# Saguaro: An Edge Computing-Enabled Hierarchical Permissioned Blockchain

Mohammad Javad Amiri[1] Ziliang Lai[2] Liana Patel[3] Boon Thau Loo[1] Eric Lo[2] Wenchao Zhou[4]
[1]*University of Pennsylvania,* [2]*Chinese University of Hong Kong,* [3]*Stanford University,* [4]*Georgetown University*
[1]{mjamiri, boonloo}@seas.upenn.edu [2]{zllai, ericlo}@cse.cuhk.edu.hk [3]lianapat@stanford.edu [4]wzhou@cs.georgetown.edu

***Abstract*—We present *Saguaro*, a permissioned blockchain system designed specifically for edge computing networks. Saguaro leverages the hierarchical structure of edge computing networks to reduce the overhead of wide-area communication by presenting several techniques. First, Saguaro proposes coordinator-based and optimistic protocols to process cross-domain transactions with low latency where the lowest common ancestor of the involved domains coordinates the protocol or detects inconsistency. Second, data are collected over hierarchy enabling higher-level domains to aggregate their sub-domain data. Finally, transactions initiated by mobile edge devices are processed without relying on high-level fog and cloud servers. Our experimental results across a wide range of workloads demonstrate the scalability of Saguaro in supporting a range of cross-domain and mobile transactions.**

***Index Terms*—Permissioned Blockchains, Edge Computing, Scalability**

## I. Introduction

Recent trends in edge computing present both new challenges and opportunities for distributed applications [2] [33]. In the edge computing paradigm, computing shifts closer to the edge of the network [30] [35] [51]. Edge devices communicate in a peer-to-peer fashion in small geographic regions known as *spatial domains*, and can communicate with edge servers, fog servers, and finally to cloud servers in a *hierarchical* fashion [50] [51] [57]. The characteristics of edge networks have led to a wide range of distributed applications being proposed [34], e.g., intelligent transportation [20], industry automation [48] and cross-border payments [28]. Many of these applications require immutability, provenance, or verifiability over wide-area networks. While point solutions exist, no general-purpose abstraction provides these capabilities in a unified manner.

Blockchain is a promising technology to realize the full potential of edge computing by providing a common substrate usable by all edge computing-enabled applications [21] [22] [58]. Increasingly, emerging uses of blockchains, in particular, *permissioned* blockchains, require transaction processing over wide-area networks among a set of mutually distrustful known entities. While distributed applications, e.g., contact tracing [44], crowdworking [9], supply chain management [5] [10] [56], and federated learning [45], benefit from the unique features of permissioned blockchains, practical deployment of edge computing-enabled blockchain applications over wide-area networks remains an elusive goal [34].

Traditional approaches for scaling distributed systems do not apply well over wide-area networks. While sharding [17] is used to partition data into multiple shards maintained by different clusters of machines, blockchain sharding, independent of the deployment scenario, backfires over a wide area due to the significant overhead of cross-shard transactions. At one end of the spectrum, flattened permissioned blockchains [7] run a consensus protocol among all nodes of involved shards to process cross-shard transactions, resulting in several messages crisscrossing high-latency low bandwidth links over the Internet. On the other end of the spectrum, coordinator-based approaches [18] do not fare much better, as the coordinator node is either close to clients or the data shards, which will not avoid slow network links when cross-shard transactions take place. Trying to avoid wide-area transactions by replicating the entire ledger on every cluster, e.g., GeoBFT [29], also merely shifts the wide-area communication from running the consensus protocol across data centers to ledger synchronization messages over a wide-area network. Moreover, current approaches do not address the mobility of nodes where a mobile edge device temporarily migrates out of its local home domain to a remote domain and initiates transactions in the remote domain.

In this paper, we present *Saguaro*, a permissioned blockchain system that leverages the hierarchical structure of edge computing infrastructures to support applications over a wide area. At a high level, in Saguaro, nodes are organized in a hierarchical structure from edge devices (*height−0*) to edge, fog, and cloud servers. Nodes at each level are further clustered into fault-tolerant *domains* where domains might follow different failure models, i.e., crash and Byzantine. In Saguaro, each height−1 domain (i.e., edge servers) maintains its own blockchain ledger, executes transactions received from their child edge devices (in parallel to other height−1 domains), constructs its ledger and propagates the ledger to higher-level domains. This hierarchical approach localizes network traffic for consensus and replication within local networks, reducing wide-area communication overhead significantly.

Saguaro leverages the hierarchical structure of edge computing networks to achieve four main purposes. First, Saguaro relies on the *lowest common ancestor* of all involved domains in the hierarchical structure (i.e., a higher-level domain with minimum total distance from the involved domains) to process cross-domain transactions in a coordinator-based fashion with low latency. Since edge servers execute transactions, the load on the internal domains, e.g., cloud servers, is highly reduced, making Saguaro suitable for edge networks.

Second, this hierarchical structure enables height−2 and

above domains to maintain only a *summarized view* (e.g., selected columns or aggregated values) of their child domain ledgers. In Saguaro, edge servers order and execute transactions and periodically, propagate the results to higher-level domains. While height$-1$ domains maintain transactions in linear ledgers, summarized ledgers at higher-level domains are structured as directed acyclic graphs to capture dependencies resulting from cross-domain transactions. These summarized views enable higher-level domains to perform aggregation functions over their sub-domains data, e.g., the total amount of exchanged assets in a micropayment application.

Third, the propagation of transactions through hierarchy enables Saguaro to optimistically process cross-domain transactions. Each involved height$-1$ domain of a cross-shard transaction orders the transaction independently without running costly cross-domain consensus protocols across height$-1$ domains, and then executes the transaction speculatively. In case of any ordering inconsistencies, the higher-level domains and eventually the lowest common ancestor of the involved domains detect the inconsistency.

Finally, the hierarchical structure enables Saguaro to efficiently support the mobility of nodes without relying on high-level fog and cloud servers. Mobile edge devices initiate transactions in different domains far from their initial local domain while Saguaro establishes *mobile* consensus by sharing a node's state only between the local and remote domains.

Saguaro makes three key technical contributions:

- Saguaro supports data aggregation over hierarchy where transactions are executed and maintained in linear ledgers of height$-1$ domains while above domains maintain only a *DAG-structured summarized view* of child domains.
- A suite of consensus protocols is provided to process transactions within and across fault-tolerant domains. Saguaro benefits from the hierarchical structure of edge networks for the geographically optimized processing of cross-domain transactions using coordinator-based and optimistic protocols.
- Saguaro supports mobility of nodes by providing a *mobile* consensus protocol where edge devices initiate transactions in different domains far from their initial local area.

## II. Background and Motivation

In an edge network, machines (i.e., devices, servers) are organized in a hierarchical structure where at the leaf level, edge devices within a local area are connected to each other and to an edge server domain (as the parent vertex). Nearby edge servers (e.g., campus area) are then connected to a fog server (e.g., metropolitan area) and finally, at the root level, cloud servers are placed [57]. The hierarchy might include multiple layers of edge, fog, or cloud servers.

We briefly describe two emerging applications: accountable ridesharing and micropayment that can realize the full potential of edge computing, and yet require technology innovations by Saguaro to make this a reality.

**Accountable ridesharing and gig economy.** In ridesharing applications, drivers give rides to travelers through platforms, e.g., Uber and Lyft. A ridesharing task usually occurs within a single domain (i.e., a local area). However, supporting the mobility of cars across domains is challenging as a driver registered in a local area might temporarily move to another domain and give rides to travelers in that area. Furthermore, a ridesharing application needs to aggregate specific data attributes from different spatial domains, e.g., the total number of tasks performed per day. The aggregated data is needed for data analysis purposes and, more importantly, for satisfying global regulations, e.g., the total work hours of a driver, who might work for multiple platforms, may not exceed $40$ hours per week to follow the *Fair Labor Standards Act* [42]. While the transparency and immutability of blockchains will aid in enforcing global regulations [8] [9], permissioned blockchain solutions today are unable to work at a global scale given that the leading ridesharing firms are all globalized.

Saguaro as a permissioned blockchain system can address the challenges above. First, in Saguaro, each height$-1$ domain (i.e., edge servers) processes ridesharing tasks initiated by edge devices within a local area. Second, within the hierarchical structure of Saguaro, while edge server domains process tasks and maintain the full record of transactions, an aggregate version of records, e.g., the travel time or working hour attribute, might be maintained by internal domains resulting in improved performance and enhanced privacy. Third, Saguaro efficiently addresses the mobility of edge devices across spatial domains. Beyond ridesharing, the ability to add accountability and verifiable global statistics collection at Internet-scale can be generally applied to any other mobile gig economy job.

**Micropayment.** Most popular micropayment infrastructures do not allow users to do cross-application payments, e.g., an Apple Pay sender cannot send money to a PayPal recipient. However, a hierarchical permissioned blockchain system can facilitate such micropayments. For micropayments within the same spatial domain and application domain (e.g., Alice pays Bob at a coffee shop, both using Apple Pay), transactions can be committed efficiently and securely within the spatial domain. For micropayments under the same spatial but different application domains (e.g., Alice pays Bob in the same coffee shop, but Alice is using Apple Pay while Bob is using PayPal), transactions can be executed efficiently if each edge server hosts ledgers from different payment companies and executes the cross-domain transactions at the edge. For micropayments that cross spatial and application domains (e.g., Alice in the West pays Bob in the East), transactions can also be executed efficiently when ledgers are deployed in the entire wide-network hierarchy, but (cross-domain) consensus is established only among the involved domains. Finally, Alice and Bob may be on the move while micropayments are happening. Saguaro aims to support such mobile micropayments as well.

## III. System Model

In a blockchain system, nodes agree on their shared states across a large network of possibly *untrusted* participants. While in a permissionless blockchain, e.g., Bitcoin [37], the network is public, and anyone can participate without a specific identity, a *permissioned* blockchain system, e.g., Hyperledger Fabric [11], consists of a set of known, identified

but possibly untrusted nodes. Saguaro is a permissioned blockchain system consisting of a distributed set of edge devices, edge servers, fog servers, and cloud servers, organized in a hierarchical tree structure. Each logical vertex of the tree, called *a domain*, consists of a number of nodes sufficient to guarantee *fault tolerance* (except for height$-0$ domains where the number of edge devices might not be known).

Nodes within each domain follow either the crash or the Byzantine failure model. In the crash failure model, nodes may fail by stopping, and may restart, whereas, in the Byzantine failure model, faulty nodes may exhibit arbitrary and potentially malicious behavior. Crash fault-tolerant (CFT) protocols, e.g., Paxos [32], guarantee safety in an asynchronous network using $2f+1$ crash-only nodes to overcome $f$ simultaneous crash failures while in Byzantine fault-tolerant (BFT) protocols, e.g., PBFT [16], $3f+1$ nodes are usually needed to guarantee safety in the presence of $f$ malicious nodes [13].

Figure 1 presents a sample 4-layer Saguaro deployment on an edge network consisting of 11 domains. For example, $D_{21}$ includes 4 nodes that follow Byzantine failure model ($3f+1$ nodes where $f=1$) while $D_{14}$ consists of 5 nodes that follow crash failure model ($2f+1$ nodes where $f=2$).

Saguaro assumes the partially synchronous communication model as it is typically used in practical fault-tolerant protocols. In the partial synchrony model, an unknown global stabilization time (GST) exists, after which all messages between correct replicas are received within some unknown bound. Saguaro further inherits the standard assumptions of existing fault-tolerant systems, including the unreliability of the network, the existence of point-to-point bi-directional communication channels to connect nodes, and a strong adversary that can coordinate malicious nodes but cannot subvert standard cryptographic assumptions. Saguaro also uses digital signatures and public-key infrastructure (PKI). We denote a message $m$ signed by node $r$ as $\langle m \rangle_{\sigma_r}$ and the digest of a message $m$ by $\Delta(m)$.

The main underlying data structure in blockchain systems is the *blockchain ledger*, an append-only replicated structure that is shared among participants. Saguaro follows the edge computing paradigm and brings computation and data closer to the network edges where height$-1$ domains execute transactions.

Saguaro targets edge computing applications where data accesses have an affinity towards locality. As a result, in Saguaro, each height$-1$ domain maintains its own ledger replicated on all nodes of the domain to provide fault tolerance. This design choice demonstrates a trade-off between performance and availability. On one hand, replicating data on a single domain leads to high performance because Saguaro does not need to deal with costly cross-domain replication protocols for every transaction. On the other hand, the availability of Saguaro is reduced in case an entire domain fails, e.g., due to natural disasters like tornadoes or earthquakes. This is in contrast to geo-replicated systems [1] [12] [17] [38] [39] where data is replicated on all domains (clusters), and the system is able to tolerate the failure of an entire domain.

Edge devices send their transaction requests to their height-1 parent domains, e.g., leaf domain $D_{02}$ is connected to a height$-1$ domain $D_{12}$. In a height-1 domain, due to data dependency among transactions of the same domain, transactions are *totally ordered* to ensure data consistency. The total order of transaction blocks in the blockchain ledger is captured by *chaining* blocks together, i.e., each block includes the cryptographic hash of the previous block.

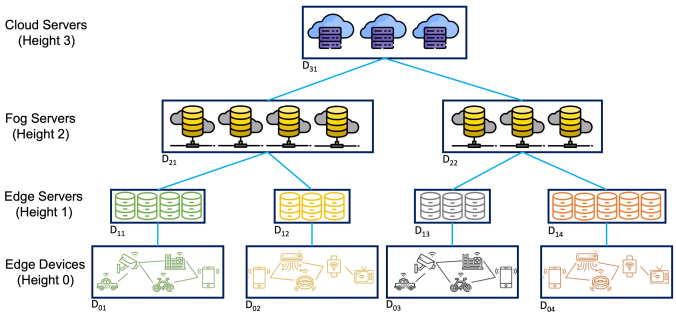


Figure 1. The Saguaro deployment on an edge commuting network

In addition to the blockchain ledger, edge servers maintain the *blockchain state*. The blockchain state is a datastore that maintains data and is being updated by executing transactions. Similar to the blockchain ledger, each domain's blockchain state is replicated on the nodes of the domain.

Saguaro uses the hierarchical structure of edge networks to provide four main functionalities. First, Saguaro processes cross-domain transactions using a coordinator-based approach by relying on the lowest common ancestor of all involved domains, resulting in lower latency (Section IV). Second, Saguaro enables data aggregation by propagating (a summarized version of) the ledgers up the hierarchy (Section V). Third, height$-1$ domains can optimistically process cross-domain transactions independent of each other and rely on higher-level nodes to detect inconsistencies (Section VI). Finally, Saguaro supports the mobility of nodes by relying on edge servers in the local and remote height$-1$ domains (Section VII).

## IV. Coordinator-based Consensus Protocol

Processing transactions requires establishing consensus on a unique order of client requests. In Saguaro, transactions are initiated by edge devices (height$-0$) and executed by edge servers in height$-1$ domains. Transactions are either internal, i.e., access records within a single domain, or cross-domain, i.e., access records across different height$-1$ domains.

The internal consensus protocol is needed among the nodes within a single domain. Edge servers within a height$-1$ domain establish consensus on every request received from edge devices (i.e., clients). The request messages are sent by edge devices to the *primary* (a pre-elected node that initiates consensus) of the corresponding height$-1$ domain. Based on the failure model of nodes, Saguaro uses a CFT protocol, e.g., Paxos [32], or a BFT protocol, e.g., PBFT [16].

Cross-domain transactions access records across different height$-1$ domains, e.g., a micropayment transaction where the sender and recipient belong to two different domains. To ensure data consistency, such transactions are appended to the ledgers of all involved domains in the same order.

The coordinator-based approach in Saguaro is inspired by the traditional coordinator-based commitment protocols in distributed databases. However, Saguaro leverages the hierarchical structure of edge networks by relying on the Lowest Common Ancestor (LCA) domain of all involved height$-1$ domains (participants) to play the coordinator role. Since the hierarchy is structured based on the geographical distance of nodes, the LCA domain has the optimal location to minimize the total distance (i.e., latency). In comparison to existing coordinator-based approaches, Saguaro deals with several new challenges.

First, in Saguaro, in contrast to distributed databases where all nodes follow the crash failure model, the coordinator and the involved domains (participants) might follow different failure models. As a result, messages from a Byzantine domain must be certified by at least $2f+1$ (out of $3f+1$) nodes of the domain (since the primary node might be malicious).

Second, in contrast to the coordinator-based approaches where a single coordinator (node or domain) sequentially orders all cross-domain transactions, in Saguaro, there are multiple independent coordinator domains in the network, i.e., any domains in height$-2$ and above could be a coordinator (an LCA domain). As a result, a participant domain in addition to its internal transactions, might be involved in several concurrent independent cross-domain transactions ordered by separate coordinator domains at the same time.

Finally, while Saguaro processes cross-domain transactions in parallel, ensuring consistency between concurrent order-dependent transactions is challenging especially when the read-set and write-set of transactions are unknown beforehand, hence, existing techniques [55] [24] [23] can not be used.

### A. Coordinator-based Cross-Domain Protocol

The normal case operation of the coordinator-based protocol is presented in Algorithm 1. Although not explicitly mentioned, every sent and received message is logged by nodes. As indicated in lines 1 to 5, $d_c$ is the coordinator domain, $\pi(d)$ represents the primary node of domain $d$, $D$ is the set of involved domains in the transaction, $\pi(D) = \{\pi(d)|d \in D\}$ is the set of primary nodes of the involved domains.

**Prepare phase.** Once the primary node of an involved domain receives a valid cross-domain transaction $m$, as shown in lines $6-7$, the primary node forwards it *directly* to all nodes of the LCA domain $d_c$ of the involved domains. Upon receiving a cross-domain transaction (lines $8-11$), the primary of the LCA domain, $\pi(d_c)$, validates the message. Since Saguaro assumes that the read-set and write-set of transactions are unknown beforehand, fine-grained locking mechanisms that lock the accessed records do not work. As a result, if the primary node $\pi(d_c)$ is currently processing another cross-domain transaction $m'$ (i.e., has not sent commit message for $m'$) where the involved domains of two requests $m$ and $m'$ intersect in at least two domains, the node does not process the new request $m$ before the earlier request $m'$ gets committed. This is needed to ensure consistency, i.e., cross-domain requests are committed in the same order on overlapping domains. Otherwise, node $\pi(d_c)$ assigns a sequence number $n_c$ to $m$ and initiates consensus on request $m$ in the coordinator domain $d_c$.

**Algorithm 1** Coordinator-based Cross-Domain Consensus

1: *init():*
2: $r$ := *node_id*
3: $d_c$ := coordinator (lowest common ancestor) domain
4: $\pi(d)$ := the primary node of domain $d$
5: $\pi(D) = \{\pi(d)|d \in D\}$
6: upon receiving request $m$ and $r \in \pi(D)$
7: forward request $m$ to $d_c$
8: upon receiving request $m$ and $r$ is $\pi(d_c)$
9: if $r$ is not processing $m'$ where $m$ and $m'$ intersect
10: establish consensus on $m$ among nodes in $d_c$
11: send signed $\langle \text{PREPARE}, n_c, \delta, m \rangle_\sigma$ to all domains $D$
12: upon receiving $\langle \text{PREPARE}, n_c, \delta, m \rangle_\sigma$ message(s) and $(r = \pi(d_i) \in \pi(D))$
13: if $r$ is not processing request $m'$ where $m$ and $m'$ intersect
14: establish consensus on the message among nodes in $d_i$
15: send signed $\langle \text{PREPARED}, n_c, n_i, \delta, r \rangle_\sigma$ to $d_c$
16: upon receiving $\langle \text{PREPARED}, n_c, n_i, \delta, r \rangle_\sigma$ from $D$ and $r == \pi(d_c)$
17: establish consensus on the order of $m$ within $d_c$
18: multicast signed $\langle \text{COMMIT}, n_i{-}n_j{-}\ldots{-}n_k, \delta, r \rangle_\sigma$ to all domains $D$
19: upon receiving $\langle \text{COMMIT}, n_i{-}n_j{-}\ldots{-}n_k, \delta, r \rangle_\sigma$ message and $r \in D$
20: append the transaction and the commit message to the ledger
21: send $\langle \text{ACK}, n_c, n_i{-}n_j{-}\ldots{-}n_k, \delta, r \rangle_{\sigma_r}$ to $\pi(d_c)$

Once consensus is established, the primary node $\pi(d_c)$ sends a signed prepare message including the sequence number $n_c$, request $m$ and its digest $\delta = \Delta(m)$ to the nodes of all involved domains. Note that if the nodes of the LCA domain follow the Byzantine failure model, a *certificate* consisting of $2f+1$ signed (commit) messages is needed.

**Prepared phase.** Upon receiving a valid prepare message, as shown in lines $12-15$, if the primary $\pi(d_i)$ of an involved domain $d_i$ is *not* processing another cross-domain transaction $m'$ where the involved domains of two requests $m$ and $m'$ intersect in at least two domains, the primary $\pi(d_i)$ assigns a sequence number $n_i$ to $m$ and initiates consensus in $d_i$ on its order. Once consensus is achieved, the primary $\pi(d_i)$ of each involved domain $d_i$ sends a signed (certified) prepared message to nodes of $d_c$ including both sequence numbers $n_c$ and $n_i$, request digest $\delta$, and node id $r = \pi(d_i)$.

**Commit phase.** When primary node $\pi(d_c)$ of the coordinator domain receives valid prepared messages from every involved domain (lines $16-18$), it establishes consensus within the coordinator domain and sends a certified commit message including a sequence number $n_i{-}n_j{-}...n_k$ (i.e., concatenation of the received sequence numbers from all involved domains) and request digest $\delta$ to every node of all involved domains. Otherwise (if some involved domain has not agreed with the transaction), the domain sends a signed abort message.

**Execution phase.** Upon receiving a valid commit message (lines $19-21$), each node considers the transaction as committed and sends an ack message to the coordinator domain. If all transactions with lower sequence numbers have been executed, the node executes the transaction. This ensures that all nodes execute transactions in the same order as required to ensure safety. Depending on the application, a reply message including the execution results might also be sent to the edge device (requester) by either the primary (if nodes are crash only) or all nodes (if nodes follow Byzantine failure) of the domain that has received the request.

Figure 2 presents four different cross-domain transactions $t_1$ to $t_4$, their involved domains and the LCA domain for

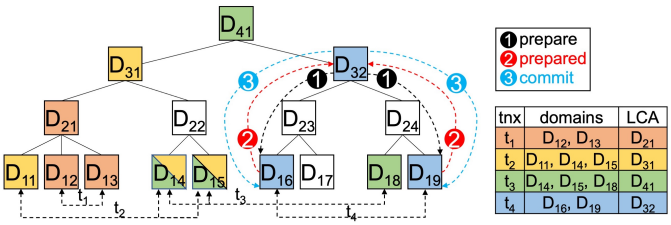



Figure 2. Coordinator-based Cross-Domain Consensus

each transaction, e.g., $D_{32}$ is the LCA domain of transaction $t_4$ between $D_{16}$ and $D_{19}$. To process transaction $t_4$, prepare, prepared, and commit messages are directly exchanged between participants ($D_{16}$ and $D_{19}$) and their LCA domain $D_{32}$ without the participation of the domains on the paths from participants to the LCA domain, e.g., $D_{23}$ and $D_{24}$.

In a situation where cross-domain transactions (1) are concurrent, (2) overlap on at least two domains, (3) are processed by different LCA domains, and (4) their prepare messages are received by overlapping domains in a different order, ensuring consistency might result in a deadlock situation. This is because domains do not process the later transaction before receiving the commit message of the earlier transaction (Algorithm 1, line 13) to ensure consistency in a coarse-grained manner. Note that if the LCA of transactions is the same, the LCA does not initiate the second transaction and deadlock will not occur (Algorithm 1, lines $9-10$).

To resolve the deadlock, once the timer of an LCA domain for its cross-domain transaction is expired, the LCA aborts the transaction and sends a new prepare message to the involved domains. Saguaro assigns different timers to different domains to prevent consecutive deadlock situations.

### B. Primary Failure Handling

If the primary of either the LCA domain or a participant domain is faulty, the primary failure handling routine of the internal consensus protocol, e.g., view change in PBFT [16], is triggered by timeouts to elect a new primary.

For cross-domain transactions, if node $r$ of an involved domain does not receive a commit message from the LCA domain for a prepared request and its timer expires, the node sends a $\langle \text{COMMIT-QUERY}, n_c, n_i, \delta, r \rangle_{\sigma_r}$ message to all nodes of the LCA domain where $n_c$ and $n_i$ are the sequence numbers assigned by the primary nodes of LCA and $d_i$ domains and $\delta$ is the digest of the request. Similarly, if node $r$ in the LCA domain has not received prepared message from an involved domain soon enough, it sends a $\langle \text{PREPARED-QUERY}, n_c, \delta, r \rangle_{\sigma_r}$ to all nodes of the involved domain.

In either case, if the message has already been processed, the nodes simply re-send the corresponding response. Nodes also log the query messages to detect denial-of-service attacks initiated by malicious nodes. If the query message is received from $n-f$ nodes of a domain, the primary will be suspected to be faulty resulting in running the failure handling routine.

Note that since in all communications between a participant and an LCA domain, the primary of the sender domain multicasts messages, e.g., request, prepare, or prepared, to all nodes of the recipient domain, if the primary of the recipient domain does not initiate consensus on the message in its domain (even after other nodes relay the message to the primary), it will eventually be suspected to be faulty.

Finally, if an edge device does not receive reply soon enough, it multicasts the request to all nodes of the domain that it has sent its request. If the request has already been processed, the nodes simply send the result back to the edge device. Otherwise, if the node is not the primary, it relays the request to the primary. If nodes do not receive prepare messages, the primary will be suspected to be faulty, i.e., it has not multicast request to the LCA domain.

### C. Correctness

We briefly analyze the safety (agreement, validity and consistency) and the liveness of the coordinator-based protocol.

*Lemma 4.1: (Agreement) If node $r$ commits request $m$ with sequence number $h$, no other non-faulty node commits request $m'$ ($m \neq m'$) with the same sequence number $h$.*

**Proof:** We assume that the internal consensus protocol of all domains ensures agreement. Let $m$ and $m'$ ($m \neq m'$) be two committed cross-domain requests with sequence numbers $h = [h_i, h_j, h_k, ...]$ and $h' = [h'_k, h'_l, h'_m, ..]$ respectively. Committing a request requires matching prepared messages from $n-f$ different nodes of *every* involved domain. Therefore, given an involved domain $d_k$ in the intersection of $m$ and $m'$, at least a quorum of $n-f$ nodes of $d_k$ have sent matching prepared messages for $m$ and at least a quorum of $n-f$ nodes of $d_k$ have sent matching prepared messages for $m'$. Since any two quorums intersect on at least one non-faulty node, $h_k \neq h'_k$, hence, $h \neq h'$.

*Lemma 4.2: (Validity) If a non-faulty node $r$ commits $m$, then $m$ must have been proposed by some node $\pi$.*

**Proof:** If nodes are crash-only, validity is ensured since crash-only nodes do not send fictitious messages. With Byzantine nodes, validity is guaranteed based on standard cryptographic assumptions which the adversary cannot subvert (as explained in Section III). Since all messages are signed (by $2f+1$ nodes) and the request or its digest is included in each message (to prevent changes and alterations to any part of the message), if request $m$ is committed by non-faulty node $r$, the same request must have been proposed earlier by some node $\pi$.

*Lemma 4.3: (Consistency) Let $D_\mu$ denote the set of involved domains (participants) for a request $\mu$. For any two committed requests $m$ and $m'$ and any two nodes $r_1$ and $r_2$ such that $r_1 \in d_i$, $r_2 \in d_j$, and $\{d_i, d_j\} \in D_m \cap D_{m'}$, if $m$ is committed before $m'$ in $r_1$, then $m$ is committed before $m'$ in $r_2$.*

**Proof:** As shown in lines $12-15$ of Algorithm 1, when node $r_1$ of a participant domain $d_i$ receives a prepare message for some cross-domain transaction $m$, if the node is involved in another uncommitted cross-domain transaction $m'$ where some other domain $d_j$ is also involved in both transactions, node $r_1$ does not send a prepared message for transaction $m$ before $m'$ gets committed. Since committing request $m$ requires a quorum of prepared messages from *every* involved domains, $m$ cannot be committed until $m'$ is committed. As a result, the order of committing messages is the same in all involved

domains. The coordinator domain $d_c$ also checks the same condition before sending prepare messages (lines $8-11$).

*Lemma 4.4: (Liveness) A request $m$ issued by a correct client eventually completes.*

**Proof:** Due to the FLP result [25], Saguaro guarantees liveness *only* during periods of synchrony. Saguaro addresses liveness in primary failure and deadlock situations. First, if the primary of a domain is faulty, e.g., does not multicast valid request, prepare, prepared, or commit messages, as explained earlier, its failure will be detected and using the primary failure handling routine of the internal consensus protocol, a new primary will be elected. Second, Saguaro addresses deadlock situations resulting from concurrent cross-domain transactions that are received by overlapping domains in different orders.

## V. Lazy Propagation of Blockchain Ledgers

Saguaro enables height$-2$ and above domains to perform data aggregation over transactions executed by edge servers in height$-1$. To this end, such domains need to maintain (a summarized version of) the ledgers of their child domains.

To send transaction blocks up the hierarchy, edge servers proceed through a succession of *rounds*. Each round ends after some predefined time interval that is identical for all height$-1$ domains. At the end of each round $r_n$, each height$-1$ domain sends a block message to its parent domain. The block message includes all transactions that are appended to the ledger in that round, and an application-dependent abstract version of the blockchain state updates in that round, i.e., $\lambda(D^{r_n} - D^{r_{n-1}})$ where $D^{r_n}$ and $D^{r_{n-1}}$ are the blockchain states at the end of rounds $r_n$ and $r_{n-1}$ and the abstraction function $\lambda$ is deterministic, predefined, and known by all nodes. For example, in a ridesharing application, it might be sufficient to send only the working hour attribute of the records that are updated to the higher-level nodes. If a domain has not received any transaction in that round, it sends an empty block message.

Depending on the failure model of the child domain, the block message is signed (certified) by either the primary (in the crash failure model) or at least $2f+1$ nodes (in the Byzantine failure model), i.e., the primary constructs a *certificate* consisting of $2f+1$ commit messages proving that consensus has been achieved on the block message within the child domain. Threshold signature can also be used to replace $2f+1$ signatures with a single threshold signature [52] [15].

Nodes in higher-level domains, on the other hand, achieve (internal) consensus on block messages that they receive from child domains. The block messages are sent by the primary node of a domain to all nodes of its parent domain. These block messages contain a collection of committed transactions in the most recent time interval. Broadcasting of block messages to all nodes in the parent domain enables nodes of the parent domain to detect malicious behavior of primary nodes.

If the primary node of the parent domain has not received the block message from a child domain after a predefined time (e.g., the primary of the child domain might be faulty), it sends a query message to all nodes of the child domain. To ensure that the completion of each round is deterministic on all nodes of a domain, the primary node puts a "cut" sign into the propose message of the last request informing other nodes of the completion of a round. Since nodes establish consensus on received messages, if a primary sends the "cut" sign maliciously, it will be easily detected.

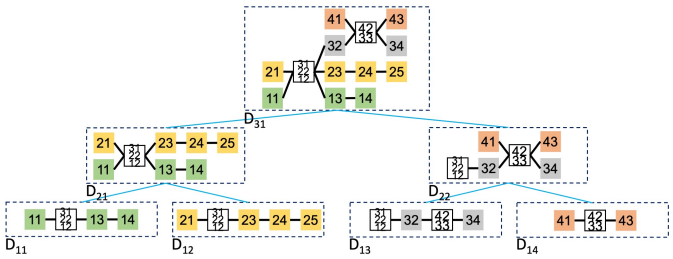



Figure 3. An Example of Saguaro Blockchain Ledger

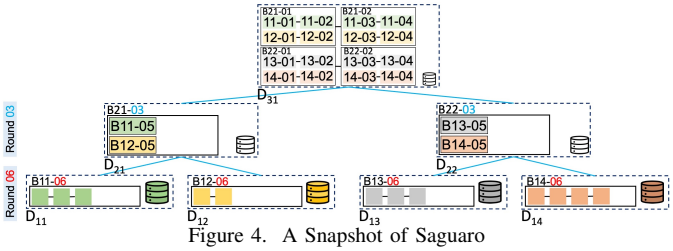



Figure 4. A Snapshot of Saguaro

Since height$-2$ and above domains might have multiple child nodes, each domain receives block messages from possibly multiple child domains and orders all transactions within received messages at each round. If there is no dependency between transactions of different child domains, any unique order of transactions is acceptable. However, cross-domain transactions, which are appended to the ledger of multiple child domains, must be appended to the ledger of the parent domain only once. Therefore, the resulting ledger is a *directed acyclic graph* to capture the order dependencies.

Similar, to height$-1$ domains, each higher-level domain sends block messages (with the same structure) to its parent domain at regular predefined time intervals in a lazy fashion. The time interval for the domains at the same level is the same, however, domains at higher levels may have larger time intervals to reduce communication overhead. Finally, the ledger of the root domain consists of all transactions that are processed in the system, and its blockchain state is a summarized view of all blockchain states in the network.

Figure 3 presents a set of transactions and shows how these transactions are appended to the blockchain ledger of different domains at different heights for the same network as Figure 1 (leaf domains are not shown because they do not maintain blockchain ledgers). The presented ledger of each domain is replicated on all nodes of that domain. In the figure, one block denotes one transaction. The sequence number of each transaction presents the order of the transaction within the ledger. While each internal transaction of a domain has a single-part sequence number, e.g., 11, 13, and 14, cross-domain transactions have multi-part sequence numbers where each part demonstrates the order of the transaction in an involved domain, e.g., $12-22-31$, is a cross-domain transaction among $D_{11}$, $D_{12}$, and $D_{13}$.

Figure 4 presents a snapshot of Saguaro for the network of Figure 1. This snapshot shows the lazy propagation of blockchain updates via the hierarchy. Each height$-1$ domain $D_x$ in its $n$-th round appends transactions to its ledger to construct block $Bx{-}n$, e.g., height$-1$ domain $D_{11}$ is in its 6th round constructing $B11{-}06$. The height$-2$ domain $D_{22}$ is in its third round constructing block $B22{-}03$. Domain $D_{22}$ has received $B13{-}05$ and $B14{-}05$ from its child domains $D_{13}$ and $D_{14}$ in this round. Finally, the root domain $D_{31}$ has appended transaction blocks $B21{-}01$ and $B21{-}02$ (received from $D_{21}$) and $B22{-}01$, and $B22{-}02$ (received from $D_{22}$) to its ledger where, for example, block $B21{-}01$ itself contains four transaction blocks $B11{-}01$, $B11{-}02$, $B12{-}01$, and $B12{-}02$. In this example, the time interval of height$-2$ domains is twice the height$-1$ domains. Note that height$-1$ domains maintain their own blockchain states while a summarized view of the blockchain state is maintained by higher-level domains.

## VI. Optimistic Consensus Protocol

Saguaro leverages the lazy propagation of ledgers presented in Section V to enable the optimistic processing of cross-domain transactions. In the optimistic protocol, each involved height$-1$ domain optimistically processes and commits a cross-domain transaction independent of other involved domains, assuming that all other involved domains also commit the transaction. Since transactions will propagate up, nodes in higher levels and eventually the LCA domain can check the commitment of the transaction.

In the optimistic approach, upon receiving a cross-domain request from an authorized edge device, the primary of the initiator height$-1$ domain multicasts the request to all nodes of the involved height$-1$ domains. The primary might behave maliciously by not sending the request to some involved domains. Hence, upon receiving the request, all nodes of the initiator domain multicast the request to the involved domains ensuring that they all received the request. Upon receiving a request, each involved domain (including the initiator domain), uses its internal consensus protocol to optimistically establish agreement on transaction order and executes it (assuming all other involved domains also execute the transaction).

For each executed cross-domain transaction $t$, nodes of a domain maintain a list of transactions (both internal and cross-domain) that are executed after $t$ and have direct or indirect data dependency to transaction $t$. If transaction $t$ gets aborted, e.g., some other involved domain does not commit the transaction, all data-dependent committed transactions need to be aborted as well. The list is deleted once transaction $t$ has eventually been committed or aborted.

Figure 5 presents the ledger of different domains using the optimistic cross-domain consensus protocol for the same network as Figure 1. In this figure, $m_b$ is a cross-domain transaction between $D_{11}$, $D_{12}$, and $D_{13}$ and $m_i$ and $m_j$ are between $D_{13}$, and $D_{14}$. Each domain maintains a list of data-dependent transactions for each cross-domain transaction, e.g., in $D_{12}$, $m_g$ has data dependency to $m_b$.

Each height$-1$ domain processes all internal and cross-domain requests and upon completion of a round, sends a

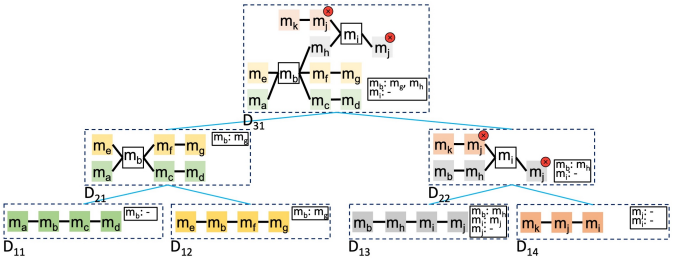



Figure 5. Example of Optimistic Cross-Domain Consensus

block message to its parent domain. In the optimistic protocol, the block message, in addition to the committed transactions (blockchain ledger) and blockchain state, consists of non-committed (aborted) cross-domain transactions (to inform other domains), and the dependency lists for cross-domain transactions (within the current and previous blocks) that have not yet been decided by all their involved domains.

Each parent domain and eventually the LCA of all involved domains in a cross-domain transaction ensures that concurrent cross-domain transactions (if any) have been appended to the ledger of the intersection domains in the same order. Otherwise, (at least) one of the transactions will be aborted. For example, in Figure 5, $m_i$ and $m_j$ are appended to the ledger of $D_{13}$ and $D_{14}$ in an inconsistent order, hence, domain $D_{22}$ aborts (only) $m_j$. Saguaro guarantees that aborting transactions is deterministic, i.e., all higher-level domains reach the same decision on choosing transactions to abort, e.g., they all abort the transaction with the lowest id. Note that intermediate domains between involved domains and the LCA domain might receive the transaction from a subset of involved domains and be able to partially check the consistency and early abort in case of inconsistency. For example, in Figure 5, domain $D_{21}$ receives $m_b$ from $D_{11}$ and $D_{12}$ (but not $D_{13}$).

Upon finding an inconsistency, the primary of the domain marks the transaction and all its data-dependent transactions as aborted, e.g., $m_j$ in Figure 5. The primary also sends a certified abort message including the request digest to the nodes of the involved domains. Involved domains need to rollback the aborted transaction and its data-dependent ones.

Each intermediate and eventually the LCA domain then checks whether the transaction is committed by the involved domains. The intermediate domains can check the commitment of the transaction by a subset of the involved domains. If the transaction is committed by all involved domains, the transaction will be appended to the ledger and upon the completion of the round sent to the parent domain. Once the primary of the LCA domain receives the transaction from all involved domains, it sends a signed commit message to all domains informing them that the transaction is committed.

If the transaction has not been appended to the ledger (block message) of an involved domain (due to the asynchronous nature of the network), the intermediate or the LCA domain does not append the transaction and waits for the next block messages. The domain also does not append the next transactions within the block message to its ledger. This is needed because there might be an inconsistency issue where

the domain needs to mark the transactions as aborted.

In the optimistic approach, the predefined time interval for completion of rounds (i.e., sending block messages to the parent domains) is smaller to detect inconsistencies in cross-domain transactions earlier. This avoids too many cascaded aborts of transactions, as any inconsistencies will result in the abort of transactions that depend on the aborted transaction.

**Correctness.** We now briefly show the safety and liveness of the optimistic approach.

*Lemma 6.1: (Agreement) If node $r$ commits request $m$ with sequence number $h$, no other correct node commits request $m'$ ($m \neq m'$) with the same sequence number $h$.*

**Proof:** We assume the internal consensus protocols, e.g., Paxos and PBFT, guarantee agreement. In the optimistic protocol, the same cross-domain transaction has different sequence numbers in different domains, however, it does not violate the agreement property, i.e., no two requests have the same sequence number in the same domain. In addition, Saguaro prevents different domains to assign the same sequence number to different requests by defining a prefix for the sequence numbers of each domain. Moreover, if the transaction is not committed in a domain, the LCA domain detects it resulting in aborting the transaction.

*Lemma 6.2: (Validity) If a correct node $r$ commits $m$, then $m$ must have been proposed by some correct node $\pi$.*

**Proof:** Validity is guaranteed in the same way as coordinator-based cross-domain consensus (lemma 4.2).

*Lemma 6.3: (Consistency) Let $P_\mu$ denote the set of involved domains for a request $\mu$. For any two committed requests $m$ and $m'$ and any two nodes $r_1$ and $r_2$ such that $r_1 \in p_i$, $r_2 \in p_j$, and $\{p_i, p_j\} \in P_m \cap P_{m'}$, if $m$ is committed before $m'$ in $r_1$, then $m$ is committed before $m'$ in $r_2$.*

**Proof:** As mentioned earlier, upon receiving a cross-domain transaction, the LCA domain first checks the consistency. Since $p_i$ and $p_j$ are involved in both $m$ and $m'$, the LCA of both $m$ and $m'$ can detect any inconsistencies in the order of transactions in both domains and resolve it by aborting either $m$ or $m'$. The aborting strategy is deterministic and results in aborting the same transaction on both LCAs, i.e., it does not matter which LCA receives the transactions first, if there is an ordering inconsistency they both either abort $m$ or abort $m'$. While transactions might be initially *optimistically* committed in an inconsistent order, eventually inconsistency will be resolved, i.e., the protocol guarantees eventual consistency.

*Property 6.4: (Termination) A request $m$ issued by a correct client eventually completes.*

The liveness of the algorithm is guaranteed in periods of synchrony based on the assumption that LCA and involved domains ensure liveness for all transactions. If the request is not committed in some predefined number of rounds by all involved domains it is considered to be aborted.

## VII. Mobile Consensus

This section addresses the next challenge of Saguaro: processing transactions initiated by mobile edge devices. When an edge device moves from its *local* to a *remote* leaf domain, reaching consensus on transactions that are initiated by the mobile device is challenging. Specifically, since edge servers of the remote height$-1$ domain do not have access to the state of the mobile node, e.g., the account balance of the node in the micropayment application, they are not able to process its requests. Moreover, any communication across domains goes through wide-area networks where bandwidth is more limited and subjected to higher latencies.

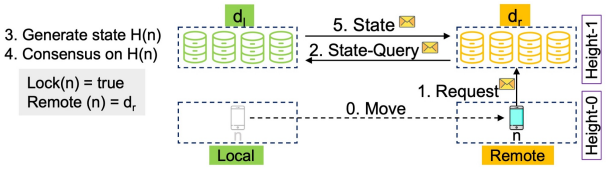


Figure 6. Mobile Consensus

In the mobile consensus protocol of Saguaro, the local height$-1$ domain shares the state of the mobile node with the remote height$-1$ domain in one round of communication to enable the remote domain executing transactions initiated by the mobile device. The state of the node includes the information that is needed to process its transactions, e.g., the account balance of the node in a micropayment application.

The normal case operation of mobile consensus is presented in algorithm 2 and shown in Figure 6 where $d_l$ and $d_r$ are the local and remote height$-1$ domains. When the primary $\pi(d_r)$ of the remote domain $d_r$ receives a valid request $m$ from an unauthorized edge device, as shown in lines 5-6, the primary $\pi(d_r)$ multicasts a signed state-query message including the request $m$ and its digest $\delta_m$ to nodes of the local domain $d_l$ to obtain the state of the node. The local domain is the domain where the node is initially registered in. The primary $\pi(d_r)$ also multicasts the state-query message to the nodes of its (remote) domain $d_r$ to inform them about request $m$.

Each domain maintains a $lock$ bit for each of its registered edge device to keep track of its mobility. When an edge device initiates a transaction in a remote domain, the $lock$ is set to FALSE, representing that the state of the edge device in the local domain is outdated. The domain also defines a variable $remote$ for each edge device to maintain the id of the remote domain that has the most recent transaction records of the node. Once the primary $\pi(d_l)$ of the local domain $d_l$ receives a valid state-query message for its edge device $n$, as shown in lines 8-9, it checks the $lock(n)$ to be TRUE (i.e., the state of node $n$ in the local domain is complete and up-to-date) and then calls GENERATESTATE function (lines 14-19). The GENERATESTATE function constructs the state of mobile node $n$ by executing a predefined application-dependent query on the blockchain. The primary $\pi(d_l)$ then runs consensus protocol among nodes of the local domain $d_l$ on the state by sending a message including both state-query message received from the remote domain $d_r$ as well as state $H(n)$. Once consensus is achieved, the primary $\pi(d_l)$ sends a signed state message including the extracted state $H(n)$, the digest $\delta_h$ of the corresponding state-query message, and the digest $\delta_m$ of request $m$ to the nodes of the remote domain. Nodes

**Algorithm 2** Mobile Consensus

1: *init():*
2: $i$ := *node_id*
3: $d_l$ := local domain
4: $d_r$ := remote domain
5: upon receiving valid request $m$ from a remote node $n$ and $i$ is $\pi(d_r)$
6: multicast $\langle$STATE-QUERY, $m$, $\delta_m\rangle_{\sigma_{\pi(d_r)}}$ to $d_l$ and $d_r$
7: upon receiving valid $\langle$STATE-QUERY, $m$, $\delta_m\rangle_{\sigma_{\pi(d_r)}}$ and $i$ is $\pi(d_l)$
8: **if** $lock(n) =$ TRUE **then**
9: $\pi(d_l)$.GENERATESATE($n, d_l, d_r$)
10: **else** ▷ $lock(n)$ *is* FALSE *and* $remote(n) = d_{r'}$
11: $\pi(d_l)$.GETSATE($n, d_l, d_{r'}$)
12: $\pi(d_l)$.GENERATESATE($n, d_l, d_r$)
13: **end if**
14: **function** GENERATESATE(node $n$, domain $d$, domain $d'$)
15: generate state $H(n)$
16: establish consensus on Sate $H(n)$ among nodes in $d$
17: $lock(n) =$ FALSE, $remote(n) = d'$
18: send $\langle$STATE, $H(n), \delta_h, \delta_m\rangle_\sigma$ to $d'$
19: **end function**
20: **function** GETSATE(node $n$, domain $d$, domain $d'$)
21: send $\langle$STATE-QUERY, $m$, $\delta_m\rangle_{\sigma_{\pi(d)}}$ to $d'$
22: $\pi(d')$.GENERATSATE($n, d', d$)
23: upon receiving valid $\langle$STATE, $H(n), \delta_h, \delta_m\rangle_\sigma$ message from $\pi(d')$
24: $lock(n) =$ TRUE
25: establish consensus on transactions of STATE message in $d$
26: append the transactions and commit message(s) to the ledger
27: **end function**

in $d_l$ also set $lock(n)$ to be FALSE and $remote(n)$ to $d_r$.

If $lock(n)$ is FALSE and $remote(n) = d_{r'}$, some other remote domain $d_{r'}$ has the most recent transaction records. As a result, as shown in the GETSTATE function, the local domain sends a state-query message to remote domain $d_{r'}$ to obtain the recent transactions that $n$ has been involved in them. Upon receiving the state of node $n$ from $d_{r'}$, the local domain $d_l$, as shown in lines 23-26, establishes consensus on the received state and updates its blockchain ledger, Finally, the local domain $d_l$ uses the GENERATESTATE function to sends the state of $n$ to the remote domain $d_r$ (line 12). This situation happens when an edge device moves to a remote domain $d_{r'}$, initiates transactions and then moves to another remote domain $d_r$. In this case, the local domain $d_l$ becomes the intermediary between remote domains $d_r$ and $d_{r'}$ by obtaining the state from $d_{r'}$, updating its state and then sending the state to $d_r$. If the mobile node returns to its local domain, the local domain updates the ledger and processes the transaction.

**Correctness.** The correctness of mobile consensus protocol is mainly ensured based on the correctness of internal consensus protocols in both local and remote height$-1$ domains. Assuming the internal consensus protocols are correct, we just need to show that communications across domains do not violate safety or liveness. Safety is guaranteed because to send a state message consensus among nodes of a domain is needed and state messages are certified by the primary of a crash-only domain or $2f + 1$ nodes of a Byzantine domain.

To provide liveness, if node $r$ of a domain has not received a state message after sending a state-query message and its timer expires, the node re-sends the state-query message to all nodes of the other domain. The nodes simply re-send the corresponding response if the message has already been processed. Nodes also log the query messages to detect denial-of-service attacks initiated by malicious nodes. If the query message is received from a majority of a domain (they already received the request, line 6), the primary will be suspected to be faulty resulting in running the failure handling routine.

If nodes of domain $d'$ receive state-query from domain $d$, however, the primary of $d'$ does not initiate consensus on state message (after nodes relay the message to the primary), nodes of $d'$ suspect that the primary is faulty. Similarly, upon receiving state messages, nodes of domain $d$ wait for the primary of $d$ to initiate consensus. Otherwise, the primary will be suspected to be faulty.

## VIII. EXPERIMENTAL EVALUATION

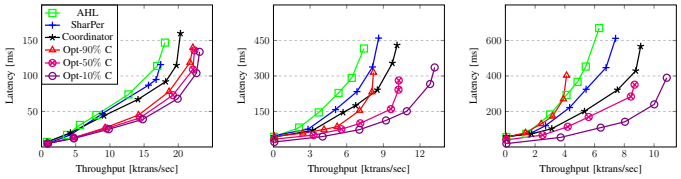
The goal of our evaluations is to measure the impact of (1) geo-distribution (i.e., nearby domains vs. far apart domains), (2) cross-domain transactions, (3) transactions initiated in a remote domain (mobile consensus), and (4) conflicting transactions (contention in the workload) in various scenarios on the performance of Saguaro.

We have implemented a prototype of Saguaro and run it on a typical four-level edge network (edge devices, edge servers, fog servers, and cloud servers) structured as a perfect binary tree (following Figure 1). Nodes follow either crash or Byzantine failure model. Each non-leaf domain (except for the last set of experiments) tolerates one failure. We use Paxos and PBFT as the internal consensus protocol for crash-only and Byzantine domains respectively.

As our experimental workload, we use a micropayment application given that it is a representative and demanding application. This application uses Saguaro, the blockchain state maintains the balance of each client (edge device), and clients continuously carry out transactions that lead to the transfer of financial assets from a sender to a recipient if all conditions are satisfied, e.g., the sender has a sufficient balance. The average measured message size (e.g., request, propose, prepare, prepared, commit, and reply) is 0.2 KB while The block messages are much larger (depending on the time interval of block propagation and the height of the tree).

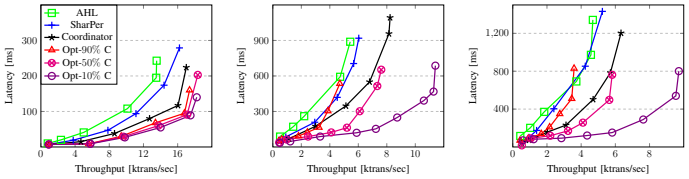
The experiments were conducted on the Amazon EC2 platform on multiple VM instances. We assigned a separate VM for each node in height$-1$ and above, e.g., four VMs are assigned to a domain with Byzantine nodes ($3f+1$). However, all nodes (clients) of a leaf domain are run on the same VM, i.e., we assigned four VMs to the four leaf domains. Each VM is a c4.2xlarge instance with 8 vCPUs and 15GB RAM, and an Intel Xeon E5-2666 v3 processor clocked at 3.50 GHz.

Saguaro follows the edge computing paradigm, processes transactions within height$-1$ domains and propagates block messages to higher-level domains. Higher-level domains receive and process block messages in parallel with transaction execution within height$-1$ domains. Given that our goal is to optimize end-user experience at the edge, we only focus on measuring the end-to-end performance of transaction execution originating with and ending at height$-1$ domains. It should be noted that since the focus of our evaluation is on transaction execution in height$-1$ domains, our evaluation




(a) 20% Cross-domain (b) 80% Cross-domain (c) 100% Cross-domain

Figure 7. Cross-Domain Transactions (Crash-only)




(a) 20% Cross-domain (b) 80% Cross-domain (c) 100% Cross-domain

Figure 8. Cross-Domain Transactions (Byzantine)

setup does not capture characteristics of edge computing networks, e.g., different bandwidth and commuting resources in different network layers. When reporting throughput measurements, we use an increasing number of requests until the end-to-end throughput is saturated.

### A. Cross-Domain Transactions

In the first set of experiments, we evaluate Saguaro in workloads with different percentages of cross-domain transactions (i.e., 0%, 20%, 80%, and 100%). Domains are distributed over four nearby AWS regions, i.e., Frankfurt (*FR*), Milan (*MI*), London (*LDN*), and Paris (*PAR*) where the average measured Round-Trip Time (RTT) between every pair of Amazon data centers is as follows; *FR* $\rightleftharpoons$ *MI*: 11 ms, *FR* $\rightleftharpoons$ *LDN*: 17 ms, *FR* $\rightleftharpoons$ *PAR*: 9 ms, *MI* $\rightleftharpoons$ *LDN*: 25 ms, *MI* $\rightleftharpoons$ *PAR*: 19 ms, and *LDN* $\rightleftharpoons$ *PAR*: 10 ms. In this scenario, each leaf and its corresponding height$-1$ domain is placed in one of the 4 data centers, while the higher-level domains are in the *FR* region.

We compare the coordinator-based and optimistic protocols of Saguaro with scalable solutions SharPer [7], and AHL [18]. SharPer and AHL are chosen because the experiments focus on studying the impact of hierarchical structure on processing cross-domain transactions. Due to the emphasis of the experiments, we only implemented the cross-shard consensus protocol of AHL where a reference committee uses 2PC to order transactions (without using trusted hardware). The internal transactions of all approaches are processed in the same way using Paxos (for crash-only domains) or PBFT (for Byzantine domains) protocol. Both SharPer and AHL are run over a network with four clusters (domains) with $f = 1$ (the same setting as height$-1$ of Saguaro). For a fair comparison, the latency in Saguaro is measured from the initiation of a transaction to when it gets committed to the blockchain of height$-1$ domain(s). Two randomly chosen domains (sender and recipient) are involved in each transaction.

In the optimistic approach, as discussed in Sec VI, a cross-domain transaction might be aborted due to inconsistency, i.e., two concurrent cross-domain transactions have been appended to the ledger of two domains in a different order, resulting in aborting all their data-dependent transactions. To measure the effects of contention on the performance of the optimistic protocol, we consider three workloads with different degrees of contention between transactions of each domain, i.e., 10% (the default value for all workloads), 50%, and 90% read-write conflicts, Figures 7 and 8 demonstrate the results with crash-only and Byzantine domains.

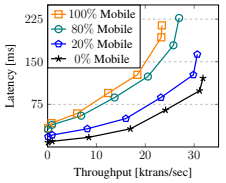
When all nodes are crash-only and all transactions are internal, Saguaro is able to process more than 31000 tps with 100 ms latency. In this scenario, each domain processes its transactions independently and the throughput of the entire system will increase linearly with the number of domains. With 20% cross-domain transactions, as shown in Figure 7(a), the optimistic approach with 10% contention shows the best performance by processing 22500 tps with 105 ms latency. This is expected because the optimistic approach does not require any communication across domains. In this scenario, only 0.17% of transactions were appended to the ledgers in an inconsistent order, hence, increasing the percentage of contention in the workload to 50% and 90% (Opt-50% C and Opt-90% C graphs) does not significantly affect the performance of the optimistic protocol. The coordinator-based approach also processes 19700 tps with 115 ms latency which is 17% more than AHL (16900 tps with the same latency).

Increasing the percentage of cross-shard transactions to 80% and 100%, as shown in Figure 7(b) and (c), results in a larger performance gap between the coordinator-based approach and the existing systems (SharPer and AHL), e.g., in the workload with 100% cross-domain transaction, the coordinator-based approach processes 63% transactions more than the AHL with the same latency. This is expected because in AHL, the single coordinator becomes overloaded by cross-domain transactions and in SharPer, consensus across domains becomes a bottleneck. However, Saguaro processes transactions efficiently by relying on multiple coordinator domains. The optimistic approach demonstrates lower performance in workloads with 50% and 90% contention due to higher inconsistencies.

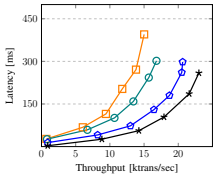
In the presence of Byzantine nodes, as shown in Fig 8, Saguaro shows similar behavior, although with lower throughput and higher latency (due to the higher cost of BFT protocols compared to CFT protocols).

### B. Transactions Initiated by Mobile Devices

In the second set of experiments, we measure the performance of the mobile consensus protocol to process remotely initiated transactions. The network is the same as Section VIII and we consider four workloads with different percentages (i.e., 0%, 20%, 80%, and 100%) of mobile nodes where a local and a remote height$-1$ domains are involved in each mobile transaction. To simulate the mobility of edge devices, we run an instance of each edge device within the VM of all leaf domains (data centers). A mobile node initiates 10 transactions within the remote domain before moving back to its local domain. The state in a micropayment application includes the balance of the mobile node. Figure 9(a) and Figure 9(b) show the results with crash-ony and Byzantine domains.

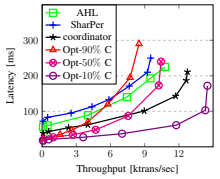



(a) Crash-only

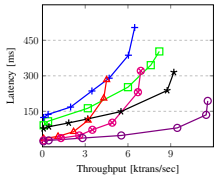



(b) Byzantine

Figure 9. Mobile Devices

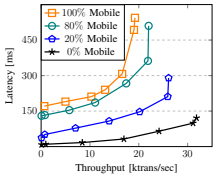



(a) Crash-only

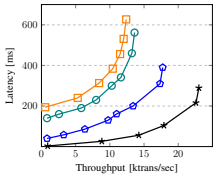



(b) Byzantine

Figure 10. Wide Area (10% cross-domain)




(a) Crash-only




(b) Byzantine

Figure 11. Wide Area (Mobile Devices)

With crash-only nodes and local transactions, Saguaro, as shown in Figure 9(a), processes 31000 tps with less than 100 ms latency (same as 0% cross-domain transaction). Adding 20% mobile transactions, Saguaro still processes 29800 transactions (only $\sim 4\%$ reduction). Similarly, with 80% and 100% mobile transactions, Saguaro processes 25700 and 23200 tps. This demonstrates the effectiveness of Saguaro in handling mobile devices: increasing the percentage of mobile devices from 0% to 100% results in only a 25% reduction in throughput. Saguaro demonstrates similar behavior with Byzantine domains (Figure 9(b)). However, since establishing consensus on state messages is more expensive with Byzantine nodes, Saguaro incurs 36% reduction in throughput by increasing the percentage of mobile devices from 0% to 100%. These results clearly demonstrate the capability of Saguaro in supporting applications that requires mobility of nodes, e.g., ridesharing.

### C. Scalability Over Wide-Area Domains

In the next experiments, the impact of long network distance on the performance of Saguaro is measured. We distribute domains over 7 far apart AWS regions all around the world, i.e., California (*CA*), Oregon (*OR*), Virginia (*VA*), Ohio (*OH*), Tokyo (*TY*), Seoul (*SU*), and Hong Kong (*HK*)[1]. In this scenario, each leaf and its corresponding height$-1$ domain is placed in one of the *TY*, *HK*, *VA*, and *OH* data-centers, the height$-2$ domains are in *SU* and *OR* and the root domain is in the *CA* region. Nodes of the same domain are placed in a single AWS region to simulate the behavior of edge networks, i.e., edge devices (servers) are within a small geographical domain. We consider workloads with 90% internal and 10% cross-domain transactions (typical settings in partitioned datastores [54]) where two randomly chosen domains are involved in each cross-domain transaction. Figures 10(a) and 10(b) depict the results for crash-only and Byzantine domains.

As shown in Figure 10(a), the optimistic protocol in the low contention workload still has the best performance (note that the workload includes only 10% cross-domain transactions). However, conflicting transactions significantly reduce the performance of the optimistic protocol in high contention workloads (Opt-50%C and Opt-90%C) compared to nearby domains (Figure 7). This is expected because when domains are far apart, resolving inconsistencies requires more time resulting in aborting more data-dependent transactions. Furthermore, the gap between the performance of the coordinator-based approach and AHL (single coordinator) has been increased, demonstrating the effectiveness of the coordinator-based approach over wide-area networks. Interestingly, AHL demonstrates better performance compared to SharPer because SharPer requires rounds of communication among nodes of domains over a wide area. In the presence of Byzantine domains, as shown in Figure 10(b), all protocols demonstrate similar behavior as the previous case.

We then use the same settings to measure the impact of network distance on mobile transactions in workloads with 0%, 20%, 80%, and 100% mobile nodes. As before, each leaf, i.e., edge devices, and its corresponding height$-1$ domain is placed in one of the *TY*, *HK*, *VA*, and *OH* data-centers. As shown in Figure 11(a), while processing mobile transactions over a wide area results in higher latency, Saguaro still demonstrate an efficient throughput: when the percentage of mobile devices increases from 0% to 100%, Saguaro incurs only a 38% reduction in its throughput (with crash-only nodes).

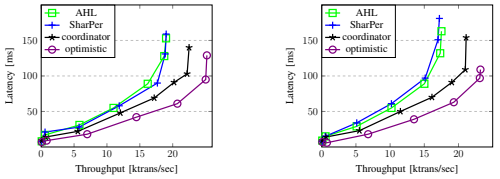
### D. Fault Tolerance Scalability

Finally, we evaluate the impact of increasing the number of nodes within each domain on the performance of protocols. Figures 12 and 13 depict the results. We consider two scenarios with $f = 2$ and $f = 4$, i.e., each crash-only domain includes 5 and 9 nodes, and each Byzantine domain includes 7 and 13 nodes respectively. All nodes are placed within an AWS region and the workload includes 90%-internal 10%-cross-domain transactions. When domains become larger, achieving consensus requires more nodes, hence, the performance of all protocols is (marginally) reduced, e.g., the throughput of the coordinator-based protocol is reduced by 6% and 11% (with the same latency) when the size increases from 3 to 5 and 9.

### E. Evaluation Summary

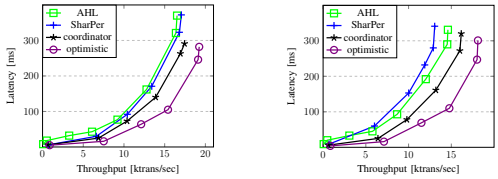
Overall, the evaluation results can be summarized as follow. First, the coordinator-based protocol outperforms SharPer and AHL, demonstrating a scalable solution that can be practically deployed over wide-area networks and used for all types of workloads. Second, in low contention workloads, the optimistic protocol processes transactions efficiently because it does not require communication across domains. However, in high contention workloads, the protocol performance is significantly reduced due to inconsistency between the ledgers of different domains, which leads to aborting all their data-dependent transactions. Third, while SharPer outperforms AHL in nearby domains, AHL demonstrates better performance in far apart domains due to its coordinator-based consensus protocol. Finally, Saguaro supports mobility over wide-area networks efficiently.

[1]The average measured Round-Trip Time (RTT) between every pair of Amazon data centers can be found at https://www.cloudping.co/grid




(a) $|p| = 5$ (b) $|p| = 9$

Figure 12. Increasing the Number Nodes (Crash-Only Domains)




(a) $|p| = 7$ (b) $|p| = 13$

Figure 13. Increasing the Number of Nodes (Byzantine Domains)

## IX. Related Work

Despite several years of intensive research, existing blockchain solutions do not adequately address the performance and scalability requirement of edge computing networks, which is characterized by cross-domain transactions and possibly mobile nodes communicating over wide-area networks.

Processing globally distributed transactions across multiple clusters (e.g., data centers) have been discussed in several studies [12] [14] [17] [19] [26] [31] [36] [43] [47] [54] [55] [60]. These systems typically shard data and replicate each data shard on multiple clusters. A coordinator-based approach, e.g., two-phase commit and two-phase locking, is then used for cross-cluster communication while a crash fault-tolerant protocol, e.g., Paxos, is used to guarantee fault tolerance within each cluster. The coordinator-based protocol of Saguaro is different from all these systems in three ways. First, in Saguaro, nodes might follow Byzantine failure model. Second, Saguaro sacrifices availability for performance by replicating data only on one (nearby) domain, and third, Saguaro leverages the hierarchical structure of edge computing networks to rely on the lowest common ancestor of all involved domains to play the coordinator role.

Processing distributed transactions across multiple clusters in the presence of Byzantine nodes has also been addressed in several studies, e.g., permissioned blockchains. Partially replicated systems replicate each data shard on a single cluster and use either coordinator-based protocols, e.g., AHL [18], or flattened protocols, e.g., SharPer [6] [7], to process cross-shard transactions. However, sharding approaches maintain data shards mainly on cloud servers with possibly large network distances from edge devices. Moreover, the far network distance either between the involved shards (in the flattened approach) or between the coordinator and involved shards (in the coordinator-based approach) results in high latency.

In geo-distributed fully replicated systems, e.g., Steward [4], Blockplane [40], and GeoBFT [29], the data is replicated on every cluster. GeoBFT proceeds in rounds where at every round, each cluster establishes consensus on a transaction and multicasts the locally-replicated transaction to other clusters. All clusters then, execute all transactions of that round in a predetermined order. Blockplane and Steward, on the other hand, present a hierarchical two-level approach where different clusters locally establish BFT consensus on disjoint transactions, and at the top level, a CFT consensus protocol is used to process all transactions globally. In contrast to geo-distributed systems, Saguaro assumes the geographical locality of data access (a reasonable assumption in edge commuting networks) and replicates data only on a nearby domain. While this design choice brings down the availability guarantee of Saguaro, it leads to higher performance. Furthermore, Saguaro leverages the hierarchical structure of edge computing networks to process cross-domain transactions more efficiently.

The blockchain model presented in [49] focuses only on the data abstraction across different levels of the hierarchy and does not address cross-domain transactions, consensus, and mobility of nodes. Plasma [46] also uses hierarchical chains to improve transaction throughput of the Ethereum blockchain, however, processing cross-domain transactions and mobility of nodes have not been addressed in Plasma.

Blockchain brings the capability of managing edge computing network data through its secure distributed ledger and provide immutability, decentralization, and transparency, all of which promise to tackle privacy and security challenges of current edge computing networks [3] [21] [27] [41] [53] [59]. Nonetheless, these studies do not address the challenges of maintaining hierarchical ledgers, processing cross-domain transactions, and consensus with mobile devices.

## X. Conclusion

In this paper, we present Saguaro, a permissioned blockchain system that leverages the hierarchical structure of edge computing networks to achieve four main purposes. First, Saguaro processes cross-domain transactions using a coordinator-based approach by relying on the lowest common ancestor of the involved domains. Second, in Saguaro domains propagate (a summarized version of) their ledgers up the hierarchy to provide data aggregation functionalities. Third, Saguaro presents an optimistic cross-domain consensus protocol by relying on higher-level nodes to detect inconsistencies. Finally, Saguaro addresses the mobility of edge devices by introducing a mobile consensus protocol. We validated these technical innovations by developing a prototype of Saguaro, where our evaluation results across a wide range of workloads demonstrate the scalability of Saguaro in processing cross-domain transactions across edge computing networks and transactions initiated by mobile devices where the involved nodes are far apart.

## Acknowledgements

This work is funded by NSF grant CNS-2104882, ONR grant N00014-18-1-2021, Hong Kong General Research Fund (14200817), Hong Kong AoE/P-404/18, Innovation and Technology Fund (ITS/310/18, ITP/047/19LP) and Centre for Perceptual and Interactive Intelligence (CPII) Limited under the Innovation and Technology Fund.

## REFERENCES


[1] D. Agrawal, A. E. Abbadi, H. A. Mahmoud, F. Nawab, and K. Salem. Managing geo-replicated data in multi-datacenters. In *International Workshop on Databases in Networked Information Systems*, pages 23–43. Springer, 2013.

[2] N. Al-Falahy and O. Y. Alani. Technologies for 5g networks: Challenges and opportunities. *IT Professional*, 19(1):12–20, 2017.

[3] M. Alaslani, F. Nawab, and B. Shihada. Blockchain in iot systems: End-to-end delay evaluation. *IEEE Internet of Things Journal*, 6(5):8332–8344, 2019.

[4] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing*, 7(1):80–93, 2008.

[5] M. J. Amiri, D. Agrawal, and A. El Abbadi. Caper: a cross-application permissioned blockchain. *Proc. of the VLDB Endowment*, 12(11):1385–1398, 2019.

[6] M. J. Amiri, D. Agrawal, and A. El Abbadi. On sharding permissioned blockchains. In *Int. Conf. on Blockchain*, pages 282–285. IEEE, 2019.

[7] M. J. Amiri, D. Agrawal, and A. El Abbadi. Sharper: Sharding permissioned blockchains over network clusters. In *SIGMOD Int. Conf. on Management of Data*, pages 76–88. ACM, 2021.

[8] M. J. Amiri, T. Allard, D. Agrawal, and A. El Abbadi. Prever: Towards private regulated verified data. In *Int. Conf. on Extending Database Technology (EDBT)*, pages 2:454–2:461, 2022.

[9] M. J. Amiri, J. Duguépéroux, T. Allard, D. Agrawal, and A. El Abbadi. Separ: Separ: Towards regulating future of work multi-platform crowdworking environments with privacy guarantees. In *Proceedings of The Web Conf. (WWW)*, pages 1891–1903, 2021.

[10] M. J. Amiri, B. Thau Loo, D. Agrawal, and A. El Abbadi. Qanaat: A scalable multi-enterprise permissioned blockchain system with confidentiality guarantees. *Proc. of the VLDB Endowment*, 15(11):1, 2022.

[11] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *European Conf. on Computer Systems (EuroSys)*, page 30. ACM, 2018.

[12] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *Conf. on Innovative Data Systems Research (CIDR)*, 2011.

[13] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[14] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, et al. Tao: Facebook's distributed data store for the social graph. In *Annual Technical Conf. (ATC)*, pages 49–60. USENIX Association, 2013.

[15] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[16] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *Symposium on Operating systems design and implementation (OSDI)*, volume 99, pages 173–186. USENIX Association, 1999.

[17] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, et al. Spanner: Google's globally distributed database. *Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[18] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards scaling blockchain systems via sharding. In *SIGMOD Int. Conf. on Management of Data*. ACM, 2019.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Operating Systems Review (OSR)*, volume 41, pages 205–220. ACM SIGOPS, 2007.

[20] S. Din, A. Paul, and A. Rehman. 5g-enabled hierarchical architecture for software-defined intelligent transportation system. *Computer Networks*, 150:81–89, 2019.

[21] A. Dorri, S. S. Kanhere, and R. Jurdak. Blockchain in internet of things: challenges and solutions. *arXiv preprint arXiv:1608.05187*, 2016.

[22] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. In *Int. Conf. on pervasive computing and communications (PerCom) workshops*, pages 618–623. IEEE, 2017.

[23] J. M. Faleiro and D. J. Abadi. Rethinking serializable multiversion concurrency control. *Proceedings of the VLDB Endowment*, 8(11):1190–1201, 2015.

[24] J. M. Faleiro, D. J. Abadi, and J. M. Hellerstein. High performance transactions via early write visibility. *Proc. of the VLDB Endowment*, 10(5):613–624, 2017.

[25] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[26] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Scalable consistency in scatter. In *Symposium on Operating Systems Principles (SOSP)*, pages 15–28. ACM, 2011.

[27] S. Guo, X. Hu, S. Guo, X. Qiu, and F. Qi. Blockchain meets edge computing: A distributed and trusted authentication system. *Transactions on Industrial Informatics*, 16(3):1972–1983, 2019.

[28] Y. Guo and C. Liang. Blockchain application and outlook in the banking industry. *Financial Innovation*, 2(1):24, 2016.

[29] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi. Resilientdb: Global scale resilient blockchain fabric. *Proceedings of the VLDB Endowment*, 13(6):868–883, 2020.

[30] M. Hu, Z. Xie, D. Wu, Y. Zhou, X. Chen, and L. Xiao. Heterogeneous edge offloading with incomplete information: A minority game approach. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2139–2154, 2020.

[31] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang, et al. H-store: a high-performance, distributed main memory transaction processing system. *Proc. of the VLDB Endowment*, 1(2):1496–1499, 2008.

[32] L. Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[33] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang. Auditing cache data integrity in the edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1210–1223, 2020.

[34] D. Loghin, S. Cai, G. Chen, T. T. A. Dinh, F. Fan, Q. Lin, J. Ng, B. C. Ooi, X. Sun, Q.-T. Ta, W. Wang, X. Xiao, Y. Yang, M. Zhang, and Z. Zhang. The disruptions of 5g on data-driven technologies and applications. *IEEE Transactions on Knowledge and Data Engineering*, 32(6):1179–1198, 2020.

[35] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[36] H. Mahmoud, F. Nawab, A. Pucher, D. Agrawal, and A. El Abbadi. Low-latency multi-datacenter databases using replicated commit. *Proc. of the VLDB Endowment*, 6(9):661–672, 2013.

[37] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[38] F. Nawab, D. Agrawal, and A. El Abbadi. Message futures: Fast commitment of transactions in multi-datacenter environments. In *CIDR*, 2013.

[39] F. Nawab, V. Arora, D. Agrawal, and A. El Abbadi. Minimizing commit latency of transactions in geo-replicated data stores. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1279–1294, 2015.

[40] F. Nawab and M. Sadoghi. Blockplane: A global-scale byzantizing middleware. In *2019 IEEE 35th Int. Conf. on Data Engineering (ICDE)*, pages 124–135. IEEE, 2019.

[41] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne. Blockchain for 5g and beyond networks: A state of the art survey. *Journal of Network and Computer Applications*, page 102693, 2020.

[42] U. D. of Labor. Wages and the fair labor standards act. https://www.dol.gov/agencies/whd/flsa.

[43] S. Patterson, A. J. Elmore, F. Nawab, D. Agrawal, and A. El Abbadi. Serializability, not serial: Concurrency control and availability in multi-datacenter datastores. *Proc. of the VLDB Endowment*, 5(11):1459–1470, 2012.

[44] Z. Peng, C. Xu, H. Wang, J. Huang, J. Xu, and X. Chu. P2b-trace: Privacy-preserving blockchain-based contact tracing to combat pandemics. In *SIGMOD Int. Conf. on Management of Data*, pages 2389–2393, 2021.

[45] Z. Peng, J. Xu, X. Chu, S. Gao, Y. Yao, R. Gu, and Y. Tang. Vfchain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Transactions on Network Science and Engineering*, 2021.

[46] J. Poon and V. Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[47] G. Prasaad, A. Cheung, and D. Suciu. Handling highly contended oltp workloads using fast dynamic partitioning. In *SIGMOD Int. Conf. on Management of Data*, pages 527–542. ACM, 2020.

[48] W. Saad, M. Bennis, and M. Chen. A vision of 6g wireless systems: Applications, trends, technologies, and open research problems. *IEEE network*, 34(3):134–142, 2019.

[49] S. Sahoo, A. M. Fajge, R. Halder, and A. Cortesi. A hierarchical and abstraction-based blockchain model. *Applied Sciences*, 9(11):2343, 2019.

[50] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim. Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. *Ieee Access*, 8:23022–23040, 2020.

[51] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[52] V. Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.

[53] A. A. Singh and F. Nawab. Wedgedb: Transaction processing for edge databases. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 482–482, 2019.

[54] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. of the VLDB Endowment*, 8(3):245–256, 2014.

[55] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD Int. Conf. on Management of Data*, pages 1–12. ACM, 2012.

[56] F. Tian. A supply chain traceability system for food safety based on haccp, blockchain & internet of things. In *Int. Conf. on service systems and service management (ICSSSM)*, pages 1–6. IEEE, 2017.

[57] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *Int. Conf. on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2016.

[58] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han. When mobile blockchain meets edge computing. *IEEE Communications Magazine*, 56(8):33–39, 2018.

[59] L. Yuan, Q. He, S. Tan, B. Li, J. Yu, F. Chen, H. Jin, and Y. Yang. Coopedge: A decentralized blockchain-based platform for cooperative edge computing. In *Proceedings of the Web Conference 2021*, pages 2245–2257, 2021.

[60] E. Zamanian, J. Shun, C. Binnig, and T. Kraska. Chiller: Contention-centric transaction execution and data partitioning for modern networks. In *SIGMOD Int. Conf. on Management of Data*, pages 511–526. ACM, 2020.