SLARM: SLA-aware, Reliable and Efficient Transaction Dissemination for Permissioned Blockchains

Paper # 264

Abstract

The blockchain paradigm has attracted diverse smart contract applications to be deployed on a blockchain consisting of a P2P network. However, no service-level agreements (SLA) has been achieved for enforcing the commit deadlines of smart contract transactions, although these applications are often interactive with clients via phones and desire stringent commit deadline (e.g., tens of seconds). Existing P2P reliable multicast protocols for enforcing stringent latency on packet dissemination are too heavyweight and incur significant traffic on existing blockchains' P2P network. Moreover, the integrity of these protocols' metadata is vulnerable on faulty P2P nodes, and their specific protocol messages are vulnerable to targeted attacks.

This paper presents SLARM, the first SLA-aware and reliable transaction dissemination system for a blockchain. SLARM leverages the strong integrity and confidentiality features of Intel SGX to develop a new message-oblivious P2P reliable multicast protocol, which defends against both the integrity and targeted attacks. Evaluation on the Ethereum blockchain system with three state-of-the-art P2P reliable multicast protocols and five diverse real-world SLA-oriented applications shows that: (1) SLARM completely eliminates targeted attacks on its SLA-enforcing messages; and (2) even with the existence of transaction spikes and attacked nodes, SLARM achieves much higher SLA satisfaction rate on transactions than the evaluated relevant protocols with a reasonable high throughput.

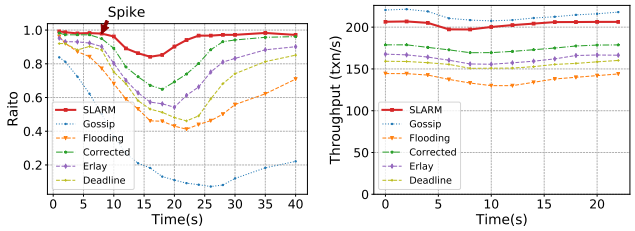
1 Introduction

The emergence of blockchains has attracted the developments of diverse Internet-wide applications (e.g., e-voting [13, 32], online auctions [36, 58], and online trading [43, 56]) on permissioned blockchain systems (e.g., Ethereum [65], Hyperledger Fabric [12], Quorum [15]), because permissioned blockchains often have high energy efficiency and throughputs. While blockchain deployments can greatly improve the reliability of these applications, these applications still naturally desire two important requirements as their past deployments in a traditional distributed database. The first requirement is the service-level agreements (SLA) on sequential execution: a client often invoke smart contract transactions updating the contracts' states [18] (e.g., trading transactions), so these transactions must be committed onto the blockchain with the same complete (gap-free) order as the order generated by the client.

The second requirement is the SLA on commit latency: many of these transactions are generated by each client on web browsers or mobile phones interactively, so the commit latency of these transactions had better not exceed a time bound (e.g., tens of seconds). We call the transactions that desire these two requirements "SLA transactions".

Unfortunately, despite much efforts on developing these applications on permissioned blockchains (e.g., Ethereum-PoA [2, 65]), fulfilling the two important SLA requirements for these applications is still especially challenging. For reliability and network bandwidth efficiency, existing blockchain systems' P2P networks, which handle the disseminations of transactions submitted by clients, often adopt a probability-based Gossip protocol [27, 42]. If a transaction updating crucial smart contract states is lost during a dissemination, the client may have to redisseminate the transaction, easily making this transaction and all the client's following transactions violate SLA. Specifically, existing blockchain systems' P2P Gossip and flooding protocols not only often miss deadlines, but also often incur gaps in per-client transaction sequences, including gaps on the sequences received by consensus nodes (confirmed in §7).

To prevent clients' retrying and redisseminating the transactions throughout the P2P network, reliable P2P multicast mechanisms [16, 30, 35, 51] make their P2P nodes exchange received transactions with their neighbors and redisseminate only the lost transactions. Erlay [51], a latest work, has integrated a P2P reliable multicast protocol [16] with Bitcoin [49] to improve both the bandwidth efficiency of Bitcoin's P2P network and privacy of clients.



(a) SLA guarantees (b) Throughputs

Figure 1: SLA guarantee for SLA transactions and throughputs for all transactions on the online trading application (details of evaluation settings are in §7). At 0s, all systems are at peak throughputs; at 8s, a spike of 200 txn/s SLA transactions lasts for 5s.

However, integrating existing P2P reliable multicast protocols [16,30,35,51] with a blockchain system still falls short in meeting the two important SLA requirements of sequential execution and commit latency for Internet-wide applications. The reason is that existing reliable multicast protocols are mainly designed for MPI [29], so they have to achieve both the gap-filling (SLA on sequential execution) and refreshing (all or certain nodes receive all latest transactions) tasks with high probability. The refreshing task is especially inefficient: each node starts from itself, uni-directionally and recursively inquiries their peers to infer the latest transactions in the network, a kind of flooding multicast [49].

Leveraging the dissemination nature of committed blocks in a blockchain, we come up with a simple, efficient P2P reliable multicast protocol: during the Gossip [42] dissemination of client transactions (the forward directional dissemination), our protocol conducts only lightweight gap-filling of transactions and guarantees that the consensus nodes of a blockchain can receive each client's SLA transactions without gaps. Then, consensus nodes can now efficiently commit the per-client gap-free transactions in a block and efficiently disseminate this committed block throughout the P2P network (the backward directional dissemination). This backward dissemination can also help the P2P nodes between a client and a consensus node to safely skip many gap-filling tasks. Overall, compared to existing reliable multicast protocols [16, 30, 35, 51], the commit latency of our simple bi-directional protocol can be greatly reduced.

This idea results in our SLARM system. However, making this new bi-directional protocol practical in the blockchain domain for SLA transactions faces two major technical challenges. First, we must create a decentralized SLA prioritization mechanism to make P2P nodes disseminate SLA-stringent transactions with high priority. Since the clocks of blockchain nodes are loosely synchronized in Internet, precisely inferring the elapsed time of a transaction's dissemination time across nodes and prioritizing the transaction's dissemination order is not an easy task. Existing P2P reliable multicast protocols [16, 30, 35, 51] simply disseminate all packets equally regardless of their elapsed dissemination time. If these protocols are used to disseminate SLA transactions, SLA-stringent transactions can be deferred by other transactions.

SLARM's core is an SLA-aware and Reliable Multicast (SLARM) protocol. To tackle this challenge, SLARM's protocol includes a new decentralized SLA prioritization mechanism, which lets each P2P node update the remaining SLA deadline of each SLA transaction conservatively according to the transaction's elapsed time during the transaction's cross-peer dissemination. Our theoretical proof (§4.3) and evaluation (§7) show that SLARM's mechanism is robust on tough network scenarios (e.g., transaction spikes).

The second challenge is security in blockchains: SLARM's SLA mechanism is run by each node, and some nodes can be faulty and corrupt the mechanism. Moreover, specific protocol messages (e.g., gap-filling) in SLARM's multicast protocol and existing P2P reliable multicast protocol [16, 30, 35, 51] are vulnerable to targeted attacks.

Fortunately, the increasingly prevalent trusted execution hardware (e.g., Intel SGX [46]) has strong confidentiality and integrity protections to make SLARM enable two important security features. The first feature is integrity: SLARM's decentralized transaction SLA dissemination mechanism is only involved in each P2P node's SGX. Second, to handle targeted attacks on specific messages (e.g., gap-filling), we present a new design with both high obliviousity and efficiency for all SLARM P2P messages (§5).

We implemented SLARM based on Ethereum [65] and evaluated it on both our cluster and AWS [8]. We compared SLARM with three state-of-the-art P2P reliable multicast protocols [30, 35, 51], and with two traditional P2P multicast protocols (Gossip [27, 42] and flooding [9]). Specifically, Erlay [51], based on bimodal multicast [16], is a latest reliable multicast protocol for blockchains. We evaluated all these protocols and SLARM with diverse real-world applications. Our evaluation and analysis show that:

1. SLARM completely eliminates targeted attacks on its SLA-enforcing messages (§5);
2. Figure 1a shows that, even with the existence of transaction spikes, SLARM achieves much higher SLA satisfaction rate for SLA transactions than all the evaluated relevant protocols;
3. Figure 1b shows that, compared to Gossip, SLARM's protocol achieves a reasonable overhead of 6.9% on the throughput of all SLA and non-SLA transactions.

The major novelty of this paper is SLARM, the first P2P reliable multicast protocol that meets the two important SLA requirements in the blockchain domain. SLARM is secure, efficient, and can support both general blockchain consensus protocols and applications. SLARM can enable people to develop even more interesting blockchain systems and applications with heterogeneous SLA requirements (§7.5). SLARM source code and evaluation results are released on github.com/osdi20p264.

The rest of the paper is as follows: §2 introduces the background. §3 gives an overview of SLARM. §4 introduces SLARM 's SLA protocol. §5 gives a security analysis, §6 presents implementation details, §7 shows our evaluation. §8 introduces related work, and §9 concludes the paper.

2 Background

2.1 Permissioned Blockchain

A blockchain runs a distributed consensus protocol among a group of nodes as consensus nodes to agree on which block can be committed in a total order onto the blockchain, where a block contains a large number of transactions submitted by many clients. A transaction is defined as committed if a block containing this transaction is committed. This paper mainly involves the Ethereum blockchain system [65]; its P2P network implementation is the most mature among open-source blockchain systems. Ethereum can be either deployed in a permissionless or a permissioned way. This paper focuses on a permissioned blockchain, which requires all nodes joining its network to go through an explicit membership registration process [12, 65], due to two main reasons. First, a permissioned blockchain often has much higher throughput (Ethereum-PoA [2] commits about 200 txns/s) and lower latency than a permissionless blockchain (e.g., Bitcoin [49] commits about 7 txn/s), so a permissioned blockchain faces a more stressful P2P network. Second, a permissionless blockchain has a cryptocurrency incentive scheme to motivate P2P nodes to disseminate a more valuable transaction (a kind of SLA scheme), where a permissioned blockchain lacks such a scheme.

It is essential for clients in a permissioned blockchain to submit transactions via P2P disseminations throughout the network in order to reach consensus nodes. In a permissioned blockchain's P2P network, all P2P nodes are granted members of the permissioned blockchain. Some P2P nodes can be elected as consensus nodes to agree on committing which block onto the blockchain. In some consensus protocols (e.g., PoS [6] and PoET [22]), any P2P node throughout the network has the right to propose a new block. Some consensus protocols (e.g., PoA [2]) periodically re-elect consensus nodes among all nodes in the P2P network, while some other protocols (e.g., Algorand [31]) make their consensus nodes be hidden from other nodes and clients. Therefore, this P2P dissemination of client transactions throughout the P2P network is essential.

The life cycle of a client transaction mainly contains two parts: (1) the dissemination of the transaction from the client to the entire network, and (2) the dissemination of a committed block containing this transaction. The second part often takes a minor portion of the time cost of the entire life cycle, because many transactions are packed in one committed block (i.e., in batches, for many clients, and gap-free) to disseminate, and each client typically connects to multiple nearby P2P nodes to fetch the committed block. As long as one of the connected nodes receives the block, the client infers the transaction as committed. The second part also often happens less frequently (due to a block's batched nature) and consumes much less network bandwidth than the first part. The first part is the major time bottleneck and improvement spot of the life cycle (§7), because many clients disseminate their transactions throughout the P2P network with the existence of ad-hoc network contention and transaction gaps, and this part is the only part that can be improved in the perspective of an individual client's perceived transaction commit latency. Our evaluation shows that the second part was often stable and took about 5.7s (only 33.5% of the entire life cycle in Ethereum with Gossip) in the PoA (Proof-of-Authority [2]) consensus protocol.

2.2 Intel SGX

Intel Software Guard eXtension (SGX) [46] is a prevalent trusted execution hardware product. SGX provides a secure execution environment called enclave. Data and code in an enclave cannot be tampered with or revealed from outside. A process running outside the enclave can invoke an SGX `ECall` to switch its execution into the enclave and to execute a statically-shielded function in the enclave; a process running in an enclave can invoke an `OCall` to switch its execution outside the enclave.

3 Overview

3.1 Failure and Threat Model

Same as typical permissioned blockchains' P2P networks, we consider SLARM's network an asynchronous, Internet-wide network. For nodes that have joined SLARM's permissioned blockchain (network), any components of these nodes running outside of SGX can be faulty. Any SLARM node with a faulty component is called a faulty node in this paper. Specifically, faulty nodes can randomly drop and delay clients' transactions as well as SLARM protocol messages. Faulty nodes can also try to alter the SLA metadata in SLARM's SLA-aware multicast protocol, and can selectively detect SLARM's gap-filling messages from all received messages and delay or drop them. All nodes have only loosely synchronized clocks, and faulty nodes can manipulate their local clocks. Due to the asynchronous nature of Internet, node failures and dropping transactions or messages cannot be distinguished by other nodes, so SLARM treats a failed node as faulty. All clients are trusted. A client's transactions are signed by the client itself and sent to connected peers (§2.1), so faulty nodes cannot modify the transactions or their SLA requirements.

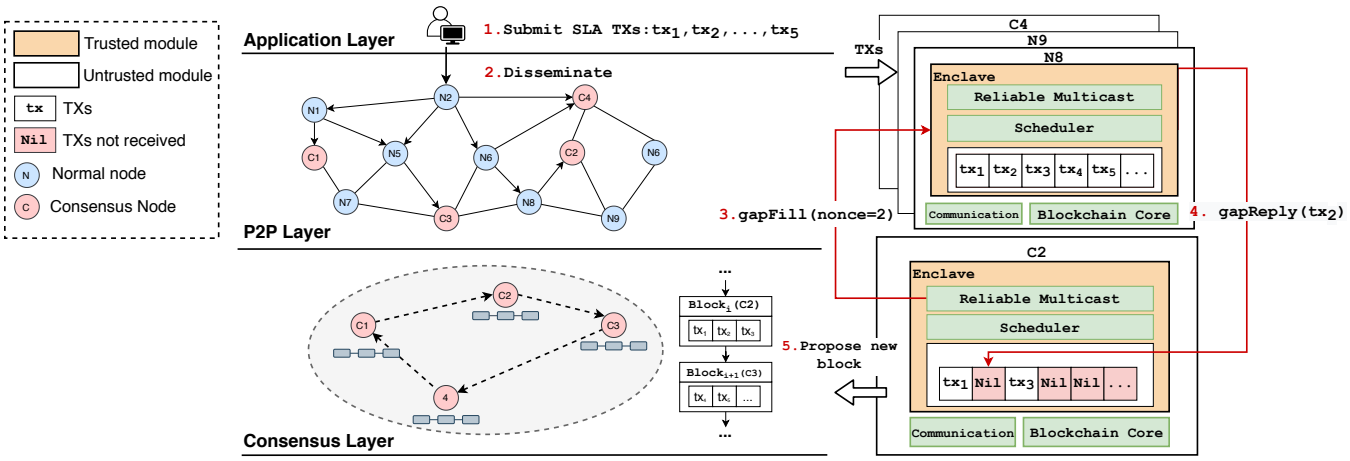


Figure 2: SLARM's architecture. All SLARM components are in green.

To prevent faulty nodes propagating an incorrect SLA deadline metadata to its peers (e.g., assigning a stringent deadline to an SLA transaction with a loose deadline), SLARM's runtime system updates only the metadata within each node's SGX enclave launched by SLARM. Outside the SLARM's enclaves, all client transactions (including both SLA and non-SLA transactions) are encrypted to make all transactions oblivious. In each node's enclave, all transactions are decrypted (§4.1). In SLARM, all the firmware and hardware components of SGX are trusted. Hiding traffic endpoints (Karaoke [40] and Stadium [60]), SGX micro-architecture side-channels [64], and SGX Iago attacks [21] are out of the scope of this paper.

3.2 SLA in SLARM

SLA is an agreement [37, 52, 59] between a customer and a service provider on the service quality (e.g., performance and reliability). Our SLARM system is the service provider, and an application and its clients are the customers. A smart contract [5, 19, 26, 65] is a stateful blockchain program that can be invoked via client transactions submitted to the blockchain. Smart contract applications often desire two SLA requirements. First, a client's transactions that invoke smart contracts must be sequentially executed (gap-free) according to the order submitted by the client. The second requirement is SLA on commit latency (§1).

In SLARM, the SLA of a transaction is a 2-tuple $SLA{:}\{Deadline, Order\}$, where *Deadline* is the desired commit latency of the transaction, and *Order* is whether the transaction need be executed in order. Suppose an uncongested network, and suppose the median commit latency of a transaction is T (e.g., 16.8s in the Ethereum-PoA system with Gossip in Table 3). Given an application (e.g., online trading in Table 2), SLARM by default considers all smart contract transactions as SLA transactions, and by default sets the two tuples as $(c \times T, Yes)$ for all the SLA transactions, where c is a configurable constant among clients. Because all clients and P2P nodes have loosely synchronized clock in SLARM, and faulty nodes can manipulate their local clock, SLARM uses latency duration instead of absolute clock time.

In SLARM, the SLA satisfaction rate p is the portion of SLA transactions meeting their SLA deadlines. Suppose in every second there are n SLA transactions with deadline $c \times T$ being submitted to the blockchain network. If n does not exceed the maximum throughput of the consensus protocol deployed in SLARM and the network capacity of SLARM's P2P network, then, SLARM guarantees that all SLA transactions can meet their SLAs with high probability p (96.0% when $c = 2$, proved in §4.3). Note that, given the same network conditions, all typical blockchains' Gossip [27, 42], flooding [7], and existing reliable multicast protocols [16, 30, 35, 51] enforce much lower SLA satisfaction rate than SLARM (Figure 1a) when they are integrated into blockchains (e.g., Erlay [51]).

3.3 SLARM Architecture

Figure 2 shows SLARM's architecture, SLARM's components are in green. Each SLARM node, including both normal nodes and consensus nodes (§2.1), has two trusted modules and two untrusted modules. The scheduler and reliable multicast module running in an SGX enclave (orange color) are trusted. The scheduler module prioritizes client transactions sent from the node's peers according to the transactions' SLAs. The multicast module conducts a gap-filling task when it finds a gap from its received per-client SLA transaction sequence; for non-SLA transactions, gap-filling is unnecessary. These two trusted modules are 1.7k LOC (§6), and the enclave memory stores only uncommitted transactions. If this node is a consensus node, transactions are forwarded to the blockchain core module.

On each SLARM node, the blockchain core module (including the consensus protocol) and network communication module (including TCP/UDP) are outside SLARM's enclave and are not trusted. The blockchain core module performs consensus on committing which block, maintains a local copy of the blockchain, and executes committed transactions extracted from committed blocks.

Figure 2 also shows the life cycle of an SLA transaction in SLARM: (1) A client submits a series of SLA transactions to a nearby blockchain node N2. (2) After N2's scheduler module receives these transactions, it prioritizes the propagation of them according to their SLAs (§4.1) and disseminates them. (3) If a node C2 detects a gap (tx_2) in received transactions, C2 sends a `gapFill` message to ask for tx_2 from a random subset of peers (§4.2). (4) If a peer in the subset has the transaction, the peer replies C2 with a `gapReply` message. In SLARM, non-SLA transactions do not involve gap-filling. (5) The consensus nodes pack transactions into blocks and agree on which block to commit next.

An open security challenge is that a P2P reliable multicast protocol for blockchains must tackle targeted attacks on protocol messages: recent work [63] shows that attackers can selectively defer certain types of P2P protocol messages during a client transaction's dissemination, which can trigger the default peer adjustment mechanism in a P2P network and maliciously make certain victim nodes adjust most of their peers to faulty nodes. Such an Eclipse attack can further arbitrarily delay the dissemination of transactions of these victim nodes and violate the transactions' SLA in SLARM.

Even if such Eclipse attacks do not exist, because the protocol behavior and sizes of `gapFill` messages and normal `disseminate` messages are explicitly distinguishable in existing P2P reliable multicast protocols (see Bimodal [16] in Figure 4), attackers can easily stop the gap-filling tasks by dropping or deferring their messages and stop the entire commit progress of many clients' SLA transactions. Existing reliable multicast protocols [16, 30, 35, 51], including Erlay [51], a latest notable P2P reliable multicast protocol tailored for blockchain security, are especially vulnerable to such attacks. §4 and §5 will present a complete of attacks SLARM aims to tackle, including such targeted attacks.

4 The SLARM Basic Protocol

On a SLARM node N, when disseminating an SLA transaction, SLARM's SLA dissemination mechanism subtracts the deadline of this transaction with two trustworthy (conservative) time variables: $N.RTT$, the recent worst-case Round Trip Time cost node N's peers sending a transaction to N; tx_N^{wait}, the time cost an SLA transaction tx spent on node N's scheduler queue. A transaction with a negative remaining deadline will still be disseminated in SLARM; the resultant remaining deadline will be reported to the transaction's client. §5 will present how SLARM makes these two variables *conservative*: SLARM can avoid faulty nodes' malicious behaviors of making the subtracted elapsed time from a transaction's remaining SLA deadline smaller than the actual elapsed time of the transaction's dissemination path, including selective deferring attacks and manipulating local clocks. To ease discussion, this section assumes these two variables are conservative first.

4.1 Prioritizing SLA Transactions

We first present SLARM's SLA prioritization mechanism in a high level. Figure 3 shows that, when a SLARM P2P node N receives a transaction tx sent from a client, N submits the transaction to its local SLARM enclave by an `ECall`. The scheduler module running in N's SLARM enclave (§3.3) verifies tx's signature and retrieves tx's SLA requirements $c \times T, Yes$ (if any). The scheduler then generates an SLA metadata m, which indicates the transaction's initial remaining SLA deadline ($c \times T$), appends m to tx, and inserts tx onto SLARM's local priority queue according to the deadline m (a smaller deadline means higher priority).

If this transaction tx is sent from N's peer, the same `ECall` is invoked to push the transaction to N's SLARM enclave. N's peer must have encrypted tx, so N's enclave decrypts tx, subtracts $N.RTT$ from the transaction's remaining deadline m, and inserts tx to the local priority queue. SLARM's own `gapFill` messages are also inserted to the queue's head. Non-SLA transactions are appended to the queue's tail. To achieve high obliviousness against attacks outside the enclave, in SLARM, the communication module does a batch fetch of transactions/messages from the queue's head using an `ECall`, subtracts each SLA transaction's m according to its wait time on the queue, encrypts each transaction/message in the batch, exits from the `ECall` with the batch, and sends the batch.

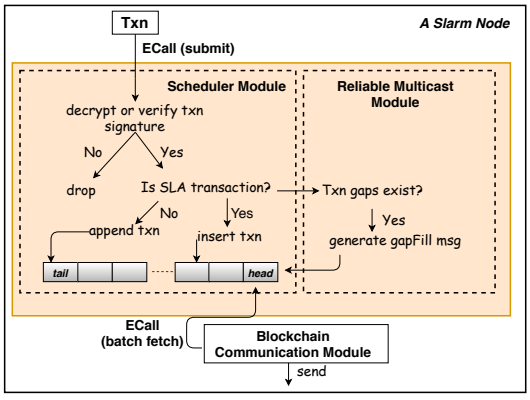


Figure 3: SLARM's enclave code invocations (via `ECall`) and outbound messages (via `ECall`). Enclave is in orange.

In a detailed level, for each scheduler module of a SLARM node, this module runs within an SGX enclave (Figure 3) to

prioritize SLA transactions over non-SLA transactions. This module is essential because typical applications often have a large portion of non-SLA transactions (Table 2). Without this module, non-SLA transactions can easily block SLA transactions and violate SLA (§7). Moreover, transactions with the same SLA requirement can have different remaining SLA deadlines in SLARM.

In the perspective of a client, the end-to-end commit latency of a transaction contains two parts: the client transaction dissemination latency τ_d (from a client to consensus nodes) and the consensus latency τ_c (including the time cost of agreeing which block to commit and that of the committed block reaching a connected peer of the client, see §2.1). According to our evaluation (§7.1), tau_d takes up about 52.3% - 86.1% of the end-to-end commit latency and varies significantly with the transaction workload. The consensus latency (τ_c) is often stable (§2.1), and the block commit and dissemination events are infrequent (e.g., happens once in the network every 5s in PoA [2]). Therefore, τ_c does not involve SLARM's scheduler module, and SLARM just subtracts τ_c from a transaction's SLA deadline once when a transaction is sent from its client to a connected P2P node. SLARM focuses on making τ_d meet the remaining SLA deadline.

To meet SLA transactions' deadlines, on each SLARM node, SLARM's scheduler module gives SLA-stringent transactions a higher priority. If a transaction is an SLA transaction, the scheduler module subtracts the remaining deadline with the actual time spent in transferring tx from N's peer (suppose it is N_{prev}) to N. One native approach is to record the local clock of N_{prev} using the code running in N_{prev}'s enclave launched by SLARM, and to subtract N's local time with this recorded time in N's SLARM enclave. This approach is problematic because if N or N_{prev} is faulty, either of the local times can be manipulated by the faulty node(s), making the time subtracted from τ_d much less than the actual time spent in the transmission (i.e., τ_d is no longer conservative).

We present a trustworthy mechanism to record N's peers' round-trip time cost conservatively as $N.RTT$, and details are in §5.3. The scheduler module subtracts each received SLA transaction's τ_d with $N.RTT$ and inserts the transaction into local priority queue according to τ_d (or m). N's next-hop node (suppose it is N_{next}) also uses its own $N_{next}.RTT$ to update the transaction's remaining τ_d conservatively.

4.2 Reliable Multicast

One key SLA requirement in SLARM is the sequential execution of per-client SLA transactions (i.e., gap-free), but the default P2P multicast protocols (typically Gossip and flooding) in existing blockchain systems are not designed to achieve the gap-filling task. Specifically, the flooding protocol [9] forwards all newly received transactions on each node to all the node's neighbors, an extremely bandwidth consuming process even for a low throughput blockchain [49]. Worse, flooding has no guarantee of filling gaps for smart contract transactions. In Gossip, when a node receives a transaction for the first time, rather than sending the transaction to all neighbors (flooding), it randomly selects a subset of neighbors, and forwards the transaction to them [41]. However, Gossip only provides a weak guarantee on gap-filling (e.g., in Figure 1a, Gossip failed to meet 91% of SLA transactions within the same SLA deadline as SLARM's).

In order to provide a high probability on the gap-filling task in a P2P network, three reliable multicast protocols are presented [16, 35, 51]. They typically follow a two-phase, disseminate-then-correct pattern, as shown in Figure 4 . The first phase is an unreliable, default Gossip that makes best-effort attempts to efficiently disseminate transactions. The second phase is an anti-entropy protocol, where each node contacts its peers to exchange both the lost and latest transactions. Erlay [51] is based on the bimodal reliable multicast [16] and integrated with Bitcoin [49]. Compared with Bitcoin's flooding multicast, Erlay reduces network consumption by 41%. In our evaluation, reliable multicast has a better SLA rate than flooding and Gossip (Figure 1a).

However, all these existing reliable multicast protocols are conceptually a uni-directional protocol: all nodes handle all the tasks of dissemination, gap-filling, and refreshing the latest transactions. Therefore, each node starts from itself, and uni-directionally and recursively inquiries their peers to infer the latest transactions throughout the network (a kind of flooding [49]). After all, all existing reliable multicast protocols [16,30,35] except for Erlay are originally designed for MPI.

Our idea is that we can leverage the refreshing nature of a blockchain's consensus protocol to design a new bi-directional P2P reliable multicast protocol for SLARM, as shown in Figure 4. This protocol is simple, efficient, yet reliable on gap-filling. Specifically, on the forward direction (Gossip [42]), SLARM lets the P2P reliable multicast protocol conducts only gap-filling of lost SLA transactions among peers and to efficiently disseminate the complete sequences of per-client transactions throughout the network.

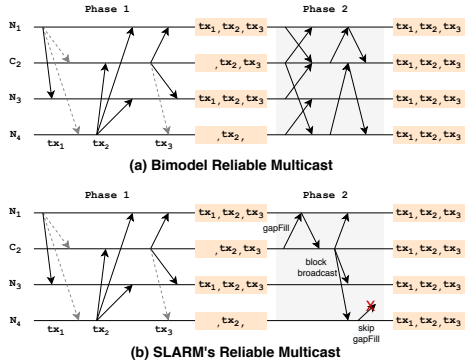


Figure 4: Bimodal [16] & SLARM's bi-directional multicast.

On the backward, the consensus nodes of SLARM's blockchain disseminate committed blocks throughout the network; P2P nodes receiving the committed blocks safely skip the disseminations of the committed transactions from SLARM's scheduler, multicast, and communication modules. SLARM also safely skips the gap-filling task of lost transactions with smaller sequence number than the committed transactions.

4.3 SLA Enforcement Probability Analysis

Now we analyze SLARM's probability in enforcing deadlines of SLA transactions. Recall SLARM's SLA variables (§3.2): $\tau_d + \tau_c$ and n. $\tau_d + \tau_c$ is the median commit latency (life cycle) of an SLA transaction in an uncongested P2P network. n is the total number of SLA transactions generated from all clients in every second. Suppose n is smaller than the consensus protocol's throughput, and we investigate the probability p: the percentage of these SLA transactions that can meet their SLAs ($c \times T$), where c is 2 by default.

SLARM's reliable multicast protocol is based on Gossip [42], where transactions are propagated to the entire network in rounds. During the propagation rounds of a transaction, each independent node randomly selects a subset of its peers and forwards the transaction to the peers. This random transmission process on each node is a Poisson process [48, 54]. Based on the fact that the superposition of independent Poisson processes is also a Poisson process [1], an SLA transaction's dissemination latency τ_d to the entire network satisfies the Poisson distribution.

The end-to-end commit latency of a transaction is $\tau_d + \tau_c$. Since we have an uncongested network, the consensus latency τ_c is small (29.0%) and relatively stable (constant), confirmed in our evaluation. Therefore, $\tau_d + \tau_c$ also satisfies Poisson distribution. Suppose the median commit latency of a transaction is T (§3.3), according to Poisson Distribution's probability function $p(x,T) = \frac{e^{-T}T^x}{x!}$, if we set the SLA transaction's deadline as $2 \times T$, 96.0% of these transactions can meet their SLA. This high theoretical rate matches SLARM's actual SLA satisfaction rate in our evaluation.

5 SLARM's Security Design and Analysis

This section presents our security guarantee: SLARM can maintain a high SLA satisfaction rate in §4.3 in the face of attacks (e.g., selective targeted attacks mentioned in §3.3). As SLARM's SLA prioritization mechanism runs in each node's SGX, an attacker can conduct attacks only on messages exchanged between different nodes' enclaves. Specifically, an attacker can corrupt, reorder, drop, and delay messages. Since SLARM's protocol does not rely on message orders to enforce SLA, we do not need to maintain message orders. Since message drop and delay cannot be distinguished in Internet, we call them deferring attacks and handle them together. The drop and delay attacks can be either selective or random. §5.1 shows how SLARM defends against message corruption attacks. §5.2 presents a message oblivious mechanism to handle selective deferring attacks. §5.3 presents a trustworthy (conservative) RTT mechanism to handle random deferring attacks.

5.1 Avoiding Message Corruption Attacks

The metadata in SLA transactions is sensitive. If faulty nodes maliciously access and modify the metadata, they can easily break SLARM's SLA guarantee. For example, faulty nodes can change the *deadline* of an SLA transaction from *5s* to *50s*; then the SLA transaction will not be prioritized even if it is stringent to disseminate, leading to SLA violation. To prevent such attacks, SLARM uses SGX and protects SLA transactions' metadata updates on all SLARM nodes (§4.1). Specifically, we put all the uncommitted transactions in enclaves and pack the scheduler logic as `ECalls` to read incoming transactions, and decide the transactions to be disseminated. In this manner, even if a faulty node runs our scheduler module and tries to poison the metadata, she cannot access the metadata and control the disseminated messages because of the shield of SGX.

5.2 Handling Selective Deferring Attacks

In SLARM, faulty nodes can selectively defer P2P messages and easily violate the SLA of many clients' SLA transactions (e.g., the selective targeted attacks mentioned in §3.3). If SLARM's protocol messages in the network have different sizes (e.g., if `gapFill` is not larger than 10 bytes while a transaction dissemination message size can reach 100 bytes), this exposes a large attack interface to attackers (§3.3). Thus, we design to make all P2P messages in SLARM oblivious by filling in extra dummy payload and encrypting messages with the same symmetric key. In this way, all messages in the SLARM network are oblivious (100 bytes), and a faulty node cannot distinguish whether a message is a dissemination message (either an SLA transaction or non-SLA transaction), a conservative RTT ping-pong message (§5.3) or a `gapFill` (§4.2).

5.3 Capturing Random Deferring Attacks

However, even if SLARM's P2P messages are oblivious, it is still challenging to meet SLA requirements of transactions in an asynchronous network. Even if messages are oblivious, faulty nodes might randomly and arbitrarily drop/delay P2P messages. For instance, faulty nodes can defer the dissemination of transactions on some faulty nodes close to consensus nodes. If a naive SLA remaining deadline updating mechanism uses the differences of local system clocks of

these faulty nodes to update the deadline of an SLA transaction (even if the updating code runs in SLARM's enclave), these faulty nodes can greatly defer the transaction's one-hop transmission and manipulate the nodes' local clocks to fool the transaction that it is far from being stringent (e.g., the transaction's one-hop elapse time measured by the clocks between two faulty nodes is made negligible). Then, other transactions will have higher priorities to reach consensus nodes and make this fooled transaction be committed in several later blocks. This is a huge SLA violation, but the fooled transaction's SLA remaining deadline does not capture this attack (the deadline is no longer conservative).

To guarantee trustworthy (conservative) time value in SLARM's scheduler module and to reveal randomly deferred attacks, SLARM achieves a trustworthy per-node RTT value with a Trustworthy RTT maintaining protocol. Since the Internet latency among two peers is asymmetric due to IP routing, SLARM uses a complete RTT value conservatively instead of its half. This conservative choice is reasonable in SLARM, as the deadline $c \times T$ often comes with $c \leq 2$ (§3.3). Within regular time interval (10s, same as Ethereum's ping interval), the SLARM enclave on each node N (including a faulty node) encapsulates a `ping` message with the same format as normal transaction dissemination (around 100 bytes) and sends that `ping` message with its latest RTT value to all its peers. At the time receiving a *ping* request, a peer node encapsulates a reply message within SLARM's local enclave and sends the reply message back to the inquiry node. The node N collects all the reply messages, all `ping` requests' RTT values from all its peers, and selects the *highest* calculation result as the latest RTT value.

For each node, this mechanism is invoked frequently enough to capture random packet deferring attacks and network congestions, as each node has 50 peers in Ethereum, and all its peers also invoke this trustworthy RTT mechanism to exchange their worst-case RTT values at random moments in every 10s ping-interval. This mechanism does not increase Ethereum's message complexity either; it is just modified from Ethereum's default ping-pong mechanism.

This protocol provides trustworthy (conservative) RTT values against the random deferring attacks. Since all faulty nodes need to forward transactions and update transactions' SLA metadata to avoid being kicked out from the network, faulty nodes have to keep sending messages, including the RTT messages. Also, a faulty node cannot manipulate an RTT reply message because all messages are decrypted within SGX. Besides, our trustworthy RTT protocol can tolerate random deferring attacks and still make SLARM's SLA update mechanism conservative. This is because we select the *highest* RTT value, so an inquiry node will get a statistical worst case of the actual network one-hop delay, including the random delay conducted by faulty nodes.

Because faulty nodes cannot distinguish RTT messages and transaction dissemination messages, so the probability of their random deferring attacks deferring only node N's transaction dissemination messages without deferring any RTT around N has an almost zero probability. Therefore, SLARM's trustworthy RTT mechanism can capture such random attacks with a high probability and make SLA transactions more stringent (some transactions' SLA deadlines can be made negative, but still stringent and conservative), as the deadline for each SLA-transaction subtracts a statistically worst-case value of RTT.

SLARM nodes also need to conservatively estimate an SLA transaction tx's trusted local elapsed time on node N, denoted as tx_N^{wait}, even if the local system clock is manipulated. This is trivial to achieve with SGX, as tx_N^{wait} is accumulated by a process in SLARM's enclave via counting CPU cycles, and the process never goes out of the enclave [24].

Overall, two critical variables used by the scheduler (§4), $N.RTT$ and local elapsed time tx_N^{wait}, are both *conservative*: if SLARM nodes infer that an SLA transaction meets the SLA deadline, the transaction actually meets it with high probability, even though the transaction's dissemination has been deferred by some faulty nodes on some hops. Of course, if a client finds its committed SLA transactions with negative SLA remaining deadlines, these transactions may have actually met their SLA deadlines, because SLARM subtracts these two conservative variables from the deadlines.

6 Implementation

We implemented SLARM based on the latest Golang version of Ethereum [65] - a fully tested and actively maintained blockchain system on the Internet. We leveraged Ethereum's P2P library to build SLARM's reliable multicast component and rewrote Ethereum's transaction logic for admitting, verifying, and scheduling SLA transactions. We carefully selected sensitive functions and put these functions into SGX enclaves. Since SGX only provides C/C++ SDKs, we rewrote all sensitive functions in C and used cgo to invoke `ECalls`. Totally, we modified 2037 lines of Golang code, and implemented the scheduler and reliable multicast component for 1705 lines of C code. For encryption/decryption, we used AES-256, a symmetric key library provided by the SGX SDK. SLARM uses Ethereum's bootstrap nodes for doing SGX attestation [46] for all member nodes' SLARM enclaves. The bootstrap nodes store a list of attested nodes and provide it to each attested node for peer discovery.

7 Evaluation

We evaluated SLARM's performance on both our own cluster and the AWS cloud [61], with other evaluation parameters shown in Table 1. In our cluster, each machine is equipped with 2.60GHZ Intel E3-1280 V6 CPU with SGX, 40Gbps NIC, 64GB memory, and 1TB SSD. On the AWS

Config	Cluster	AWS Cloud
# Nodes per-machine/Total	20/500	100/5,000
Default consensus Protocol	Clique-PoA	Clique-PoA
Block commit interval	5s	5s
Avg RTT	20ms	200ms
SLA	{16s, Yes}	{32s, Yes}
pingInterval	10s	10s
Workload	Online trading, 200 txn/s	Online trading, 200 txn/s
Bandwidth Limitation	20Mbps	30Mbps

Table 1: Default evaluation settings (unless specified).

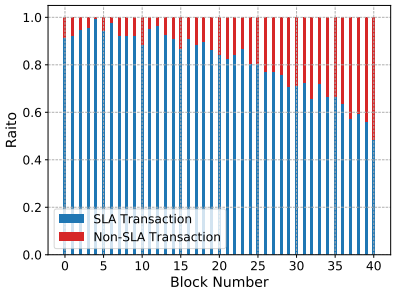


Figure 5: Distribution of SLARM's SLA and non-SLA transactions in 40 committed blocks, starting at 8s in Figure 1a. Since the block commit latency is 5s, non-SLA transactions have deferred commit latencies in SLARM.

cloud, we launched 50 m5d.24xlarge instances with 96 vCPUs, 4x900(SSD) and up to 25 Gigabit network bandwidth. We ran 100 SLARM nodes on each VM instance (5k nodes in total), with each SLARM node running in a docker container. To collect and analyze all SLARM member nodes' performance data, we also launched one t2.xlarge instance with 4 vCPUs and moderate network bandwidth. All AWS instances are run in the same zone (Ohio). We ran SGX in cluster with actual SGX hardware and in AWS with Intel's SGX simulator, because AWS does not provide SGX hardware. Table 5 show that SLARM's performance in simulation mode is roughly the same as hardware mode because SLARM's performance is bound to network latency.

We compared SLARM's performance with five P2P messaging protocols for blockchain systems, including three reliable multicast protocols (Erlay [51], Corrected protocol [35], and Deadline protocol [30]), the basic Gossip protocol [42] and the flooding protocol [9]. Among these baseline protocols, we evaluated the flooding protocol by running origin Ethereum [65] (version eth/64). Gossip is a popular P2P messaging protocol that broadcasts transactions to random peers in rounds (§4.2). Among these reliable multicast protocols, Erlay is the only reliable multicast protocol that developed on a blockchain system (i.e., Bitcoin [49]). Since Erlay is not open-source, we implemented Erlay on our own system; Deadline protocol is implemented upon Gossip; Corrected protocol is also a recent reliable multicast protocol.

We ran five applications with different portions of SLA transactions (Table 2). We also generated traffic spikes in the network. Our default benchmark workload is *Online trading* because it has interactive transactions and is prevalent on the Internet. We set the transaction size as 100 bytes. The transaction size for baseline systems are either equal to or smaller than that of SLARM.

For SLARM, we define SLA satisfaction rate as the percentage of SLA transactions with positive remaining deadlines (§4.1) when their committed blocks reach their submitted clients. For other systems, we define their SLA satisfaction rates by checking the clients' elapsed clocks. We report throughput as txn/s for both SLA and non-SLA transactions. We define commit latency as the client perceived elapsed time for all committed transactions.

Our evaluation focuses on the following questions:

§7.1 How does SLARM meet SLA for SLA transactions and what is the performance for all transactions?

§7.2 How resilient is SLARM's SLA and throughput on spikes of SLA transactions?

§7.3 How resilient is the SLA and throughput to node failures in SLARM?

§7.4 How is the SLA and throughput for five applications in SLARM?

§7.5 What are the limitations and potential future works of SLARM?

7.1 SLA and Performance

Figure 1a shows all systems' SLA satisfaction rate for SLA transactions(for SLARM, 70% of all transaction; for other systems, 100%, because they cannot distinguish SLA transactions). Figure 1b shows the throughput of all systems for all transactions. From 0 to 8s, the network launched 200 clients to generate 200 txn/s totally (in order to make the PoA consensus protocol reach its peak throughput), and 70% of them were SLA transactions. On 8s, we generated 200 additional SLA transactions for five seconds as a spike. Overall, SLARM's SLA satisfaction rate for SLA transactions dropped from 98.0% to 82.1% at about 15s and then quickly recovered. The other systems' SLA satisfaction rate all dropped significantly and recover much slower than SLARM. Gossip achieved the lowest SLA satisfaction rate because it selects only a subset of peers to disseminate transactions without any gap-filling. This indicates that SLARM achieves the highest SLA satisfaction rate on the spike (all systems incur the same number of additional SLA transactions). Figure 1b shows that Gossip's throughput is the best, and SLARM incurred roughly 6.9% overhead on all transactions' throughput compared to Gossip. This is because SLARM's reliable multicast incurs a lightweight bidirectional gap-filling in addition to Gossip (Figure 4). Other reliable multicast protocols and flooding incur much higher throughput overhead compared to Gossip.

To investigate the reasons of SLARM's high SLA satisfaction rate and its throughput overhead, we collected all sys-

Applications	SLA Txs	Non-SLA Txs	Traffic Spike
Mobile carriers [14, 20]	Pre-paid users (10%)	Post-paid users (90%)	×
Disease control [3, 4, 68]	High-risk events (50%)	Low-risk events (50%)	×
Online trading [43, 56]	Real time trading (70%)	Non-real time trading (30%)	✓
Voting [13, 32, 34]	Election management (10%)	Cast votes (90%)	×
CDN accounting [10, 33, 62]	Settlements (10%)	Usage data (90%)	✓

Table 2: SLARM's evaluated blockchain applications. Parameters are from the cited blockchain papers on this table. All SLA transactions are written in smart contracts.

System	Consensus Latency(s) (τ_c)	Avg P2P Latency(s) (τ_d)	Gap-filling		SLA Txn Miss Rate
			num	cost (s)	
SLARM	6.1	6.0	5	1.1	3.1%
Flooding	6.2	8.8	N/A	N/A	0.05%
Gossip	5.9	11.1	N/A	N/A	15%
Corrected	6.1	7.1	13	2.6	3.3%
Erlay	6.3	7.3	21	4.2	3.6%
Deadline	6.2	7.8	23	4.6	4.1%

Table 3: SLARM micro-events before the spike launched at the 8s in Figure 1a on AWS. Miss Rate means the ratio of missed SLA transactions in gaps on consensus nodes.

System	Consensus Latency(s) (τ_c)	Avg P2P Latency(s) (τ_d)	Gap-filling		SLA Txn Miss Rate
			num	cost (s)	
SLARM	7.5	26.8	33	6.6	21.4%
Flooding	7.2	123.7	N/A	N/A	11.1%
Gossip	7.8	143.1	N/A	N/A	64.1%
Corrected	8.6	75.3	157	31.4	31.9%
Erlay	7.8	88.3	198	39.8	35.9%
Deadline	8.0	93.1	203	41.6	36.2%

Table 4: SLARM micro-events at the 16s (lowest SLA satisfaction rate) in Figure 1a on AWS. Miss Rate means the ratio of missed SLA transactions in gaps on consensus nodes.

tems' mirco-events at the 7s in Table 3 and the same events at the 16s (lowest SLA rate for SLARM) in Table 4. On Table 3, SLARM's mean commit latency ($\tau_d + \tau_c$) is much less than $2 \times T = 34s$, where T is 17 according to Gossip in Table 3. This table explains SLARM's high satisfaction rate of 98.X% before the spike comes at 8s. After 8s, SLARM incurs higher burden on disseminating SLA transactions, so its per transaction τ_d increases from 6s in Table 3 to 26.8s in Table 4, leading to a decreased SLA satisfaction rate. However, SLARM's SLA satisfaction rate was still much better than the other three reliable multicast protocols, because their τ_d increased by at least one order of magnitude, much larger than the SLA deadline $2 \times T = 34s$. The reason SLARM's τ_d is much lower than all the other systems in Table 4 is two folds: (1) for reliable multicast protocols, SLARM incurred a much fewer number of gap-filling invocations and spent much fewer time in gap-filling; (2) for traditional flooding and Gossip, they either incurred high transaction miss rate on consensus nodes (64.1% for Gossip) or incurred severe P2P network congestion (flooding's τ_d is 123.7s).

To understand the throughput of SLARM's SLA prioritization mechanism (§4) for SLA and non-SLA transactions, we broke down SLARM's portion of SLA transactions and non-SLA transactions in each committed block in Figure 5, which shows that SLARM's mechanism gave high priority for SLA transactions in the first half of the committed blocks, and the non-SLA transactions took the majority in the second half. This implies that SLARM always schedules SLA transactions first (§4.2) and avoids non-SLA transactions to block SLA transactions' dissemination.

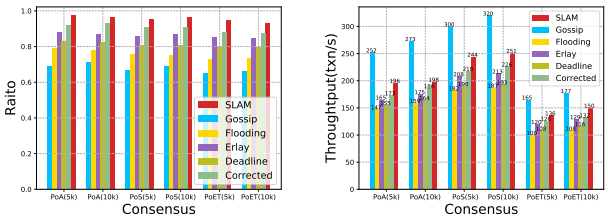
Regarding all systems' client perceived latency on committed SLA transactions, Table 4 can give a good indication. By summing up the τ_d and τ_c columns, SLARM achieved the lowest latency for SLA transactions among all systems since the spike occurred in the 8s. Note that this spike is fair for all systems, because the added 200 txn/s for 5s in Figure 1a were all SLA transactions. For non-SLA transactions, SLARM does sacrifice their commit latency, indicated in Figure 5. Note that the first committed block in Figure 5 happened at the 8s in Figure 1a, and Figure 5 shows that, in the first 20 committed blocks (each has a commit latency of 5s in PoA), non-SLA transactions took less than 10%. Since the online trading application has 30% non-SLA transactions, Figure 5 indicates that SLARM greatly sacrifices the commit latency of non-SLA transactions, which matches SLARM's design goal on favoring SLA transactions. SLARM's performance tradeoff between SLA and non-SLA transactions also makes SLARM's throughput for all transactions slightly lower than Gossip.

Since there is no cloud provider that can run the SGX hardware on clouds, we ran SLARM's enclave code using Intel's SGX simulator on AWS. Table 5 shows the micro-events of running SLARM's enclave code on both the SGX simulator and running the same code on the SGX hardware of our cluster, which shows similar performance cost. SLARM's time spent in SGX is not the bottleneck of SLARM's performance, and we consider SLARM's performance results reported on AWS would be close to running physical SGX hardware on future clouds (if they can provide SGX hardware).

In addition to PoA, we also evaluated SLARM on different P2P network scales (5k and 10k nodes) with two other

SGX	# ECalls	Enc/Dec	No spike	Spike
Cluster	93	1.8ms	3.4s	20.8s
AWS	115	2.3ms	4.8s	21.3s

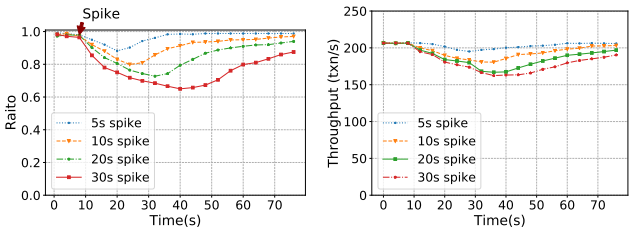
Table 5: SLARM's SGX micro-events in a SLA transaction's entire life cycle. **# ECalls** means the total number of `ECalls` invoked for this transaction on all SLARM nodes that disseminate this transaction on all route paths, **Enc/Dec** means the corresponding total encryption and decryption time in SGX, **No spike** means the corresponding total wait time of this transaction on all route paths before the spike in Figure 1a. **Spike** means an SLA transaction's corresponding total wait time on all route paths during the spike.



(a) SLA guarantees (b) Throughputs

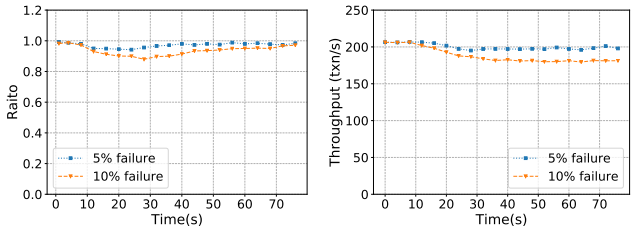
Figure 6: SLA guarantees and normalized throughput of PoA [2], PoS [38], and PoET [23] with 5k and 10k P2P nodes.

high-throughput consensus protocols on Ethereum, shown in Figure 6. Systems' SLA transaction rate in Figure 6a was the same as Figure 1a before the 8s. Overall, SLARM achieves reasonable SLA satisfaction rate and reasonable overhead on throughput (Figure 6b) compare to Ethereum running with default Gossip multicast. Since typical applications (Table 2) are often deployed with 5000 P2P nodes [10], we consider SLARM scalable to the network scale. Figure 1b also suggests that SLARM's throughput is mainly determined by consensus's throughput, due to SLARM's light-weight gap-filling mechanism (§4.3).



(a) SLARM's SLA guarantee (b) SLARM's throughput

Figure 7: SLARM's SLA and throughput under different lasting time lengths of transaction spikes.



(a) SLARM's SLA guarantee (b) SLARM's throughput

Figure 8: SLARM SLA and throughput with node failures.

7.2 Resilience on SLA Transaction Spikes

We also studied all systems' resilience to different degrees of SLA transaction spikes. We started with the same setting as the setting of 0s of Figure 1a, and we varied the lasting time lengths of the 200 additional SLA txn/s from 5s to 30s. Figure 7 shows that, for the 30s spike curve, SLARM's $N.RTT$ among all nodes at the 40s (lowest SLA rate) is 0.6s to 1.3s. At this 40s, the $N.RTT$ value on each SLARM node was larger than all one hop SLA transactions' dissemination time cost observed on the node. This indicates that SLARM's trustworthy RTT mechanism (§5.3) is able to capture the random packet delay attacks or network congestions with high probability and makes SLARM's SLA deadline update mechanism conservative (§4). Figure 7 shows that, on the 30s of lasting SLA spikes, SLARM' SLA satisfaction rate dropped to as low as 67% and then recovered. This is much better than Corrected Gossip even if the spike of SLA transactions lasts for only 5s.

7.3 Robustness on Node Failures

Node failures in a P2P network can be triggered by hardware failures or DoS attacks, and such failures are more severe than traffic spikes. We measured SLARM's SLA satisfaction rate and throughput with node failures, as shown in Figure 8. On 8s, we randomly selected 10% of nodes in SLARM's network and killed them. SLARM's SLA satisfaction rate dropped from 98.1% to about 89.2%, and its throughput for all transactions dropped from about 201 txn/s to 170 txn/s. In fact, given a more sparse P2P network topology, SLARM's SLA prioritization mechanism has to reconnect peers and recompute the conservative RTTs (§5.3) for nodes, and new peers are often farther, leading to more stringent SLA deadline during the dissemination.

7.4 SLA and Performance on Applications

The above evaluation focuses on evaluating the online trading application with different systems on different scenarios. We deployed each of the five applications (Table 2) in SLARM individually and measured their SLA and through-

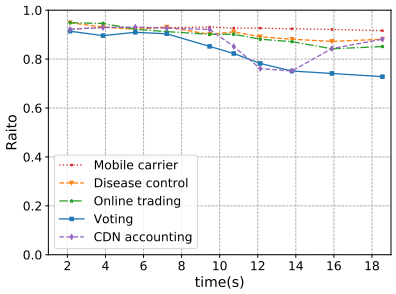


Figure 9: SLA guarantees for five applications (Table 2).

put according to their own application settings (e.g., portions of SLA transactions and spikes). Figure 9 shows the results for these applications. Overall SLARM's SLA mechanisms (§4 and §5) are generic for diverse applications.

7.5 Discussion

SLARM has two limitations. First, SLARM's transaction scheduling mechanism is based on Intel SGX. It is essential to protect the SLA metadata in an untrusted environment with faulty nodes and to conservatively schedule transactions when traffic spikes and node failures occur. Nowadays, SGX is prevalent in modern CPUs. More and more blockchain consensus protocols are developed with SGX (§8).

Second, SLARM's performance is designed to favor SLA transactions over non-SLA transactions. Figure 5 indicates that SLARM highly favors SLA transactions and defers non-SLA transactions. After all, more and more Internet-wide blockchain applications are developed with the stateful smart contracts, and some transactions are submitted in an interactive way in web browsers or mobile apps. These transactions generally desire a more stringent SLA guarantee, while the deadlines for non-SLA transactions do not matter, so SLARM's trade-off is worthwhile.

SLARM, the first SLA-aware transaction dissemination system for permissioned blockchains, enables the research community to develop even more exciting blockchain systems and applications with diverse SLA requirements. For instance, SLARM can facilitate the development of new heterogeneous blockchain applications consisting of fine-grained SLA requirements (e.g., a blockchain application consisting of online trading, auction, and clearing transactions). In addition, because SLARM's SLA scheduling mechanism is conservative (§4.2), SLARM can make SLA guided verifications feasible (e.g., trustworthy SLA incentive and auditing mechanisms) and can match the demands of blockchain-driven CDN networks [10] and auditing applications [57]. Last but not least, SLARM may also be used in permissionless blockchains [49], because permissionless blockchains prioritize transactions based on transaction fees without any SLA guarantee. We leave these exciting innovations for future works.

8 Related Work

SGX-powered Blockchains. SGX improves diverse aspects of blockchain systems. Intel's PoET [53] and its variant [47] replaces the PoW puzzles with a trusted timer in SGX. REM [69] uses SGX to replace the "useless" PoW puzzles with "useful" computation (e.g., big data). Microsoft CCF [55] (originally named COCO) is a permissioned blockchain platform using SGX to achieve transaction privacy. Hawk [39] and zkLedger [50] focus on enhancing confidentiality of smart contracts [17]. Ekiden [25] and ShadowEth [67] offload the execution of smart contracts to a small group of SGX powered computing nodes to avoid the redundant smart contract executions on all consensus nodes. BlockStack [11] builds a decentralized DNS and storage services on blockchains. TeeChain [45] is a payment network and leverages SGX to prevent parties from misbehaving. SLARM is complementary to these SGX blockchain systems and can be integrated into them.

P2P reliable multicast. Reliable multicast protocols exist [16, 30, 35, 51]. They typically follow the disseminate-correct scheme. Bimodal multicast [16] first disseminates messages, then corrects lost messages with anti-entropy. Corrected Gossip [35] follows the same pattern and achieves much lower message complexity with stronger reliability guarantees. Chryssis et al. first considers on-time message delivery in Gossip [30]. Erlay [51] integrates the reliable multicast with Bitcoin [49] to improve both the bandwidth efficiency of Bitcoin and the security of the system. SLARM's protocol differs from these protocols in that SLARM emphasizes achieving a no-gap guarantee efficiently by co-designing the P2P and consensus level (bi-directional) rather than correcting all lost messages (uni-directional).

P2P and SLA. Several SLA provisioning protocols have been proposed for P2P networks [28, 44, 66]. They all focus on one-to-one message routing, while blockchain's transaction delivery requires one-to-all multicast. Chryssis et al. show how to ensure on-time message delivery in Gossip multicast protocol. Unlike SLARM, these protocols do not consider DoS or targeted deferring attacks toward protocol messages, crucial in blockchain deployments.

9 Conclusion

We have present SLARM, the first blockchain transaction dissemination system that can meet diverse SLAs of different applications. SLARM's integration with Ethereum has the potential to attract broad Internet-wide applications with SLA requirements to be developed upon. This will not only greatly improve the reliability of these applications but also greatly improving the efficiency of both user-perceived latency and network bandwidth. All SLARM source code and evaluation results are released on github.com/osdi20p264.

References

[1] Thinning and superposition the poisson process. https://www.randomservices.org/random/poisson/Splitting.html.

[2] Clique poa protocol & rinkeby poa testnet. https://github.com/ethereum/EIPs/issues/225, 2017.

[3] CDC is Testing Blockchain to Monitor the Country's Health in Real Time. https://tinyurl.com/y748cm38, 2018.

[4] How IBM and the CDC are testing blockchain to track health issues like the opioid crisis. https://tinyurl.com/y8trjxyn, 2018.

[5] Introduction to smart contracts. https://solidity.readthedocs.io/en/v0.4.24/introduction-to-smart-contracts.html, 2018.

[6] Ethereum's proof of stake faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ, 2019.

[7] Flooding. https://en.wikipedia.org/wiki/Flooding_(computer_networking), 2019.

[8] Amazon EC2. https://aws.amazon.com/ec2/, 2020.

[9] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *ACM SiGMOD Record*, 32(3):29–33, 2003.

[10] Elif Ak and Berk Canberk. Bcdn: A proof of concept model for blockchain-aided cdn orchestration and routing. *Computer Networks*, 161:162–171, 2019.

[11] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J Freedman. Blockstack: Design and implementation of a global naming system with blockchains, 2016. Accessed: 2016-03-29.

[12] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[13] Ahmed Ben Ayed. A conceptual secure blockchain-based electronic voting system. *International Journal of Network Security & Its Applications*, 9(3):01–09, 2017.

[14] Jere Backman, Seppo Yrjölä, Kristiina Valtanen, and Olli Mämmelä. Blockchain network slice broker in 5g: Slice leasing in factory of the future use case. In *2017 Internet of Things Business Models, Users, and Networks*, pages 1–8. IEEE, 2017.

[15] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421*, 2018.

[16] Kenneth P Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2):41–88, 1999.

[17] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2014. Accessed: 2016-08-22.

[18] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[19] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[20] Abdulla Chaer, Khaled Salah, Claudio Lima, Pratha Pratim Ray, and Tarek Sheltami. Blockchain for 5g: opportunities and challenges. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2019.

[21] Stephen Checkoway and Hovav Shacham. Iago attacks: why the system call api is a bad untrusted rpc interface. *ACM SIGARCH Computer Architecture News*, 41(1):253–264, 2013.

[22] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.

[23] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.

[24] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. Detecting privileged side-channel attacks in shielded execution with déjá vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 7–18, 2017.

[25] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *arXiv preprint arXiv:1804.05141*, 2018.

[26] Christopher D Clack, Vikram A Bakshi, and Lee Braine. Smart contract templates: foundations, design landscape and research directions. *arXiv preprint arXiv:1608.00771*, 2016.

[27] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[28] Emad Felemban, Chang-Gun Lee, and Eylem Ekici. Mmspeed: multipath multi-speed protocol for qos guarantee of reliability and. timeliness in wireless sensor networks. *IEEE transactions on mobile computing*, 5(6):738–754, 2006.

[29] Message Passing Interface Forum. Mpi: A message-passing interface standard version 2.2, September 2009.

[30] Chryssis Georgiou, Seth Gilbert, and Dariusz R Kowalski. Meeting the deadline: on the complexity of fault-tolerant continuous gossip. *Distributed Computing*, 24(5):223–244, 2011.

[31] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[32] Rifa Hanifatunnisa and Budi Rahardjo. Blockchain based e-voting recording system design. In *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–6. IEEE, 2017.

[33] Nicolas Herbaut and Nicolas Negru. A model for collaborative blockchain-based video delivery relying on advanced network services chains. *IEEE Communications Magazine*, 55(9):70–76, 2017.

[34] Friðrik Þ Hjálmarsson, Gunnlaugur K Hreiðarsson, Mohammad Hamdaqa, and Gísli Hjálmtỳsson. Blockchain-based e-voting system. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 983–986. IEEE, 2018.

[35] Torsten Hoefler, Amnon Barak, Amnon Shiloh, and Zvi Drezner. Corrected gossip algorithms for fast reliable broadcast on unreliable systems. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 357–366. IEEE, 2017.

[36] Yutao Jiao, Ping Wang, Dusit Niyato, and Zehui Xiong. Social welfare maximization auction in edge computing resource allocation for mobile blockchain. In *2018 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2018.

[37] Li-jie Jin, Vijay Machiraju, and Akhil Sahai. Analysis on service level agreement of web services. *HP June*, page 19, 2002.

[38] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[39] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Symposium on Security & Privacy*. IEEE, 2016.

[40] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 711–725, 2018.

[41] Joao Leitao, José Pereira, and Luis Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 419–429. IEEE, 2007.

[42] Joao Leitao, José Pereira, and LuÍs Rodrigues. Gossip-based broadcast. In *Handbook of Peer-to-Peer Networking*, pages 831–860. Springer, 2010.

[43] Zhetao Li, Jiawen Kang, Rong Yu, Dongdong Ye, Qingyong Deng, and Yan Zhang. Consortium blockchain for secure energy trading in industrial internet of things. *IEEE transactions on industrial informatics*, 14(8):3690–3700, 2017.

[44] Zhi Li and Prasant Mohapatra. Qron: Qos-aware routing in overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):29–40, 2004.

[45] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gun Sirer, and Peter Pietzuch. Teechain: A secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM Symposium on*

Operating Systems Principles, SOSP '19, pages 63–79, New York, NY, USA, 2019. ACM.

[46] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *HASP @ ISCA*, 10, 2013.

[47] Mitar Milutinovic, Warren He, Howard Wu, and Maxinder Kanwal. Proof of luck: An efficient blockchain consensus protocol. In *SysTEX '16 Proceedings of the 1st Workshop on System Software for Trusted Execution*, pages 2:1–2:6. ACM, 2016.

[48] Yves Mocquard, Bruno Sericola, and Emmanuelle Anceaume. Probabilistic analysis of rumor-spreading time. *INFORMS Journal on Computing*, 2019.

[49] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[50] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. *auditing*, 17(34):42.

[51] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. Erlay: Efficient transaction relay for bitcoin. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 817–831, 2019.

[52] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[53] Giulio Prisco. Intel develops 'sawtooth lake'distributed ledger technology for the hyperledger project. *Bitcoin Magazine*, 2016.

[54] S Sundhar Ram, A Nedić, and Venugopal V Veeravalli. Asynchronous gossip algorithms for stochastic optimization. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 3581–3586. IEEE, 2009.

[55] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, CÃ©dric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olya Ohrimenko, Felix Schuster, Roy Schwartz, Alex Shamis, Olga Vrousgou, and Christoph M. Wintersteiger. Ccf: A framework for building confidential verifiable replicated services. Technical Report MSR-TR-2019-16, Microsoft, April 2019.

[56] Moein Sabounchi and Jin Wei. Towards resilient networked microgrids: Blockchain-enabled peer-to-peer electricity trading mechanism. In *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–5. IEEE, 2017.

[57] Bruce Schneier and John M Kelsey. Event auditing system, November 2 1999. US Patent 5,978,475.

[58] Alberto Sonnino, Michał Król, Argyrios G Tasiopoulos, and Ioannis Psaras. Asterisk: Auction-based shared economy resolution system for blockchain. *arXiv preprint arXiv:1901.07824*, 2019.

[59] Douglas B Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 309–324, 2013.

[60] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[61] Jinesh Varia. Migrating your existing applications to the aws cloud. *A Phase-driven Approach to Cloud Migration*, 2010.

[62] Thang X Vu, Symeon Chatzinotas, and Björn Ottersten. Blockchain-based content delivery networks: Content transparency meets user privacy. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2019.

[63] Matthew Walck, Ke Wang, and Hyong S Kim. Tendrilstaller: Block delay attack in bitcoin. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 1–9. IEEE, 2019.

[64] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2421–2434, 2017.

[65] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[66] Hao Yang, Minkyong Kim, Kyriakos Karenos, Fan Ye, and Hui Lei. Message-oriented middleware with qos awareness. In *Service-Oriented Computing*, pages 331–345. Springer, 2009.

[67] Rui Yuan, Yu-Bin Xia, Hai-Bo Chen, Bin-Yu Zang, and Jan Xie. Shadoweth: Private smart contract on public blockchain. *Journal of Computer Science and Technology*, 33(3):542–556, 2018.

[68] Xiao Yue, Huiju Wang, Dawei Jin, Mingqiang Li, and Wei Jiang. Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control. *Journal of medical systems*, 40(10):218, 2016.

[69] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert van Renesse. Rem: Resource-efficient mining for blockchains. http://eprint.iacr.org/2017/179, 2017. Accessed: 2017-03-24.