Chain of Trust: Can Trusted Hardware Help Scaling Blockchains?

Hung Dang, Anh Dinh, Ee-Chien Chang, Beng Chin Ooi
School of Computing
National University of Singapore
{hungdang, dinhtta, changee, ooibc}@comp.nus.edu.sg

Abstract
As blockchain systems proliferate, there remains an unresolved scalability problem of their underlying distributed consensus protocols. Byzantine Fault Tolerance (BFT) consensus protocols offer high transaction throughput, but can only support small networks. Proof-of-Work (PoW) consensus protocol, on the other hand, can support thousands of nodes, but at the expense of poor transaction throughput. Two potential approaches to address these scalability barriers are by relaxing the threat model, or by employing sharding technique to deal with large networks. Nonetheless, their effectiveness on data-intensive blockchain workloads remains to be seen.

In this work, we study the use and effectiveness of trusted hardware on scaling distributed consensus protocols, and by their extension, blockchain systems. We first analyze existing approaches that harness trusted hardware to enhance scalability, and identify their limitations. Drawing insights from these results, we propose two design principles, namely scale up by scaling down and prioritize consensus messages, enabling the consensus protocols to scale. We illustrate the two principles by presenting optimizations that improve upon state-of-the-art solutions, and demonstrate via our extensive evaluations that they indeed offer better scalability. In particular, integrating our optimizations into Hyperledger Fabric achieves up to 7× higher throughput, while enabling the system to remain operational as the network expands. Another optimization that we introduce to Hyperledger Sawtooth allows the system to sustain high throughput as the network grows. Finally, our new design for sharding protocols reduces the cost of shard creation phase by up to 32×.

1. INTRODUCTION
Blockchain, the technology that underlies Bitcoin and several hundreds other crypto-currencies, is essentially a distributed, append-only ledger that stores ordered transactions. The ledger (or blockchain) consists of blocks forming a chain using cryptographic hash pointers, each block contains a sequence of transactions. The blockchain is maintained by a set of mutually distrusting nodes (often called replicas, or validators). These nodes run a distributed consensus protocol to ensure the blockchain’s consistency under arbitrary node failures.

Blockchain systems can be broadly classified as permissioned or permissionless. In the former, nodes are authenticated and they run Byzantine Fault Tolerance (BFT) protocols such as PBFT [24] to reach consensus. Permissioned blockchains are being adopted by many industries for applications such as asset management [68], supply chain, and settlement [8, 9]. However, BFT protocols are communication bound and have been shown not to scale beyond a dozen of nodes [33]. In a permissionless blockchain, any node can join the network, and the nodes run lottery-based (or Nakamoto) consensus protocols such as Proof of Work (PoW) or Proof of Stake (PoS) [69, 22]. The most well known application of permissionless blockchain is cryptocurrency, namely Bitcoin and Ethereum which run on PoW. These systems can support thousands of nodes, but at the expense of transaction throughput. For instance, Bitcoin can only process up to seven transactions per second.

One approach to overcome the scalability barrier of these consensus protocols is to use trusted hardware. Such hardware, for example Intel SGX [62], creates a secure (tamper proof and confidential) execution environment in which a trusted code base (or TCB) can run. The key idea behind this approach is to use the hardware to relax the Byzantine failure model, which can lead to better performance [55, 49]. Previous works [26, 53, 10, 82, 17] have shown that the hardware enables BFT protocols to tolerate $f$ failures using only $2f+1$ replicas, as opposed to $3f+1$ without the hardware. Proof-of-Elapsed-Time (PoET) [7], recently introduced by Hyperledger Sawtooth, uses Intel SGX [62] to completely replace Proof-of-Work (PoW). By avoiding the massive computation requirement in PoW, PoET expects to achieve high throughput.

Another approach takes advantage of sharding, which is a common technique in distributed database systems, to partition the network and the state for better throughput. For instance, Elastico [56], OmniLedger [48], and Chainspace [13] divide the network into smaller committees (or shards) where each shard runs an independent BFT protocol. However, Elastico and OmniLedger incur high overhead in forming the committees. Furthermore, all three solutions do not exploit trusted hardware, and they need hundreds of nodes per committee in order to ensure security.

In this paper, we address the problem of scaling the blockchain systems with trusted hardware. Our focus is on scalability of BFT and PoW protocols which have been shown to be the performance bottleneck [33]. Instead of designing a new protocol from scratch, we take a principled approach that improves upon the state-of-the-art solutions. First, we conduct extensive experimental evaluations of consensus protocols that use trusted hardware. We quantify their performance against data-intensive blockchain workloads. For scaling BFT, we consider two trusted hardware based PBFT variants: one follows the protocols specified in [26] and is called AHL, while the other adds the optimizations proposed in Byzcoin [47] and is dubbed AHLR. The comparison is based on our own implementations of these variants on top of Hyperledger Fabric v0.6 [2] and using Intel SGX, due to their source code unavailability. For scaling PoW, we consider Hyperledger Sawtooth’s PoET. We summarize these scaling proposals in Table 1.

Second, our results show that although these solutions have better performance than the baselines (without trusted hardware), their scalability remains limited. In particular, both PBFT variants do
not prevent the system from crashing after a small number of nodes join the network. AHL, for example, can sustain only up to four more failures than the baseline before crashing. PoET’s throughput decreases sharply as the network size increases, which is inconsistent with the expectation of stable performance demonstrated by Bitcoin and Ethereum. Furthermore, our analysis of sharding protocols [48, 56] shows that the cost of forming shards becomes prohibitive as the network grows, limiting the overall scalability. Drawing insights from these observations, we propose two design principles that address the weaknesses of the systems under consideration. The first principle is scale up by scaling down, which leverages trusted hardware to reduce the effective network size. The second principle is prioritize consensus messages, which minimizes the impact of increased network load on the consensus protocol.

Following the two principles, we propose optimizations to the PBFT variants and PoET called AHL+ and S-PoET, respectively. Despite their simplicity, these optimizations are derived from extensive evaluations of the blockchain systems. They involve small but important changes to the underlying protocols. In addition, we design a more efficient sharding protocol called GRANBEA which is based on OmniLedger but harnesses trusted hardware to avoid the former’s prohibitive overheads. Figure 1 summarizes the design space and where our work is positioned.

We show experimentally that our optimizations and new design achieve better scalability. More specifically, AHL+ has 7× higher throughput than the baseline and remains operational as the network expands. S-PoET sustains high throughput at scale. Finally, GRANBEA reduces shard formation time by up to 32× compared to the state-of-the-art sharding protocol.

In summary, we make the following contributions:

1. We conduct extensive evaluations of hardware-assisted distributed consensus protocols in the context of blockchain systems. To the best of our knowledge, this work provides the first implementations of PBFT protocol on Intel SGX. It is also the first systematic study of these protocols as parts of real blockchain systems.

2. Based on the evaluation results, we identify scalability bottlenecks of existing scaling proposals, and propose two design principles that address these bottlenecks.

3. We describe two optimizations, namely AHL+ and S-PoET, and show experimentally that they have better performance as well as scalability compared to the baselines. AHL+ achieves 7× higher throughput and remains operational as the network size increases, while S-PoET maintains high throughput at scale.

4. We design an efficient sharding protocol, called GRANBEA, that harnesses trusted hardware to reduce the cost of shard formation by 32×.

The remaining of the paper is structured as follows. Section 2 provides backgrounds on distributed consensus protocols, focusing on PBFT and PoW. It explains the high-level design of Hyperledger, and key properties of Intel SGX. Section 2 also details the system model, threat model and our experimental setup. The next section focuses on evaluations of three major approaches for scaling consensus protocols. The evaluation results reveal scalability limitation in state-of-the-art solutions. Section 4 discusses the insights drawn from these results, and codifies them into two design principles for addressing scalability bottlenecks. It is followed by specific optimizations and a new sharding design based on the principles. Section 5 evaluates these optimizations and design, confirming that they deliver better performances. Section 6 discusses related works, before Section 7 concludes our work.

2. PRELIMINARIES

In this section, we first provide backgrounds of two major classes of distributed consensus protocols used in blockchain systems, namely PBFT and PoW. We then describe key characteristics of Hyperledger Fabric and Intel SGX. Finally, we discuss the system model and our evaluation framework.

2.1 Distributed Consensus Protocols

The goal of a distributed consensus protocol is to reach agreement among multiple nodes. The key challenge in distributed consensus is node failures. Distributed database systems, for example Megastore [16] and Spanner [36], replicate data over multiple nodes, and rely on consensus to ensure consistency between the replicas. They assume the crash failure model wherein a faulty
node fails by crashing (i.e., it stops responding to requests). Many consensus protocols have been designed to operate under this failure model. Examples include Paxos [52] and Raft [70].

The crash failure model does not protect against arbitrary hardware failure, or against malicious applications and users [77, 72, 42]. Blockchain systems assume a stronger failure model, called Byzantine failure, which accounts for arbitrary deviation from the expected behaviours. A typical blockchain system uses consensus to establish consistent views of the ledger among mutually distrustful nodes. It therefore can be seen as a decentralized data management system specifically designed for Byzantine failures. In fact, most disruptive applications of blockchain are traditionally built on databases, for example financial [8], supply chain management [38] and data sharing applications [63].

In this paper, we focus on consensus protocols for blockchain systems. Two important properties of the consensus protocol are safety and liveness. The former means that honest (non-Byzantine) nodes agree on the same value, while the latter means they eventually agree on a value. In this section we briefly discuss two major classes of consensus protocols, namely BFT and PoW, and refer readers to [32] for a more comprehensive treatment of the design space. To better motivate and illustrate relevant design choices, we use the following example throughout the paper.

Running example. Consider a group of financial institutions that want to offer cross-border payment services to their customers. They implement a blockchain solution that provides a shared ledger for recording payment transactions and settlements. The ledger is distributed and its content is agreed upon by group members via distributed consensus. Given the amount of money at stake, the consensus protocol must be tolerant to Byzantine failures, so that group members are protected against attacks that compromise the ledger in order to double-spend or to revoke transactions. As the group can admit new members, the protocol should not assume an upper bound on the group size. In summary, the consensus protocol must support large groups with strong Byzantine fault tolerance, as well as high service request rates.

Byzantine fault tolerant protocols. BFT protocols assume a permissioned network setting, in which nodes and their messages are authenticated. The most well known BFT protocol is the Practical Byzantine Fault Tolerance (PBFT) [24]. This protocol allows the nodes (or replicas) to agree on a sequence of requests to be executed. It consists of three phases: a pre-prepare phase in which the leader broadcasts a next request in a pre-prepare message, the prepare phase in which the replicas agree on the ordering of the request by sending out a prepare message, and the commit phase in which the replicas commit to the request together with its order by sending out a commit message. Each node collects a quorum of prepare messages before moving to the commit phase, and executes the request only after it collects a quorum of commit messages. The leader is replaced by a view change protocol when nodes suspect leader failure. A node triggers view change when it fails to make progress after a time-out t. The protocol uses $O(n^2)$ messages for n nodes, and it achieves both safety and liveness property. Other BFT protocols [55, 49] extend PBFT to optimize for normal case (without view change) performance.

BFT protocols require $n \geq 3f + 1$ nodes and a quorum size of $2f + 1$ to tolerate $f$ Byzantine failures. In the cross-border payment example above, $f$ represents the number of Byzantine group members that are either malicious or are compromised by an attacker. BFT protocols assume that $f$ is less than a third of the group size. Given this condition, safety property is guaranteed under any network condition (i.e., even in fully asynchronous network). On the other hand, liveness requires partially synchronous network wherein messages are delivered within an unknown yet finite bound. BFT’s safety implies consensus finality, meaning that once reached, consensus decision will not be changed.

Proof of Work protocol. PoW protocols assume a permissionless setting in which any node can join and leave the network at will [69, 78], as typically a case of miners in Bitcoin and Ethereum cryptocurrency networks. The protocol randomly selects a leader to propose the next block that the network will accept, requiring a single round of multicast to reach agreement [69]. This alleviates the network communication bottleneck that PBFT and the likes observe, allowing the network size to scale. Leader selection is a probabilistic process in which a node must show a proof, which is a solution to a computational puzzle, to claim leadership. The probability of solving the puzzle is proportional to the amount of computational power the node possesses over the total power of the network, protecting the leader selection against Sybil attacks [35].

Being probabilistic, PoW does not prevent multiple nodes from claiming leadership and proposing new blocks roughly at the same time. It is possible to have two valid blocks extending the same parent block, creating a fork in the blockchain. A common approach to resolve such a conflict is for the nodes to pick the longest branch of the fork, such that after some number of blocks, one branch will be adopted by the entire network with very high probability.

PoW can scale to large number of nodes: 3000 and 4000 nodes for Bitcoin and Ethereum respectively [39]. However, its throughput is limited. The security model of PoW is different to that of BFT. It considers Byzantine tolerance in terms of the cumulative resource belonging to the Byzantine nodes (e.g., fraction of the total computational power), as opposed to the number of Byzantine nodes in BFT. Furthermore, safety property depends not only on Byzantine threshold, but also on network latency. For instance, under a fully synchronous network (i.e., zero network latency), safety is guaranteed against 50% Byzantine power [67]. However, this threshold drops quickly as network latency increases in a partially synchronous network (e.g., below 33% when the latency is equal to the block time [71]). Recent works study systems of rational and Byzantine nodes as opposed to honest and Byzantine nodes that BFT protocols consider. Under this model, PoW is more vulnerable to attacks such as selfish mining [37].

2.2 Hyperledger Fabric

Hyperledger Fabric is a platform for permissioned blockchain applications. It contains an implementation of PBFT. It uses Docker to execute smart contracts, and a key-value store to persist the blockchain data. Each node maintains fixed size queues to receive messages from other nodes. The network queue handles both request (transaction) messages and consensus messages. When a node receives a request from a client, it broadcasts the request to the rest of the network. All messages are forwarded to an internal work queue, which are then picked up by the consensus engine. Once consensus is reached for a batch of requests (or a block), the batch is placed on the work queue to be executed by the execution engine. Results of the execution is placed to the work queue to be committed to the storage.

2.3 Trusted Execution Environment with SGX

Enclave protection. A trusted execution environment (TEE) is a secure area within which confidentiality and integrity of the data and computation are protected. Intel SGX, a recently introduced set of CPU extensions, provides TEE support in the form of hardware enclave. An enclave is essentially a CPU-protected address space...
which is accessible by no other entities but the enclave owner. Multiple enclaves can be instantiated by non-privilege user processes. The enclaves are isolated from each other, from the operating system (OS) and from other user processes. The code within the enclave, however, can invoke OS services such as IO and memory management. Memory pages can be swapped out of the enclave memory, but they are encrypted before leaving the enclave.

Attestation. A user can verify if a specific TEE is correctly instantiated and running at a remote host via a remote attestation protocol. Intel SGX implements the protocol as follows [15]. Once the enclave in question has been initiated, the CPU computes a measurement of such enclave (i.e., the hash of its initial state). The CPU then signs the computed value with its private key under the Enhance Privacy ID (EPID) scheme. Given a signed attestation message, the user first verifies the signature using Intel’s Attestation Service (IAS), and then compares the measurement value against a known value.

Data sealing. The TEE can persist its state to non-volatile memory via the data sealing mechanism, which allows for recovery after crash failures. An SGX enclave seals its data by first requesting the CPU for a unique key bound to its measurement, then encrypting the data before storing it on persistent storage. This mechanism ensures the data can only be decrypted by the enclave that sealed it. However, enclave recovery using sealed data is subject to rollback attacks wherein an attacker (e.g., the malicious OS) provides stale data to the enclave [21, 58].

TEEs offer a number of cryptographic operations to the applications running inside. In particular, SGX provides two important functions, namely sgx_read_rand and sgx_get_trusted_time. The former uses hardware instructions to generate unbiased random numbers, while the latter returns the elapsed time relative to a reference point.

2.4 System and Threat Model

System model. We consider a blockchain system of \( n \) nodes which are well connected. Unless otherwise stated, the network is partially synchronous, in which messages sent repeatedly with finite time-out will eventually be received. Let \( f < n \) be the number of Byzantine nodes under the attacker’s control, the remaining \( n - f \) nodes are honest. We say that the system is making progress, or live, if it can generate blocks. Otherwise, when no blocks are generated, the system is said to have crashed.

Every node is provisioned with TEEs. While this work leverages Intel SGX, our solutions can be implemented in any other TEE instantiations. More specifically, we can replace SGX with other hardware-based TEEs such as TrustZone [14], Sanctum [28], TPMs [11], or with software-based TEEs such as Overshadow [25], without compromising security. A SGX enclave integrates easily with any blockchain software stack (e.g., Hyperledger Fabric), for it is used in the same way as a typical user-space library.

Threat model. The attacker has full control over the Byzantine nodes. It can read and write to memory of any running process, even the OS. It can modify data on disk, intercept and change the content of any system call. It can modify, reorder and delay network messages arbitrarily. However, the attacker is computationally bounded, and it cannot break standard cryptographic assumptions (e.g., cannot forge a valid signature or MAC of a message). Furthermore, it cannot compromise the TEE implementation, hence both confidentiality and integrity of the data and computation within the TEE are protected. We assume the software running inside the TEE (i.e., the trusted code base) is secure. We do not consider side-channel attacks against the TEEs and DoS attacks against the system. Since SGX is used, we note that rollback attacks are possible. We refer the readers to the Appendix A for counter measures against rollback attacks on the enclaves.

2.5 Evaluation Setup

We used BLOCKBENCH [33] to generate blockchain workloads and drive the experiments. The workload consists of read and write transactions for a smart contract implementing a key-value store (YCSB contract). We conducted the experiments using an in-house cluster of 48 servers, each equipped with ES-1650 3.5GHz CPU (with 6 cores and 12 threads), 32GB RAM, 2TB hard drive, and running Ubuntu 14.04 Trusty. The servers are connected via a 1GB switch. We used Intel SGX SDK [4] to implement the trusted code base. Since SGX is not available on our server hardware, we configured the SDK to run in simulation mode. We measured the latency of each SGX operation on Skylake 6970HQ 2.80 GHz CPU with SGX Enabled BIOS support, and injected it to the simulation. The results reported in the following sections are averaged over 5 independent runs. Table 2 lists the notations used in the paper.

3. SCALABLE DISTRIBUTED CONSENSUS PROTOCOLS REVISITED

This section presents an in-depth analysis of existing works on scaling distributed consensus protocols. Both BFT and PoW protocols are limited in their scalability, either in terms of the network size (i.e., number of nodes) or the overall throughput. The design space for improving them is vast. In this paper we focus on two design approaches, namely using trusted hardware and sharding. Table 1 compares the systems being considered. We discuss other related approaches in Section 6.
For each protocol, we first describe its design before presenting its performance evaluation. Then, we discuss the design choices that lead to limitations in the protocol’s scalability.

3.1 Scaling BFT

The most prominent BFT consensus protocol is the Practical Byzantine Fault Tolerance (PBFT) [24]. The original PBFT protocol requires \( n = 3f + 1 \) nodes to tolerate up to \( f \) Byzantine failures. Further, it has been shown not to scale beyond a dozen of nodes due to its quadratic communication complexity [33]. Repeating the cross-border payment example mentioned earlier, this means the distributed ledger facilitating settlement of the assets transfer can only be shared among a small number of institutions, and that as soon as an adversary compromises one third of that small number, he can perpetrate a double-spend attack.

As a consequence, scaling proposals for PBFT focus either on improving the protocol’s communication complexity to allow larger network size [47], or reducing the number of nodes required to tolerate \( f \) Byzantine failures [26, 53, 17].

3.1.1 AHL - Reducing the number of nodes

The complexity of PBFT – its need for three distinct phases – is necessary to ensure security when a Byzantine node can equivocate (i.e., issue conflicting statements to different nodes without being detected). To tolerate \( f \) Byzantine nodes that can equivocate in a quorum system like PBFT, quorums must be intersected by at least \( f + 1 \) nodes [57]. Consequently, the system requires \( n = 3f + 1 \) nodes, and the quorum size is \( 2f + 1 \).

Chun et al. [26] show that without equivocation, it is possible to tolerate \( f \) Byzantine failures with only \( n = 2f + 1 \) nodes, reducing the quorum size to \( f + 1 \). The smaller \( n \) means the lower communication cost incurred in tolerating the same number of failures. It also means that for the same number of nodes \( n \), the network can tolerate more failures.

One way to eliminate equivocation is to run the entire consensus protocol inside a TEE or enclave. This approach, adopted by Coco [10], effectively reduces the failure model from Byzantine to fail-crisis. Thus, any non-Byzantine consensus protocols, such as Raft [70] or Paxos [52], can be used. However, this approach incurs a large trusted code base (TCB) which is undesirable for security [76, 60, 61]. A large TCB makes it difficult, if not impossible, to conduct security analysis of the implementation, and increases the number of potential vulnerabilities. In fact, a popular security guideline is to keep the code base inside a TEE small [12].

Chun et al. propose PBFT-A2M [26] that incurs a small TCB. They leverage a trusted log abstraction called attested append-only memory. The log can be appended, truncated, and looked up. By running inside a TEE, operations on the log cannot be equivocated. Once a record is appended to the log, it cannot be modified.

AHL is the implementation of PBFT-A2M for Hyperledger Fabric v0.6, using SGX enclave to realize the TEE. AHL maintains different logs for different consensus message types (e.g., prepare, commit, checkpoint logs). Before sending out a new message, each node has to append its digest to the corresponding log. The proof of such append operation generated by the enclave is included in the message. AHL requires all valid messages to be accompanied by such a proof. Since the Byzantine node cannot tamper with the enclave’s operation nor forge the proof, it cannot equivocate messages. Each node collects and verifies \( f + 1 \) prepare messages before moving to the commit phase, and \( f + 1 \) commit messages before executing the request. However, the view change protocol in PBFT-A2M needs one extra round of communication in comparison to its counter-part in the original PBFT.

Algorithm 1 Message Aggregation Enclave.

```plaintext
procedure E-AGGRSIGN((m_1, m_2, ..., m_{f+1}))
(r, p, o) ← PARSE(m_1);
for each m_i do
  if PARSE(m_i) ≠ (r, p, o) ∨ ¬VERIFY(m_i) then
    return 1
  end if
end for
M ← SIGN((r, p, o));
return M;
end procedure
```

AHL maintains the logs in the enclave’s volatile memory, which will be lost if the enclave is stopped. To recover it states when restarting, the enclave needs to periodically seal the logs to persistent storage. However, enclave recovery is vulnerable to rollback attacks [58]. We extend AHL to protect defend the attacks, the details of which can be found in Appendix A.

Other proposals, such as TrInc [53], MinBFT [82], CheapBFT [44] or Hyster [17], implement simpler abstraction of monotonic counters inside the trusted hardware. They allow for binding a message with a certificate of trusted counter value. However, a faulty node can get two separate valid certificates for two conflicting messages. In this case, equivocation can only be detected when an honest node receives both certificates.

3.1.2 AHLR - Reducing communication complexity

While reducing the network size needed for tolerating \( f \) failures, AHL has the same communication complexity of \( O(n^2) \). This complexity is necessary because each node needs to collect authenticated messages from others in order to form quorums. Byzcoin [47] proposed an optimization wherein the leader uses a collective signing protocol (CoSi) [81] to aggregate other nodes’ messages into a single authenticated message. In this way, each node only needs to forward its messages to the leader, and verify the aggregate message from the latter. By avoiding broadcasting, the communication is reduced to \( O(n) \).

AHLR adds collective signing to AHL. Nonetheless, it does not follow Byzcoin approach of implementing the complex CoSi protocol. Instead, AHLR implements an enclave that verifies and aggregates messages. We dub this enclave E-AGGRSIGN, and details its logic in Algorithm 1. In particular, upon receiving \( f + 1 \) signed messages for a request \( req \), at phase \( p \) of consensus round \( o \), the E-AGGRSIGN enclave verifies the messages. If all of them are valid, it issues an authenticated proof \( M \) indicating that there has been a quorum for \( (req, p, o) \), with which other nodes can proceed to the next phase. We note that AHLR can be realized without increasing the TCB, but with increased computation and communication cost due to CoSi’s complexity.

3.1.3 Performance evaluation

We measure the throughput of the original PBFT protocol (denoted by HL), AHL, and AHLR based on Hyperledger Fabric v0.6 implementation. Each node is single-threaded and runs on a separate physical server. We launch one client per node, each client uses 16 threads, each thread generates \( r \) transactions per second (tps). Figure 2 compares performance of the three systems with increasing \( f \) and \( r \). We note that for a given \( f \), HL runs with \( n = 3f + 1 \) nodes, whereas AHL and AHLR run with \( n = 2f + 1 \) nodes.

We observe that AHL has higher throughput than HL for small value of \( f \), with peak throughput of 1200 tps compared to 900 tps. Moreover, AHL sustains more failures than HL. It crashes at \( f = 7 \).
The performance of AHLR is interesting. Since AHLR reduces the communication cost to $O(n)$, it is expected to scale better than AHL. Nonetheless, the opposite is observed. AHLR’s throughput is worse than those of HL and AHL, and the protocol crashes even earlier than HL. Careful examination reveals that the AHLR communication pattern makes the leader a single point of failure, due to two reasons. First, the leader incurs one extra communication round to coordinate other nodes. Second, a message drop at the leader is more consequential, for the rest of the network depends on the leader to make progress. If the leader fails to collect and multicast the aggregate message before the time out, the system triggers the view change protocol which is expensive. Figure 2 demonstrates the number of view change messages observed in different protocols, showing that AHLR suffers more view changes than AHL. In Appendix B, we derive an analytical model for the probability of view changes in AHL and AHLR, showing that AHLR communication pattern is less robust than AHL’s.

Discussion. The performance of AHLR suggests that there are other factors besides asymptotic complexity that affect scalability. Figure 2 shows that as $f$ increases, there are more view changes during which nodes stop processing requests. The large number of view changes demonstrates that the nodes get stuck in the view change phase. Indeed, view changes are triggered by two local timers: a request timer specifying an expected duration within which a request should be completed, and a view change timer indicating that of the view change phase.

We note that given a fixed network queue length, the probability of requests and consensus messages being dropped increases with $f$. A higher rate of message drop leads to more frequent triggering of view change. When it is high enough, the view change timer also fires repeatedly (i.e., the nodes are stuck in view change). One possible remedy is to lengthen the queue. However, no improvement is observed, because longer queues lead to longer processing time per request that exceeds the timer duration.

We observe that the network queue at each node has a disproportionate number of requests compared to consensus messages. We created a separate queue for consensus messages, but this did not lead to any performance gain, for the request queue is being filled as a faster rate. The reason is that Hyperledger Fabric strictly follows PBFT’s formal specification [23], requiring a non-leader node to broadcast any request it receives. We content that this is unnecessary, and the node only needs to forward the request to the leader.

3.2 Proof-of-Elapsed-Time

PoW protocols are considered resource-wasteful. For example, Bitcoin network consumes up to 56TWh of electricity annually, more than most countries in Africa [1]. Hyperledger Sawtooth’s Proof of Elapsed Time (PoET) proposes a replacement to PoW, enforcing a mandatory random wait time before a leader is elected.

PoET assumes nodes are equipped with Intel SGX CPU running a trusted enclave called E-PoET. Each node asks its E-PoET enclave for a wait time. Only after such wait time has passed does the enclave issue a wait certificate or create a new wait time. The node with the shortest wait time (i.e., the first to obtain a wait certificate) becomes the leader. The enclave generates the wait time as follows:

\[
\text{waitTime} = \text{MinimumWait} - \text{localMean} \cdot \log(d)
\]

wherein MinimumWait is a fixed system parameter, localMean is the product of expected block time and an estimated network size, and $d \in (0; 1]$ is the hash value of the previous certificate [6]. Assuming that the hash function is a random oracle [45], $d$ is a random seed. localMean is used to adapt waitTime’s deviation to the network size (i.e., larger network size leads to larger deviation and therefore lower probability of collision). We note that both Bitcoin and Ethereum have similar self-adjusting mechanisms that adjust PoW difficulty to the network size.

The node attaches the wait certificate to the block it proposes, which will be adopted by the network if the certificate contains the shortest wait time. Similar to PoW, PoET suffers from forks and stale blocks. Because of delays in blocks propagation, if multiple nodes get their certificates roughly at the same time, they will propose conflicting blocks, being unaware of the other. This conflict creates a fork in the blockchain, which is resolved as follows: the fork branch with highest aggregate localMean wins [5]. When the network size is fixed, the longest branch is chosen. All other blocks not on the selected branch are discarded as stale blocks.

One effect of stale blocks is the overhead of verifying them, which can affect the overall throughput. A more important implication is security (i.e., the probability that a block is unconfirmed at a later time). PoET has similar security guarantee as other PoW protocols. In particular, with zero network latency, a transaction that has $k$ block confirmations can only be unconfirmed with probability \( (\frac{s}{1-s})^k \) where $s$ is the fraction of nodes that the attacker controls. As the network latency grows, the system tolerance reduces [73]. In Bitcoin and Ethereum, the recommended value of $k$ is 6 and 30 respectively. It remains unclear what the appropriate value for PoET is.
Suppose that $n$ in which message propagation delay has a lower bound it issues a block $b$ work size, block time, and block sizes. We consider a simplified between stale block rate and other system parameters such as net- and throughput of the system. They also illustrate the relationship which show negative impact of stale block rate on both security
Discussion.

We observe that a larger block size leads to a higher stale block rate, the node has to spend more time verifying them, thus ending up with less capacity for processing the valid blocks. We evaluate performance of PoET based on Hyperledger Sawtooth v0.8 implementation by running up to 128 nodes instances on 32 physical servers. Each node runs on a separate CPU core with two threads, and is connected to $\sqrt{n}$ other nodes selected at random. We impose 50 Mbps limit to the bandwidth and 100ms latency on the network links. In our experiments, a single node can validate up to 1150 tps, which is the upper bound of the system throughput. We vary the block size from 4MB to 8MB, and the corresponding block time from 12 to 24 seconds. We observe that smaller block times (e.g., 8s for 4MB blocks) leads to a stale block rate of up to 40%. Figure 3 shows the throughput and stale block rates for increasing $n$. The throughput is peaked at one node where there is no stale block. As $n$ increases, block propagation time, which depends on block time and block size, increases and leads to degradation in both throughput and stale block rate.

We note that the throughput is not depicted against $f$ as with PBFT. The reason is that PoET and other PoW protocols make assumption on network synchrony in order to achieve safety. Therefore, in practice, it is difficult to quantify their fault tolerance directly in terms of $f$. Although we did not compare PoET with Ethereum due to their design differences, we note that PoET’s throughput is higher than Ethereum’s, which was reported in [33] to be around 100 tps for 32 nodes.

Recall that Equation 1 is expected to stabilize the throughput as network size increases. Instead, we observe a sharp drop of up to 83% (Figure 3 (left)). This can be explained by the increased stale block rate (Figure 3 (right)). More specifically, with more stale blocks, the node has to spend more time verifying them, thus ending up with less capacity for processing the valid blocks. We also observe that a larger block size leads to a higher stale block rate, and as a consequence, a lower throughput. This finding is in line with performance of Bitcoin network [30, 40], wherein larger block size means higher probability of forks and stale blocks.

Discussion. Our results are in keeping with those reported in [40] which show negative impact of stale block rate on both security and throughput of the system. They also illustrate the relationship between stale block rate and other system parameters such as network size, block time, and block sizes. We consider a simplified model in which message propagation delay has a lower bound $\delta$. Suppose that $n_1$ is the first node to obtain the wait certificate, and it issues a block $b_1$. Further suppose that $n_2$ obtains its wait certificate slightly after $n_1$, but before it receives $b_1$. Consequently, $n_2$ issues another block $b_2$ which is in conflict with $b_1$. Clearly, conflicting blocks happen not only when the two nodes obtain the same wait time (which is rare), but also when their wait time difference

is smaller than the propagation delay $\delta$. Let $T$ be the average block time, then in our simplified model, the probability of some node proposing a conflicting block is $\frac{1}{T}$. In the network of $n$ nodes, the expected number of conflicting blocks is $C \approx \frac{n^2}{T}$. Given the same block size and block time (i.e., fixing $T$ and $\delta$), a larger network (i.e., larger $n$) leads to higher $C$. When block time increases linearly with block size (in order to keep the expected throughput unchanged), we observed that $\delta$ grows super linearly: $2.3s$ for 4MB blocks and $6.2s$ for 8MB blocks in a network of 128 nodes. This results in higher $C$ when the block size increases.

3.3 Sharding Protocols

The distributed consensus protocols considered so far are leader based, which ultimately makes the leader’s processing capacity an upper limit of their overall throughput. To overcome this fundamental limitation, a common approach is to use sharding. A popular technique in database systems, sharding protocols partition the nodes (network partition) and the states (state partition) into smaller shards or committees. In the context of blockchain, sharding offers two benefits. First, each committee can be made reasonably small so that it is possible to run PBFT protocol among its nodes, which can achieve high throughput. Second, when there is light contention between transactions, the overall throughput can increase linearly with the number of committees. In other words, the throughput could scale linearly with the network size.

Three major sharding protocols for permissionless blockchains are Elastico [56], OmniLedger [48], and Chainspace [13]. Elastico uses network sharding, and does not support cross-shard transaction. Its sharding protocol splits the transactions per block to be processed in parallel by multiple committees. For each block, the miners perform PoW with low difficulty such that most miners can find a solution in reasonable time. The system specifies the number of committees as $2^k$. Two miners belong to the same committee if their PoW solutions share the last $k$ bits. Transactions are forwarded to corresponding committees based on their last $k$ bits.

Unlike Elastico, OmniLedger splits the global states into multiple shards, thus a transaction may be cross-shard (i.e., it has to be processed by multiple committees). In OmniLedger, committees are fixed for a long period called epoch (e.g., a day). At each epoch, every nodes are known using a permissionless identity blockchain. At the beginning, one node is selected at random to serve as a coordinator, using a verifiable random function (VRF) [65]. Next, the coordinator drives the RandHound protocol [80] that collects inputs from the nodes and generates a fresh, non-bias random value. Us-
ing this random value, each node computes a random permutation \( \pi \) of \([1..n]\). The node divides \( \pi \) into approximately equally-sized chunks, each chunk represents IDs of members in one committee. The global states are assigned to committees based on their last few bits. OmniLedger ensures consistency for cross-shard transactions by relying on a two-phase client-driven “lock/unlock” protocol.

Similar to OmniLedger, Chainspace partitions the global states into shards. However, it abstracts away the shard formation protocol and allows smart contracts to assign nodes to shards. Its security relies on such assignment to maintains 33% Byzantine threshold per shard. Chainspace uses a complex atomic commit protocol which incurs more cross-shard communication than OmniLedger. To disincentivize Byzantine nodes (and shards) from misbehaving, Chainspace provides an auditing mechanism that enables honest nodes in honest shards (i.e., shards that satisfy the 33% Byzantine threshold) to detect inconsistency and to trace the malicious shard. Nonetheless, the system does not specify how to recover from inconsistency. In this paper, we focus on Elastico and OmniLedger because of their secure shard formation protocols and simple cross-shard communication, and leave Chainspace to future work.

**Performance Analysis**

To ensure security of sharding, the nodes must be assigned to committees at random, otherwise the adversary could penetrate any committee to gain a majority. Both Elastico and OmniLedger assume synchronous communication during committee formation, wherein messages are delivered within a known bounded delay \( \Delta \). They rely on unbiased randomness to form good committees. Elastico uses PoW as source of randomness, which is not ideal as miners can selectively discard PoW solutions to influence the result [20]. We examine the cost of RandHound for committee formation by running up to 512 nodes on 48 physical servers. Each node is single-threaded and runs in a virtual machine. Communication between nodes goes through the network. Table 3 shows that the cost of RandHound is significant (e.g., it takes nearly 3 minutes to generate a random value from 512 nodes). We attribute this cost to the communication complexity of \( O(n^2) \) [80]. We ran the same experiments with 36 servers and observed an overhead increase of less than 12%. This suggests RandHound is communication bound, and therefore oversubscribing the servers did not significantly affect the results.

Security of sharding protocols also depends on the committee sizes. In particular, the committee cannot be too small as the attacker can penetrate a committee to control its a majority, thereby subverting the PBFT protocol. We define committee failure as the scenario in which a committee of size \( n' \) has more than \( \frac{2n' - 1}{3} \) Byzantine nodes. To tolerate attackers that control up to 25% nodes in the network (i.e., 25%-attacker), both Elastico and OmniLedger require more than 600 nodes per committee to keep committee failure probability below \( 2^{-20} \). We remark that a network of more than 600 nodes per committee brings only a slight increase of less than \( 12\% \) in the Hyperledger Fabric implementation. In particular, longer epochs reduce the frequency of forming new committees, amortizing the cost of committee formation. However, it is easier for the attacker to adapt and compromise some committees. Alternatively, the Byzantine tolerance threshold can be lowered in order to use smaller committees. Our goal is to exploit trusted hardware to address the two limitations of sharding protocols without sacrificing security guarantee.

<table>
<thead>
<tr>
<th>n</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>15.73</td>
<td>22.82</td>
<td>40.97</td>
<td>76.34</td>
<td>151.76</td>
</tr>
</tbody>
</table>

**Table 3: RandHound runtime with group size \( c = 16 \).**

<table>
<thead>
<tr>
<th>( s% )</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Committee size</td>
<td>1</td>
<td>28</td>
<td>55</td>
<td>106</td>
<td>227</td>
<td>648</td>
<td>4625</td>
</tr>
</tbody>
</table>

**Table 4: Required committee size against \( s\% \) attackers, to ensure failure probability below \( 2^{-20} \).**

4. **DESIGN TO SCALE**

In this section, we draw insights from the results presented in the previous section, and codify them into two design principles. Following these principles, we describe optimizations that improve AHL, AHLR, and PoET. Finally, we present a new, efficient design for shard creation.

**4.1 Design Principles**

The performance analysis in the previous section reveals two insights. First, AHL, AHLR, and PoET demonstrate a negative effect of network size on the overall throughput. These protocols essentially implement a replicated state machine (RSM) system which is driven by a leader. That is, the leader processes requests sequentially and the replicas merely follow. Therefore, the system throughput can only be as high as the leader’s. When the network grows in size, message delay and message drop rate in AHL and AHLR increase, and the probability of nodes proposing conflicting blocks in PoET escalates. All these consequences cause the overall throughput to suffer.

The second insight is that frequent dropping of consensus messages in PBFT triggers repeated view change protocols and consequently paralyzes the system. Although it is expected that more messages are dropped as the number of nodes grows, we observed an unusually high drop rate in the Hyperledger Fabric implementation. The reason is that this implementation uses the same queue for both request and consensus messages. With more nodes, there are more requests filling up the network queue more quickly, causing consensus messages to be rejected. Worse still, every replica multicasts requests it receives to the rest of the network, further exacerbating the imbalance in the queue. This multicast is included in PBFT’s formal specification as a means to ensure all the nodes receive the request [23]. However, it is unnecessary as the leader is already obliged to broadcast the request in the pre-prepare phase.

**Design Principle 1 - Scale up by scaling down.** This principle follows directly from the first insight. It recommends exploiting trusted hardware to reduce the number of nodes that take parts on the consensus (i.e., the effective network size). A smaller number of nodes means less communication overhead and lower rate of stale blocks, which translate to better throughput. The challenge is to maintain the same level of security as before scaling down. AHL and AHLR are examples of this principle, in which the effective network size is reduced from \( 3f + 1 \) to \( 2f + 1 \).

**Design Principle 2 - Prioritize consensus messages.** Generalizing from the second insight, this principle recommends assigning importance scores to different types of messages, with consensus messages carrying the highest score. This could prevent consensus messages from being rejected from the queue or delayed after other message types.
4.2 AHL+: Prioritizing Consensus Messages

Following the second principle, the drop rate of consensus messages should be minimized, which ideally should be zero. To this end, we introduce the following optimizations to AHL. First, we divide the network queue into two parts: one for request, and the other for consensus messages. Specifically, we split the original queue, implemented as a Go channel, into two separated channels. The received message from the network socket contains metadata that determines its type (request or consensus) and is forwarded to the corresponding channel. While this isolation prevents consensus messages from being being overwhelmed by request messages, it did not deliver the expected improvement. The request queue is filled at a much higher rate than the consensus queue, causing long delay in processing the latter. Our second optimization removes request multicast at the replicas. In particular, upon receiving a request, the replica only forwards it to the leader, instead of broadcasting it to the network.

We note that the ideal optimization is to maintain a priority queue out of consensus and request messages, giving higher priority to the former. However, we opted not to introduce intrusive change to the Fabric code base, and implemented instead the more simple optimization above. As shown in the next section, even such simple optimization leads to significant improvement. We leave the priority queue implementation and evaluation to future work.

Security analysis. Both optimizations do not affect safety of the consensus protocol because no existing messages are changed and no new messages are introduced. However, they may affect liveness since a request has higher chance of being dropped at the leader due to smaller queues. Furthermore, since there is only one copy of the request, if dropped by the leader, the request is lost and the client must resend. However, we show in next section that the overall liveness of the system, in fact, improves significantly.

4.3 S-PoET: Scaling Down Effective Network

We apply the first principle to PoET so as to reduce the stale block rate, allowing the system to sustain higher throughput as the network size grows. More specifically, we limit the number of nodes competing to propose the next block (i.e., smaller effective network), thus decreasing the expected number of conflicting blocks. To address the challenge of maintaining the security level, we exploit SGX to ensure that the effective network consists of nodes selected uniformly at random.

We introduce a small change to PoET trusted code base, and call the result S-PoET. When invoked to generate a wait certificate, the enclave in S-PoET uses sgx_read_rand to generates a random l-bit value $q$ that is bound to the wait certificate. This value $q$ is used to determine if the node belongs to the effective network. In particular, only wait certificates containing $q = 0$ are considered valid. The node with a valid certificate and the shortest wait time becomes the leader.

Without loss of generality, S-PoET leader selection can be deemed a two-stage process. The first stage samples uniformly at random a subset of $n' = n \cdot 2^{-\ell}$ nodes. The second stage selects uniformly at random a leader among these $n'$ nodes. Following the model in Section 3.2, the expected number of stale blocks is $C' \approx n'^2$, which is smaller than that of PoET, for $n' < n$.

Security analysis. Let $s$ be the fraction of nodes controlled by the attacker. Compared to PoET, S-PoET does not grant the attacker any advantage in becoming the leader, because the probability of this event remains at $\frac{n'}{n} = s$. While both waitTime and $q$ are generated by the same enclave, we note that they are independent.

The former uses the hash value of the enclave’s previous certificate as seed (Section 3.2), whereas the latter is generated using sgx_read_rand. The adversary cannot repeatedly invoke the enclave to obtain different $q$ for the same wait certificate because $q$ can only be generated once for each certificate.

4.4 GRANBea: Scaling-Out Sharding Protocols

Existing sharding protocols, namely Elastico and OmniLedger, already follow our first design principle to some extent. They split the network into smaller committees, each of which runs an independent instance of PBFT. However, their committee formation incurs prohibitive overhead, and the established committees are still too large for PBFT. We describe a new sharding protocol, called GRANBea, that further follows the first design principle to address the above mentioned limitations without sacrificing security. A high level comparison of GRANBea and the state-of-the-art is shown in Figure 5.

4.4.1 Efficient committee formation

Similar to OmniLedger, GRANBea first collects a fresh unbiased random value, based on which it assigns nodes into committees. However, GRANBea implements a trusted randomness beacon — a function that returns fresh, unbiased random numbers — inside a secure enclave. For this phase, GRANBea makes the same network assumption to that of OmniLedger and Elastico (i.e., the network is synchronous with the bounded delay $\Delta$).
One naive solution to implement a trusted randomness beacon is to follow the design of PoET. More specifically, each node asks its enclave for a random value $\text{rnd}$, and broadcasts it to the network. After $\Delta$ has passed, all nodes lock in the lowest $\text{rnd}$ that they have seen. This solution incurs $O(n^2)$ communication complexity.

The trusted randomness beacon is implemented in an enclave called E-RANDGEN whose logic is detailed in Algorithm 2. The enclave samples uniformly at random a subset of nodes that could propose and broadcast $\text{rnd}$. At the beginning of each epoch, each node invokes the enclave with an epoch number $e$. The enclave generates two random values $q$ and $\text{rnd}$ using two independent invocations of the $\text{sgx_read_rnd}$ function. It returns a signed certificate containing $(e, \text{rnd})$ if and only if $q = 0$. The certificate is broadcast to the network. After a time $\Delta$, every node locks in the lowest $\text{rnd}$ it has seen for the epoch $e$, and uses it to compute the committee assignment. We note that E-RANDGEN is configured such that it can only be invoked once per epoch. This is to prevent an attacker from selectively discarding the enclaves output in order to bias the final randomness. If the nodes fail to receive any message after a time $\Delta$, which happens when no node is able to collect $(e, \text{rnd})$ from its enclave, they increment the epoch number $e$ and repeat the process.

The bit length $l$ of $q$ defines the probability that none of the nodes could collect $(e, \text{rnd})$ in epoch $e$, and the system has to repeat the process, which is $P_{\text{repeat}} = (1 - 2^{-l})^n$. It can be tuned to achieve a desirable trade-off between $P_{\text{repeat}}$ and the communication overhead, which is $O(2^{-1}n^3)$. For example, when $l = \log(z)$ for some constant $z$, $P_{\text{repeat}} \approx 0$ and the communication cost is $O(n^3)$. When $l = \log(n)$, $P_{\text{repeat}} \approx \frac{1}{2^n}$ and the communication cost is $O(n)$.

Security analysis. Because $q$ and $\text{rnd}$ are generated independently by the CPU, their randomness is not influenced by the attacker [3]. Furthermore, the enclave only generates them once per epoch, therefore the attacker cannot selectively discard the outputs to bias the final randomness and the committee assignment.

4.4.2 Reduce committee size

The committee size is critical to the security of a sharding protocol. Specifically, let $s$ and $s'$ be the Byzantine threshold that the entire network, and each committee can tolerate, respectively. The probability that a committee of size $n'$ contains more than $f' = s' \cdot n'$ (i.e., security is broken) is given by:

$$P_{\text{fail}} = 1 - \sum_{k=0}^{f'} \binom{n'}{k} s^k (1 - s)^{n' - k} \quad (2)$$

Elastico and OmniLedger assume $s = 0.25$ and $s' = \frac{1}{2}$, which means in order to keep $P_{\text{fail}} \leq 2^{-20}$, $n'$ must be greater than 600. In GRANBEA, each committee runs AHL+, or $s' = \frac{1}{3}$. This results in significantly smaller $n'$ (e.g., $n' = 80$ for $P_{\text{fail}} \leq 2^{-20}$). Smaller committees can lead to better throughput for two reasons. First, each committee attains higher throughput due to lower communication overhead. Second, there are more committees in the network, which increases the overall throughput under light contention between transactions.

5. PERFORMANCE EVALUATION

In this section, we present comprehensive evaluations of our proposed optimizations and new design, namely AHL+, S-PoET, and GRANBEA. We show that AHL+ attains $7 \times$ higher throughput and $5 \times$ lower latency than the original Hyperledger Fabric, scales to a larger network sizes, and tolerates more failures. S-PoET reduces the stale block rate by $4 \times$, which enables it to sustain a high throughput as network size increases. In GRANBEA, committee formation is two orders of magnitude more efficient, and the resulting committees are two orders of magnitude smaller.

We quantify the cost of using SGX in terms of security and runtime overhead. Table 5 details the TCB sizes of the trusted components in the protocols. Recall that a large TCB increases the number of potential software vulnerabilities, which is undesirable for security. Each enclave in our systems are implemented in than 1310 lines of code, which is small and hence amenable to formal verification. Table 6 details runtime costs of enclave operations on SGX-enabled processor. The most expensive operations involve public key signatures (e.g., signing and signature verification cost about 450µs and 844µs, respectively). Context switching and other symmetric key operations take less than 5µs.

5.1 AHL+

We compare AHL+ against three baselines: the original Hyperledger Fabric (HL), HL+ which is the original Hyperledger Fabric enhanced with the two optimizations discussed in Section 4.2, and AHLR+ which is the AHLR protocol (Section 3.1.2) enhanced with the two optimizations.

Figure 6 and Figure 7 show performance of the three systems with increasing $n$ and $f$.

\begin{table}[h]
\centering
\small
\begin{tabular}{cccccc}
\hline
PoET & S-PoET & AHL & E-AGGRSign & E-RANDGEN \\
1300 & 1310 & 780 & 130 & 100 \\
\hline
\end{tabular}
\caption{TCB size (LoC) of different trusted modules.}
\end{table}

\begin{table}[h]
\centering
\small
\begin{tabular}{lcc}
\hline
Operations & Time ($\mu$s) \\
ECDSA Signing & 458.4(±0.4) \\
ECDSA Verification & 844.2(±0.8) \\
SHA256 & 2.5(±0.1) \\
AHL Append & 5.1(±0.1) \\
AHL Lookup & 465.3(±0.8) \\
AHL Truncate & 2.1(±0.1) \\
E-AGGRSign ($f = 8$) & 8031.2(±2.3) \\
E-RANDGEN & 482.2(±0.5) \\
PoET-GetWaitTime & 465.2(±0.5) \\
PoET-GetWaitCertificate & 1311.5(±0.9) \\
\hline
\end{tabular}
\caption{Runtime costs of enclave operations (excluding enclave switching cost which is roughly 2.7µs).}
\end{table}

The number of clients $c$ is the same as the number of nodes, which means the offered load grows with $n$. We note that for the same number of nodes, as in the left figure, AHL+ and AHLR+ tolerate more Byzantine failures than HL. In other words, for the same $f$, as in the middle figure, HL uses more nodes. For example, at $n = 13$, HL can only tolerate $f = 4$ failures, as opposed to $f = 6$ in the other systems.

At $n = 1$, the throughput is around 400 tps for all systems, because Hyperledger server restricts the rate of external client RESTful requests. As $n$ increases, AHL+ scales much better than HL. The former continues to offer over 300 tps with $f = 12$, while the latter crashes at $f = 3$. We observe that HL’s throughput before crashing is several times lower than that of AHL+ ($7 \times$ with
Another observation is that the two optimizations deliver significant improvement, even without the use of trusted hardware. In particular, the throughput of HL+ is close to that of AHL+. However, the former’s throughput decreases faster and the system crashes at \( f = 12 \). In other words, trusted hardware is an important factor that prevents the system from crashing. The performance of AHLR+ is even closer to that of AHL+, but it is still worse because of the communication bottleneck at the leader. As discussed in Section 3.1.2, the leader in AHLR+ performs one extra round of communication, and incurs higher probability of triggering view changes. The latency of these systems are shown in Figure 8. AHL+ always has the lowest latency, whereas those of HL and AHL are only shown before they crash. HL+ maintains relatively low latency, but inflates quickly after \( f = 8 \).

Figure 9 shows performance of the three systems when the offered load is fixed with \( c = 5 \). The throughputs in all systems decrease as \( f \) increases, because of the inherent communication overhead. However, the degradation is at a much lower rate than

**Figure 6:** Throughput comparison with varying \( n \) where \( c = n \).

**Figure 7:** Throughput comparison with varying \( f \) where \( c = n \).

**Figure 8:** Latency comparison with varying \( f \).

**Figure 9:** Performance with a fixed number of clients: \( c = 5 \).

**Figure 10:** Throughput with varying \( r \) and \( t \), where \( f = 2 \).
that in Figure 7, due to fewer messages in the systems. This suggests that admission control can help sustain performance at scale.

Figure 10 shows the effect of transaction rate $r$ and time out $t$ on the overall throughput. $r$ is varied from 1 tps to 256 tps, $t$ from 1s to 256s. When $r$ increases, in all three systems the throughputs increase and then decrease. This is because the systems are not saturated for small values of $r$, and they get overloaded (more dropped messages and view changes) when $r$ is large. For small values of $t$, HL fails to generate any blocks because view change is triggered prematurely before a consensus round completes. On the other hand, a consensus round in AHL+ and AHLR+ finishes faster, therefore blocks are generated before view change happens. The degradation at higher $t$ is because the leader has already failed before $t$ elapsed, thus the nodes wait unnecessarily before starting view changes.

Finally, we examined the cost breakdown for a block of transactions to better understand the performance differences. Figure 11 compares the systems in terms of consensus time, execution time and view change time. The cost of transaction execution is an order of magnitude smaller than consensus. HL stops generating blocks after $f = 2$, thus there are no consensus time after this point. Furthermore, its view change time indicates that the system is stuck in repeated view changes for $f > 2$. The costs of AHLR+ are higher than AHL+, because of the bottleneck at the leader.

### 5.2 S-POET

We compare S-POET’s performance with that of the original PoET using the same setting as in Section 3.2. Moreover, we set the bit length of $q$ to $l = \frac{\log(n)}{\log(\log(n))}$, which results in the expected effective network size of roughly $\sqrt{n}$. We report their throughput and stale block rates in Figure 12.

The results demonstrate that S-POET indeed reduces the stale block rate. As discussed in Section 4.3, this is due to the use of $q$ to restrict expected number of conflicting blocks. Consequently, it allows the system to maintain a high throughput at scale. In particular, at $n = 128$, PoET’s stale block rates is as high as 0.15, while that of S-POET is as low as 0.03. This translate to a significant difference in throughput, with PoET’s drops to 125 tps, whereas that of S-POET remains well above 500 tps. In another word, our optimization enables S-POET to attain 4× higher throughput in comparison with PoET.

### 5.3 GRANBEA

We compare GRANBEA against OmniLedger in terms of committee formation time and the required committee size such that $P_{\text{bad}} \leq 2^{-20}$ using the same experimental setup as in Section 3.3. In GRANBEA, we set $l = \log(n) - \log(\log(n))$, attaining $O(n \log(n))$ communication overhead while keeping $P_{\text{repeat}}$ small (e.g., below $2^{-11}$ for $n = 256$). We measure empirically the maximum propagation delay in different network sizes for a 1KB message, and conservatively set $\Delta$ to be 3× the measured values. In our experiments, $\Delta$ ranges from 2s to 4.5s.

**Committee formation.** Figure 13 (left) compares the costs of committee formation in OmniLedger against that in GRANBEA. It shows that GRANBEA’s committee formation is up to 32× faster than OmniLedger’s. We attribute this gap to the difference in their communication complexity: $O(n \log(n))$ versus $O(nc^2)$.

**Committee size.** Figure 13 (right) compares the committee sizes against increasing Byzantine threshold. OmniLedger requires exponentially large committees to tolerate more Byzantine failures. On the other hand, GRANBEA maintains a slower growth, and keeps the committees up to two orders of magnitude smaller.
5.4 Discussion of Future Work

This section has demonstrated how AHL+, S-POET and GRABEA improve the scalability of existing blockchain systems. However, the current implementations and evaluations have a number of limitations that can be addressed in future work. First, AHL+ does not fully prioritize consensus over request messages. It is not clear whether using a priority queue in such a way that the consensus messages are always drained before the request messages will result in better performance. Second, our evaluations have not considered end-to-end performance of GRABEA, in particular the overall throughput when running with smaller committee sizes and larger number of committees. We note that a full-fledged sharded blockchain requires many other components, including identity blockchain, two-phase lock/unlock protocol to handle cross-shard transactions, network, and execution engine. Third, the evaluations did not simulate any Byzantine failures. In fact, there are no failures during the experiments; a node only appears to have failed because it is overloaded with network messages. We plan to evaluate our solutions under simulated Byzantine failures in future work. However, we believe that the performance in the presence of such failures will be significantly lower [27]. Finally, the experiments were run in SGX simulation mode instead of real SGX hardware, due to the unavailability of SGX-enabled processors on our in-house servers. We believe that using real hardware will not affect the relative performance presented in the paper.

6. RELATED WORKS

Scaling BFT. Early works on scaling BFT protocols have focused on reducing communication cost [29, 49], number of replicas [26, 53, 82, 44] or execution cost [34]. They either do not use commodity trusted hardware or rely on hardware with limited performance such as smart cards or FPGA. Our work, on the other hand, achieves scalability using Intel SGX.

Using SGX for distributed consensus. PoET is one of many consensus protocols that exploit SGX. Hyperledger [17] also relies on SGX to implement TrInc instances [53]. It focuses on scaling consensus in multi-core settings, and implements a two-phase ordering protocol which can run in parallel. Hyperledger allows honest nodes to detect equivocation upon receiving conflicting messages, but unlike AHL+, it cannot prevent the malicious node from issuing such conflicting messages. Proof-of-Useful-Work (PoUW) [84] is a variant of PoW that uses SGX to certify that a miner has executed some useful workloads. Instead of solving PoW puzzles, the miners execute some workloads. The execution is measured by the PoUW enclave which issues a leader certificate based on the amount of computation the miner has completed. Similar to PoET, only nodes with valid certificates can propose a new block. We remark that Hyperledger and PoUW can serve as other case studies in our work, besides AHL, AHLR, and PoET. Given the source code unavailability, we plan to include them in future work.

Microsoft Coco [10] implements the entire consensus protocol in a trusted execution environment, reducing the failure model to crash failure. As a result, more efficient protocols such as Raft [70] and Paxos [52, 51] can be used. However, this approach incurs a much larger TCB than AHL+, making it more vulnerable to exploitation in the software stack. Our work only considers BFT and PoW based protocols. Nevertheless, in our preliminary evaluation of Quorum [43], a similar system that implements Raft consensus, we observed very low throughputs (i.e., less than 200 tps). This poor performance is due to the overhead of the Ethereum software stack used for the non-consensus components.

Other consensus protocols for blockchains. Stellar [9] and Ripple [8] assume a federated network setting. A node belongs to a federate comprising a set of other nodes that it trusts. Such a federate is called a “quorum slice” in Stellar or a “unique node list” in Ripple. The node only communicates with members of its federate during consensus. Ripple’s underlying consensus is a variant of PBFT. Stellar, on the other hand, proposes a new Byzantine agreement protocol in which a node ratifies a transaction if and only if its quorum does so. These protocols demonstrate high throughput at large scale, but their security depends on correct federate configuration, which is either non-trivial in large scale or subject to high degree of centralization.

The most popular alternative to PoW in permissionless settings is Proof of Stake (PoS) protocol. In PoS, nodes are required to stake their resources (e.g., cryptocurrencies) to participate in the protocol. PoS protocols can be divided into two types: chain-based and BFT-based [22]. Chain-based PoS protocols [18, 46] choose availability over consistency, and their security model is similar to that of PoW. In each round, a node is selected to be the leader based on its stake. The leader is granted a right to propose the next block that the network will adopt. To guarantee fair and random leader selection, these protocols require a source of randomness. Bentov et al. [18] use the blockchain’s current state to derive the required randomness. Ouroboros [46] employs a publicly verifiable secret sharing scheme (PVSS) [79] to implement a secure multi-party protocol that computes the randomness for the leader selection process. In contrast, PoET and S-PoET, which can be seen as instances of chain-based PoS protocols wherein every node has equal stake, rely on commodity trusted hardware to enable unbiased and secure leader selection. To the best of our knowledge, our work is the first performance study of chain-based PoS.

BFT-based PoS protocols [50, 64, 41], on the other hand, require nodes to cast votes over multiple rounds in order to reach consensus. The votes may be weighted based on voters’ stakes. Examples of BFT-based PoS protocols include Tendermint [50], Algorand [64], and Dfinity [41]. Tendermint [50] is a variant of PBFT in which consensus messages are weighted according to the nodes’ stakes, and certificates are formed based on the messages’ weight as opposed to number of messages as in PBFT. Algorand [64] employs a Verifiable Random Function (VRF) [65] to select a subset of nodes that participate in a Byzantine Agreement protocol for the next block. Similarly, Dfinity [41] relies on a threshold signature scheme [19] to implement a decentralized randomness beacon whose output is used to select a subset of nodes that are authorized to notarize new blocks. These protocols aim to overcome scalability issues of BFT and PoW. As such, our work can be extended to include them; that is, to evaluate their potential improvement with trusted hardware.

Scaling blockchain with database techniques. While consensus protocols have been the main focus of scaling blockchains, there are a number of works that exploit database techniques to improve other parts of the blockchain software stack. Forkbase [83] is the first storage designed for blockchains. Its data model and architecture are optimized for blockchain workloads. In particular, Forkbase supports analytical queries at orders of magnitude lower cost than the current key-value backends. Dickerson et al. [31] add concurrency to Ethereum execution engine by using software transaction memory primitives. They demonstrate up to 60% speed up in execution time using only three threads. We expect more works in improving the blockchain storage and execution engine. While orthogonal to our work in scaling consensus protocols, they can be combined to offer highly scalable blockchain solutions.
Off-chain scaling. In addition to improving consensus and other blockchain components, an orthogonal approach to scalability is to move as many transactions off the blockchain as possible. Such off-chain solutions allow users to execute transactions and reach consensus directly among each other, requiring minimal interaction with the blockchains. In particular, the blockchain is used only for disputes and settlements. Examples of off-chain solutions include payment channel [54] and state channel [66].

7. CONCLUSIONS

In this paper, we examined three scaling proposals for distributed consensus protocols and showed that there remain scalability limitations. Following the performance evaluation, we propose two design principles which are scale up by scaling down, and prioritize consensus messages. We then presented optimizations that harness trusted hardware, in particular Intel SGX, to improve upon the state-of-the-art solutions. We demonstrated that the optimizations indeed offer significant performance enhancement over the existing systems. More specifically, a SGX-based PBFT protocol achieves up to 7× higher throughput than Hyperledger Fabric, and prevents the system from crashing as the network size increases. The optimization over PoET enables the system to maintain high throughput at scale. Finally, we proposed a new sharding protocol GRANBEA that improves the shard creation phase by 32× while keeping the shard sizes order of magnitude smaller in comparison to existing techniques.

8. REFERENCES

AOLN E. Pavlou and R. T. Snodgrass. Forensic analysis of
[74] B. Schoenmakers. A simple publicly verifiable secret sharing
scheme and its application to electronic voting. In Annual
[75] D. Schwartz, N. Youngs, A. Britto, et al. The ripple protocol
open enclaves to
open enclaves halt or crash. This mechanism, however, open enclaves to
AHL+ prevents equivocation by enforcing that one sequence number
cannot be associated with two contradicting messages. This
enforcement is realised by requiring each node to append an digest of
the message to the attested append-only log before sending it
out. The log is indexed using the sequence number. The append-
only property ensures that once a message is appended to a log,
it cannot be equivocated (i.e., no other message can be associated
with that sequence number). It is essential for the enclave in AHL+
to keep track of the heads, or the latest sequence numbers, in its
logs. The enclaves periodically save its latest log heads to persist-
ent storage, so as they can correctly resume if they are restarted.
Rollback attacks, however, threatens to compromise the append-
only property. In particular, the adversary can cause the enclave
to restart, and supply it with a stale log heads upon its resumption.
The enclave that resumes with stale log heads effectively “forget”
all the messages that are appended to its logs after those sequence
numbers, allowing the adversary to equivocate. Assuming the last
sequence number the enclave processes prior to its restart is \(H\),
on resumption, it must not accept any messages with sequence
number lower than or equal to \(H\). Safety of the protocol, however,
does not strictly require the enclave to resume exactly at sequence
number \(H + 1\). It suffices for the enclave to resume at any sequence
number that is higher than \(H\).

In the presence of roll back attack, the resuming enclave cannot
blindly trust the host to supply it with a correct log heads. Instead,
it runs an estimation protocol that estimates an upper bound, \(H_M\), on the latest sequence number it would have observed if it were
not crashed. Clearly, \(H_M \geq H\). The enclave starts the estimation
protocol by querying all its peers for sequence number of their last
checkpoint, denoted by \(ckp\). The queries and responses include
nonces to guarantee freshness. The resuming enclave uses the re-
sponses to select \(ckp_M\), which is a value \(ckp\) it receives from one
node \(j\) such that there are \(j\) replicas other than \(j\) reported values
for \(ckp\) less than or equal to \(ckp_M\). It then sets the value \(H_M\)
to \(H_M = L + ckp_M\) where \(L\) is the preset difference between
the node’s high water mark and low water mark. The test against \(ckp\)
responses of other replicas ensures that \(ckp_M\) is greater than the
sequence number of any stable checkpoint the resuming enclave
may have; otherwise, there must be at least \(ckp\) responses that are
larger than \(ckp_M\), which is not possible due to quorum inter-
section.

The resuming enclave and its host node passively observe the
protocol, receiving and processing messages from other nodes.
Nonetheless, the enclave will not append any message to its logs
until it collects a correct stable checkpoint with a sequence number
greater than or equal to \(H_M\). At this point, we consider the resuming
enclave is recovered and fully functional. Since \(H_M\) is an upper
bound on the sequence number the AHL+ enclave would observe
had it not been crashed, and that the host node cannot send any
message with sequence number lower than \(H_M\) once its enclave
is restarted, the protocol is safe from equivocation.

S-PoET. Recall that the random value \(q\) is bound to a specific
wait certificate, which, in turn, is bound to a waitTime and a par-
ticular block height. When the adversary restart the enclave, not
only the value \(q\) is lost, but also are the wait certificate and its
concerning waitTime. In another word, it cannot imme-
diately request for the wait certificate after the restarting enclave
resumes its operation. Instead, it has to first request for another
waitTime, waits until such duration expires, and then collects a
wait certificate. As such, the adversary cannot apply the rollback
attack to bias the eligibility of the wait certificate.

APPENDIX

A. ON ROLLBACK ATTACKS

Data scaling mechanism enables enclaves to save their data and
internal states to persistent storage so that they are not lost when the
enclaves halt or crash. This mechanism, however, open enclaves to
rollback attack \([21, 58]\), where in the resuming enclaves are tricked
to load properly sealed but stale data, posing serious risks on state-
ful enclaves.

AHL+. AHL+ assumes stateful enclaves that implement attested
append-only log to combat equivocation. More specifically, each
message in the PBFT protocol is assigned a sequence number, and
AHL+ prevents equivocation by enforcing that one sequence number
cannot be associated with two contradicting messages. This
enforcement is realised by requiring each node to append an digest of
the message to the attested append-only log before sending it
out. The log is indexed using the sequence number. The append-
only property ensures that once a message is appended to a log,
it cannot be equivocated (i.e., no other message can be associated
with that sequence number). It is essential for the enclave in AHL+
to keep track of the heads, or the latest sequence numbers, in its
logs. The enclaves periodically save its latest log heads to persis-
tent storage, so as they can correctly resume if they are restarted.
Rollback attacks, however, threatens to compromise the append-
only property. In particular, the adversary can cause the enclave
to restart, and supply it with a stale log heads upon its resumption.
The enclave that resumes with stale log heads effectively “forget”
all the messages that are appended to its logs after those sequence
numbers, allowing the adversary to equivocate. Assuming the last
sequence number the enclave processes prior to its restart is \(H\),
on resumption, it must not accept any messages with sequence
number lower than or equal to \(H\). Safety of the protocol, however,
does not strictly require the enclave to resume exactly at sequence
number \(H + 1\). It suffices for the enclave to resume at any sequence
number that is higher than \(H\).

In the presence of roll back attack, the resuming enclave cannot
blindly trust the host to supply it with a correct log heads. Instead,
it runs an estimation protocol that estimates an upper bound, \(H_M\), on the latest sequence number it would have observed if it were
not crashed. Clearly, \(H_M \geq H\). The enclave starts the estimation
protocol by querying all its peers for sequence number of their last
checkpoint, denoted by \(ckp\). The queries and responses include
nonces to guarantee freshness. The resuming enclave uses the re-
sponses to select \(ckp_M\), which is a value \(ckp\) it receives from one
node \(j\) such that there are \(j\) replicas other than \(j\) reported values
for \(ckp\) less than or equal to \(ckp_M\). It then sets the value \(H_M\)
to \(H_M = L + ckp_M\) where \(L\) is the preset difference between
the node’s high water mark and low water mark. The test against \(ckp\)
responses of other replicas ensures that \(ckp_M\) is greater than the
sequence number of any stable checkpoint the resuming enclave
may have; otherwise, there must be at least \(ckp\) responses that are
larger than \(ckp_M\), which is not possible due to quorum inter-
section.

The resuming enclave and its host node passively observe the
protocol, receiving and processing messages from other nodes.
Nonetheless, the enclave will not append any message to its logs
until it collects a correct stable checkpoint with a sequence number
greater than or equal to \(H_M\). At this point, we consider the resuming
enclave is recovered and fully functional. Since \(H_M\) is an upper
bound on the sequence number the AHL+ enclave would observe
had it not been crashed, and that the host node cannot send any
message with sequence number lower than \(H_M\) once its enclave
is restarted, the protocol is safe from equivocation.
GRANBea. Recall that the random values \( q \) and \( \text{rnd} \) are bound to the epoch number \( e \) and a counter \( v \) to prevent the adversary from selectively discarding the enclave’s output to bias the randomness. These values, nonetheless, are stored in the enclave’s volatile memory. The adversary may attempt to restart the enclave and invoke it using the same epoch number \( e \) to collect different values of \( q \) and \( \text{rnd} \). Fortunately, the adversary only has a window of \( \Delta \) from the beginning of epoch \( e \) to bias its \( q \) and \( \text{rnd} \) in that same epoch (after \( \Delta \) nodes have already locked the value of \( \text{rnd} \) used in epoch \( e \)). Thus, to prevent the adversary from restarting the enclave to bias \( q \) and \( \text{rnd} \), it suffices to bar the enclave from issuing these two random values for any input \( e \neq 0 \) for a duration of \( \Delta \) since its instantiation. The genesis epoch, however, requires a more subtle set-up wherein participants are enforced to not restart their enclaves during that first epoch. This can be realised by involving the use of CPU’s monotonic-counter. While this process is much more involved, it is conducted only once during the system’s bootstrap.

B. VIEW-CHANGE PROBABILITY OF AHL AND AHLR

Our experiment results show that AHLR’s communication pattern inspired by Byzcoin [47] might have a reverse effect on the throughput if the network communication is not sufficiently robust. In the following, we compare the probability of view-change embodied in AHL and AHLR. We quantify these probabilities in term of message delivery, and show that the latter is more prone to view-change.

AHL communication. Under AHL communication pattern, each node sends and receives \( n-1 \) messages in each phase, except for the Pre-prepare phase where in only the leader is multicasting the pre-prepare messages. Let us denote by \( \alpha \) the probability that a pre-prepare message sent by the leader fails to deliver at a receiving node\(^2\); by \( \beta \) the probability that a prepare or commit message multicasted by a node fails to deliver at its destination. For simplicity, we assume that the delivery of one message is independent of another.

Let us denote by \( F(x, n, p) \) and \( CF(x, n, p) \) a cumulative distribution function and a complementary cumulative distribution function of a random variable that follows a binomial distribution \( B(n, p) \), respectively:

\[
F(x, n, p) = \sum_{k=0}^{x} \binom{n}{k} p^k (1-p)^{n-k}
\]

\[
CF(x, n, p) = 1 - F(x, n, p)
\]

Let \( v = \lceil \frac{n-1}{2} \rceil \). In both AHL and AHLR, the quorum size \( v+1 \).

The probability \( \mu \) that a quorum has received pre-prepare message, yet a node fails to collect a prepare certificate (i.e., a quorum of prepare messages) is:

\[
\mu = F(v, n, \alpha) \times CF(v, n, \beta)
\]

The probability \( \epsilon \) that a quorum has received prepare message, yet a node fails to collect a commit certificate (i.e., a quorum of commit messages) is:

\[
\epsilon = F(v, n, \mu) \times CF(v, n, \beta)
\]

The probability \( \gamma_{AHL} \) that a node requests for a view-change is:

\[
\gamma_{AHL} = 1 - (1-\alpha)(1-\mu)(1-\epsilon)
\]

\(^2\)We note that in Byzcoin’s experimental evaluation, the authors do not take into consideration events of view-change and their cost

\(^3\)More precisely, it is not delivered within the time-out

\[\text{Figure 14: View-change probability of AHL and AHLR w.r.t failure of message deliver.}\]

The probability that AHL enters view-change is:

\[
P_{AHL} = CF(v, n, \gamma_{AHL})
\]

AHLR communication. Unlike AHL, the leader in AHLR is tasked to collect messages and distribute a proof that there has been a quorum for a message in question, and each node only needs to communicate with the leader. Let us again denote by \( \alpha \) the probability that a message sent by the leader fails to deliver at a receiving node, and by \( \beta \) the probability that a message sent by a node fails to deliver at the leader.

The probability \( \phi \) that the leader fails to collect a quorum of responses for a message that it has sent is:

\[
\phi = F(v, n, \alpha) \times CF(n, v, \beta)
\]

The probability \( \omega \) that a node fails to receive a proof that there has been a quorum for prepare or commit message from the leader is:

\[
\omega = \phi + (1-\phi)\alpha
\]

The probability \( \gamma_{AHL} \) that a node raises a view-change request is:

\[
\gamma_{AHL} = 1 - (1-\alpha)(1-\omega)^2
\]

The probability that AHLR enters view-change is:

\[
P_{AHLR} = CF(v, n, \gamma_{AHL})
\]

\[\text{Figure 14 plot probability } P_{AHLR} \text{ and } P_{AHL} \text{ with respect to varying } \alpha \text{ and } \beta. \text{ It can be observe that } P_{AHLR} \text{ is consistently higher than } P_{AHL} \text{ for almost all } \alpha \text{ and } \beta \text{ that are lower than } 0.45. \text{ As these two values approach 0.5, both } P_{AHLR} \text{ and } P_{AHL} \text{ approach 1. The reason is that when } \alpha \text{ and } \beta \text{ approach 0.5, the expected number of messages that a node could collects in a particular consensus phase } p \text{ reduces sharply to } \lceil \frac{n-1}{2} \rceil, \text{ preventing it from forming a necessary quorum. This, in turn, forces the node to raise a view change request. Our model suggests that that AHLR communication pattern is less robust than AHL’s all-to-all communication pattern. This makes the former more prone to view changes and deteriorates its throughpout when the network is congested.}\]