DeCross: Toward Accountable and Efficient Cross-Chain Collaboration in IIoT

Yuchao Zhang, Xiaofeng He, Xiaotian Wang, Haiyang Wang, and Ke Xu, *Fellow, IEEE*

***Abstract*—Blockchain-based industrial Internet-of-Things (IIoT) systems have seen rapid adoption and development in recent years. The increasing diversity of IIoT application scenarios is driving the growth of multichain ecosystem, making cross-chain communication a key issue in multichain collaboration. However, existing centralized cross-chain architectures risk derailing the blockchain's trust-free decentralization and suffer from single point failure. The design of decentralized cross-chain collaboration mainly faces two key challenges. First, implicit cross-chain accountability, which is caused by collusion among malicious distributed participants and, thus, compromises cross-chain security. Second, low cross-chain efficiency, which is induced by the highly dynamic environment in practical cross-chain networks, i.e., changing memberships, and adaptive attacks to corrupt honest nodes. In this article, we propose a cross-chain consensus protocol *DeCross* to solve the abovementioned problems. *DeCross* achieves decentralized cross-chain collaboration with explicit accountability and high efficiency. Specifically, we audit participants with a succinct auditable data object constructed from the protocol to hold nodes accountable for misbehaving. Furthermore, we propose a parallel processing workflow that leverages both CPUs and GPUs to guarantee efficient and stable cross-chain communication. Finally, we implement a prototype based on the hyperledger fabric with both local and geo-distributed clusters. Our extensive experiments show that *DeCross* achieves 44% better throughput over the existing cross-chain approaches.**

***Index Terms*—Blockchain, consensus, cross-chain communication.**

Received 8 August 2024; revised 18 October 2024 and 16 February 2025; accepted 19 February 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62172054, in part by the Beijing Nova Program under Grant 2023140, in part by the National Natural Science Foundation of China for Distinguished Young Scholars under Grant 62425201, and in part by the University of Minnesota Grant-in-Aid 739819. Paper no. TII-24-4028. *(Corresponding author: Yuchao Zhang.)*

Yuchao Zhang, Xiaofeng He, and Xiaotian Wang are with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100088, China (e-mail: yczhang@bupt.edu.cn; xiaofenghe@bupt.edu.cn; wangxiaotian@bupt.edu.cn).

Haiyang Wang is with the Department of Computer Science, University of Minnesota Duluth, Duluth, MN 55812 USA (e-mail: haiyang@d.umn.edu).

Ke Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: xuke@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TII.2025.3547061

I. Introduction

THE evolution of the industrial Internet-of-Things (IIoT) is driving rapid growth of various applications such as smart cities, e-healthcare, and intelligent transportation [1]. With the increasing scale of IIoT networks, massive real-time data processing and storage are put forward with high-quality requirements. Furthermore, there exist non-negligible security and trust issues in information exchange. It has become a nontrivial task to enable confidential collaborations in IIoT [2].

With the characteristics of decentralization, immutability, and transparency, blockchain-based IIoT has emerged as a prospective solution for providing secure and collaborative services in different IIoT scenarios such as circular economy, smart energy, and intelligent transportation [3], [4], [5]. The blockchain technology adopted to the IIoT ecosystem enables secure and traceable data exchange, reducing the risk of data tampering and leakage. However, the rapid growth of heterogeneous IIoT applications necessitates flexible data sharing among IIoT devices across various cooperative domains and organizations. The resulting cross-domain and cross-organization collaboration typically involves multiple blockchains processing transactions collaboratively. Consequently, cross-chain operation among a set of collaborative members in IIoT has become increasingly prominent. For example, in the circular economy, cross-chain communication allows nontrusting parties from different organizations or regions to cooperate to create the circular economy of solar panels and make consensus on specific indexes, i.e., recycling cost per module and solar panel lifetime [3]. In smart energy, cross-chain transactions enable energy trading between prosumers even if they register accounts on different chains [5]. Cross-chain collaboration significantly improves productivity and enhances the delivery of services, thereby becoming a promising industrial production mode. Currently, most of the cross-chain solutions adopt a centralized architecture, where a trusted third party (e.g., relay chains) [6] and the notary mechanism [7]) is introduced. Although this method can achieve interoperability across chains, it relies on the trust premise of committees or an honest majority assumption. The extra trust presuppositions significantly lower the decentralization of blockchains. Besides, relying on a central party has a higher possibility of causing single point failures.

A blockchain system typically relies on a consensus protocol for reaching an agreement and upholding fault-tolerance in a decentralized manner. Hence, a cross-chain consensus protocol is viewed as a prospective solution to handle the cross-chain collaboration in blockchain-based IIoT without compromising decentralization and security. However, the design of a cross-chain consensus protocol introduces two key challenges. 1) *Explicit cross-chain accountability*: In cross-chain systems, consensus membership is more likely to become dishonest since the nodes (replicas) may originate from multiple heterogeneous blockchain networks. Although existing Byzantine fault tolerance (BFT) consensus protocols tolerate a subset of participants behaving arbitrarily, they ensure safety and liveness when fewer than $1/3$ of N replicas misbehave. With more misbehaving replicas, the consensus is unreliable and misbehavior is unable to be identified. Besides, malicious replicas may misbehave during ledger state updating, which can destroy consistency and cause ledger conflicts across replicas. Thus, the accountability of replicas is blurred, making it intractable to prevent the global ledger stored in multiple chains from being tampered with. In this case, explicit replica accountability in blockchain ledgers is quite important to mitigate the issue. 2) *High cross-chain efficiency*: On one hand, stability demand. As real IIoT applications are highly dynamic, cross-chain communication should be able to remain stable under dynamic environments. Specifically, the dynamic environment includes changing membership of the cross-chain consensus committee due to the joining/leaving of nodes, and fluctuating network conditions such as changing bandwidth and delay. On the other hand, low latency demand. Since a single cross-chain transaction involves multiple transactions to register, record, and finalize, the frequent read/write operations from these transactions lead to high processing latency. Besides, the corresponding read/write dependency conflicts between transactions incur extra time to solve. Consequently, low cross-chain efficiency poses a significant challenge that hinders the large-scale adoption of cross-chain collaboration in practice.

To tackle the aforementioned problems, we propose *DeCross*, a trusted and decentralized cross-chain consensus protocol for blockchain-enhanced IIoT. It achieves fully distributed interoperation without depending on any trusted centralized intermediary. To guarantee explicit accountability, we design an auditing mechanism based on the protocol that can assign blame regardless of the number of misbehaving replicas. The auditing relies on a lightweight auditable data object (ADO) computed from the consensus messages, Merkle tree, and the digital signatures. The universal verifiable proof can be generated from the ADO to provide auditors with evidence for succinct proof of misbehaving. To ensure cross-chain efficiency, we then present a parallel processing workflow utilizing the GPU resources for consensus configuration to reduce latency in dynamic cross-chain networks. We then propose the conflict-free multithread transaction execution with CPU to further accelerate the cross-chain communication. In a nutshell, our work is a novel attempt to achieve high-performance cross-chain communication without compromising security guarantees in a decentralized way.

Our contributions are discussed as follows.

1) We propose a decentralized cross-chain protocol *DeCross* to achieve auditable collaboration in blockchain-based IIoT without a trusted third party or relay committee.
2) We propose a parallel processing workflow that leverages both CPU and GPU to achieve efficient cross-chain communication and improve resource utilization in a highly dynamic network.
3) Furthermore, we give a comprehensive evaluation of the entire *DeCross* scheme. The results show that our proposed solution achieves 44% higher throughput than existing cross-chain schemes.

The rest of this article is organized as follows. The related work is reviewed in Section II. Section III presents the system design. Sections IV and V describe our protocol in detail. Section VI presents the analysis of security and correctness. The implementation and evaluation are presented in Section VI. Finally, Section VII concludes this article.

II. Related Work

In this section, we review the cross-chain technology and accountability in both distributed systems and blockchains. Table I compares the representative related works across various aspects. λ denotes the predefined security parameter, where the failure probability P_r of the N replicas satisfies $P_r \leq 2^{-\lambda}$.

A. Cross-Chain Communication

Cross-chain communication has drawn attention in both research and industry areas. Existing cross-chain solutions mainly include the following four types: sidechains, relays, notary mechanism, and hashed time-lock contracts (HTLC). Sidechains are the most common type of cross-chain communication, which realizes interactions through a centralized two-way peg [8]. Relays are trusted parties incorporating execution of smart contracts from the sender and providing verification of executing results to receiver [6]. The notary mechanism applies monitors across multiple blockchains and undertakes the interactions [7]. Compared with relay modes, which commonly extend the existing blockchain structure, the notary acts as a third-party processing operations on both chains. HTLC combine hash-lock and time-lock to generate cryptographic proofs for cross-chain transactions within a timeout period [9].

However, the abovementioned solutions rely on centralized servers or trusted third parties to bridge the communication between clients and the blockchain networks, which is unavoidable for single point failure and misbehaving of the central server although these solutions may achieve high throughput. To solve the issue, Polkadot [10] achieves interoperability through a global data structure called parachains. These parallelized parachains connect different chains and validate cross-chain state transition proofs via the cross-chain message passing protocol. Cosmos [11] ensures cross-chain data transfer through specific channels called Hubs. The relayer in Hubs handles the cross-chain transactions from the sender. However, Polkadot and Cosmos still suffer from semicentralized governance and low scalability.

Recent works aim to utilize decentralized cross-chain schemes to eliminate centralization. AC^{3}WN [12] proposes the first decentralized atomic cross-chain commitment protocol, which ensures that conflicting events never occur simultaneously and allows concurrent processing of atomi3c swaps. Ghosh et al. [13] introduced a decentralized gateway architecture that

TABLE I
COMPARISONS OF CROSS-CHAIN SYSTEMS AND ACCOUNTABLE PROTOCOLS

Systems/protocols	**Cross-chain approach**	**Fault tolerance**	**Communication complexity**	**Accountability**	**Reliability**
Cosmos [11]	Relayer	$\frac{1}{2}N$	$O(\lambda n^2)$	✗	✗
zkBridge [14]	Zero-knowledge proofs	$\frac{1}{2}N$	$O(n(\log n + \lambda))$	✗	✓
AC^{3}WN [12]	Atomic cross-chain commitment	$\frac{1}{2}N$	$O(\lambda n^2)$	✗	✗
CrossChannel [15]	Off-chain payment channel	$\frac{1}{2}N$	$O(\lambda n \log n)$	✗	✓
Prosecutor [16]	✗	$\frac{1}{3}N$	$O(\lambda n^2)$	Implicit	✓
Polygraph [17]	✗	$\frac{1}{3}N$	$O(\lambda n^2)$	Implicit	✓
DeCross(Ours)	**Cross-chain consensus**	**N**	$O(\lambda n \log n)$	**Explicit**	✓

interfaces between public and private blockchains. Although it uses off-chain multisignature collection to improve efficiency, the Consensus on Consensus mechanism may introduce unacceptable latency in real networks. Zkbridge [14] designs a trusted and decentralized cross-chain architecture without any honesty assumptions and proposes a concise proof for on-chain verification acceleration. CrossChannel [15] tries to accelerate cross-chain transactions through off-chain channels but compromises safety. All these works above suffer from low cross-chain efficiency due to the dynamic states of nodes and networks. In addition, replica accountability is ignored, making security guarantees impractical in the face of targeted attacks.

B. Accountability in Blockchains

Accountability is a fundamental concept in distributed systems and it serves as an alternative safety guarantee in blockchain to strict security measures, enabling the detection and attribution of misconduct by malicious participants. Moreover, accountability can incentivize replicas to either prevent or disclose misbehavior. PeerReview [18] uses accountable virtual machines to ensure accountability in distributed systems without violating the liveness. It maintains auxiliary information from re-execution to provide evidence for detecting faults. However, it incurs high overhead since all consensus messages are signed. Du et al. [19] proposed the first pragmatic data auditing protocol applied in the decentralized storage system for efficient and privacy-preserving on-chain verification. However, it relies on prebuilt cryptography primitives to implement before auditing and is limited to be applied in the smart contract.

Accountability with more than $f + 1$ consensus participants misbehaving has been explored to identify and blame misbehaving parties in some cases without relying on an honest majority assumption. Prosecutor [16] examines the auditability of BFT consensus protocols, proposing an auditable ledger that defines dishonest behavior based on the review of consensus protocol messages and the blockchain ledger. Polygraph [17] is another accountable BFT consensus protocol designed to detect misbehavior with more than $f + 1$ replicas. However, the accountability mechanisms in these works fail when $2f + 1$ participants are corrupted because these malicious replicas can collude with each other to tamper with the ledger. Besides, the reconfiguration of the consensus membership is not well-defined to prevent the persistent misbehaving replicas from slowly compromising honest replicas over time.

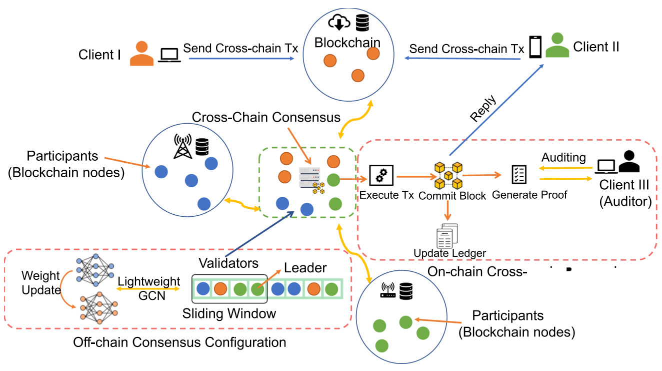

Fig. 1. Proposed cross-chain architecture.

III. SYSTEM DESIGN OF DECROSS

In this section, we present the system design of *DeCross*. We first give the system model of *DeCross* and then explain the overview of *DeCross* scheme.

A. System Model

Similarly to intrachain consensus, *DeCross* proceeds in epochs within a set of nodes, which can be either honest or malicious. *DeCross* follows the Byzantine failure model and consists of N nodes with f malicious nodes. Malicious nodes may collude with each other and behave arbitrarily, e.g., delay, tamper, and discard messages, send conflicting content to other nodes, keep silent when receiving messages, or pretend unable to send messages. Like existing blockchain systems, the nodes in *DeCross* rely on a partially synchronous network assumption, where all messages can be successfully delivered after an unknown global stabilization time.

B. Overview of DeCross

The architecture of the system is shown in Fig. 1, where cross-chain interactions are configured and dealt with. The proposed scheme mainly involves four entities: clients, participants, validators, and auditors.

1) Clients: Clients connect to the blockchain network via IIoT devices to send, receive, and validate intrachain and cross-chain transactions between blockchain nodes.

2) Participants: Participants are blockchain nodes in cross-chain systems. They act as the intrachain consensus nodes and

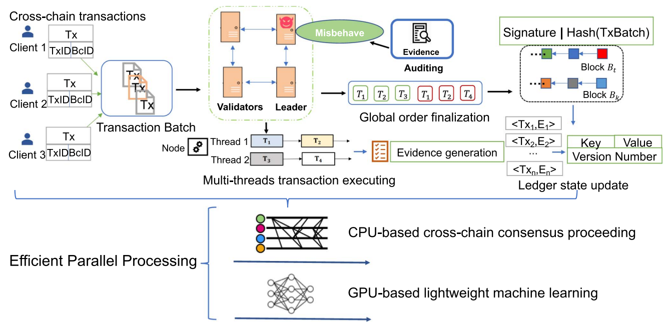

Fig. 2. Workflow of *DeCross* cross-chain scheme.

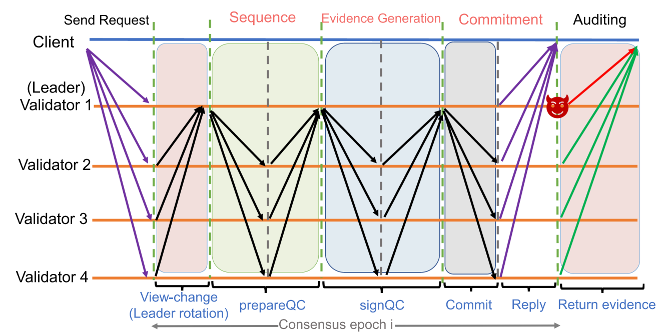

Fig. 3. *DeCross* protocol specification.

are responsible for ordering, executing, and storing the transactions.

3) Auditors: Auditors can be any clients who find the ledger is nonlinear. They initiate the auditing process upon observing inconsistencies in the execution results or the ledger state. The related malicious nodes are then identified and blamed by generating the universal proof of misbehaving, and the invalid transactions can be replayed to reconstruct a valid global ledger state.

4) Validators: Validators are elected from different chains as cross-chain consensus committee members. They are involved in the cross-chain consensus during which the evidence used for auditing is generated and stored.

Based on the architecture above, we design a cross-chain consensus protocol named *DeCross*, as shown in Fig. 2. *DeCross* achieves both accountability in a malicious environment and efficiency in the dynamic and resource-constrained cross-chain network. The workflow can be summarized as follows: The clients send cross-chain transactions that are processed through cross-chain consensus. The cross-chain consensus consists of several blockchain nodes called validators, which are elected based on the output of a lightweight GCN model. During the cross-chain consensus, the validators order and execute the transactions, pack them into a block, and commit it. We build a parallel processing framework leveraging both CPU and GPU to enhance efficiency. Then, the blockchain ledger is updated and the execution result is replied to the client. A lightweight ADO is also generated and stored. Whenever a client finds inconsistencies in the ledger state, it can instantly begin auditing to check the misbehavior. A universal proof of the incorrect transaction execution is constructed from the ADO. Then, it can assign blame to those validators colluding with each other to participate in the malicious behavior.

IV. Accountability-Preserving Cross-Chain Protocol

In this section, we formalize the cross-chain protocol *DeCross* and explain the auditing based on this protocol.

A. *Basic* DeCross

DeCross is an accountability-preserving Byzantine consensus protocol for cross-chain transactions. The protocol proceeds with an increasing view number v and collects $N - f$ votes from replicas in each phase to ensure liveness and safety, similar to Hotstuff, the state-of-the-art Byzantine fault-tolerant consensus protocol [20]. The collection of $N - f$ votes in a specific view is referred to as a quorum certificate (QC). A *view-change* proposal is initiated to enter the new view and perform leader rotation if any replica does not obtain QC for a period of time (timeout). Fig. 3 describes the consensus workflow: After receiving requests from a client, the consensus proceeds by collecting several QCs and replies to the client. *DeCross* achieves auditing by providing auditors with succinct proof and universally verifiable evidence generated from signed protocol messages. Clients can start auditing when inconsistencies by checking the ADO, which is computed from some auxiliary data embedded in the consensus messages.

B. *Protocol Specification*

As shown in Algorithm 1, *DeCross* proceeds in three phases for each round, i.e., *sequence, evidence generation, and commitment*.

1) Phase I Sequence: The transaction requests sent from the client are first received by cross-chain consensus replicas named validators and then broadcast. Different from the conventional consensus, the requests in *DeCross* can be sent to multiple validators in parallel, instead of only to the leader. This means cross-chain transactions can be individually disseminated and ordered by validators through multiple instances of consensus. If one validator does not receive sufficient messages for a period of time or suspects the leader of misbehaving, a *view-change* request is initiated for the leader rotation. A new leader begins broadcasting after receiving $N - f$ votes. We design an efficient, secure, and fair leader rotation approach, which will be discussed in Section V.

When a leader collects a batch of cross-chain transactions, it first locally orders the requests and carries out a pre-execution result $\text{res}_{\text{tx}} = \langle R_{\text{tx}}, W_{\text{tx}}, G_{\text{cur}} \rangle$ for each transaction. We record the read set R_{tx} and the write set W_{tx} in a transaction execution. G_{cur} is the state root of the current block that represents the latest ledger state. After pre-execution, the leader computes the Merkle proof of all R_{tx} and W_{tx} of the received transactions, denoted as π_r and π_w. The hash value of the consensus proposal is defined as

$$H_{\text{prepare}} = \text{hash}(\pi_r || \pi_r || G_{\text{cur}}). \quad (1)$$

Algorithm 1: Accountability-Preserving Cross-Chain Protocol.

```
1:  procedure SEQUENCE
2:    ON sendPrepare($request = \{tx\}$)
3:    Pre: verify($t$)
4:    $\mathcal{T} \leftarrow \mathcal{T} \cup \{tx\}; \mathcal{G} \leftarrow \{\}$
5:    $\mathcal{R} \leftarrow \{\}; v \leftarrow getCurrentView()$
6:    if $|v_{local}| < v$ then
7:      $sendViewChange()$
8:    end if
9:    for all $tx \in \mathcal{T}$ do
10:     $\langle R_{tx}, W_{tx} \rangle \leftarrow$ execute($v, tx$);
11:     $\mathcal{R} \leftarrow \mathcal{R} \cup \langle tx, R_{tx}, W_{tx} \rangle$;
12:     $\pi_r, \pi_w \leftarrow getMerkleProof(R_{tx}, W_{tx})$;
13:     $\mathcal{G} \leftarrow generateRoot(pre)$
14:     $H_{prepare} \leftarrow hash(\pi_r || \pi_r || G_{cur})$
15:     $broadcast(H_{prepare})$
16:   end for
17: end procedure
19:   procedure EVIDENCE GENERATION
20:   On receiving $N - f$ votes:
21:     broadcast($\mathcal{R}, prepareQC, sendSign$)
22:   $pre \leftarrow$ receivePrepare($prepareQC, v$)
23:   $G_{PMT} \leftarrow generateRoot(PMT(p))$
24:   $EV_p \leftarrow \{H(H(H_{prepare}||H(\mathcal{G}))||G_{PMT})\}$
25:   $SIG \leftarrow getSignature(k)$
26:   $broadcast(commit, SIG)$
27: end procedure
29:   procedure COMMITMENT
30:   On receiving $N - f$ votes:
31:     broadcast($signQC, sendCommit, p, EV_p$)
32:   $p \leftarrow$ receiveCommit($signQC, v, \mathcal{M}, EV_p$)
33:   for all $r \in \mathcal{R}$ do
34:     $t \leftarrow$ removeTx($r$); $\mathcal{L} \leftarrow \mathcal{L} \parallel \langle t \rangle$
35:     $Sig \leftarrow sig_{pre}, sig_{sig}$;
36:     $ADO$
        $\leftarrow \langle EV_p, \pi_r, \pi_w, Cp, \{SIG_{pre}\}, \{SIG_{sig}\} \rangle$
37:   end for
38: end procedure
```

Then, the leader adds H_{prepare} into the prepare consensus message and broadcasts the ordered transaction requests to other validators. Upon receiving the messages, a validator executes the transactions in the consensus proposal and packs the transaction execution records into a global Merkle tree GMT_p. If the reconstructed proof of executing results and the updated Merkle tree root match that received from the leader, it adds a digital signature and replies to the leader with a prepare vote. The signature is generated via a hash function based on i) its private key, ii) the hash of the consensus proposal. After receiving $N - f$ confirmation from other validators, the leader combines them into a prepareQC. If the match fails, the record of this proposal is aborted and the leader initializes a new consensus instance.

2) Phase II Evidence Generation: In this phase, the leader first broadcasts prepareQC in a sign consensus message. The evidence is to prove the write set execution is signed by at least $N - f$ validators. We design a partial Merkle tree PMT_p of a consensus proposal p for evidence generation. It can declare whether operations in R_{tx} and W_{tx} of all transactions in the proposal have been executed correctly. In addition, it helps to update only the state root digest without accessing the full Merkle tree. PMT_p shares the same state root as the full Merkle tree but only the tree nodes corresponding to transactions that have W_{tx} after a checkpoint Cp, along with their Merkle paths, are reserved. In other words, a PMT_p only records the W_{tx} from transactions that are received after a checkpoint index. The design has two reasons: On the one hand, the execution of read-only transactions will not tamper the ledger state, and the cost of constructing a PMT_p is minimal. On the other hand, every validator stores a checkpoint Cp that represents the latest auditing index. It enables auditing without replaying the ledger from the start since the transactions before the checkpoint are verified. After the auditing process is completed, the checkpoint is updated through consensus. We then define the evidence

$$\text{EV}_p = \left\{\text{hash}\left(\text{hash}\left(H_{\text{prepare}}||\text{hash}(G_{\text{GMT}_p})\right)||G_{\text{PMT}_p}\right)\right\}. \quad (2)$$

It describes the unique digest of execution results within the path from the leaf to the root in GMT_p and PMT_p. Although the evidence is generated partly from the ledger state, it can be hard for third parties to alter the evidence from which we can carry out the proof of collusion. Each validator then computes an EV_p using the GMT_p and replies to the leader along with its digital signature. When the leader receives $N - f$ sign votes, it combines them into a signQC, packs the transactions into a block, and then broadcasts a commit message.

3) Phase III Commitment: In the commitment phase, two components need to be committed to complete the cross-chain transaction: i) block commitment and ii) evidence commitment.

The block commitment is to verify the generated block and decide whether it can be appended to the ledger. After the leader proposes a block in the commit message, validators check the block hash to verify the integrity and the correctness of the received block. To avoid misbehaving during the block dissemination, validators will reply to the block hash to preclude malicious parties from tampering with the content. As for evidence commitment, validators verify the validity of the prepare and sign messages through digital signatures and recompute the Merkle tree root using the transactions in the block. If the abovementioned results match, validators send the execution results to the client. After the client receives $f + 1$ identical replies, the block is committed and appended to the chain. The ledger state is also updated according to the write operations in the PMT_p. Finally, the validators store the lightweight ADO for each block as

$$\text{ADO}_p = \langle \text{EV}_p, \pi_r, \pi_w, Cp, \{\text{SIG}_{\text{pre}}\}, \{\text{SIG}_{\text{sig}}\} \rangle. \quad (3)$$

$\{\text{SIG}_{\text{pre}}\}$ and $\{\text{SIG}_{\text{sig}}\}$ indicate the signatures in the prepare and sign consensus messages, respectively. The ADO is returned to an auditor for auditing if inconsistency in the ledger is found or validators are suspected of misbehaving.

The storage cost is acceptable since except for the signatures, which have $O(N)$ space complexity, all other elements take up a small constant storage size. The packed transactions and related consensus messages are maintained in validators until the check index exceeds the transactions.

C. Auditing

When any client finds that no successive and linearizable execution in the stored state can produce the same sequence of evidence, it is deemed that dishonest behavior occurs and the auditing is booted up. We apply auditing to detect misbehaving regardless of the number of malicious validators. The auditor first receives a collection of ADO to find proof of misbehavior regardless of the number of misbehaving validators.

As shown in Algorithm 2, the auditor first retrieves the checkpoint index Cp related to the abnormal results and a portion of the ledger that refers to the evidence marked with Cp from the validator. Then, it begins verifying from the index, it first checks the transaction order and the signatures of consensus messages $\{\mathrm{SIG}_{\mathrm{pre}}\}$ and $\{\mathrm{SIG}_{\mathrm{sig}}\}$ to confirm that i) the index is a right and valid mark, ii) the evidence is not expired. Then, the auditor checks π_r, π_w to confirm that the transactions referenced by the evidence are recorded at the right positions in the portion. Finally, we check the execution result, if the evidence EV_p does not match the result in the portion or exception occurs before this phase, then we hold accountability within $f+1$ malicious validators. The misbehaving can be induced by a certain proportion of malicious validators M:

1) $M <= 2f+1$: In this case, if the proposal that contains the transaction has the same view as that in the prepare messages, we can assert to blame those $f+1$ validators that have signed and committed for the batch. If the view number is unequal, the $M = C_{\mathrm{vc}} \bigcap C_{\mathrm{sig}}$ validators can be blamed where the C_{vc} and C_{sig} represent the validators having sending the view change message and having signed the prepared messages.

2) $M > 2f+1$: If no misbehavior is found still, we conclude that more than $2f+1$ or more validators may collude to have misbehaved and lead to faulty transaction execution results. Therefore, the auditor can replay the transactions from the check index since the corrupted transactions before the check index is legitimate. If re-executed a transaction fails to match the result in the portion of the ledger throughout this process, the auditor generates a global verifiable proof

$$\pi_{\left\langle \mathrm{tx},\, \mathrm{SIG}_{\mathrm{pre}||\mathrm{sig}},\{\mathrm{GMT}_{\text{tx-root}}\},\{\mathrm{PMT}_{\text{tx-root}}\}\right\rangle}. \tag{4}$$

$\mathrm{SIG}_{\mathrm{pre}||\mathrm{sig}}$ represents the signatures corresponding to the transaction in the prepare and sign consensus messages. $\mathrm{GMT}_{\text{tx-root}}$ and $\mathrm{PMT}_{\text{tx-root}}$, respectively, indicate the sets that include all paths from the global and partial Merkle tree roots to the leaves containing the inconsistent transaction. Then, any validator that signed the proposal containing the ledger fragment is blamed for the misbehaving.

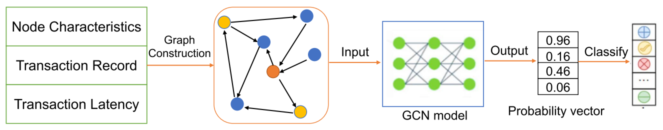

Fig. 4. LightGCN model for validator election.

Algorithm 2: Auditing Algorithm.

```
1:  procedure AUDITING
2:    Pre: $Cp, \mathcal{F} \leftarrow getCheckpointAndLedger()$
3:    verify($SIG_{pre}, SIG_{sig}$)
4:    if getRoot($\mathcal{M}$) $\cup$ getRoot($PMT$) $\neq \mathcal{G}$ then
5:      undo($t$); return
6:    end if
7:    $c = \langle commit, v, \mathcal{G}\rangle$
8:    sendToAllReplicas($c$); $\mathcal{M} \leftarrow \mathcal{M} \cup \{c\}$
9:    for all $\langle tx\rangle \in$ getTxForBatch($\mathcal{L}, v$) do
10:     sendReplyToClient($\langle tx, v, EV_p\rangle$)
11:     if verifyEvidence() $\wedge$ getConflicts() $= \emptyset$ then
12:       $MisbehaveNodes \leftarrow replayLedger(\mathcal{F}, Cp)$
13:     end if
14:   end for
15: end procedure
```

V. Efficient Parallel Processing

In this section, we give a specific explanation of how we design a novel parallel processing workflow for efficient cross-chain collaboration.

The key observation is that in the Byzantine failure mode, the most common mode of transaction execution is CPU-centric processing, the tasks in consensus like message broadcasting and transaction executing are merely CPU-based workloads. Based on this fact, we apply the hardware accelerator by utilizing the idle GPU resource to improve cross-chain efficiency and system performance. Algorithm 3 describes the parallel processing workflow. In our system architecture, the GPU-based workload handles tasks related to the consensus committee configuration. We utilize a lightweight improved GCN model, LightGCN, for validator election and committee formation. The CPU-based workload manages consensus tasks such as broadcasting, ordering, and executing transactions. To optimize throughput and latency, we implement parallel transaction execution using multiple threads on the CPU. Concurrent conflicts are resolved through deterministic execution strategies, ensuring consistent and reliable transaction processing.

A. Consensus Committee Configuration

First, the validator election proceeds for every chain during which the participating nodes independently execute LightGCN model training and return the model output to the leader of the intrachain consensus. Then, the leader in each chain constructs the consensus committee based on the node classification from the output. The workflow of the LightGCN model is shown in

Algorithm 3: Efficient Parallel Processing.

```
Input: 1.A set of received transactions $\{tx_1, tx_2 ...\}$ 2.The
    snapshot of the ledger state $LS_i$ based on last block $B_i$
Output: Block $B_{i+1}$ and the ledger state snapshots $LS_{i+1}$
1:  Initialize two transaction batches $\{txBatch\}_{i+1}$ and
    $\{txBatch\}_{i+2}$;
2:  Initialize version number of all keys $\{key.VeNum\}$
3:  for all validators parallel do
4:    GPUExecute($B_i$);
5:    CPUExecute($\{txBatch\}_{i+1}, LS_i$);
6:  end for
8:    function GPUEXECUTE$B_i$
9:    $G = (V, E) \leftarrow GraphConstruction(B_i)$;
10:   $\vec{P_i} \leftarrow ModelTraining(G)$;
11:   $\{Validators\}_{chain} \leftarrow MembershipConfig(\vec{P_i})$;
12: end function
14:   function CPUEXECUTE$\{txBatch\}_{i+1}, LS_i$
15:   for all$tx$ in $txBatch$ parallel do
16:     $\{tx.R, tx.W\} \leftarrow ExecuteTx(tx)$
17:     $\{txBatch\}_{i+1} \leftarrow DetectConflict(tx)$
18:   end for
19:   Commit $\{txBatch\}_{i+1}$ to update $B_{i+1}$ and $LS_{i+1}$
20: end function
22:   function DETECTCONFLICT$tx$
23:   Initialize $key.NewVeNum = key.VeNum$
24:   for each key in $tx.W$ do
25:     if $key.NewVeNum = key.VeNum$ then
26:       $key.NewVeNum++$;
27:     else
28:       $\{txBatch\}_{i+1} \leftarrow Remove(tx)$
29:       $\{txBatch\}_{i+2} \leftarrow \{txBatch\}_{i+2} \cup tx$;
30:     end if
31:   end for
32: end function
```

Fig. 4, which includes graph construction, graph learning, and node classification.

1) LightGCN: Specifically, we apply an improved graph convolutional network LightGCN as the learning model. This model is trained for cross-chain consensus committee configuration. The key challenges are: i) Since the IIoT nodes on different chains may be deployed across geographic regions, the stable and low-latency network for cross-chain communication is critical. ii) The states and performance of the IIoT nodes also determine the system throughput, optimal and honest nodes should be elected as validators. Besides, the joining/leaving of the nodes to the committee requires reconfiguration of the consensus membership. In this section, we propose LightGCN to simultaneously consider the network condition and the node status for highly efficient cross-chain collaboration through graph construction and graph learning.

Graph construction: We construct a directed graph $G = (V, E)$ from the transaction records in each block as model input. Each vertex $v \in V$ describes the blockchain node characteristics and is represented as $\vec{v_k} = [f_1, f_2, s]$, where f_1 and f_2 represent the transaction frequency sent from v_k and v_k received, respectively. s describes the historical frequency of the node chosen as validators. Each edge $e \in E$ is represented as (v_j, v_k, t), where $v_j, v_k \in V$ indicates the transaction sent from v_j to v_k. We introduce t to represent the type of the transaction between nodes since the transaction type may vary in different scenarios, e.g., read/write key-value, modify smart contracts. We then split the graph into multiple subgraphs, each with a particular edge type. To represent different types of transactions, we use the adjacent matrix set $\{A_1, A_2 ..., A_t\}$ instead of a single adjacent matrix to calculate for t type edges. To further increase the accuracy of the result of the validator election, we add a set of latency matrices for each edge type to synthetically evaluate the network condition.

1) *Transaction propagation latency* P^t: Each $p_{ij}^t \in P^t$ represents the average transaction propagation latency from node i to node j in the last block.
2) *Transaction processing latency* Q^r: Each $q_{ij}^t \in Q^t$ represents the average transaction processing latency from node i to node j in the last block.

Graph learning: we then define the layer-wise propagation rule for graph learning based on the graph input, the forward updating is formulated as

$$h_i^{(l+1)} = \sigma\left(\sum_{t\in T}\sum_{j\in N_i^t}\frac{\text{Sigmoid}\left(\alpha p_{ij}^t + \beta q_{ij}^t\right)}{d_{i,t}}W_t^{(l)}h_j^{(l)}\right) \tag{5}$$

where $h_i^{(l)}$ is the hidden state of vertex v_i in the l_{th} layer of LightGCN and the l_{th} layer input is composed of the set of all N vertexes' hidden states. α, β are hyperparameters that represent the weights of transaction propagation latency and transaction processing latency. The weights can be adjusted according to the link stability, bandwidth, and congestion status of the intrachain network. $\frac{1}{d_{i,t}} = |N_i^t| \cdot \tanh(\sum_j a_{ij}^t)$ represents similarity coefficient where $|N_i^t|$ is the number neighboring nodes of v_i in edge type t and $a_{ij}^t \in A_t$. $W_r^{(l)}$ is the model weight matrix of edge type t in the l_{th} layer.

2) Consensus Committee Formation and Leader Rotation: *Node classification:* After training, we get the classification probability vector of being a high-performance node among N nodes from the model output: $\vec{p} = (m_1, m_2 ... m_n)$. Based on the results of node classification, we rank the probability among the N nodes. We then calculate the consensus node penalty (CNP) for each node to select the validators:

$$\text{CNP}_i = m_i(1 - \text{Sigmoid}(N_i^{\text{blame}})). \tag{6}$$

m_i is the classification probability of node i and N_i^{blame} is the number of times a node has been blamed for misbehaving during the last t consensus epochs. The historical malicious behavior is considered to evaluate the node as well. The Sigmoid function is for normalization. Then, all nodes are ranked according to CNP. We apply a sliding window for the election of validators and the leader. The window size k is the same as the number of cross-chain consensus nodes. It is adaptive to make sure that at least one node of each involved chain is elected. The nodes with the top k highest CNP are elected as validators for cross-chain communication. If a validator is found to be misbehaving and is blamed during auditing, the window slides to the right to replace the faulty node.

For leader rotation, since Byzantine nodes may collude with clients to order requests unfairly. In this case, they may behave honestly but repeatedly acquire leadership to continuously derive benefits. To prevent monopolies in leadership, we calculate the consensus leader penalty (CLP) for each validator in the sliding window

$$\begin{aligned}\mathrm{CLP}_i &= \alpha \mathrm{CNP}_i \\ \alpha &= 1 - \mathrm{Sigmoid}(N_i^{\text{leader}}).\end{aligned} \tag{7}$$

To calculate CLP, the CNP is multiplied by an attenuation factor α ($\alpha < 1$). α is related to N_i^{leader}, which represents the number of times a node has become a leader during the last t consensus epochs. α takes into account the historical leadership of the validators to avoid monopoly from a particular node. Our design provides efficiency, security, and fairness in leader rotation. When a view change request is requested, the validator with the highest CLP in the sliding window excluding the current leader is elected to be the new leader.

B. Multithread Transaction Execution

Since serialized transaction execution can cause high latency, in this section, we introduce the parallel multithread transaction execution to accelerate cross-chain transaction processing. As blockchain systems handle transactions and update the ledger in batches, we adopt the snapshot isolation where transaction execution is based on the snapshot of the last block, and each state is stored as $\langle \text{key}, \text{value}, \text{version} \rangle$ in the blockchain. In *DeCross*, transactions are executed concurrently in different threads and then committed. However, this method may induce many read-write conflicts: i) Write-and-write (WW) Conflict. One transaction writes a key and a later transaction reads the key. The stale read occurs since the updating of the key is not included in the reading result of the later transaction. ii) Write-and-read (WR) Conflict. One transaction writes a key and a later transaction rewrites the key. The stale update occurs since the modification operation of the later transaction is based on the key in the snapshot, not the latest value of the key. Normally the write of a key is allowed once in each block to avoid WW conflicts and a high abort ratio.

We address the WW and WR conflicts easily by leveraging the pre-execution in the sequence phase. The key observation is that if each key is strictly written at most once in a block, the conflicts can be eliminated. Thus, in *DeCross*, the leader will prepare two consensus proposals $\{\text{txBatch}\}_{i+1}$ and $\{\text{txBatch}\}_{i+2}$ simultaneously, but broadcast them one by one. If a key in a transaction's write set has been written, the transaction is moved to the second consensus proposal. During the sequence phase, the leader packs transactions into $\{\text{txBatch}\}_{i+1}$ and pre-executes the transactions based on the last snapshot to produce the read and write set. Each version number of a key VeNum is increased by 1 if a transaction writes it. As shown in Algorithm 3, the transactions in $\{\text{txBatch}\}_{i+1}$ are detected using the write set. For each key in a transaction's write set, if the key's version number has been increased, the transaction is moved to $\{\text{txBatch}\}_{i+2}$ to avoid abortion.

VI. Security and Correctness

In this section, we give a detailed analysis of the security and correctness properties of *DeCross*.

A. Security Analysis

Lemma 1: DeCross satisfies reliability: *Proof.* Each ordering block is certified with at least $2f+1$ signatures, of which at least $f+1$ signatures are from honest replicas. An honest replica only accepts one ordering block at a position after the current checkpoint. Based on the quorum intersection, the blocks delivered by honest replicas at the same position in the ledger must be identical.

Theorem 1: DeCross safety (accountability guarantee): All honest nodes within the same chain commit the same sequence of blocks; all honest replicas across different chains commit cross-chain blocks in the leader-defined order.

Proof: To prove the accountability always held in *DeCross*, let S_k be a sequence number where there is an invalid signature in the consensus messages. The auditor can get the first checkpoint transaction that follows S_k that has no invalid signatures in its evidence. If it exists, the auditor can assign blame to all $N-f$ replicas that signed that checkpoint transaction. If no such checkpoint transaction exists, the auditor can assign blame to the responding replicas.

For malicious leaders, accountability is guaranteed by combining the auditing evidence and the view-change protocol. When the leader tampers with the ordering and execution of requests, a view-change is triggered by honest replicas if they do not receive $N-f$ attestations within a timeout after receiving the request. The corresponding evidence fragment is returned to the client to prevent misbehaving view-change proposal.

An auditing request returns all evidence that is necessary for the auditor to assign blame to misbehaving replicas if the receipts reflect any linearizability violation. Auditing ensures that each replayed transaction can be used to reconstruct the corresponding raw block. The checkpoint mechanism ensures consistency of the global ordering across different replicas and provides necessary prerequisites for auditing. All honest nodes commit the same sequence of blocks after applying the deterministic accountability-preserving cross-chain protocol.

Theorem 2: DeCross liveness: All blocks proposed by honest nodes are eventually committed or aborted.

Proof: Given our assumption that the system operates under a partial-synchronous network, both the sequence stage and evidence generation stage can make progress with the participation of $2f+1$ malicious nodes. In the commitment stage, the timer mechanism utilized in the partial-synchronous BFT protocol ensures the liveness of the consensus when leader blocks are missed, while the timer mechanism used during cross-chain validation ensures the liveness in the event of malicious validation leaders. Consequently, all replicas in the cross-chain committee are eventually committed or aborted.

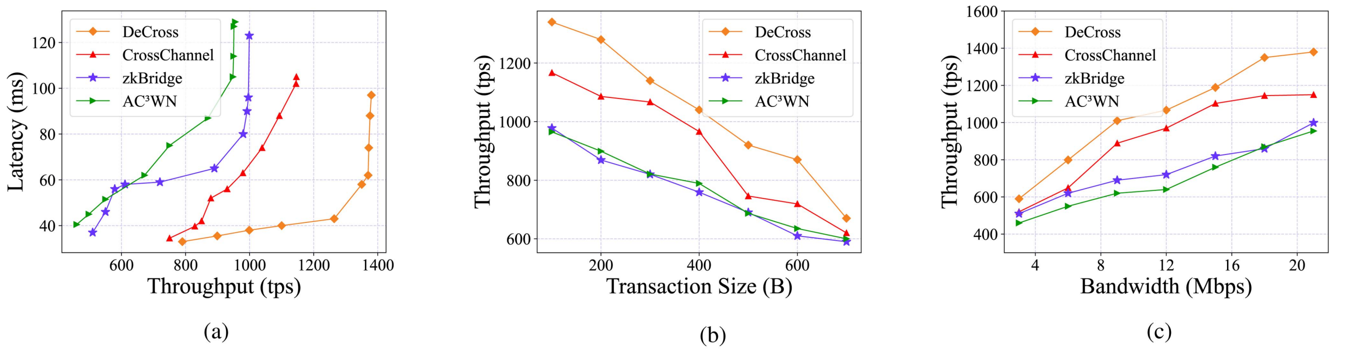

Fig. 5. Performance compared with different cross-chain schemes. (a) Throughput versus latency in different modes. (b) Throughput with various transaction sizes. (c) Throughput in different bandwidths.

B. Correctness Argument

Lemma 2: DeCross satisfies completeness and validity.

Proof: Lemma 1 has already proven that the blocks delivered by honest replicas at the same position in the ledger are identical. Since each ordering block carries at least $2f + 1$ references to blocks from the previous consensus round, honest replicas simultaneously reach an agreement on the referenced blocks from the previous round. Besides, a committed transaction tx has been endorsed by the committed consensus confirmation, which consists of $N - f$ votes, which means it must have been signed and verified by at least $2f + 1$ validators.

Theorem 3: Consistency: If two honest validators are involved in two consensus instances containing blocks with the same sequence number, the two blocks are identical.

Proof: We denote the two blocks as BC_1 and BC_2. If they are processed in the consensus instances with the same view, it holds that BC_1 equals BC_2. If they are committed in two instances with different views, since a view-change message is successfully advanced after collecting $2f + 1$ view-change messages, we could declare that at least one honest replica has verified the block or sent the view-change message. Thus, the conflicting block is ordered in the consensus with a new view with the same sequence number.

VII. Experimental Results

We implement a prototype of the *DeCross* based on Hyperledger Fabric v2.4 and build on both LAN and WAN. Our LAN deployment contains 96 locally connected nodes and the WAN deployment contains 64 nodes on 11 cloud servers. To simulate in a real environment, we extract historical Ethereum transactions and randomly sample 30% to replay using the proposed scheme.

We explore and tune hyperparameters by cross-validation on a rolling basis to achieve the peak cross-chain throughput. The LightGCN model has two hidden layers and each layer has 16 units. The optimal number of layers is small due to the low node feature dimension and the 2-layer network is the most informative through validation. The model is trained every 10 consensus rounds with Adam optimizer for 100 epochs. We set the dropout rate to 0.5 to avoid overfitting. The graph neural network runs on a server with an Intel Xeon (Ice Lake) Platinum 8369B CPU (32 cores), 128 GB of RAM, and Nvidia 3060 GPU.

We consider three baselines: zkBridge [14] achieves secure cross-chain communication through zero-knowledge proofs without trust assumptions. AC^{3}WN [12] is another decentralized cross-chain protocol for atomic transaction commitment. CrossChannel [15] establishes cross-and-off-chain payment channels to realize efficient cross-chain transactions.

1) Overall Performance: We measure the peak throughput results and the corresponding latency. Fig. 5 reports the overall performance of *DeCross* compared to the baselines. We can see that the proposed scheme exhibits higher throughput than the others under all workloads.

Fig. 5(a) illustrates the throughput and the latency. CrossChannel outperforms the other two decentralized cross-chain approaches since it off-loads the cross-chain consensus to the off-chain payment channels thus reducing the on-chain overhead. *DeCross* even achieves higher performance than CrossChannel due to our efficient parallel workflow design. Fig. 5(b) and (c) describes the throughput with varying transaction sizes and bandwidths. Fig. 6(b) describes the throughput with different IIoT applications deployed in the cross-chain system. *DeCross* continuously achieves the highest throughput regardless of changes in the above metrics. Besides, *DeCross* improves the throughput by 38%–44% against the two decentralized schemes and 21% against the centralized scheme.

2) Scalability and Failure Case: In this part, the scalability parameter w represents the scale of the network and the maximum scale is 64 nodes. Fig. 6(a) shows the commit ratio of cross-chain transactions with scaling networks. The commit ratio of *DeCross* maintains a high level continuously (more than 85%), and the latency remains the lowest. The performance degrades slightly with larger network scales due to extra communication costs and network bandwidth constraints. We also conducted experiments under Byzantine failures. In Fig. 6(c), the throughput of our method exhibits the highest level with increasing dishonest nodes. The auditing procedure timely detects collaborative misbehavior and corrects the ledger.

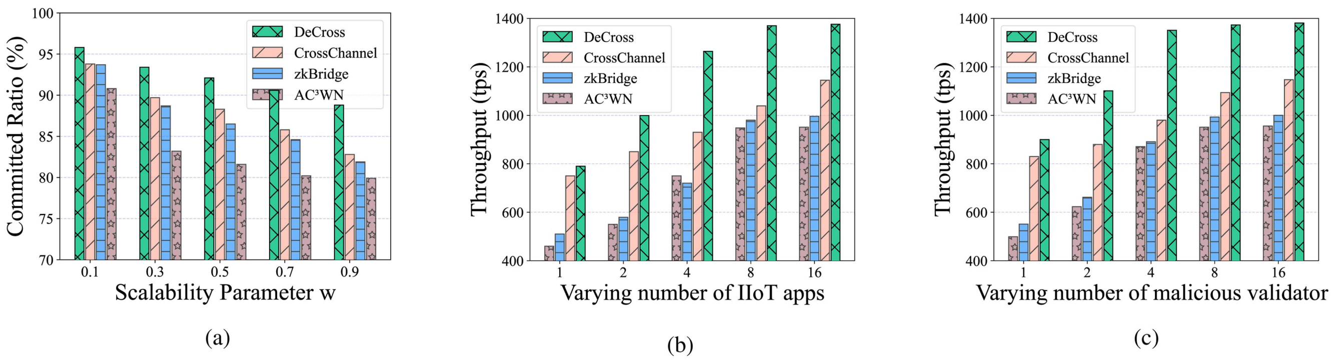

Fig. 6. Performance with varying conditions. (a) Commit ratio with scaling network. (b) Throughput with different number of IIoT apps. (c) Throughput with different number of malicious validators.

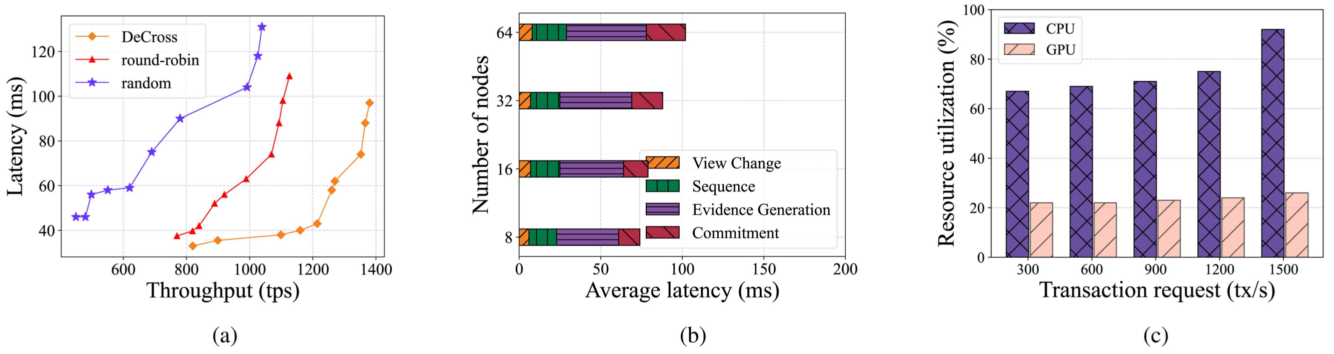

Fig. 7. Consensus Performance. (a) Performance of leader-rotation. (b) Latency of different phases. (c) Resource utilization with varying loads.

3) View-Change and Resource Utilization: Fig. 7(a) presents consensus performance with different leader-rotation methods. Our method outperforms the two baselines in terms of throughput and latency. Besides, our method evaluates nodes in terms of efficiency, security, and fairness simultaneously, which means the elected leader has a higher probability of behaving honestly and correctly. To evaluate the cost of the view change process, we present the average latency of each consensus phase, shown in Fig. 7(b). The latency of the view change phase maintains the lowest. On the one hand, our communication complexity is $O(n)$ since the broadcast pattern is similar to Hotstuff [20]. On the other hand, our CLP of validators is calculated after the sliding window is updated, instead of calculated during leader rotation, which further reduces view change cost. Fig. 7(c) describes the average resource utilization of GPUs and CPUs under different transaction loads. The CPU utilization has a slight increase when the transaction request is less than the peak throughput (about 1400 tps), and has a rapid increase when the load becomes higher because the input rate exceeds the output rate. The average GPU utilization remains at a low level since it collects transactions after several rounds of consensus and then begins processing. The periodic information exchange between the CPU and GPU such as replicating model output and sharing transaction information may induce small resource competition. The probability of scheduling conflicts occurring is low since the CPU and GPU execute relatively independent tasks in our work: cross-chain transaction execution and GCN model training. However, through the experiment, we find the potential to improve GPU utilization, such as scheduling tasks of executing transaction batches in GPU.

VIII. Conclusion

In this article, we present *DeCross*, a cross-chain consensus protocol for multichain collaboration in IIoT. *DeCross* achieves accountable and efficient cross-chain communication by introducing an accountability-preserving cross-chain protocol with parallel transaction processing. Empirical evaluation shows that *DeCross* achieves 44% higher throughput compared to the advanced baseline protocol. In our future work, we will study fairness-aware ordering of the cross-chain transactions.

References

[1] J. Sengupta, S. Ruj, and S. D. Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT," *J. Netw. Comput. Appl.*, vol. 149, 2020, Art. no. 102481.

[2] A. H. Khan et al., "Blockchain and 6G: The future of secure and ubiquitous communication," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 194–201, Feb. 2022.

[3] M. J. M. Chowdhury et al., "A blockchain-enabled circular economy: Collaborative responsibility in solar panel recycling," *IEEE Ind. Electron. Mag.*, vol. 18, no. 3, pp. 45–62, Sep. 2024.

[4] M. B. Mollah et al., "Blockchain for the Internet of vehicles towards intelligent transportation systems: A survey," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4157–4185, Mar. 2021.

[5] N. U. Hassan, C. Yuen, and D. Niyato, “Blockchain technologies for smart energy systems: Fundamentals, challenges, and solutions,” *IEEE Ind. Electron. Mag.*, vol. 13, no. 4, pp. 106–118, Dec. 2019.

[6] V. Buterin, “Chain interoperability,” *R3 Res. Paper*, vol. 9, pp. 1–25, 2016.

[7] N. Shadab, F. Houshmand, and M. Lesani, “Cross-chain transactions,” in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency*, 2020, pp. 1–9.

[8] S. Lin, Y. Kong, and S. Nie, “Overview of block chain cross chain technology,” in *Proc. IEEE Int. Conf. Meas. Technol. Mechatron. Autom.*, 2021, pp. 357–360.

[9] J. Poon and T. Dryja, “The Bitcoin lightning network: Scalable off-chain instant payments,” 2016. Accessed: Jan. 14, 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[10] G. Wood, “Polkadot: Vision for a heterogeneous multi-chain framework,” *White Paper*, vol. 21, no. 2327, 2016, Art. no. 4662.

[11] J. Kwon and E. Buchman, “Cosmos whitepaper,” *Netw. Distrib. Ledgers*, vol. 27, pp. 1–32, 2019.

[12] V. Zakhary, D. Agrawal, and A. El Abbadi, “Atomic commitment across blockchains,” in *Proc. VLDB Endow.*, 2020, pp. 1319–1331.

[13] B. C. Ghosh, T. Bhartia, S. K. Addya, and S. Chakraborty, “Leveraging public-private blockchain interoperability for closed consortium interfacing,” in *Proc IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[14] T. Xie et al., “zkBridge: Trustless cross-chain bridges made practical,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 3003–3017.

[15] X. Luo, K. Xue, Q. Sun, and J. Lu, “CrossChannel: Efficient and scalable cross-chain transactions through cross-and-off-blockchain micropayment channel,” *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 1, pp. 649–663, Jan./Feb. 2025.

[16] G. Zhang and H.-A. Jacobsen, “Prosecutor: An efficient BFT consensus algorithm with behavior-aware penalization against Byzantine attacks,” in *Proc. Int. Middlewave Conf.*, 2021, pp. 52–63.

[17] P. Civit, S. Gilbert, and V. Gramoli, “Polygraph: Accountable Byzantine agreement,” in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 403–413.

[18] A. Haeberlen, P. Aditya, R. Rodrigues, and P. Druschel, “Accountable virtual machines,” in *Proc. USENIX Symp. Oper. Syst. Des. Implement.*, 2010, pp. 119–134.

[19] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, “Towards privacy-assured and lightweight on-chain auditing of decentralized storage,” in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 201–211.

[20] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, “HotStuff: BFT consensus with linearity and responsiveness,” in *Proc. 2019 ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 347–356.