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

Ji Qi, Tianxiang Shen, Jianyu Jiang, Xusheng Chen, Xiapu Luo, Fengwei Zhang, *Senior Member, IEEE*, and Heming Cui, *Member, IEEE*

***Abstract*—The blockchain paradigm has attracted diverse applications to be deployed upon. However, no service-level agreement (SLA) mechanism has been proposed to enforce the SLA disseminating deadlines to commit blockchain transactions, although these transactions are often interactively submitted by clients and desire short SLA deadlines (e.g., tens of seconds). Existing peer-to-peer (P2P) multicast protocols for blockchains take the unidirectional approach to disseminate transactions regardless of their SLA deadlines, making transactions easily violate their deadlines. Moreover, these protocols are vulnerable to malicious P2P nodes, and their protocol messages (e.g., SLA-stringent transactions) are vulnerable to deferring attacks. We propose SLARM, the first bidirectional P2P multicast protocol for permissioned blockchains, which conservatively adjusts transactions' dissemination speed to satisfy their SLA deadlines according to the trustworthy SLA feedback of previously disseminated transactions. SLARM guarantees transactions' SLAs in a decentralized way and defends against the deferring attacks using TEE. Evaluation of SLARM with five notable P2P multicast protocols and five diverse real-world applications shows that: even with transaction spikes and attacked nodes, SLARM achieves a much higher transaction SLA satisfaction rate with reasonably high commit throughput.**

***Index Terms*—Network attacks, permissioned blockchain, service level agreement, trusted execution environment.**

I. Introduction

THE blockchain [1] is an immutable ledger maintained by mutually untrusted participants (i.e., clients and nodes) on a large peer-to-peer (P2P) network (e.g., the Internet) consisting of nodes. Some of these nodes are *consensus nodes* and run a block *consensus protocol* to agree on the order of clients' transactions in committed blocks.

The permissioned blockchain (e.g., Quorum [2]) runs among explicitly identified participants [2], [3]. In contrast to permissionless blockchains like Bitcoin [1], a permissioned blockchain is decoupled from the cryptocurrency, requiring nodes to contribute either high energy (e.g., PoW [1]) or wealth (e.g., PoS [4]). Therefore, permissioned blockchains can run fast Byzantine-fault-tolerant (BFT) protocols like PBFT [5] to achieve better energy efficiency and performance than permissionless blockchains [6]. This makes permissioned blockchains especially suitable for deploying performance-sensitive smart contract applications (e.g., trading [6]).

As more performance-sensitive applications are deployed on permissioned blockchains, the service-level agreements (SLA) on deadlines of transactions' commit latencies are becoming increasingly important. For instance, Ethereum [7] allows a smart contract to specify a time deadline for accepting transaction invocations [8], and such a deadline can be short (e.g., a blockchain-driven flash auction opens for only a few minutes [9]). More importantly, different transactions may have diverse SLA requirements. In a trading application [6], the real-time payments need to be committed within a short deadline, while the non-real-time payments often have no deadline requirements. We call the transactions with deadline requirements "SLA transactions", and call the actual fraction of transactions meeting the deadline "SLA satisfaction rate."

A blockchain transaction's commit latency mainly hinges on the transaction's dissemination latency throughout the large blockchain network (Section IV-A). Permissioned blockchains generally adopt fast and bandwidth-efficient P2P multicast protocols [10], [11] based on *partial view Gossip* [12] to disseminate transactions. Specifically, for scaling to a large blockchain network (e.g., 10 K nodes [13]), P2P multicast protocols disseminate messages with the *partial view* (i.e., a small subset of all P2P nodes) rather than the global view (i.e., the set of all P2P nodes): when a node performs a multicast step, it sends messages to random nodes from its partial view.

However, despite the great advancements of existing partial view P2P multicast protocols [10], [11], [12], [14], [15], they are not suitable for meeting the transactions' SLA deadline requirements due to two critical issues. First, since the network latency among nodes in different regions varies drastically, nodes with only the partial view can easily mispredict a transaction's dissemination latency to the entire network and

Received 27 August 2023; revised 9 May 2025; accepted 15 July 2025. Date of publication 24 July 2025; date of current version 9 October 2025. This work was supported in part by the National Key R&D Program of China under Grant 2023YFB4503905, in part by HKU-CAS Joint Laboratory for Intelligent System Software, and in part by SupernetAI. *(Ji Qi and Tianxiang Shen contributed equally to this work.) (Corresponding authors: Ji Qi; Jianyu Jiang.)*

Ji Qi is with the Institute of Software Chinese Academy of Sciences, Beijing 100190, China, and also with the Key Laboratory of System Software (Chinese Academy of Sciences), Beijing 100190, China (e-mail: qiji@iscas.ac.cn).

Tianxiang Shen and Xusheng Chen are with Huawei Technologies Company, Ltd., Shenzhen, Guangdong 518129, China (e-mail: stx635@connect.hku.hk; chenxus@connect.hku.hk).

Jianyu Jiang is with ByteDance Ltd., Beijing 100086, China (e-mail: jianyu@connect.hku.hk).

Xiapu Luo is with Hong Kong Polytechnic University, Hung Hom, Hong Kong (e-mail: csxluo@comp.polyu.edu.hk).

Fengwei Zhang is with the Southern University of Science and Technology, Shenzhen, Guangdong 518055, China (e-mail: zhangfw@sustech.edu.cn).

Heming Cui is with the University of Hong Kong, Pok Fu Lam, Hong Kong (e-mail: heming@cs.hku.hk).

Digital Object Identifier 10.1109/TSC.2025.3592411

select a suboptimal speed to disseminate transactions. Second, as each node is unaware of the SLA transactions submitted by other nodes in a partial view, SLA-stringent transactions can be easily delayed by the non-stringent transactions. Worse, since transactions submitted by the same client must be committed in order without losing one (Section IV-B), if an SLA transaction is delayed, all subsequent SLA transactions will also be delayed and violate their SLAs, further aggravating the SLA satisfaction rate. As a result, non-stringent SLA transactions may be disseminated throughout the network much faster than their SLA deadlines (consumes more bandwidth and CPU [10]), making SLA-stringent transactions be delayed by these non-stringent transactions, inhibiting the overall SLA satisfaction rate under traffic spikes.

Overall, existing P2P multicast protocols face the dilemma between achieving high scalability and high SLA satisfaction rate. On the one hand, constraining each node's view (i.e., the partial view) to increase the scalability inevitably sacrifices the SLA satisfaction rate. On the other hand, utilizing a global view to achieve a high SLA satisfaction rate cannot scale up.

In this study, our key insight to tackle the dilemma is that, the remaining deadlines of previous disseminated transactions, which reflect the statistical global view of the entire network, can be used as important feedback information to the P2P multicast protocol. With this global feedback, we can solve the above two critical issues by making nodes select near-optimal dissemination speed for transactions to match their SLA deadlines, and nodes can obtain the stringency of SLA transactions to prioritize SLA-stringent transactions over non-stringent transactions in dissemination. By collecting the statistical feedback reflecting the global view, we can achieve high SLA satisfaction rates, as well as high scalability because we still disseminate all transactions via the partial view (i.e., via a small subset of all P2P nodes).

This insight leads to SLARM, the first SLA-aware *bidirectional* reliable multicast protocol for permissioned blockchains. SLARM disseminates each transaction twice. In the *forward directional multicast*, SLARM disseminates transactions via Gossip with the default dissemination speed and tracks transactions' remaining time to their SLA deadlines. In the *backward directional multicast*, SLARM disseminates transactions along with their remaining deadlines (when they are committed) in blocks throughout the network. In typical blockchains, blocks must be disseminated to all nodes [1], [7]. SLARM leverages this block dissemination as the feedback to the forward directional multicast, adjusting SLA transactions' dissemination speed in the forward directional multicast according to transactions' remaining deadlines in committed blocks. Compared with existing *unidirectional* P2P multicast protocols lacking the SLA feedback, SLARM aligns the dissemination speed of SLA transactions with their SLA deadlines, achieving a high SLA satisfaction rate (Section VII-A1).

The main research obstacle to making SLARM's bidirectional multicast protocol practical in the blockchain domain is that SLARM runs on mutually untrusted blockchain nodes, where some nodes can be malicious and corrupt the integrity of any SLA mechanism running on nodes. Fortunately, the pervasive usage of the Trusted Execution Environments (TEE), in particular Intel SGX, in blockchain systems (e.g., Teechain [16]) shows that the integrity and confidentiality features of TEE can regulate the misbehavior of malicious nodes.

However, even with TEE, ensuring the end-to-end SLA deadlines for SLA transactions still faces two subtle challenges. First, SLARM must generate trustworthy feedback on transactions' remaining deadlines to the forward directional multicast in a decentralized way without a globally synchronized clock, which can be arbitrarily manipulated by malicious nodes (Section V). The absence of a synchronized clock makes it fundamentally impossible for nodes to track transactions' precise remaining deadlines on the Internet. Second, SLARM must tackle deferring attacks on the Internet toward SLARM's protocol messages, such as SLA transactions (Section VI).

We address the aforementioned challenges by creating a new *conservative deadline* mechanism which infers the stringency (i.e., remaining deadline) of SLA transactions *conservatively:* each SLA transaction's inferred remaining deadline by SLARM is more stringent than its actual remaining deadline. SLARM ensures that SLA transactions are distributed faster than their actual deadlines, even with malicious nodes and deferring attacks in the blockchain domain, without needing to track the precise remaining deadlines. This remaining deadline is piggybacked to each transaction and is updated conservatively in a decentralized way, allowing nodes to evaluate the stringency of each transaction without a globally synchronized clock.

This paper makes three main contributions.

1) We propose SLARM, the first *bidirectional* P2P multicast protocol that meets transactions' SLA deadlines in permissioned blockchains. SLARM uses blockchain's block dissemination (backward) to provide SLA deadline feedback to the transaction dissemination (forward) for adjusting transactions' dissemination speeds to match SLA deadlines (Section V).
2) We identify the attacks when satisfying transactions' SLA deadlines in permissioned blockchains. SLARM tackles the attacks with a novel *conservative deadline mechanism* to infer SLA transaction's remaining SLA deadlines in a trustworthy and conservative way (Section VI). Existing P2P multicast protocols (e.g., Erlay [10] and Corrected Gossip [11]) are not designed to tackle such attacks, so SLARM is more secure than existing P2P multicast protocols [10], [11], [14].
3) We implemented SLARM on the permissioned blockchain Quorum [2] and evaluated SLARM on Azure [17]. We compared SLARM with two traditional P2P multicast protocols (Gossip [12] and Flooding [15]) and three state-of-the-art reliable multicast protocols (Erlay [10], Corrected Gossip [11], and Deadline Gossip [14]). Specifically, Erlay [10] is the latest reliable multicast protocol for blockchains. We evaluated all protocols with diverse real-world applications. Evaluation results (Section VII) demonstrated that SLARM achieves much higher SLA satisfaction rate for SLA transactions than all the evaluated baselines. Compared to Gossip, the most lightweight protocol among the five evaluated baselines, SLARM achieves a reasonable low overhead of 11.3% on the throughput of all transactions.

II. Background and Related Work

A. P2P Multicast Protocols

In a blockchain, it is essential to disseminate transactions to **all nodes** to reach consensus nodes, which can be unknown, hidden, or constantly changing during transaction dissemination. For instance, the Proof of Authority (PoA) consensus protocol Clique [18], one of the most widely adopted permissioned blockchain consensus protocols [7], irregularly

TABLE I
COMPARISON OF SLARM AND EXISTING STUDIES

Systems	**High SLA Rate**	**Scalability**	**Tackle Network Attacks**
Global View P2P multicast protocols			
✧ CloudCast [20], Segment EDF [21]	✓	✗	✗
❖ FireFlies [22]	✓	✗	✗
❖ Deadline Gossip [14]	✓	✗	✗
Partial View P2P multicast protocols			
✧ FRing [23]	✗	✓	✗
❖ Gossip [12], Flooding [15]	✗	✓	✗
❖ Erlay [10]	✗	✓	✗
❖ Corrected Gossip [11]	✗	✓	✗
❖ **SLARM (this study)**	✓	✓	✓

'✧' indicates *structured* P2P multicast protocols; '❖' indicates randomized P2P multicast protocols.

re-elects consensus nodes among all nodes. Proof of Stake (PoS [4]) and Proof of Elapsed Time (PoET [19]) allow any node to propose blocks. EGES [13] conceals consensus nodes from other nodes and clients to tackle DoS attacks toward the consensus nodes.

1) Blockchains' P2P Multicast Protocols: A P2P multicast protocol can be randomized or structured (Table I). Typical blockchains use randomized P2P multicast protocols, which can be classified into two types: Gossip [12] and reliable multicast [10], [11].

In Gossip protocols (used by Ethereum [7], Quorum [2], and Hyperledger Fabric [3]), P2P nodes periodically send messages to subsets of the nodes' connected **peers** (i.e., a random set of P2P nodes of the entire network [10]). Flooding [15] is a special case of Gossip, wherein each node periodically sends messages to all of its peers. In Gossip, each node receives each message multiple times, thereby addressing both node and link failures. Although Gossip can disseminate a message to all nodes with high probability and moderate communication and computation overhead, Gossip is susceptible to message losses, causing gaps in nodes' received messages.

In reliable multicast protocols [10], [11], P2P nodes generally adopt the *Gossip* $\rightarrow$ *fix* scheme to detect and retransmit lost messages, ensuring nodes receive the latest gap-free messages with a high probability. Erlay [10] integrates the reliable multicast protocol with Bitcoin [1] to enhance its robustness. However, to achieve strong reliability one employs costly protocols, whose high communication and computation overhead lead to unstable performance under traffic spikes. In sum, existing randomized P2P multicast protocols either excel in efficiency or in reliability but not both.

Structured P2P multicast protocols [20] disseminate messages through dissemination trees composed of P2P nodes and reconstruct the tree on node failures. In a benign environment, structured tree-based multicasts attain both high reliability and high efficiency. However, they are not suitable for blockchains for two reasons. First, permissioned blockchain nodes can join or leave the blockchain network at any time, causing frequent reconstructions of the dissemination tree. Second, and more important, the permissioned blockchain nodes are mutually untrusted, so malicious nodes in the dissemination tree can easily break the tree structure.

2) SLA-Aware P2P Multicast Protocols: Several SLA-aware P2P multicast protocols rely on a global view to manage per-link message delivery [20], which is impractical for large scale blockchain networks, where nodes can dynamically join or leave at any time. They also do not consider malicious nodes and network attacks, crucial in blockchain deployments.

Existing partial view P2P multicast protocols are unidirectional (Section I). They disseminate transactions as fast as possible, unaware of the varying stringency and SLA deadline requirements of different transactions, resulting in poor SLA satisfaction rates (Section VII-A1).

A potential approach for tracking transactions' stringency (remaining deadlines) for enforcing SLA is to use the globally synchronized clock among nodes, but it is impractical for two reasons. (1) The local clock maintained by each node is subject to the drift [24], causing eventual divergence unless frequently resynchronized. (2) Achieving global clock synchronization is challenging due to the unpredictable network delay, and existing protocols are either inaccurate or expensive. For example, the Network Time Protocol (NTP) suffers from poor accuracy, with time errors reaching 100ms [25]. This is because NTP relies on round-trip time measurements to estimate the time offset between the node and NTP server, fluctuations in network latency can introduce inaccurate time estimates. Google's TrueTime [26] achieves high accuracy by running nodes in datacenters and equipping GPS and atomic clocks on per-datacenter master nodes, which are expensive and unsuitable for blockchain nodes on the Internet.

B. SLA-Aware Permissioned Blockchains

Many studies [27], [28] use permissioned blockchains to monitor SLA guarantees of external services (e.g., the availability of cloud services [27]) in a trustworthy manner. These studies collect real-time SLA logs using the blockchain and analyze the logs using smart contracts in a transparent and trustworthy way. These studies are orthogonal to SLARM, which focuses on the permissioned blockchains' own SLA guarantee in terms of transactions' commit deadline.

Overall, meeting transaction SLA deadlines in malicious and mutually untrusted permissioned blockchain remains an open problem. By co-designing the P2P multicast protocol with blockchain's consensus protocol, we create SLARM, the first bidirectional P2P multicast protocol that satisfies transactions' SLA deadlines while tolerating malicious nodes and attacks without compromising blockchain's end-to-end performance. SLARM can be integrated with existing partial view P2P protocols, leveraging trustworthy feedback to adjust the dissemination speed.

C. Trusted Execution Environment

A Trusted Execution Environment (TEE) is a secure area in a processor designed to execute sensitive operations while ensuring confidentiality and integrity for code and data. It provides an isolated environment separate from the main operating system, protecting against both software and hardware-based attacks. Popular implementations, like Intel SGX and ARM TrustZone, enable applications to perform sensitive tasks within secure enclaves or trusted applications. TEE is extensively used by notable blockchains (e.g., Teechain [16] and PoET [19]) to regulate malicious participants' misbehavior.

TEE is essential for SLARM for three reasons. (1) SLARM needs TEE's trusted timer to measure the elapsed time of SLA transactions spent on nodes and during node-to-node dissemination (Section VI-B1). (2) SLARM needs TEE's integrity feature to regulate nodes' behavior (Section III-B). (3) We use TEE's confidentiality feature to design a uniform message dissemination protocol, tackling deferring attacks (Section VI-B3). While cryptographic techniques like homomorphic encryption and non-interactive zero-knowledge proof can also provide integrity and confidentiality respectively, they are unsuitable for SLARM due to their high computational cost [29].

We implemented SLARM on Intel SGX; however, SLARM's design is not limited to SGX and its principles are general to other TEEs. We chose SGX due to its wide adoption in prior secure blockchains [19], [30]. SLARM supports other TEEs as long as the TEE provides integrity, confidentiality, and the trusted time source to implement the trusted timer (Section VI-B1). These features are common in existing TEEs [31], [32], [33]. Porting SLARM to other TEE platforms requires moderate engineering effort. For example, in ARM TrustZone, porting SLARM involves: (1) map SGX ECALLs to GlobalPlatform TEE Client APIs on the host side; (2) migrate enclave logic to Trusted Applications using GlobalPlatform TEE Core APIs; and (3) replace SGX's timer with TrustZone's secure time source [33].

III. SLARM's Overview

A. Deployment Requirements and Threat Model

SLARM targets permissioned blockchains, whose participants include P2P nodes (denoted as N) and clients [2]. The nodes can be *normal* or *consensus* nodes. Normal nodes perform three main tasks: (1) networking: gossiping transactions over the P2P network, (2) execution: validating and executing transactions, and (3) storage: maintaining the blockchain and its states. Consensus nodes execute all tasks of normal nodes and also run a BFT consensus protocol to commit blocks.

Same as existing permissioned blockchains [3], [6], [7], to ensure client transactions are eventually committed to the blockchain (i.e., liveness), SLARM requires a partial synchrony network (e.g., the Internet): there exists a global stabilization time (GST), after which all messages between correct nodes are delivered within a known maximum delay. All blockchain nodes and clients run in this network.

SLARM's threat model is the same as typical TEE-based blockchain systems [16]. All firmware and hardware components of TEE are trusted. Any nodes' components running in TEE are trusted; any nodes' components running outside TEE are untrusted and can be arbitrarily malicious. A SLARM node with malicious components is a malicious node (Section VI-B). Malicious nodes can drop or delay SLARM's protocol messages. All nodes' hosts have only loosely synchronized clocks [7], and malicious nodes can arbitrarily manipulate their local clocks. We use a fine-grained trusted timer to conservatively measure transactions' elapsed time during dissemination (Section VI-B1). All clients can be malicious (Section VI-A). A malicious client can send illegal transactions (e.g., invoking an invalid smart contract).

We deployed SLARM on Quorum [2], one of the most notable open-source permissioned implementations of Ethereum. Quorum extends Ethereum by incorporating essential permissioned blockchain features, such as account management, access control, and the integration of diverse consensus protocols tailored for permissioned blockchains (e.g., IBFT [34]). SLARM can be generalized to run with other permissioned blockchains. Specifically, SLARM can seamlessly integrate with permissioned blockchains that use P2P multicast protocols to disseminate transactions (e.g., FISCO BCOS [35] and Hyperledger Besu [36]). SLARM is also compatible with Hyperledger Fabric [3] by running as a transaction dissemination network on Fabric's peer nodes.

Why permissioned blockchain? We focus on the permissioned blockchain setting instead of the permissionless setting for two reasons. First, the permissioned blockchain is decoupled from the cryptocurrency, and thus is more suitable for general enterprise applications with diverse SLA requirements on transactions' commit latency (Table V). Second, the permissionless blockchains already have cryptocurrency-based incentive schemes [7], motivating nodes to prioritize the dissemination and commitments of more valuable transactions (a kind of SLA scheme). The client can assign a transaction with a higher transaction fee to achieve lower commit latency. However, the permissioned blockchain lacks such a scheme.

B. SLARM's Protocol Overview

Fig. 1 shows a SLARM node's architecture. Same as existing P2P multicast protocols, nodes execute SLARM's protocol independently without coordination. Each node has two trusted modules (green) running in TEE (orange). The *scheduler* module prioritizes SLA-stringent transactions and adapts transactions' dissemination speeds. The *reliable multicast* module disseminates transactions and conducts a gap-filling task when it finds a gap in its received per-client SLA transaction sequence. The blockchain core module (e.g., the consensus protocol) and P2P network module (e.g., TCP/UDP) are outside the TEE and are not trusted.

❶ A client submits SLA transactions to a nearby node N_2 on the TLS link. A transaction is successfully submitted once the client receives an ACK signed by N_2's TEE (Section IV-A).

❷ After N_2's scheduler module receives the transactions, N_2 determines their SLA requirements according to the transaction types and the SLA specification on-chain (Section IV-B), then piggybacks each transaction with an *SLA metadata* indicating the remaining time to the transaction's SLA deadline (in short, *remaining deadline*, denoted as τ_d). N_2 initializes the remaining deadline as the transaction's SLA deadline, then disseminates the transaction $\{tx, \tau_d\}$ using Gossip with a specific fanout $\mathcal{F}$ (SLARM's forward dissemination). During the dissemination, each node updates transactions' remaining deadlines and prioritizes the propagation of transactions with short remaining deadlines (Section V-A). To prevent malicious nodes from altering the remaining deadlines of SLA transactions, the deadline updating is only involved in each node's TEE.

❸~❹ When a node N_6 detects a gap tx_2 in its received transactions (Section V-B), if the transactions require sequential execution in their SLA specification (Section IV-B), N_6 requests tx_2 from random peers in its partial view (Section II-A). In SLARM, non-SLA transactions do not involve gap-filling.

❺ The consensus nodes pack transactions with their remaining deadlines (signed by SLARM in TEE) into blocks and agree on which block to commit next. Nodes disseminate committed blocks to the entire network (SLARM's backward dissemination). Nodes adjust transactions' dissemination speeds by adapting the Gossip fanout $\mathcal{F}$ according to transactions' remaining deadlines in blocks (Section V-C).

By leveraging the integrity feature of TEE (Section IV-A), running SLARM's protocol in TEE can prevent the adversary (e.g., malicious nodes) from tampering with the SLA metadata (i.e., the remaining deadline τ_d), solving **Threat 1** in Fig. 1.

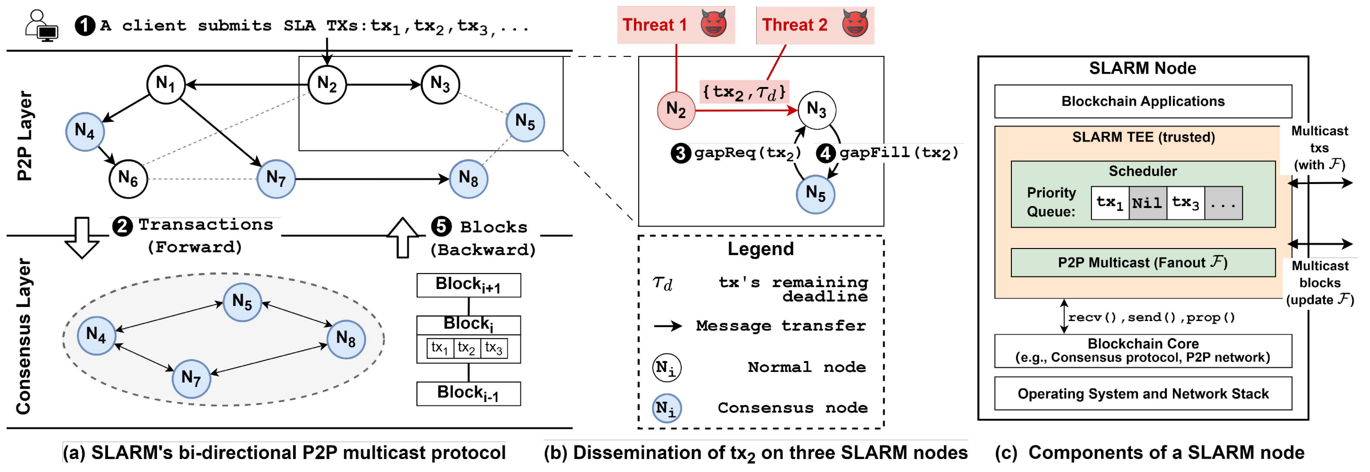


Fig. 1. SLARM's architecture in a zoom-in manner: (a) SLARM's bi-directional P2P multicast protocol; (b) Dissemination of $rmtx_2$ on three SLARM nodes; (c) Components of an SLARM node. The SLARM TEE is in orange.

However, even with SGX, SLARM still faces a security challenge. Specifically, SLARM is susceptible to the network deferring attacks (e.g., deferring a transaction during node-to-node dissemination, see **Threat 2** in Fig. 1). Recent work [37] shows that the adversary can perform **random deferring attacks** to random protocol messages or **targeted deferring attacks** to specific protocol messages (Section VI). For example, the adversary can selectively defer certain clients' transactions and stop their commit progress.

SLARM addresses deferring attacks (**Threat 2**) in two steps. First, each SLARM node detects deferring attacks by measuring the round-trip time (RTT) with peers using probe messages (Section VI-B2). Upon detecting deferring attacks, the node excludes the compromised peer from transaction dissemination. Second, SLARM disseminates all messages with the same traffic pattern (e.g., message size and fanout), preventing the adversaries from distinguishing SLARM's probe messages from other messages, thereby evading detection by selectively deferring only non-probe messages (Section VI-B3).

Overall, the highlights of SLARM stem from achieving a high SLA satisfaction rate in highly malicious permissioned blockchains, which we illustrate in two aspects. First, SLARM's bidirectional dissemination protocol adjusts its P2P dissemination speed according to the *trustworthy SLA feedback* from blockchain's consensus protocols (Fig. 1), making SLA transactions' dissemination speeds match their SLA deadlines. Existing P2P multicast protocols [10], [11], [14], [38] do not provide such trustworthy feedback. Second, SLARM is robust to the deferring attacks toward SLA transactions by conservatively adjusting SLA transactions' dissemination speeds. Existing P2P multicast protocols [10], [11], [14], [38], including Erlay [10], the latest notable P2P reliable multicast protocol tailored for blockchain security, are especially vulnerable to the deferring attacks.

IV. SLARM's SLA Definition

Before introducing SLARM's protocol (Section V), we first define SLARM's SLA latency (Section IV-A) and SLA specification (Section IV-B).

A. *SLARM's SLA Latency*

Fig. 2 shows the life cycle of a transaction tx in a permissioned blockchain (e.g., Quorum [2]) with six stages (**S**). **S1.** A client submits tx to the blockchain. **S2.** Nodes disseminate tx to the entire network using a P2P multicast protocol (Section II-A). **S3.** A consensus node (i.e., the ***proposer***) executes and proposes tx in a block. **S4.** Other consensus nodes agree to commit this block. **S5.** The block is relayed to all nodes. **S6.** All nodes commit the block to their local blockchains and execute tx. In this study, we take the first step to define the transaction's *SLA latency* in the blockchain domain.

Definition 1 (SLA latency): A transaction's **SLA latency** is the time interval between when a client submits the transaction and when the transaction is *proposed* (**S2+S3**).

Our SLA latency (**S2+S3**) for SLA assessment differs from the traditional commit latency (**S2+S3+S4**). This distinction arises because nodes execute transactions' time-dependent logic (e.g., assessing if a transaction is committed prior to its SLA deadline [9]) based on timestamps (i.e., transactions' *remaining deadlines*) *determined by the proposer in* **S3**. Specifically, for state consistency (blockchains' safety guarantee [6]), the blockchain must ensure that all nodes follow the same timestamp while executing transactions' time-dependent logic. By following the propose timestamp assigned by the block's proposer (**S3**), all blockchain nodes can produce consistent execution results. Note that the commit timestamp indicating when the block is committed by consensus nodes (**S4**) does not exist, as different consensus nodes generally commit the same block at different times [4], [5], [18].

Our SLA latency differs from the client perceived latency (**S2+S3+S4+S5**). In **S4** and **S5**, transactions are processed in blocks, making it difficult to adjust the commit or dissemination speed for individual transactions. As blocks contain the most stringent transactions, we prioritize the dissemination of blocks with the highest priority among all protocol messages.

Hence, the SLA latency ($\tau_d + \tau_p$) of a blockchain transaction is composed of two parts: the dissemination latency τ_d (**S2**) and the propose latency τ_p (**S3**). The propose latency τ_p is generally upper bounded by the **propose timeout** of permissioned blockchain's BFT consensus protocols. When the proposer is

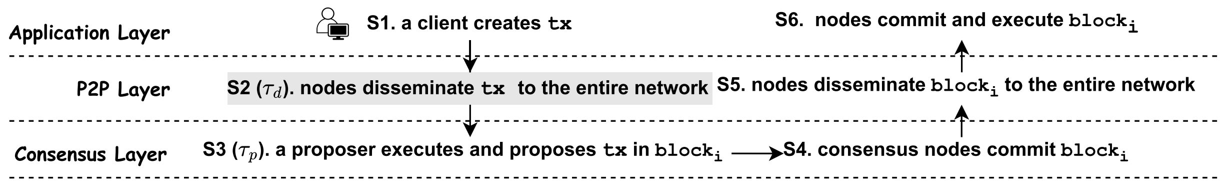

Application Layer
S1. a client creates tx
S6. nodes commit and execute block_i
P2P Layer
S2 (τ_d). nodes disseminate tx to the entire network
S5. nodes disseminate block_i to the entire network
Consensus Layer
S3 (τ_p). a proposer executes and proposes tx in block_i
S4. consensus nodes commit block_i

Fig. 2. Life cycle of a typical permissioned blockchain transaction. SLARM is an SLA-aware P2P multicast protocol for disseminating transactions (the *gray box*, see Section IV-A). **'S'** stands for *Stage*.

correct, it regularly proposes blocks at a fixed time interval regardless of whether there exist transactions to propose [18]. When the proposer is malicious and does not propose any blocks after the propose timeout, the consensus protocols allow other consensus nodes to propose blocks. Typical BFT consensus protocols invoke a failover mechanism to replace the current proposer with another one [5], [39], [40]. During failover, it is impossible to satisfy transactions' SLA deadlines because the blockchain cannot commit any transactions. In sum, we conservatively consider τ_p to be the propose timeout of blockchain's BFT consensus protocol, so τ_d (the **gray box** in Fig. 2) is the major bottleneck and improvement spot of a transaction's SLA latency, which is detailed in Section V.

B. SLARM's SLA Specification

Service-level agreements (SLAs) define the expected service quality between providers and customers. In this paper, a permissioned blockchain using SLARM as the P2P multicast protocol is the service provider, and a blockchain application with its clients are the customers. A blockchain application for SLARM contains two parts:

1) A *smart contract*, which is a stateful program invoked by client transactions [3]. Smart contract applications often desire the SLA requirement on transactions' SLA latency.
2) An *SLA specification*, which defines the SLA requirements (e.g., SLA latency) for different transactions in the application. The SLA specification is stored on the blockchain, and only the application's **administrator** can modify the SLA specification. SLARM determines the transaction SLA requirements in TEE based on the *SLA specification* when the client submits a transaction. For security concerns, a client is prohibited from determining the SLA of its transactions (e.g., malicious clients may assign stringent deadlines for non-stringent transactions, see Section VI-A).

Definition 2 (SLARM's SLA): A transaction's SLA is a 2-tuple SLA :$\{deadline, sequential\}$, where *deadline* is the desired SLA latency (Section IV-A) of the transaction and *sequential* indicates whether the transaction needs to be committed gap-free and sequentially according to its sequence number issued by the client (e.g., {32 s, yes}).

In SLARM, transactions with SLA requirements are SLA transactions; other transactions are non-SLA transactions. Note that the smart contract is stateful. The same client's transactions that invoke the smart contracts of one blockchain application must be sequentially committed by the consensus nodes without gaps. The client assigns the transactions that require sequential execution with continuous sequence numbers.

TABLE II
SUMMARY OF TERMS/SYMBOLS

Symbol	Definition
N	A blockchain participant node (§III-A).
$\mathcal{F}$	The Gossip fanout (§III-B).
tx	A transaction (§IV-A).
τ_d	SLA transaction's dissemination latency (§IV-A).
τ_p	SLA transaction's propose latency (§IV-A).
$rtt_{N,N'}$	The recent worst-case round-trip times (RTT) between the TEEs of nodes N and N' (§V-A).
tx_N^{wait}	The time a transaction spends in node N's TEE (§V-A).
n	The total number of all nodes (§V-C).
R_n	The number of rounds for Gossip to disseminate tx to all the n nodes in a network (§VI-C2).

A transaction's actual SLA latency is $\tau_d + \tau_p$ (defined in Section IV-A). In an uncongested network, with the default fanout (the number of selected peers to send a message in Gossip, by default 50 [41]), suppose the mean dissemination latency of a transaction is $\overline{\tau_d}$, and the mean propose latency of a transaction is $\overline{\tau_p}$ (upper bounded by the propose timeout, see Section IV-A). SLARM by default sets the SLA as $\{c \times \overline{\tau_d} + \overline{\tau_p}, yes\}$ for all SLA transactions, where c is a configurable constant. For each SLA transaction, SLARM conservatively considers τ_p as the propose timeout, and strives to meet the transaction's SLA deadline by making $\tau_d < c \times \overline{\tau_d}$.

In SLARM, the SLA satisfaction rate p is defined as the portion of committed SLA transactions meeting their SLA deadlines. Suppose clients submit SLA transactions with deadline $c \times \overline{\tau_d} + \overline{\tau_p}$, and the number of submitted transactions per second does not exceed the maximum commit 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 $c = 2$), which is theoretically proved in Section VI-C2 and evaluated in Section VII-A1.

V. SLARM's Bidirectional Multicast

SLARM strives to make each transaction's actual SLA latency less than its SLA deadline. As discussed in Section IV-A, a transaction's SLA latency consists of the dissemination latency τ_d and the propose latency τ_p (we summarize all symbols in Table II). As τ_p is smaller than the block propose timeout (a consensus protocol parameter, orthogonal to SLARM) in normal cases, upon receiving a transaction tx from the client, SLARM deducts τ_p from tx's SLA deadline, and focuses on making τ_d meet the transaction's remaining SLA deadline.

To this end, SLARM's *bidirectional* reliable multicast protocol disseminates each transaction twice. First, in the *forward dissemination* (Sections V-A and V-B), SLARM disseminates an SLA transaction throughout the network using Gossip with the default fanout $\mathcal{F}$ and traces the transaction's remaining deadline to prioritize SLA-stringent transactions. Second, in the *backward dissemination* (Section V-C), SLARM disseminates transactions in committed blocks and adapts $\mathcal{F}$ of SLA transactions in subsequent dissemination according to transactions' remaining deadlines in blocks (i.e., the *SLA feedback*).

A. Forward Dissemination

In the forward dissemination, SLARM prioritizes the dissemination of SLA-stringent transactions according to their remaining deadlines. Each SLARM node N maintains (subtracts) transactions' remaining deadlines with two trustworthy (conservative) time variables: $rtt_{N,N'}$ and tx_N^{wait}. Specifically, $rtt_{N,N'}$ is the recent worst-case round-trip times (RTT) between node N and N' (N's peer) outside nodes' TEEs. tx_N^{wait} is the time a transaction spends in N's TEE. We present a trustworthy mechanism (Section VI-B2) to conservatively record the round-trip time ($rtt_{N,N'}$) between N and N' as tx's elapsed time in the transmission from N' to N. Section VI will present how SLARM makes these two variables **conservative:** the subtracted elapsed time from a transaction's remaining deadline is larger than the actual elapsed time on the transaction's dissemination path. To ease discussion, this section assumes these two variables are conservative, and we will discuss the conservative deadline mechanism in Section VI.

When node N receives a new transaction tx from a peer N', N subtracts $rtt_{N,N'}$, the conservative elapsed time in transferring tx from N' to N, from tx's remaining deadline τ_d. Since the Internet latency among two nodes is asymmetric due to IP routing [42], SLARM uses the RTT value instead of its half as the transaction's conservative one-hop transfer time among nodes. N prioritizes the dissemination of SLA-stringent transactions with a local priority queue (`TxQueue` in Fig. 1(c)). Before inserting tx into `TxQueue`, SLARM's scheduler module updates the τ_d of all transactions in `TxQueue` by subtracting their elapsed time tx_N^{wait} on N since the last τ_d update from their τ_d. Then, the scheduler inserts tx into `TxQueue` according to tx's τ_d, smaller τ_d indicates higher priority.

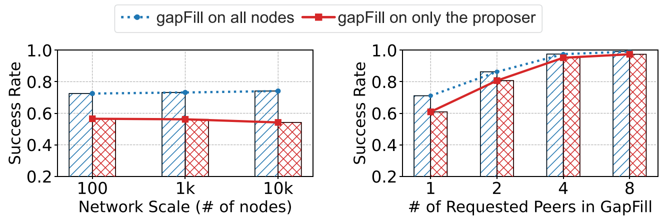
SLARM sends the most stringent transactions in `TxQueue` to peers every certain time interval T. To send a transaction, node N's reliable multicast module retrieves an SLA transaction tx from the head of `TxQueue`. N then subtracts tx's time cost tx_N^{wait} on N since the last τ_d update from tx's τ_d. Following this, N encrypts and sends tx to a randomly selected peer.

To prevent non-SLA transactions from being infinitely obstructed by SLA transactions, SLARM assigns a large deadline (e.g., 128 s in Fig. 5, where the SLA deadline for SLA transactions is 32 s) to non-SLA transactions, thereby ensuring eventual commitment.

B. Gap-Filling in the Forward Dissemination

In stateful blockchain applications, SLA transactions must be committed sequentially and without gaps, based on the client-issued sequence numbers (Section IV-B). Two gap-filling approaches are possible: (1) performing gap-filling on all nodes, or (2) performing gap-filling on only the proposer (i.e., the consensus leader, see Section IV-B). We evaluated the gap-filling success rate for both approaches.

As shown in Fig. 3, performing gap-filling on only the proposer resulted in a 15.3% lower success rate than performing



(a) Success rates in different network scales. Each gap-filling requests missing transactions from only *one* peer.

(b) Success rates under varying number of requested peers. Each gap-filling requests *one* to *eight* peers.

Fig. 3. The success rates of two *gap-fill* strategies in SLARM. (a) Perform gap-fill on all nodes; (b) Perform gap-fill on only the proposer.

gap-filling on all nodes (Fig. 3(a)). However, the success rate improved significantly as the number of request peers increased (Fig. 3(b)). When requesting the missing transaction from four peers, the above two approaches achieved similar success rates. This improvement is attributed to the random selection of peers from the entire network [43], which ensures that the probability of none of the proposer's peers holding the missing transaction is extremely low.

Therefore, SLARM performs gap-filling tasks for SLA transactions on only the proposer during forward dissemination (Section V-A). The proposer detects gaps according to the sequence numbers of transactions from the same client. Upon detecting a gap, the proposer requests the missing transactions from a randomly selected subset (by default, four peers) of its peers.

C. Backward Dissemination

In the backward dissemination, SLARM disseminates transactions in committed blocks. To propose a block, the proposer N retrieves a batch of transactions piggybacked with their remaining deadlines from SLARM TEE. The transactions' remaining deadlines serve as the trustworthy feedback for adjusting transactions' dissemination speed.

SLARM's initial Gossip fanout is $\mathcal{F} = ln(n)$ [12], where n is the number of all nodes (explicitly identified in a permissioned blockchain [2]), and $ln(n)$ is the theoretical fanout threshold for Gossip to disseminate a transaction to all nodes with high probability [44].

Upon receiving a new committed block, a SLARM node adjusts the fanout of SLA transactions by calculating the *moving average remaining deadline* of transactions in recently committed blocks to avoid oscillations caused by sudden traffic spikes or deferring attacks. Suppose the average remaining deadline of SLA_1 transactions is negative, indicating that SLARM cannot disseminate SLA_1 transactions before the deadline with the current fanout $\mathcal{F}_{SLA_1}$. Each node sets $\mathcal{F}_{SLA_1}$ as $\mathcal{F}_{SLA_1} + 1$, until $\mathcal{F}_{SLA_1}$ equals the maximum peer number (by default, 50 peers [41]). If the average remaining deadline is positive and larger than a specific threshold (by default, half of the SLA deadline), nodes set $\mathcal{F}_{SLA_1}$ to $\mathcal{F}_{SLA_1} - 1$, until $\mathcal{F}_{SLA_1} = ln(n)$. For high consensus performance, the blockchain can first disseminate the block header then disseminate block transactions in parallel. In such cases, SLARM adjusts the fanout for SLA transactions after receiving the block transactions.

When the network experiences traffic spikes, SLARM cannot disseminate transactions in a timely way, resulting in negative remaining deadlines for SLA transactions in committed blocks; increasing the Gossip fanout in such cases can worsen the network congestion. To address this, the proposer should include the size of its TxQueue in each block. When proposing a block, the proposer retrieves a batch of transactions along with the current TxQueue size from SLARM. The TxQueue size, signed by SLARM in TEE, indicates the number of pending transactions and serves as a trusted indicator of the current congestion level. If a block's TxQueue size is large (by default, the block size), SLARM will omit the block when calculating the moving average remaining deadlines.

SLARM's protocol operates within each node's TEE. The updating (i.e., subtraction) of the SLA metadata τ_d during forward dissemination and the adjustment of the Gossip fanout during backward dissemination are performed exclusively in each node's TEE. Additionally, the transactions' remaining deadlines and the proposer's TxQueue size are managed and signed within TEE. This prevents malicious nodes from altering SLARM's protocol and its messages.

VI. SLARM's Security Design and Analysis

A. Tackling Client's SLA Unfairness Attacks

A malicious client may attempt to assign its transactions with arbitrarily short SLA deadlines. This can make their transactions be committed faster than other clients' transactions, leading to SLA unfairness.

To tackle this attack, SLARM determines transactions' SLAs in nodes' TEEs rather than on clients. Each SLARM application, as explained in Section IV-B, incorporates an SLA specification which defines the SLA deadlines of various types of transactions (e.g., real-time and non-real-time payments in trading [6]). Only the application's administrator can create and update this SLA specification. Following the SLA specification, when an SLA transaction tx is received, N_i determines tx's SLA deadline within N_i's TEE, preventing malicious clients from seizing unfair commit precedence by assigning unreasonably short SLA deadlines to their transactions.

B. Tackling Node's Deferring Attacks

SLARM runs its SLA mechanism in nodes' TEE (Section V), preventing the adversary from tampering with SLARM's protocol and messages. However, even with SGX, the adversary can still conduct attacks outside the TEEs by dropping or delaying SLARM's protocol messages. Specifically, during messages' transmissions among nodes on the Internet, message dropping and delaying are indistinguishable, and both cause the recipient node to fail to receive the message (e.g., an SLA transaction) in a timely way, thereby violating the transaction's SLA deadline. Thus, we term message dropping and delaying as "deferring attacks" and tackle them together. The deferring attacks can be either random or targeted. The *random deferring attacks* randomly defer SLARM's protocol messages, while the *targeted deferring attacks* selectively defer specific protocol messages (e.g., SLA-stringent transactions).

To combat deferring attacks on malicious nodes, we propose a novel *conservative deadline* mechanism, containing three parts. First, we introduce a *millisecond-level trusted timer* based on SGX to measure transactions' elapsed time during dissemination (Section VI-B1). Second, based on the trusted timer, we propose a *trustworthy RTT protocol* to capture the random deferring attacks by measuring the RTT among nodes (Section VI-B2). Third, we present a *uniform message dissemination protocol* to eliminate the targeted deferring attacks by disseminating messages in the same traffic pattern. (Section VI-B3).

1) Fine-Grained Trusted Timer: To maintain SLA transactions' remaining deadlines and defend against the deferring attacks, SLARM relies on a trusted timer to measure transactions' elapsed time on nodes and during node-to-node dissemination.

Many fine-grained trusted timers have been proposed [33], [45] to tackle this problem. In this study, we use the notable millisecond-level trusted timer of TimeSeal [45] to measure each transaction's elapsed time. TimeSeal divides the coarse-grained SGX timestamps into fine-grained sub-millisecond resolution sub-timestamps by counting the CPU cycles between SGX timestamps in SLARM's SGX enclave. TimeSeal securely bounds the relative time accuracy to millisecond, even under a malicious OS. SLARM is compatible with other trusted timers [31], [46]. The fine-grained trusted timer's design is orthogonal to SLARM, so we eliminate the timer's detailed security analysis [45] and focus on SLARM's own security design for handling the random and targeted deferring attacks.

2) Capturing Random Deferring Attacks: In the random deferring attacks, an adversary can randomly defer SLARM's protocol messages (e.g., the SLA transactions) to reduce SLARM's SLA satisfaction rates (evaluated in Section VII-B2). We design a *trustworthy RTT mechanism* to capture this attack by calculating the trustworthy per-peer RTT to detect the extra delay imposed by the attack.

For a random time interval, each SLARM node N sends an RTT ping message to a random peer N' piggybacked with the recent *highest* RTT value measured between N and N'. On receiving the returned RTT pong message from N', node N can calculate the RTT value. Node N selects the *highest* RTT value measured by either N or N' as the latest RTT value $rtt_{N,N'}$. Under the random deferring attacks, the RTT messages are also deferred. If the observed RTT with a peer experiences a significant surge, the peer may be under attack, and SLARM nodes will replace the peer with another one.

To ensure a high probability for detecting the attacks, for a certain time interval T, (SLARM's message sending interval, see Section VI-B3), each node sends f (SLARM's message sending fanout, see Section VI-B3) RTT ping messages toward random peers with probability p. If the adversary randomly defers m messages from the node, the probability for SLARM to detect the deferring attack is $1-(1-p)^m$. We set $p=10\%$ to maintain a low communication overhead. Following the deferral of 50 messages, SLARM exhibits an overwhelmingly high probability of 99.9% in detecting the attack, confirmed in our evaluation (Section VII-B2).

Overall, SLARM's trustworthy RTT mechanism captures the random deferring attacks with a high probability and makes the deferred SLA transactions more stringent, as the deadline for each SLA transaction subtracts a statistically worst-case RTT value. Although some transactions' SLA deadlines are made negative, they are still conservative.

3) Eliminating Targeted Deferring Attacks: The aforementioned *trustworthy RTT mechanism* (Section VI-B2) can detect the random deferring attacks. However, the adversary can elude SLARM's detection by selectively deferring specific protocol messages (e.g., SLA transactions), while leaving SLARM's RTT messages unaffected. With this targeted deferring attack, the adversary can easily make SLA transactions fail to meet their deadlines without being detected by SLARM.

Specifically, although the adversary can only observe encrypted P2P messages (Section V-A), a smart adversary can still distinguish different messages by analyzing the distinct traffic patterns (**P1** and **P2**) associated with different P2P messages.

- Message size (**P1**): The size of SLARM's encrypted P2P messages in the network varies. For instance, the RTT messages are no larger than 10 bytes, whereas SLA transactions can reach 250 bytes.
- Message fanout (**P2**): SLARM disseminates an SLA-stringent transaction with a large fanout $\mathcal{F}$ to many peers simultaneously (Section V-A), while disseminating an RTT message with a small fanout.

To thwart the adversary's ability to identify different messages based on their distinct traffic patterns, we present a *uniform message dissemination protocol* to disseminate all P2P messages with the same traffic pattern. We achieve this goal in two steps. First, to address **P1**, SLARM uniformly expands the size of all P2P messages to the same size with random dummy payloads before sending the messages. By doing so, an adversary cannot distinguish different messages according to the message sizes.

Second, to address **P2**, SLARM shuffles the dissemination of all messages. Each SLARM node maintains one message priority queue MsgQueue for all SLARM's P2P messages, including transactions, RTT messages (Section VI-B2), and gapFill messages (Section V-B). For a transaction with fanout $\mathcal{F}$ (Section V), a SLARM node inserts the transaction into MsgQueue for $\mathcal{F}$ times (assigned with $\mathcal{F}$ random destination peers). Meanwhile, the node also inserts other protocol messages, including RTT and gapFill messages, to the tail of MsgQueue.

To dispatch the messages in MsgQueue, at a predefined time interval T, the node retrieves f messages from MsgQueue and transmits them to corresponding peers. For a message with fanout $\mathcal{F}$, instead of simultaneously disseminating the message to $\mathcal{F}$ peers, SLARM sends the messages to f peers at a time and repeats the process for $\lceil \mathcal{F}/f \rceil$ times. This prevents the adversary from inferring the message fanouts based on the number of messages a node is sending simultaneously. SLARM selects an appropriate T and f to ensure that the peak per-peer transaction dissemination speed (by default, 300 tx/s) on each node is larger than the peak commit throughput of the blockchain's consensus protocol (no larger than 300tx/s in our evaluation, see Section VII-A1).

Overall, by uniformly disseminating all SLARM's protocol messages with the same traffic pattern, the adversary cannot distinguish different protocol messages and perform targeted deferring attacks.

C. Discussion

1) Discussion of SLARM's Security Design: SLARM's *conservative deadline mechanism* conservatively measures the two time variables (tx_N^{wait} and $rtt_{N,N'}$, defined in Section V-A) to track SLA transactions' remaining deadlines. Specifically, with SLARM's fine-grained trusted timer (Section VI-B1), SLARM can precisely measure an SLA transaction tx's local elapsed time tx_N^{wait} in each node's TEE. With SLARM's trustworthy RTT protocol (Section VI-B2) and uniform message dissemination protocol (Section VI-B3), SLARM can conservatively measure tx's node-to-node transfer latency $rtt_{N,N'}$. Therefore, the two critical variables $rtt_{N,N'}$ and tx_N^{wait} are both *conservative:* if SLARM infers that an SLA transaction meets the SLA deadline, the transaction actually meets it with overwhelmingly high probability (Section VI-C2), even though the transaction's dissemination is deferred by some malicious nodes on some hops.

SLARM's *conservative deadline mechanism* plays a pivotal role in realizing SLARM's bidirectional multicast protocol (Section V). In the forward dissemination (Section V-A), the conservative remaining deadlines enable nodes to prioritize SLA-stringent transactions. In the backward dissemination (Section V-C), the conservative remaining deadlines are used as the trustworthy feedback for adjusting the transactions' dissemination speed. These two unique strengths enable SLARM to attain a high SLA satisfaction rate even under tough scenarios (Section VII-B), making SLARM suitable for blockchains.

2) Discussion of SLARM's SLA Enforcement Probability: Now we analyze SLARM's probability in enforcing deadlines of SLA transactions. Recall that SLARM's default SLA deadline is $c \times \overline{\tau_d} + \overline{\tau_p}$, where $c \times \overline{\tau_d}$ is the expected dissemination latency, and $c = 2$ by default (Section IV-B). We now investigate the probability of an SLA transaction being disseminated to the entire network before $2 \times \overline{\tau_d}$.

In SLARM, a transaction's dissemination latency is determined by the Gossip protocol [38]. The Gossip model in which we analyze SLARM is a set of independent nodes communicating by pushing messages to the node's peers in rounds. Let n be the total number of nodes, then the expected number $\mathbb{E}(R_n)$ of rounds for all the n nodes to receive the transaction is [47]: $\mathbb{E}(R_n) = log_{\mathcal{F}}(n) + ln(n) + O(1) = O(log_{\mathcal{F}}(n))$.

As proved in existing studies [48], the variance of the rounds is $Var(R_n) \approx \pi^2/3$. According to Chebyshev's inequality $P(|X - \mathbb{E}(X)| \geq c) \leq Var(X)/c^2$ [48], We have $P(|R_n - \mathbb{E}(R_n)| \geq b) \leq \frac{\pi^2}{3b^2}$. If we set the default dissemination rounds (Section IV-B) for SLA transactions as $R_n = 2 \times \mathbb{E}(R_n)$ (i.e., $c = 2$) and set $b = \mathbb{E}(R_n)$, at least 98.7% of all SLA transactions can be disseminated to the entire network within $2 \times \mathbb{E}(R_n)$ rounds by theory.

Given the distribution of the per-round relaying latency (including the transfer latency among nodes and the waiting latency on nodes), we could obtain the distribution of the end-to-end dissemination latency τ_d of transactions [38]. As existing studies [38], we use the normal distribution to model the per-round relaying latency over time. In Gossip, each intermediary node can be viewed as an independent process. The end-to-end dissemination latency is the sum of the relaying latencies on all intermediary nodes. Due to the fact that the sum of independent normally distributed random variables is still normal [38], the end-to-end dissemination latency (consisting of R_n rounds) also satisfies a normal distribution. Its mean equals $R_n \times \mu$, and variance equals $\sqrt{R_n} \times \sigma$, where μ and σ are the mean and variance of the per-round relaying latency, respectively [38]. Therefore, we would expect the variance of the end-to-end dissemination latency to grow slowly, as the square root of the log of the network size. Therefore, in an uncongested network where σ is small, the variance of τ_d is also small, and the random variable τ_d highly concentrates around its mean value. Our evaluation results also show that the actual SLA satisfaction rate (96.0%) matches our theoretical analysis (98.7%).

Same as the previous study [38], the above analysis assumes that the relaying latency distribution is reasonably tight with a small variance. However, on the Internet, the relaying latency may be unstable in some scenarios. Compensating for this is SLARM's bidirectional multicast protocol in Sections V and VI, which is not treated in our theoretical analysis, but in practice would enable nodes to detect this relaying latency

TABLE III
SLARM'S API EXPOSED TO THE BLOCKCHAIN

SLARM's API	Description
DISSEMINATION API (invoked by normal nodes)	
❶ **recv**(MSG)	Forward MSG to SLARM's TEE.
❷ {MSG, N_i} **send**()	Fetch a message from SLARM's TEE for sending to peer N_i.
PROPOSER API (invoked by only consensus proposers)	
❸ {BLK} **prop**(num)	Fetch a batch of *num* transactions for block consensus from SLARM's TEE.

TABLE IV
DEFAULT EVALUATION SETTINGS

Config	Setting
# Nodes per-machine/Total	50/5k
# Consensus nodes	10 (PoA [18], [34])/N.A.(PoET [19])
# Node peers	50
Consensus protocol	Clique-PoA [18]
Block propose interval	1s
Avg RTT	200ms
SLA	{32s, Yes}
Workload	Online trading
Bandwidth limitation	20Mbps [13]

'#' means number. n.a. Means PoET has no dedicated consensus nodes (sectionVII).

fluctuation and adapt the dissemination speeds of transactions conservatively, which "hides" the resulting variations in the end-to-end dissemination latency. Our experimental data, reported in Section Section VII-B2, is for a setting in which the RTT among nodes is quite unstable, and we indeed observe very stable SLA satisfaction rates for the default $2 \times \overline{\tau_d} + \tau_p$ SLA setting (Section IV-B).

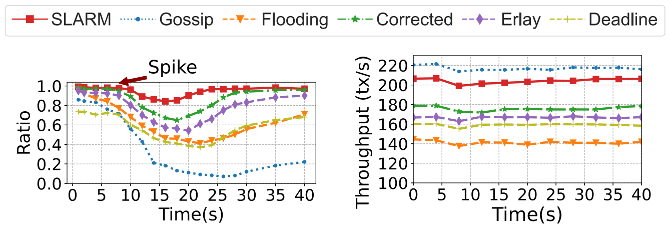
3) Integration With Existing Blockchains: SLARM can easily integrate with existing permissioned blockchains. Blockchain nodes run SLARM in TEE and interact with it via APIs in Table III. On receiving a message, the node forwards the message to SLARM via the `recv( Msg )` API. To send a message, the node invokes the `{ Msg ,` N_i`} send()` API, where N_i is the receiver node. A proposer can obtain a batch of transactions to propose via the `prop(num)` API. SLARM treats consensus protocol as a blackbox, enabling compatibility with different consensus protocols.

These three APIs suffice for SLARM to provide reliable and efficient message dissemination. If SLARM detects a message intended for other nodes, the receiver considers the sender node specified in the message malicious and removes the sender from its peers. A node that delays message dissemination is identified through SLARM's trustworthy RTT mechanism (Section VI-B2). A node that does not retrieve messages from SLARM in a timely manner can also be detected by SLARM and exit the network.

VII. Evaluation

We evaluated SLARM on Azure [17] with 100 VM instances. Each instance has 48 vCPUs, 384 GB RAM, and 24 Gbps NIC. Same as existing studies [13], to simulate the Internet, we set each SLARM node's bandwidth as 20 Mbps and the node-to-node RTT as 200 ms. We ran 50 nodes on each VM instance. Each node runs in a docker container. We summarize the default evaluation settings in Table IV.

We evaluated SLARM with three consensus protocols: PoET [19], Clique [18], and IBFT [34]. PoET is a notable



(a) SLA guarantee (b) Throughput

Fig. 4. SLA guarantee for SLA transactions and throughput for all transactions in the online trading application (evaluation settings are in Section VII): (a) SLA satisfaction ratio for SLA transactions; (b) Throughput for all transactions. At 0 s, all systems are at peak throughput; at 8 s, a spike of 200 tx/s SLA transactions lasts for 5s.

blockchain consensus protocol based on SGX. PoET has no dedicated consensus nodes and offers each node an equal chance of proposing blocks in random propose intervals. Clique and IBFT are popular PoA consensus protocols deployed by Ethereum private network [53] and Quorum [2], respectively. We ran Clique as the default consensus protocol. We deployed ten consensus nodes and set the block interval as 1 s, same as existing PoA permissioned blockchains [2].

We compared SLARM with five unidirectional P2P multicast protocols: Gossip [12], Flooding [15], Erlay [10], Corrected Gossip [11], and Deadline Gossip [14]. Gossip and Flooding are default P2P multicast protocols of permissioned blockchains [1], [7]. Erlay, Corrected Gossip, and Deadline Gossip are reliable multicast protocols. Erlay is a notable reliable multicast protocol designed for blockchains [1]. Corrected Gossip (the *Corrected* in figures) is the most recent reliable multicast protocol. Deadline Gossip (the *Deadline* in figures) is a deadline-aware reliable multicast protocol. For a fair comparison, all five protocols were run with a static transaction priority: SLA transactions were always prioritized over non-SLA transactions during dissemination. Same as existing permissioned blockchains [2], [36], we ran Gossip as the default P2P multicast protocol.

We evaluated five typical blockchain applications with different portions of SLA transactions (Table V). We also generated transaction spikes to emulate real-world workloads. Our default workload was *online trading*, one of the most prevalent blockchain applications [6]. We set the transaction size as 250 bytes. The transaction sizes for baseline systems are either equal to or smaller than that of SLARM.

We set the SLA requirements for all SLA transactions as {32 s, Yes}. We define *SLA satisfaction rate* (*Ratio* in figures) as the percentage of SLA transactions with positive remaining deadlines committed in all blocks. We report throughput as *tx/s* for the sum of SLA and non-SLA transactions. We focus on the following questions:

VII-A How does SLARM meet the transaction's SLA and what is the system's throughput (tput) under traffic spikes?

VII-B How robust is SLARM to complex network conditions?

A. SLA and Performance

1) End-to-End SLA and Performance: Fig. 4(a) shows all systems' SLA satisfaction rates for SLA transactions. Fig. 4(b) shows the throughput of all systems. From 0 s to 8 s, we made the consensus protocol reach its peak throughput. At 8 s, we

TABLE V
SLARM'S EVALUATED BLOCKCHAIN APPLICATIONS

Applications	**SLA Txs**	**Non-SLA Txs**	**Traffic Spike (Extra Txs)**	**Tx Submission Rate**
App 1: Mobile carriers [49]	Pre-paid users (30%)	Post-paid users (70%)	×	Peak commit throughput
App 2: Voting [50]	Election management (30%)	Cast votes (70%)	×	1000 tx/s
App 3: CDN accounting [51]	Settlements (30%)	Usage data (70%)	200 tx/s SLA txs lasts 5s	Peak commit throughput
App 4: Disease control [52]	High-risk events (50%)	Low-risk events (50%)	×	Peak commit throughput
App 5: Online trading [6]	Real-time trading (70%)	Non-real-time trading (30%)	200 tx/s SLA txs lasts 5s	Peak commit throughput

Parameters are from the cited papers in the table. SLA transactions are written in smart contracts. The peak commit throughput is no more than 300 tx/s in our evaluation (figure 7).

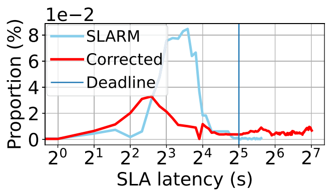


Fig. 5. SLA violations (x-axis is log-scaled).

TABLE VI
SLARM MICRO-EVENTS BEFORE THE SPIKE LAUNCHED AT THE 8S OF FIG

System	Avg Propose Latency(s) (τ_p)	Avg P2P Latency(s) (τ_d)	Gap-filling num	Gap-filling cost (s)	SLA Rate
Gossip	4.9	8.7	N/A	N/A	68.1%
Flooding	5.1	6.8	N/A	N/A	69.5%
Corrected	5.1	12.1	20	2.6	98.7%
Erlay	5.0	11.3	21	4.2	85.4%
Deadline	5.2	13.2	19	4.6	65.1%
SLARM	4.9	9.9	18	1.1	98.9%

Miss Rate means the ratio of missed SLA transactions in gaps on consensus nodes.

TABLE VII
SLARM MICRO-EVENTS AT THE 16S (LOWEST SLA SATISFACTION RATE) OF FIG. 4(A)

System	Avg Propose Latency(s) (τ_p)	Avg P2P Latency(s) (τ_d)	Gap-filling num	Gap-filling cost (s)	SLA Rate
Gossip	4.8	53.1	N/A	N/A	19.5%
Flooding	5.2	78.7	N/A	N/A	41.2%
Corrected	5.1	75.3	197	31.4	61.7%
Erlay	4.8	88.3	198	39.8	48.8%
Deadline	5.1	63.7	200	41.6	45.6%
SLARM	4.9	26.8	187	6.6	92.6%

generated a traffic spike of additional SLA transactions for five seconds. In Fig. 4(a), SLARM's SLA satisfaction rate was the highest and dropped from 98.1% to 82.1% at about 15 s then quickly recovered. Other systems' SLA satisfaction rates all dropped significantly and recovered much slower than SLARM. Gossip achieved the lowest SLA satisfaction rate because it caused many gaps in nodes' received transactions. The SLA satisfaction rate for all systems dropped at the 8 s because all transactions (including the SLA transactions) were waiting in proposers' queues due to the spike. Then, SLARM's SLA satisfaction rate grew more quickly because SLARM's bidirectional multicast protocol adaptively prioritizes more stringent SLA transactions in P2P nodes' priority queues (with the conservative remaining deadlines) to meet their SLA deadlines (Section V-A).

In Fig. 4(b), Gossip achieved the highest throughput. SLARM incurred an 11.3% drop in transactions' throughput compared to Gossip. We also evaluated the overhead separately for SLARM's forward and backward dissemination. The results show that SLARM's overhead compared to Gossip was mainly attributed to forward dissemination, whereas the overhead of backward dissemination was negligible. This is because committed blocks are disseminated backward in plaintext (no encryption/decryption overhead), and the Gossip fanout adjustments happen asynchronously when nodes are less busy. The overhead from SLARM's conservative deadline mechanism (Section VI) was only 3.3%, mainly due to message encryption/decryption.

Other reliable multicast protocols and Flooding's performance were much lower than SLARM. SLARM's high throughput is due to SLARM's lightweight gap-filling upon Gossip with less communication and computation overhead compared to existing reliable multicast protocols (Section V-B). Overall, SLARM can achieve a high SLA satisfaction rate while still maintaining a high throughput, both of which are highly desirable for blockchain applications.

2) Comparison of SLA Violations: We evaluated how SLA transactions could violate their deadlines by comparing SLARM with Corrected Gossip, the P2P multicast protocol achieved the highest SLA satisfaction rate among all baselines (Fig. 4(a)). As shown in Fig. 5, in SLARM, the SLA latency of SLA transactions that violated their deadlines was about 48 s in the worst case. While in Corrected Gossip, the violated SLA transactions took a significantly longer time (about 129 s) to be committed. This indicates that SLARM's bidirectional multicast protocol not only has a higher SLA satisfaction rate for SLA transactions, but also ensures the SLA transactions that have already violated their deadlines to be committed as fast as possible.

3) Breakdown and Micro-Events: To investigate the reasons for SLARM's high SLA satisfaction rate and its throughput overhead, we collected all systems' micro-events at the 7 s of Fig. 4(a) in Table VI and the same events at the 16 s (the lowest SLA satisfaction rate for SLARM) in Table VII. In Table VI, SLARM's mean SLA latency ($\tau_d + \tau_p$) was less than 32s, which explains SLARM's high satisfaction rate. We also broke down a transaction's latency on a single SLARM node in terms of TEE latency, scheduling latency, and queueing latency. The results show that a transaction's latency on a node is dominated by the queueing latency (0.4 s); the latency caused by TEE and scheduling was about 0.2 s and 0.1 s respectively.

The P2P latency for Flooding was low due to its use of a large fanout to disseminate messages. The P2P latency for Corrected Gossip and Erlay was high, because they incurred high overhead for fixing the missed transactions (e.g., set reconciliation in Erlay [10]). Additionally, Deadline Gossip incurred significant overhead in large networks due to maintaining a global view of all nodes.

In Table VII, SLARM incurred a higher burden on disseminating SLA transactions, so its per-transaction τ_d increased leading to a decreased SLA satisfaction rate. SLARM's SLA satisfaction rate was still much better than the other three reliable multicast

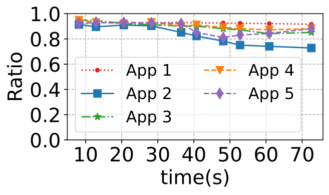


Fig. 6. SLA guarantees for five applications (Table V).

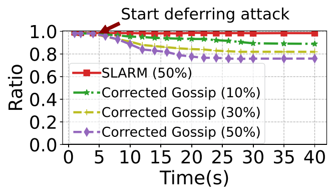


Fig. 8. Portion of P2P nodes under deferring attack.

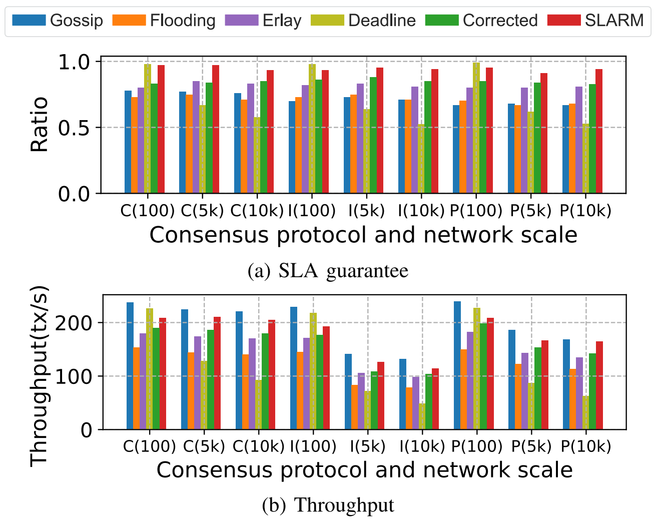


Fig. 7. Performance of Clique [18], IBFT [34], and PoET [19] with 100, 5 k, and 10 k P2P nodes. C, I, and P denote settings running Clique, IBFT, or PoET consensus protocols, respectively. (a) SLA satisfaction ratio of SLA transactions; (b) Throughput of all transactions. C, I, and P denote settings running Clique, IBFT, or PoET consensus protocols, respectively.

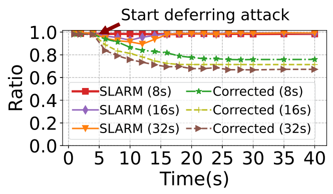


Fig. 9. Extra latency added by the deferring attack.

protocols under traffic spikes because (1) though only performing the light-weight gap-filling task (Section V-A), SLARM achieved a surprisingly high fix rate ($> 90\%$) for lost SLA transactions; (2) SLARM's bidirectional multicast adaptively prioritizes the dissemination of SLA-stringent transactions over non-stringent transactions.

4) SLARM's SLA on Diverse Applications: We deployed the five applications in Table V on SLARM individually and measured their SLA satisfaction rates according to their own application settings (e.g., portions of SLA transactions and spikes). As shown in Fig. 6, SLARM can achieve high SLA satisfaction rates for different percentages of SLA transactions and can recover fast under traffic spikes (CDN accounting and online trading). The SLA satisfaction rate decline in the voting application is because the number of SLA transactions submitted by clients per second is always much higher than the system's peak throughput. Overall, SLARM is generic for different applications with diverse transaction patterns (e.g., ratios of SLA transactions).

5) Scalability: We further evaluated SLARM at different network scales (100 & 5 k & 10 k nodes) with three permissioned blockchain's consensus protocols, shown in Fig. 7. We ran ten consensus nodes for PoA consensus protocols (Clique [18] and IBFT [34]) in all network scales and set the default SLA deadline proportionally to the network scale. Overall, SLARM achieved a reasonable SLA satisfaction rate and overhead on throughput (Fig. 7(b)). Notably, the global view P2P multicast protocol, Deadline Gossip, attained the highest SLA satisfaction rate in the small network (100 nodes) but performed poorly at large scales due to the high cost of maintaining a global view. Given that typical applications (Table V) generally involve no more than 5 k P2P nodes [51], we consider SLARM's scalability sufficient for practical deployments.

B. Performance Under Complex Network Conditions

1) Cross-Region Performance: We evaluated SLARM's performance in the cross-region (i.e., multi-datacenter) setting. We grouped nodes into four regions, reflecting common multi-region deployments. We set cross-region RTTs to 200ms [3], [39], and set in-region RTTs to 2ms [39]. We set cross-region bandwidth to 100Mbps [3], [39], and set in-region bandwidth to 10Gbps [6].

Under the cross-region setting, Gossip experienced a modest throughput drop, and SLARM's throughput declined more. Specifically, SLARM's throughput was 17.7% lower than Gossip—whereas in the default setting, the gap was only 8.3% (Fig. 4(b)). This additional overhead was due to SLARM's `gap-filling` (Section V-B) and `RTT` (Section VI-B2) messages, which increased the cross-region bandwidth usage. Despite this, SLARM maintained high SLA satisfaction rates, similar to the default setting, thanks to our bidirectional multicast protocol (Section V) that dynamically adjusted transaction's dissemination speed based on the actual network latency.

2) Robustness to Latency Variants: We injected latency variants to the network by randomly increasing the RTT among nodes (i.e., deferring attack). Thanks to SLARM's uniform message dissemination protocol (Section VI-B3), the adversary can only perform the random deferring attacks but not the targeted deferring attacks. We compared the SLA satisfaction rates of SLARM and Corrected Gossip (the baseline with the highest SLA satisfaction rate in Fig. 4(a)) under the random deferring attacks.

We first evaluated the SLA satisfaction rate of SLARM and Corrected Gossip when the adversary randomly defers 50% messages by 8 s on 10% ∼50% P2P nodes (Fig. 8). Then, we evaluated two systems' SLA satisfaction rates when the adversary randomly defers 50% messages by 8 s∼ 32s (32s is equal to transactions' SLA deadlines) on 50% P2P nodes (Fig. 9). In each experiment, we started the deferring attack at 5 s.

In Fig. 8, even 50% of P2P nodes are under attack, SLARM's SLA satisfaction rates were still high. In contrast, the SLA satisfaction rate of Corrected Gossip gradually dropped and finally converged to a lower SLA satisfaction rate. In Fig. 9, when the adversary deferred transactions by 8 s or 16 s, SLARM's SLA satisfaction rates remained constantly high. Under the most severe attack scenario, where the adversary deferred transactions by 32 s, SLARM's SLA satisfaction rate first dropped gently then recovered quickly. SLARM's high SLA satisfaction rate is because (1) SLARM's trustworthy RTT mechanism (Section VI-B2) can capture the deferring attacks and SLARM's P2P nodes will prioritize the dissemination of the deferred transactions to compensate for the delay caused by the adversary; (2) upon receiving valid committed blocks containing SLA transactions with negative remaining deadlines, SLARM's bidirectional multicast protocol (Section V) will disseminate the SLA transactions faster by increasing the Gossip's fanout, ensuring that the SLA transactions can meet their deadlines under attacks; and (3) when SLARM nodes captured the dramatic RTT increase, the node replaces the peer with increased RTT with a new peer (Section VI-B2), thus subsequent transactions are disseminated through the new peer and will not miss their SLA deadlines.

Overall, SLARM's SLA satisfaction rate is robust under complex network conditions, making SLARM suitable for the highly malicious blockchain environments.

VIII. Conclusion

We present SLARM, the first bidirectional transaction multicast protocol for permissioned blockchains that can meet diverse SLA requirements of different applications. Extensive evaluation shows that SLARM not only greatly improves the user-perceived SLAs but also improves the system's performance and efficiency. SLARM's integration with permissioned blockchains has the potential to attract broad deployments of Internet-wide applications with SLA requirements.

Acknowledgment

The authors sincerely thank the editor and reviewers for their valuable feedback.

References

[1] S. Nakamoto, "Bitcoin: A Peertopeer electronic cash system," Oct. 31, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] Quorum, 2020. [Online]. Available: https://consensys.net/quorum/

[3] E. Androulaki et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

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

[5] M. Castro et al., "Practical Byzantine fault tolerance," in *Proc. 3rd Symp. Operating Syst. Des. Implementation*, 1999, pp. 173–186.

[6] J. Qi et al., "Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 18–34.

[7] Ethereum Whitepaper, 2020. [Online]. Available: https://ethereum.org/en/whitepaper/

[8] Ethereum Time Units, 2021. [Online]. Available: https://solidity.readthedocs.io/en/v0.8.4/units-and-global-variables.html

[9] UbidUwin, 2020. [Online]. Available: https://www.ubiduwin.com

[10] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 817–831.

[11] T. Hoefler, A. Barak, A. Shiloh, and Z. Drezner, "Corrected gossip algorithms for fast reliable broadcast on unreliable systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 357–366.

[12] J. Leitao, J. Pereira, and L. Rodrigues, "Gossip-based broadcast," in *Handbook of Peer-to-Peer Networking*. Berlin, Germany: Springer, 2010, pp. 831–860.

[13] X. Chen et al., "Efficient and dos-resistant consensus for permissioned blockchains," *Perform. Eval.*, vol. 153, 2022, Art. no. 102244.

[14] C. Georgiou, S. Gilbert, and D. R. Kowalski, "Meeting the deadline: On the complexity of fault-tolerant continuous gossip," *Distrib. Comput.*, vol. 24, no. 5, pp. 223–244, 2011.

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

[16] J. Lind, I. Eyal, F. Kelbert, O. Naor, P. Pietzuch, and E. G. Sirer, "Teechain: Scalable blockchain payments using trusted execution environments," 2017, *arXiv: 1707.05454*.

[17] Azure confidential computing, 2017. [Online]. Available: https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/

[18] Clique PoA protocol & Rinkeby PoA testnet, 2017. [Online]. Available: https://github.com/ethereum/EIPs/issues/225

[19] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (PoET)," in *Proc. Int. Symp. Stabilization Saf. Secur. Distrib. Syst.*, Springer, 2017, pp. 282–297.

[20] S. Wooders et al., "Cloudcast: High-throughput, cost-aware overlay multicast in the cloud," in *Proc. 21st USENIX Symp. Netw. Syst. Des. Implementation*, 2024, pp. 281–296.

[21] W. Xiao, S. Zhao, X. Wang, and C. Zhou, "Segment EDF: A scheduling policy with tight deterministic latency under multi-hop networks," *IEEE/ACM Trans. Netw.*, vol. 33, no. 1, pp. 446–461, Feb. 2025.

[22] H. D. Johansen, R. V. Renesse, Y. Vigfusson, and D. Johansen, "Fireflies: A secure and scalable membership and gossip service," *ACM Trans. Comput. Syst.*, vol. 33, no. 2, pp. 1–32, 2015.

[23] H. Qiu et al., "A geography-based P2P overlay network for fast and robust blockchain systems," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1572–1588, May/Jun. 2023.

[24] C. Drift, "Clock drift," 2024. Accessed: Oct. 10, 2024 . [Online]. Available: https://en.wikipedia.org/wiki/Clock_drift

[25] How accurate will my clock be?, Jun. 27, 2022. Accessed: Oct. 10, 2024 . [Online]. Available: https://www.ntp.org/ntpfaq/NTP-s-algo/#5131-how-accurate-will-my-clock-be

[26] J. C. Corbett et al., "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 1–22, 2013.

[27] R. B. Uriarte, H. Zhou, K. Kritikos, Z. Shi, Z. Zhao, and R. De Nicola, "Distributed service-level agreement management with smart contracts and blockchain," *Concurrency Computation: Pract. Experience*, vol. 33, no. 14, 2021, Art. no. e5800.

[28] A. Alzubaidi, K. Mitra, P. Patel, and E. Solaiman, "A blockchain-based approach for assessing compliance with SLA-guaranteed IoT services," in *Proc. 2020 IEEE Int. Conf. Smart Internet of Things*, 2020, pp. 213–220.

[29] S. Steffen, B. Bichsel, R. Baumgartner, and M. Vechev, "ZeeStar: Private smart contracts by homomorphic encryption and zero-knowledge proofs," in *Proc. 2022 IEEE Symp. Secur. Privacy*, 2022, pp. 179–197.

[30] R. Cheng et al., "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2019, pp. 185–200.

[31] Porting guidelines, 2021. [Online]. Available: https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html#secure-clock

[32] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proc. 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–16.

[33] A. Nasrullah and F. M. Anwar, "Trusted timing services with timeguard," in *Proc. IEEE 30th Real-Time Embedded Technol. Appl. Symp.*, 2024, pp. 1–14.

[34] H. Moniz, "The Istanbul BFT consensus algorithm," 2020, *arXiv: 2002.03613*.

[35] FISCO BCOS, 2022. [Online]. Available: http://fisco-bcos.org/

[36] Hyperledger Besu, 2020. [Online]. Available: https://besu.hyperledger.org/en/stable/Concepts/Consensus-Protocols/Overview-Consensus/

[37] M. Walck, K. Wang, and H. S. Kim, "TendrilStaller: Block delay attack in bitcoin," in *Proc. IEEE Int. Conf. Blockchain*, 2019, pp. 1–9.

[38] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, 1999.

[39] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "ResilientDB: Global scale resilient blockchain fabric," 2020, *arXiv: 2002.00160*.

[40] S. Gupta, J. Hellings, and M. Sadoghi, *Fault-Tolerant Distributed Transactions on Blockchain*. San Rafael, CA, USA: Morgan & Claypool, 2021.

[41] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, “Measuring Ethereum network peers,” in *Proc. Internet Meas. Conf.*, 2018, pp. 91–104.
[42] Y. Pi, S. Jamin, P. Danzig, and F. Qian, “Latency imbalance among internet load-balanced paths: A cloud-centric view,” in *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 1–29, 2020.
[43] Networking layer, 2024. [Online]. Available: https://ethereum.org/en/developers/docs/networking-layer/
[44] S. Verma and W. T. Ooi, “Controlling gossip protocol infection pattern using adaptive fanout,” in *Proc. 25th IEEE Int. Conf. Distrib. Comput. Syst.*, 2005, pp. 665–674.
[45] F. M. Anwar, L. Garcia, X. Han, and M. Srivastava, “Securing time in untrusted operating systems with timeseal,” in *Proc. IEEE Real-Time Syst. Symp.*, 2019, pp. 80–92.
[46] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, “Detecting privileged side-channel attacks in shielded execution with Déjá Vu,” in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 7–18.
[47] M. Jelasity, “Gossip,” in *Self-Organising Software*. Berlin, Germany: Springer, 2011, pp. 139–162.
[48] A. J. Ganesh, “Rumour spreading on graphs,” lecture notes, University of Bristol, 2015. [Online]. Available: https://people.maths.bris.ac.uk/~maajg/teaching/complexnets/rumours.pdf
[49] J. Backman, S. Yrjölä, K. Valtanen, and O. Mämmelä, “Blockchain network slice broker in 5G: Slice leasing in factory of the future use case,” in *Proc. 2017 Internet of Things Bus. Models Users Netw.*, 2017, pp. 1–8.
[50] R. Hanifatunnisa and B. Rahardjo, “Blockchain based e-voting recording system design,” in *Proc. 11th Int. Conf. Telecommunication Syst. Serv. Appl.*, 2017, pp. 1–6.
[51] E. Ak and B. Canberk, “BCDN: A proof of concept model for blockchain-aided CDN orchestration and routing,” *Comput. Netw.*, vol. 161, pp. 162–171, 2019.
[52] CDC is testing blockchain to monitor the country’s health, 2018. [Online]. Available: https://www.nextgov.com/emerging-tech/2018/11/cdc-testing-blockchain-monitor-countrys-health-real-time/152622/
[53] Ethereum Private Network, 2019. [Online]. Available: https://geth.ethereum.org/docs/interface/private-network

Ji Qi received the BS degree from the Beijing Institute of Technology, the MS degree from Tsinghua University, and the PhD degree from the University of Hong Kong. He is an assistant professor with the Institute of Software Chinese Academy of Sciences.

Tianxiang Shen received the BEng degree from Jilin University and the PhD degree in computer science from HKU. He is a researcher with Huawei.

Jianyu Jiang received the bachelor’s degree from Computer Science Department, Xi’an Jiaotong University, and the PhD degree form Computer Science Department, HKU. He is a researcher with ByteDance.

Xusheng Chen received the bachelor’s and PhD degrees in computer science from HKU. He is a researcher with Huawei. His research interests include distributed consensus protocols, distributed systems, and system security.

Xiapu Luo received the PhD degree from the Hong Kong Polytechnic University and was a postdoctoral fellow with Georgia Tech. He is an associate professor with the Department of Computing, Hong Kong Polytechnic University. His research interests include network and system security, Blockchain, and IoT security.

Fengwei Zhang (Senior Member, IEEE) is an associate professor with the Department of Computer Science and Engineering, SUSTech. His research focuses on systems security, particularly trustworthy execution, and hardware-assisted security. Prior to joining SUSTech, he was an assistant professor with Wayne State University.

Heming Cui (Member, IEEE) is an associate professor in computer science with HKU. His research interests include operating systems, programming languages, distributed systems, and cloud computing, with a focus on building software infrastructures to improve reliability and security of real-world software.