are calculated for the worst case scenario when the adversary targeted the committee of size $(c \log n)k/n$.

□

6.6 Performance Analysis

We summarize the theoretical analysis of the performance of RapidChain in Table 3.

Complexity of Consensus. Without loss of generality, we calculate the complexity of consensus per transaction. Let tx denote a transaction that belongs to a committee C with size m. First, the user sends tx to a constant number of RapidChain nodes who route it to C. This imposes a communication and computation overhead of $m \log(n/m) = O(m \log n)$. Next, a leader in C drives an intra-committee consensus, which requires $O(\frac{m^2}{b})$ communication amortized on the block size, b. Moreover, with probability 96.3% (see Section 6.7) tx is a cross-shard transaction, and the leader requires to requests a verification proof from a constant number of committees assuming tx has constant number of inputs and outputs. The cost of routing the requests and their responses is $O(m \log n)$, and the cost of consensus on the request and response is $O(m^2/b)$ (amortized on the block size due to batching) which happens in every input and output committee. Thus, the total per-transaction communication and complexity of a consensus iteration is equal to $O(m^2/b + m \log n)$.

Complexity of Bootstrap Protocol. Assuming a group size of $O(\sqrt{n})$ and $O(\sqrt{n})$ groups, we count the total communication complexity of our bootstrap protocol as follows. Each group runs the DRG protocol that requires $O(n)$ communication. Since each group has $O(\sqrt{n})$ nodes, the total complexity is $O(n\sqrt{n})$. After DRG, a constant number of members from each group will gossip the result incurring a $O(n\sqrt{n})$ overhead, where $O(n)$ is the cost of each gossip. Since the number of nodes in the root group is also $O(\sqrt{n})$, its message complexity to generate a randomness and to gossip it to all of its members for electing a reference committee is $O(n)$.

Storage Complexity. Let $|B|$ denote the size of the blockchain. We divide the ledger among n/m shards, thus each node stores $O(m \cdot |B|/n)$ amount of data. Note that we store a reference block at the start of each epoch that contains the list of all of the nodes and their corresponding committees. This cost is asymptotically negligible in the size of the ledger that each node has to store as long as $n = o(|B|)$.

6.7 Probability of Cross-Shard Transactions

In this section, we calculate the probability that a transaction is cross-shard, meaning that at least one of its input UTXOs is located in a shard other than the one that will store the transaction itself.[1] Let tx denote the transaction to verify, k denote the number of committees, $u > 0$ denote the total number of input and output UTXOs in tx, and

[1]A similar calculation is done in [42] but the presented formula is, unfortunately, incorrect.

$v > 0$ denote the number of committees that stores at least one of the input UTXOs of tx. The probability that tx is cross-shard is equal to $1 - F(u, v, k)$, where

$$F(u,v,k) = \begin{cases} 1, & \text{if } u = v = 1 \\ (1/k)^u, & \text{if } v = 1 \\ \frac{k-v}{k} \cdot F(u-1, v-1, k), & \text{if } u = v \\ \frac{k-v}{k} \cdot F(u-1, v-1, k) + \\ \qquad \frac{v}{k} \cdot F(u-1, v, k), & \text{otherwise.} \end{cases} \tag{10}$$

For our target network of 4,000 nodes where we create $k = 16$ committees almost all transactions are expected to be cross-shard because $1 - F(3, 1, 16) = 99.98\%$. In comparison, for a smaller network of 500 nodes where we create only 3 committees, this probability is equal to $1 - F(3, 1, 3) = 96.3\%$.

6.8 Estimating Unreported Overheads

We estimate the bandwidth overhead of an epoch of OmniLedger using the numbers reported in [42]: a total of 1,800 nodes, transaction size of 500 B, throughput of 3,500 tx/sec, and a probability of 3.7% for a transaction to be intra-shard. OmniLedger requires at least three gossip-to-all invocations per cross-shard transaction. Therefore, the bandwidth required by each node is at least 3,500 tx/sec $\cdot$ 0.967 $\cdot$ 500 B $\cdot$ 3 $\approx$ 45 Mbps.

To estimate the throughput of Elastico, we use the reported transaction confirmation latency of 800 seconds for n =1,600, m =16, and 1 MB blocks. Assuming 500 B/tx, the throughput of Elastico can be calculated as 16 $\cdot$ 2,000 / 800 = 40 tx/sec.

7 Conclusion

We present RapidChain, the first 1/3-resilient sharding-based blockchain protocol that is highly scalable to large networks. RapidChain uses a distributed ledger design that partitions the blockchain across several committees along with several key improvements that result in significantly-higher transaction throughput and lower latency. RapidChain handles seamlessly churn introducing minimum changes across committee membership without affecting transaction latency. Our system also features several improved protocols for fast gossip of large messages and inter-committee routing. Finally, our empirical evaluation demonstrates that RapidChain scales smoothly to network sizes of up to 4,000 nodes showing better performance than previous work.

8 Acknowledgment

The authors would like to acknowledge support from NSF grants CNS-1633282, 1562888, 1565208, and DARPA SafeWare W911NF-15-C-0236 and W911NF-16-1-0389. We are also grateful for kind help from Loi Luu (NUS) and Aditya Sinha (Yale), and invaluable comments from Dominic Williams (Dfinity), Timo Hanke (Dfinity), Abhinav Aggarwal (UNM), Bryan Ford (EPFL), and Eleftherios Kokoris Kogias (EPFL).

References

[1] Blockchain charts: Bitcoin's hashrate distribution, March 2017. Available at https://blockchain.info/pools.

[2] Blockchain charts: Bitcoin's blockchain size, July 2018. Available at https://blockchain.info/charts/blocks-size.

[3] Jerasure: Erasure coding library, May 2018. Available at http://jerasure.org.

[4] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. In *Proceedings of the 21st International Conference on Principles of Distributed Systems*, OPODIS '17, Lisboa, Portugal, 2017.

[5] Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and JP Stern. Addendum to scalable secure storage when half the system is faulty. *Information and Computation*, 2004.

[6] Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and Julien Stern. Scalable secure storage when half the system is faulty. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, 2000.

[7] Marcin Andrychowicz and Stefan Dziembowski. *PoW-Based Distributed Cryptography with No Trusted Setup*, pages 379–399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[8] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 318–327, New York, NY, USA, 2006. ACM.

[9] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.

[10] Elwyn Berlekamp and Lloyd R. Welch. Error correction for algebraic block codes, US Patent 4,633,470, December 1986.

[11] Richard E Blahut. *Theory and practice of error control codes*, volume 126. Addison-Wesley Reading (Ma) etc., 1983.

[12] Gabriel Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 154–162, New York, NY, USA, 1984. ACM.

[13] Gabriel Bracha. An $o(\log n)$ expected rounds randomized byzantine generals protocol. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 316–326, New York, NY, USA, 1985. ACM.

[14] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, November 1987.

[15] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 12–26, New York, NY, USA, 1983. ACM.

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

[17] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 123–132, 2000.

[18] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 42–51, New York, NY, USA, 1993. ACM.

[19] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[20] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, 1999.

[21] 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. pages 251–264, 2012.

[22] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *23rd Annual Network and Distributed System Security Symposium, NDSS*, 2016.

[23] 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.

[24] Christian Decker and Roger Wattenhofer. Information propagation in the Bitcoin network. In *P2P*, pages 1–10. IEEE, 2013.

[25] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[26] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 2009.

[27] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology — CRYPTO' 92: 12th Annual International Cryptology Conference Santa Barbara, California, USA August 16–20, 1992 Proceedings*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[28] David S. Evans. Economic aspects of Bitcoin and other decentralized public-ledger currency platforms. In *Coase-Sandor Working Paper Series in Law and Economics*, No. 685. The University of Chicago Law School, 2014.

[29] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 45–59, Berkeley, CA, USA, 2016. USENIX Association.

[30] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 427–438, Washington, DC, USA, 1987. IEEE Computer Society.

[31] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[32] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. ACM.

[33] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[34] Egor Homakov. Stop. calling. bitcoin. decentralized. https://medium.com/@homakov/stop-calling-bitcoin-decentralized-cb703d69dc27, 2017.

[35] Min Huang and Vernon J. Rego. Polynomial evaluation in secret sharing schemes, 2010. URL: http://csdata.cs.purdue.edu/research/PaCS/polyeval.pdf.

[36] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 565–, Washington, DC, USA, 2000. IEEE Computer Society.

[37] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. In *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer Berlin Heidelberg, 2006.

[38] Valerie King and Jared Saia. Breaking the $o(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 420–429, New York, NY, USA, 2010. ACM.

[39] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 990–999, Philadelphia, PA, USA, 2006.

[40] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 87–98, Washington, DC, USA, 2006. IEEE Computer Society.

[41] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium, USENIX Security '16*, pages 279–296, 2016.

[42] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (S&P)*, pages 19–34, 2018.

[43] Hugo Krawczyk. Distributed fingerprints and secure information dispersal. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, PODC '93, pages 207–218, New York, NY, USA, 1993. ACM.

[44] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[45] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. Vault: Fast bootstrapping for cryptocurrencies. Cryptology ePrint Archive, Report 2018/269, 2018. https://eprint.iacr.org/2018/269.

[46] Eric Limer. The world's most powerful computer network is being wasted on Bitcoin. May 2013. Available at http://gizmodo.com/the-worlds-most-powerful-computer-network-is-being-was-504503726.

[47] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. ACM.

[48] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[49] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, London, UK, UK, 1988. Springer-Verlag.

[50] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[51] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 120–, Washington, DC, USA, 1999. IEEE Computer Society.

[52] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42, New York, NY, USA, 2016. ACM.

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

[54] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at https://bitcoin.org/bitcoin.pdf.

[55] Rafail Ostrovsky, Sridhar Rajagopalan, and Umesh Vazirani. Simple and efficient leader election in the full information model. 1994.

[56] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. http://eprint.iacr.org/2016/917.

[57] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[58] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, April 1989.

[59] Irving Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, pages 300–304, 1960.

[60] Ling Ren, Kartik Nayak, Ittai Abraham, and Srinivas Devadas. Practical synchronous byzantine consensus. *CoRR*, abs/1704.02397, 2017.

[61] Alexander Russell and David Zuckerman. Perfect information leader election in $\log^* N + o(1)$ rounds. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 576–, Washington, DC, USA, 1998. IEEE Computer Society.

[62] Siddhartha Sen and Michael J. Freedman. Commensal cuckoo: secure group partitioning for large-scale services. *ACM SIGOPS Operating Systems Review*, 46(1):33–39, 2012.

[63] Alex Tapscott and Don Tapscott. How blockchain is changing finance. *Harvard Business Review*, March 2017. Available at https://hbr.org/2017/03/how-blockchain-is-changing-finance.

[64] The Zilliqa Team. The zilliqa technical whitepaper. https://docs.zilliqa.com/whitepaper.pdf, August 2017.

[65] Hsiao-Wei Wang. Ethereum sharding: Overview and finality. https://medium.com/@icebearhww/ethereum-sharding-and-finality-65248951f649, 2017.