IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received 27 April 2024, accepted 9 May 2024, date of publication 13 May 2024, date of current version 21 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3400672

RESEARCH ARTICLE

A Blockchain Oracle Interoperability Technique for Permissioned Blockchain

ASMA A. ALHUSSAYEN, KAMAL JAMBI, MAHER KHEMAKHEM, AND FATHY E. EASSA
Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia

Corresponding author: Asma A. Alhussayen (aalhussayen@stu.kau.edu.sa)

ABSTRACT Blockchain interoperability has become an essential requirement for the advancement of blockchain technology in numerous fields. Enterprise organizations are increasingly utilizing permissioned blockchains to manage and store their organizations' data and transactions in a private immutable ledger. Interoperability enables permissioned blockchain platforms to communicate and exchange information which is paramount for fully exploiting permissioned blockchains as facilitators for B2B applications. Additionally, the cross-network invocation of smart contracts under agreed conditions enhances business operations. Blockchain oracles can enable permissioned blockchain interoperability and cross-network transactions in a seamless and private manner. However, they have not been studied in the literature as interoperability techniques between permission blockchains. This study proposes a blockchain oracle interoperability technique designed specifically for permissioned blockchain platforms. We presented the architecture of the blockchain oracle interoperability technique and a prototypical implementation to demonstrate the practicality of the proposed technique. In addition, we obtained cross-network transaction latency measurements and analyzed the results.

INDEX TERMS Permissioned blockchain, blockchain interoperability, blockchain oracle, cross-network transactions.

I. INTRODUCTION

Distributed Ledger Technology (DLT) is a digital transaction record system that allows multiple parties in different locations to have access, validation, and data manipulation across a networked database. The DLT infrastructure and protocols were designed to facilitate data record management among several distributed users by recording the history of modifications, ensuring data reliability and providing immutable records. This technology is enabled by cryptography, consensus algorithms, validity rules, governance and smart contracts. Blockchain is the most famous form of DLT that has attracted the interest of many researchers and practitioners leading to technological advancements in the area and the development of various types of blockchains and DLT systems that serve different applications [1].

Blockchain has gained popularity owing to its applications in the field of cryptocurrencies and digital asset management [2]. The proliferation of cryptocurrency blockchain-based applications and the wide variety of digital currencies have necessitated the development of asset transfer and exchange techniques among different cryptocurrencies. In addition, several blockchain platforms have begun to utilize smart contracts to automate the execution of business logic between participants when certain predefined conditions are met. This resulted in extending the application domain of blockchains beyond cryptocurrencies to other industries such as supply chain management, health care insurance, smart city applications and governmental services [3]. Various blockchain systems with various structures and protocols have been designed and implemented to satisfy the distinct requirements of each application domain. According to a survey paper conducted by Khan et al. [4], blockchain systems can be considered homogeneous when they share similar data structures, execution environments and programming language of smart contracts. Heterogeneous blockchains on the other hand have dissimilar data structures, execution environments and

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Destefanis.

use different programming languages to develop their smart contracts.

Interoperation between blockchain networks, whether homogeneous or heterogeneous, soon became an area of interest to many researchers in both academia and industry [5]. Techniques for achieving interoperation and supporting cross-network transactions varied based on the purpose of the interoperation and the type of blockchain network supported. Consequently, diverse interoperability solutions have been developed to address the need for blockchain-based systems to communicate and expand the capabilities of blockchain systems. The majority of the developed techniques were designed to allow asset transfer and exchange between different blockchain platforms, mostly supporting public blockchains and had the disadvantage of increased transaction costs [5]. When it comes to arbitrary data exchange among private blockchain networks, there is noticeable lack in the interoperability techniques designed specifically to allow private blockchain networks perform cross-network transactions and exchange data.

A blockchain oracle is a type of interoperability technique developed to integrate blockchain systems with external sources such as legacy systems, web applications and IoT devices to feed external data to smart contracts within a blockchain network or retrieve data from the blockchain to external systems [6]. Oracles have enhanced the capabilities of smart contracts and enabled blockchain-based systems to be integrated with existing business applications. However, oracles have not yet been studied as an interoperability technique between heterogeneous private blockchain networks to allow for cross-network transactions and arbitrary data exchange. According to a survey paper [7], oracles have the potential to overcome many of the limitations of the existing interoperability approaches, and there is a need to perform an in-depth study and analysis of blockchain oracle performance as an interoperability technique between blockchain networks.

In this study, we present the design of an interoperability technique for permissioned blockchain platforms to communicate and execute cross-network transactions through blockchain oracles. We explain the technique in the context of two widely known permissioned blockchain platforms: Hyperledger Fabric and R3 Corda. To the best of our knowledge, this is the first study to demonstrate blockchain oracles as an interoperability tool for cross-network transactions between heterogeneous permissioned blockchains. Employing the proposed technique does not require any changes in the interoperating blockchain platforms, except for the deployment of interoperability software components to manage cross-network communication and transactions. In addition, the technique preserves the decentralized feature of the blockchain network because each blockchain includes oracle nodes with interoperability components inside the blockchain network itself. In addition, we conducted a performance evaluation to measure the transaction latency of cross-network transactions, and the results show that our proposed solution can achieve interoperability between permissioned blockchains with a minimal increase in transaction latency. The main contributions of this study are threefold:

1. We present the blockchain oracle interoperability technique and its components.
2. We implement a prototype of the interoperability technique and demonstrate its feasibility.
3. We perform an analysis and acquire performance measurements of the cross-network transaction latency.

II. PAPER ORGANIZATION

The remainder of this paper is organized as follows: Section III provides the background on blockchain, blockchain interoperability and blockchain oracle. Then, we discuss research papers that cover private blockchain interoperability and blockchain oracle topics in Section IV. Section V describes the research methodology followed by a detailed explanation of the Blockchain oracle interoperability technique in Section VI. Then, we present the prototype implementation of the proposed solution and the performance evaluation in Section VII. We then compare the features of our interoperability technique with the techniques discussed in the related work in Section VIII. Next, we discuss the limitations and future work in Section IX. Finally, we conclude the paper in Section X by summarizing the results of the study and future work.

III. BACKGROUND

This section provides the background information required to understand the concepts investigated in this study. We begin with the definition of blockchain, its key elements and different types of blockchain networks. Next, a definition of the interoperability of blockchain systems is presented along with different approaches to interoperability. Blockchain oracles were subsequently introduced.

A. BLOCKCHAIN NETWORKS

Blockchain is a distributed database system that stores validated transaction records in a linked list called chains [3], [1]. Transactions stored in blocks are known to be immutable and publicly available for participating peers to view [2]. Blockchain has gained popularity as an enabling technology for Bitcoin, which was introduced by Satoshi Nakamoto in 2008. Blockchain was a decentralized network of anonymous peers participating in submitting transactions and reaching consensus through the Proof of Work (PoW) protocol.

Security, immutability, transparency, independence, and anonymity are strong features of blockchains that make cryptocurrency systems possible [8]. The blockchain structure that enabled these likable features also gave blockchain certain disadvantages, namely, high cost and low transaction throughput, which limited its applicability to specific application domains. Different blockchain network structures and consensus mechanisms have been developed to meet the specific requirements of additional application domains such as business applications, governmental services, and hospital management systems.

Currently, blockchain platforms are classified into three categories, and each category is suitable for a specific set of application areas. The categories of blockchain platforms are as follows:

1) PUBLIC (PERMISSIONLESS) BLOCKCHAINS

Public blockchains are the first blockchains to appear; they are also known as permissionless blockchains because any one can join the network and contribute without the permission of a central authority. Public blockchains are completely decentralized and offer openness and transparency such that any peer node can participate in the reading, writing, verification, and consensus processes of transactions submitted and stored in the chain. Economic incentives are used to reward participating nodes as encouragement for their role in mining activities to verify transactions and reach a consensus [9]. Public blockchains also offer anonymity; hence, transactions are not traceable to specific users. In addition, public blockchains are known to be trustless and tamperproof because of the distributed ledger stored at each node. Therefore, if any node holds a modified copy of the ledger, it is considered corrupt and rejected by the majority of the peer nodes. Consensus mechanisms are an important part of public blockchains and play an essential role in their performance and security. The two most prevalent methods for reaching consensus in public blockchains are Proof-of-Work (PoW) and Proof-of-Stake (PoS). Furthermore, the source code is usually open source, and anyone can verify transactions, detect bugs, and propose changes because no valid record or transaction can be changed on the network [10].

2) PRIVATE (PERMISSIONED) BLOCKCHAINS

Private blockchains, also known as permissioned chains, are centralized blockchains with a central authority granting permission to peer nodes and authorizing access to data. Unlike public blockchains, only authorized nodes can join the network and participate in the management of data on the chain. Optionally, some private blockchains allow public non-permissioned nodes to read data for auditing in specific organizations. These characteristics of private blockchains limit their application to business applications involving a small group of entities [9]. Similar to public blockchains, private blockchains are peer-to-peer networks with distributed ledger that offer immutable and transparent storage of transactions. Limiting access to permissioned nodes makes private blockchain networks much smaller than public blockchain networks. The consensus mechanisms of private blockchains are also different from those of public blockchains because only authorized peer nodes inside the organization participate in the consensus process rather than anyone with computing power [10]. Proof-of-Authority (PoA) and Practical Byzantine Fault Tolerance (pBFT) are two of the most widely used consensus algorithms for private blockchains.

3) HYBRID (CONSORTIUM) BLOCKCHAINS

A hybrid blockchain is a combination of both public and private blockchains in a network. In a hybrid blockchain, the network is partially distributed such that every participating node has a copy of the ledger and can commit transactions but only authorized permissioned nodes participate in the block generation and consensus process. This type of blockchain has emerged as a solution for organizations that share common data and transactions with other organizations, but also has private data that require restricted access [9]. A hybrid blockchain employs smart contracts to verify transactions and records and maintain confidentiality of the information inside the network. The identity of a member node is only revealed to peer nodes that share common transactions. Hybrid blockchains usually incorporate a mix of consensus algorithms to achieve the security, scalability, and decentralization features required in this type of blockchain network. For example, PoW may be used for public access and PoS for permissioned access, thus providing both security and efficiency. The combination of public and private blockchain features has allowed the wide adoption of hybrid blockchains in various application areas. The Hyperledger Project is a prominent example of a hybrid blockchain platform designed for business solutions [10].

Each category described above has certain challenges and limitations. Public blockchains suffer from limited scalability, a lack of privacy, and high computational power. Private blockchains avoid the limitations of public blockchains but are only accessible to a set of people and are costly to set up and maintain compared to public blockchains. Although hybrid blockchains are designed to combine the features of public and private networks, they do not offer complete transparency for the public users of the network, and they offer fewer incentives than public blockchains [10], [11]. Among these categories, private blockchains provide the most secure and privacy-preserving blockchain solution for organizations that require enhanced privacy and security, while providing immutable records of transactions for auditing and inner organization management.

B. BLOCKCHAIN INTEROPERABILITY

As blockchain technology advances with different technological approaches and architectures to serve a wide range of application areas, it has become imperative to develop interoperability solutions that allow blockchain based systems to communicate with each other. In recent years, research studies have discussed blockchain interoperability definitions, types, and techniques to propose an interoperability solution that enables communication between different blockchain systems. Blockchain interoperability allows blockchains to communicate with each other and with external systems and hardware, such as legacy systems and IoT devices.

In [12], Abebe et al. defined blockchain interoperability as the ''semantic dependence between distinct ledgers to transfer or exchange data or value, with assurance of validity.'' In [13], Hardjono et al. presented blockchain interoperability from an application-level view, as they focused on the completion of a transaction and its sub-transactions, independent of the blockchain systems executing the transactions. Pang in the research study presented in [14] provides a broad definition of blockchain interoperability that goes beyond cross-chain asset transfers and includes the ability to execute smart contracts across heterogeneous blockchains in a seamless and secure process. A comprehensive definition of blockchain interoperability is provided by the National Institute of Standards and Technology (NIST) in their technical report in [15], where they describe blockchain interoperability as ''a composition of distinguishable blockchain systems, each representing a unique distributed data ledger, where atomic transaction execution may span multiple heterogeneous blockchain systems, and where data recorded in one blockchain are reachable, verifiable, and referable by another possibly foreign transaction in a semantically compatible manner.''

Blockchain interoperability enables the wider adoption of blockchain networks in various business applications that need to integrate different blockchain systems to meet their business requirements. Various interoperability techniques have been developed to fulfill the demands and features of interoperability required by different application areas. As a result, three categories of interoperability techniques were identified: cryptocurrency-directed interoperability approaches, Blockchain Engines, and Blockchain Connectors [16]. The following is a brief description of each category:

1) CRYPTOCURRENCY-DIRECTED INTEROPERABILITY (PUBLIC CONNECTORS)

Techniques in this category are designed specifically to allow digital asset transfers/exchanges using protocols of public blockchains. Three known methods belong to this category: side chains, notaries and Hashed Time Locks [16], [17], [14]. These methods were designed with public blockchains in mind and therefore assume that gateways are not trusted; and as a result, they are not suitable for permissioned (private) blockchain networks.

2) BLOCKCHAIN ENGINES (BLOCKCHAIN OF BLOCKCHAINS - BOB)

Blockchain of Blockchains are similar to the side chains method in that they connect secondary chains to the main chains. However, interconnected blockchains are customized to interoperate and are built for specific applications. BOB platforms support flexibility, high throughput, and compatibility, but only blockchains built using their platform and protocols may interoperate. The Cosmos and Polkadot platforms are popular examples of BOBs [16].

3) BLOCKCHAIN CONNECTORS (HYBRID CONNECTORS)

As the name suggests, interoperability between public and private blockchains is allowed in this approach. Techniques in this category utilize DApps to communicate with participating blockchain networks through a layer of abstraction that incorporates a group of standard procedures. ''Blockchain Migrators, ''Blockchain Agnostic Protocols'' and ''Trusted Relays'' are popular example of interoperability methods under this category [16].

Kayıkcıet al. [18] described a different categorization of available blockchain interoperability techniques, as defined by the World Economics Forum in their white paper published in 2022:

4) CROSS AUTHENTICATION

This includes methods that enable interoperability between different blockchain networks or between blockchain and other existing systems. Three known methods belong to this category: 1) Notaries, which is a centralized method that allows both asset and arbitrary data transfer; 2) Relays, which use side chains and are completely decentralized for asset and arbitrary data transfer, but require the use of specific developed protocols and platforms; 3) Hash-Locking, an efficient technique that has gained wide adoption but only supports digital asset exchange.

5) ORACLE

Oracle nodes are used to provide external data to the blockchain or extract data from the blockchain through the implementation of specific smart contracts. They are considered by several researchers to be interoperability techniques between blockchain systems and other non-blockchain systems. However, a recent survey paper published by Ezzat et al. in 2022 [7] presented blockchain oracles as a promising interoperability technique between blockchain systems and compared it with existing techniques.

6) API GATEWAYS

In this approach, several APIs are developed to act as gateways that interact with participating blockchain platforms and applications to perform assets or arbitrary data exchange. APIs are programs written to control and govern access to participating blockchain networks and perform cross-authentication between interacting blockchains. Although APIs are easy to develop, they centralize trust and are unsuitable for a wide range of blockchain networks.

In addition to the categorizations described above, Hewett et al. [17] classified the data types included in blockchain interoperability as either digital asset exchange or arbitrary data exchange. For digital asset exchange, interoperating blockchains transfer or swap assets in a decentralized system without an intermediary. Arbitrary data exchange involves cross-network transaction execution and external smart contract invocation between interoperating blockchains, which results in the transfer of data or the modification of stored data on participating blockchains.

C. BLOCKCHAIN ORACLE

Oracles are trusted nodes that provide external data feed to blockchain networks. Blockchain oracles were first used to provide real-time data from external sources (typically web based sources) to blockchain networks to be used during smart contract execution. Later, oracles were used to push data into external sources from blockchain networks [19]. The correction of smart contract execution is therefore dependent on the correctness of the fetched data, which makes it critical for blockchain oracles to verify the validity and authenticity of the provided data. The operation of blockchain oracles and the type of data being fetched to or pulled out of the blockchain network relies on the Service Level Agreement (SLA), instructions, and end points [20]. A popular example of a blockchain oracle system is Chainlink [21], which is a decentralized data-feed oracle system. The aim of Chainlink is to fetch external data and verify its authenticity for smart contracts using a network of computers and incentives as motivation.

The main advantage of Oracles is that they are easy to implement and integrate into any blockchain system [17]. Nevertheless, they have several disadvantages. First, oracle nodes are considered as central agents between blockchains and external data sources, thus centralizing trust. Second, oracle nodes present reliability issues because they are considered as single points of failure of the system. Third, oracle nodes gather data from multiple data sources, which poses some computational complexity and lead to poor performance [17].

Various blockchain oracles are used and can be categorized based on different characteristics. According to Beniiche [22], there are three different classifications of blockchain oracles. When considering the network administration of nodes, oracles can either be centralized or decentralized, as described in the following:

- **Centralized Oracles:** A single oracle node is utilized to run on a single server and gather data from a single data source. This type of Oracle system leads to a single point of failure because it is the main source that provides the required external data to smart contracts.
- **Decentralized Oracles:** designed to solve the single point of failure limitation, a distributed network of oracle nodes running on different servers gathers data from multiple data sources, thereby delivering more reliable and trusted data to the blockchain.

Another categorization of Oracle systems depends on the data source from which the external data are gathered, which can be of three types:

- **Software Oracles:** In this type, data are gathered from software systems, such as cloud servers, APIs, websites, or other blockchain systems.
- **Hardware Oracles:** This type of oracle system delivers data collected from the real world, such as from sensors, medical equipment, QR scanners, RFID tags, robots, and IoT devices.
- **Human Oracles:** Experts and specialists' input.

The final categorization considers the direction of data flow and identifies two types:

- **Inbound Oracles:** Pull data from data sources (off-chain) to smart contracts (on-chain).
- **Outbound Oracles:** push data from smart contract to the external world.

Many researchers have directed their studies towards developing blockchain oracle systems that integrate blockchain networks with other non-blockchain systems and have not considered blockchain oracles as an interoperability technique between blockchain networks. Ezzet et al. [7] presented blockchain oracles as a possible blockchain interoperability technique and discussed its capabilities and compared it with other existing interoperability techniques. The study also discussed design implementations to ensure the reliability and trust of blockchain oracles as an efficient interoperability technique.

IV. RELATED WORKS

In this section, we discuss research papers that have studied interoperability techniques between private blockchain networks, followed by related work on blockchain oracles in the area of blockchain interoperability.

A. PRIVATE BLOCKCHAIN INTEROPERABILITY

Private permissioned blockchain systems are suitable for Business-to-Business (B2B) applications because they apply strict privacy and security measures and deliver business requirements with smart contracts and workflows. Organizations are increasingly showing interest in private blockchain systems to store their data and manage and monitor business transactions. Therefore, it is pivotal to design an interoperability technique capable of allowing business transactions between private blockchain systems in a seamless and secure manner. However, research on interoperability solutions for private blockchain networks is limited. Among the studies on blockchain interoperability, we selected those that focused on permissioned blockchain platforms. The following is a summary of related work on private blockchain interoperability and its limitations.

Ghaemi et al. [23] proposed an interoperability solution for permissioned blockchains, based on the publish-subscribe design pattern. Interoperation is achieved between a source blockchain and a destination blockchain through an intermediary blockchain (broker). To avoid centralization, peers from participating blockchains participate in the governance of the broker blockchain. A participating blockchain can be a *publisher* in the interoperability process and create *topics* to share specific information and update it when needed. *Subscriber* blockchains can select *topics* to subscribe to and view published information. The *broker* blockchain is responsible for storing published topics and notifying *subscriber* blockchains when the *topics* are updated. A prototype of the proposed solution was implemented, and an evaluation was conducted to assess the performance and measure transaction throughput and latency. The interoperability method described in this paper only deals with data sharing but does not allow cross-blockchain transactions or smart contract invocations.

Dinh et al. [24] described a blueprint for interoperable blockchains by focusing on the requirements of private blockchain networks. The proposed architecture is designed to achieve cross-chain transactions and communication with controlled access. The designed solution adds three components for handling interoperability-related functionalities to the existing blockchain software stack, thereby avoiding the need to build a new blockchain software system. They presented a high-level description of a general-purpose interoperation approach for private blockchain networks with access control. A detailed description of the proposed architecture and its components is required, and further research is required to evaluate the efficiency of the proposed solution.

Khan et al. [4] conducted an extensive survey of blockchain interoperability techniques for public and private blockchains. An analysis of the interoperability techniques and requirements revealed that complete interoperability can be achieved through identity management, consensus mechanisms, cryptographic management, and smart contracts of interoperating blockchain systems. They further stated that smart contracts are the most critical part of private blockchain interoperability. In addition, they identified privacy, security, scalability, degree of confidence, and direction of transactions as characteristics of the interoperability techniques by which they are compared and analyzed. The study revealed that there is a lack of research analyzing and evaluating smart contract-based interoperability for blockchain systems.

Bayraktar and Gören [25] studied private blockchain interoperability to provide design guidelines and principles for developing interoperability systems among private blockchains. In addition, they described an interoperability architecture that was developed following the identified design principles. The researchers proposed the publish-subscribe method as the most appropriate for private blockchains' interoperability and suggested the following features for the interoperability of smart contracts:

1. Smart contract to publish the desire to become a member in interoperability system.
2. Smart contracts to publish the services available for external blockchains.
3. Authorization smart contracts that allow invited users to interoperate with participating networks.
4. Smart contracts to subscribe to published services by member blockchains.
5. Smart contracts to become a member of the end-to-end chain.
6. Smart contracts to sign up as a member chain to trace the end-to-end process.

The developed architecture for interoperability aims to avoid centralization and single point of failure by deploying the interoperability solution on each interoperating blockchain. The solution is described at a high level and consists of publishing and subscription modules and separate ledgers for interoperation transactions. This study can be considered a good reference for developing a more detailed interoperation architecture, but it is only applicable to homogeneous blockchains.

Bradach et al. in [26], acknowledged the lack of interoperability solutions for permissioned blockchain platforms and proposed an interoperability gateway following the design principles suggested by Abebe et al. [12] and Ghaemi et al. [23]. The design of the solution was based on a social security exchange scenario between Hyperledger Fabric and Corda. The interoperability gateway comprises three main components: two connectors (one for each platform) and a router. The router coordinates communication between the two platforms and employs a static blockchain discovery mechanism. The connectors are implemented to communicate with the blockchain platform they belong to and provide an interface to communicate with the router. The proposed solution focuses on routing messages from one blockchain to another but does not provide details on handling transactions and replying to incoming queries. Important features of blockchain, such as consensus and validating transactions, have not been discussed. The study also lacks performance evaluation of the implemented prototype.

In [27], the researchers continue their study on the gateway-based solution described previously by Bradach et al. in [26]. They describe additional components to their solution to make it more general and applicable to wider range of blockchain platforms. Additionally, they provide an initial prototype evaluation to test the effectiveness of the gateway-based solution. Two frameworks, LCIM and CBIDD, were used to assess the solution, providing different perspectives on its effectiveness and suitability for permissioned blockchain interoperability. The LCIM model helped determine the level of interoperability achieved by the solution, while the CBIDD framework verified that the gateway-based solution was appropriate for the applied scenario. The gateway-based approach is simple to use and does not require any change to participating blockchain networks. It is however a centralized solution which does not comply with the essential decentralization property of blockchain systems.

The authors in [28] introduce T-ODAP, a secure protocol designed to enable secure asset exchange between permissioned blockchains without needing a trusted third party during the interoperation process. The proposed protocol uses a Decentralized View Storage (DVS) implemented with Polkadot and a connector built with Hyperledger Cactus. The paper models participants as rational agents using game theory, which helps in understanding how participants are likely to behave in this system, ensuring that the protocol is designed in a way that encourages participants to act honestly.

The system was evaluated to test its robustness against attacks, ensuring that it can withstand malicious attempts to disrupt that asset transfer process. The proposed solution is complex and costly compared to simpler counterparts and the paper described an initial implementation of the solution. A full implementation and performance evaluation of this interoperability technique is required to assess its feasibility. Also, the use of Polkadot and Hyperledger Cactus for implementing the main components of the system limits the applicability of the solution to supported blockchain platforms.

PIECHAIN is a Kafka-based interoperation framework described in [29]. The aim of the designed framework is to allow cross-chain read/write transactions between participating blockchains focusing on ensuring the atomicity and liveness feature required in such transactions. In addition, the researchers acknowledge the importance of eliminating the need for a trusted third party to facilitate the cross-chain transactions by replacing it with cross-chain services that communicate through an event log making the process more efficient and reliable. The researchers showcase a practical use of PIECHAIN through a cross-chain auction, demonstrating how users with tokens on different blockchains can participate in bidding for an asset on another blockchain, highlighting blockchain's real-world application. The approach described in this paper is limited in its description to auction applications and allowing asset exchange, while it could be explored as a method of interoperability for wider range of applications. Further research is required to evaluate the performance of this approach and compare it with existing methods.

1) DISCUSSION

Although existing interoperability solutions for private blockchains have succeeded in meeting the requirements of interoperability, they have several limitations that hinder their application. Some solutions require interoperable blockchains to have some degree of compatibility between them. Others are designed only to support asset exchange, which limits their application area. In addition, the majority of the studies described in the literature lack detailed design and architecture descriptions and require thorough evaluation to acquire performance measurements. Among the interoperability approaches described in the reviewed literature, the publish-subscribe approach [25], [23] for private blockchain interoperability appears to be the most promising and fulfilling for business applications. The solution proposed in this study explores blockchain oracles as an interoperability tool between permissioned blockchain networks as an alternative to existing techniques. Oracles do not require compatibility between interacting blockchain systems, and they achieve both digital and arbitrary data exchange. In addition, blockchain oracles are relatively easy to implement and do not require changes to blockchain platforms.

Similar to our approach, is the gateway-based technique described in [26] and summarized in the related work section. Although their approach is based on Hyperledger Fabric and Corda, their design lacks some important features of blockchain interoperability such as the ability to execute cross-network write/update of a state in participating blockchain networks. Also, the ability to handle failed transactions was not included in their design. Their evaluation of the implemented prototype was limited to functional testing without conducting performance testing and analysis. Our research study aims to explore the feasibility of blockchain oracles as interoperability technique by describing in detail the design of the main components and their interactions to execute both cross-network read and write transactions between participating blockchain networks. In addition, we conduct a performance evaluation to measure the transaction latency, which is an important metric to be considered when assessing the feasibility and applicability of an interoperability solution.

B. BLOCKCHAIN ORACLE

Many of the related works in the blockchain oracle focus on guaranteeing the validity of the gathered data and ensuring trust in the blockchain oracle. The following is a summary of related work in blockchain oracle.

The research paper in [30] described a blockchain oracle framework designed to guide blockchain developers who need to employ blockchain oracles to provide external data to and from the blockchain network. The researchers conducted a systematic literature review (SLR) to explore the different origins of information used by oracles, validation techniques of gathered data, encryption mechanisms of incoming and outgoing data, properties of oracles, and how they can be integrated with blockchain platforms. According to this study, smart contracts and software modules are the most widely used technique for integrating blockchain oracles with blockchain platforms. The results of the SLR were summarized in a table to guide developers and provide a reference for different techniques to utilize and integrate blockchain oracles with a specific blockchain platform. For permissioned blockchain-based systems, the framework suggests using custom-developed smart contracts to integrate a single-source Oracle and using a third-party validation technique to validate the data.

Researchers in [31] proposed a Decentralized Oracle Network (DON) to act as a trusted source of external information gathered from different web sources and verified by a network of oracle nodes. In addition to the nodes that gather and verify data, bridge nodes are used to integrate the oracle network with public smart contract platforms, such as Ethereum. Bridge nodes are used to listen to and detect requests for external data on the Ethereum network, format the requests, broadcast them on the DON, and then gather and submit the result to the Ethereum network. The interoperability technique with the public blockchain, Ethereum, was only briefly discussed, giving key points on how to achieve interoperability without technical details.

Beniiche [22] studies different types of blockchain oracles, how they were classified, their technical architecture, design patterns and the use case scenario of each type. The main focus of this study is the trust issue known in blockchain oracles and existing solutions to verify the authenticity of the data feed. They explain two examples of widely used blockchain oracles: Oraclize and Chainlink. The researcher further discusses the human oracle type as a solution to the centralized trust issue and how human oracles can strengthen the trust of fetched data.

In [6], the researchers study the different blockchain oracles technologies developed to solve what is known as the ''oracle problem,'' which refers to the challenge of verifying the integrity and correctness of the data feed. In this research, blockchain oracle implementation techniques are categorized in to two main types: voting-based approaches and reputation-based approaches. The voting-based approach relies on a group of users called verifiers or certifiers who vote on the correctness of certain data and are rewarded if their vote is correct. The reputation-based approach consists of two main processes. First, an authenticity-proof technique is utilized to assess the trustfulness and integrity of the gathered data. In each category, variable methods were developed to overcome the security weaknesses introduced by blockchain oracles. Researchers have also presented several challenges that required further research, such as increasing the performance speed of blockchain oracles, reducing costs, enhancing security, and managing different query types. Although this paper presented a thorough investigation of blockchain oracle techniques, their focus was on enhancing the trust of the data feed and quarantining security, specifically for public blockchains.

Another study discussing the challenge of enhancing trust in blockchain oracles was conducted by Al-Breiki et al. [32]. The researchers identified four main characteristics of blockchain oracles, by which they were categorized. Blockchain oracles can be classified as software, hardware, or human oracles when considering the data source. Depending on the number of oracle nodes utilized to provide the data feed, the blockchain oracle can implement either a centralized trust model or a decentralized trust model. Request-response, publish-subscribe, and immediate-read are three design patterns chosen depending on the type of data needed by the blockchain system. Considering the direction of the data feed, blockchain oracles can be either inbound oracles (insert data from external sources into smart contracts) or outbound oracles (deliver data from smart contracts to external systems). Existing widely used blockchain oracle systems such as Provable, TownCrier, and Chainlink are explained in detail, discussing their trust-enabling mechanisms, strength and weaknesses. Several challenges and research areas have been identified, such as the need for a decentralized and reliable design of oracle systems, and the need for decentralized identity management and registration for the oracle nodes to increase trust.

Mühlberger et al. [33] performed an analysis to measure the performance of four oracle system design patterns. Design pattern categorization is based on the direction of the data flow (inbound or outbound) and the initiator of the data flow (push or pull) from the point of view of the blockchain system. The researchers implemented proof-of-concept implementations of four oracles on an Ethereum test network and acquired transaction latency and cost measurements that helped characterize each design pattern. The results showed that oracles that push data into the blockchain have higher transaction latency than pull-based oracles, which read data from the blockchain. The characterization of the oracle design patterns was based on their performance on an Ethereum test network. Further examination of the design patterns on different blockchain networks may lead to different performance results.

Another study focusing on enhancing the trust and reliability of blockchain oracles was conducted by Liu and Feng [34]. This paper proposes a new blockchain oracle method based on the aggregation signature technique of Bohen-Lynn-Shacham (BLS) to provide reliable off-chain data. The method consists of five main steps: sending the data request to all the oracle nodes, collecting signatures from oracle nodes, committing the query data, verifying the validity of the oracle signatures, and finally performing the aggregation signature function. An analysis of the system performance was performed on an Ethereum private network to measure the cost and time consumption of the aggregate signature method. This method verifies each signature before performing aggregation; therefore, it consumes more time and gas as the number of oracle nodes increases.

Pupyshev et al. [35] proposed GRAVITY architecture, for an interoperability system that enables cross-chain communication between different blockchain platforms and interoperability with other external systems through oracles. The proposed architecture comprises a network of oracle nodes that own accounts in all participating blockchain platforms in addition to software modules that perform data extraction, task scheduling, and manage communication with supported blockchain networks and an internal ledger. Two consensus algorithms are designed: an Oracle consensus is implemented in the target blockchain to decide which data feed is valid, and a Pulse consensus is implemented in the GRAVITY network to select a set of oracle nodes that verify the data based on their reputation. The researchers provided a detailed description of the components of the proposed system, oracle smart contracts, and workflow pattern. The described interoperability system is suitable for public blockchains and complex to implement. Additionally, performance analysis is required to measure the latency of such a complex system.

Ezzet et al. [7] conducted a thorough study on blockchain oracle capabilities as an interoperability technique and compared it with existing popular blockchain interoperability techniques. According to their study, many blockchain interoperability solutions suffer from some limitations, such as requiring some degree of compatibility between blockchain platforms, introducing central authority, and supporting digital asset exchange without support for arbitrary data exchange. The researchers examined the blockchain oracle tools that exist in the literature and presented a detailed comparison between them, highlighting their strengths and weaknesses. The study concludes that blockchain oracles need to be studied and examined as an alternative interoperability tool to the existing techniques. In addition, performance analysis studies are required to measure the main aspects of blockchain oracle interoperability systems such as transaction latency, scalability, and throughput.

1) DISCUSSION

So far, published studies on blockchain oracles have mainly focused on validating incoming and outgoing data and providing a trusted source of data to blockchain networks. Although the benefits of blockchain oracles as interoperability techniques are apparent and can overcome some of the challenges of existing interoperability techniques, there are no published research studies that describe such an interoperability system. To the best of our knowledge, blockchain oracles have not yet been studied or analyzed as interoperability tools between permissioned blockchain systems. In this paper, we present interoperability based on blockchain oracles between two widely used permissioned blockchain platforms, and we perform a performance analysis to evaluate the transaction latency of cross-network transactions.

V. RESEARCH METHODOLOGY

This research study is prepared with the aim to explore the feasibility of an Oracle-based system to allow interoperability between two permissioned blockchain networks. The study is conducted in the following steps:

1. The requirements for the Oracle-based interoperability system are explained in the context of two widely used permissioned blockchain platforms: Hyperledger Fabric and R3 Corda.
2. The main operations for achieving interoperability through the Oracle-based system are defined.
3. Using the five-layer architecture of blockchain platform, the main components responsible for delivering the interoperability operations are explained in detail using Hyperledger Fabric and Corda as examples to simplify the explanation.
4. Detailed description of the interactions between the components of the interoperability system are also described.
5. Performance evaluation is conducted to measure the transaction latency using a prototypical implementation of the designed Oracle-based system with minimal network setup for Hyperledger Fabric and Corda.
6. Test cases are designed to measure both local and cross-network transactions for read and write operations to compare between the local transaction latency and cross-network transactions and analyze the results.

VI. BLOCKCHAIN ORACLE INTEROPERABILTY SYSTEM

Blockchain networks have reached multiple application areas, leading to the development of numerous blockchain platforms that target specific requirements. Hyperledger Fabric [36] is a leading permissioned blockchain platform that target all types of business areas. The platform provides high security and performance, as well as access to management tools and support for business logic implementation. Several projects based on Fabric are already in production with a growing number of participating parties and operations. Famous members of Hyperledger Fabric that have running blockchain systems include Walmart, FedEx, and Visa.

Another widely used permissioned blockchain is the R3 Corda framework [37], which was designed and built for regulated financial institutions. Corda provides strong privacy assurance through its distinct architecture, where each peer has its own view of the ledger, storing only a subset of the ledger. To offer a strong degree of security and privacy, communication in a Corda network is P2P with any broadcast having to use the Network Map Service to loop over. Several large companies use the Corda blockchain, such as Citibank, HSBC, and Metlife Insurance.

Both the Hyperledger Fabric and Corda platforms are becoming important facilitators for large business and financial corporations. Consequently, their interoperation is an evident requirement for these businesses to conduct daily transactions seamlessly. Therefore, the Oracle based interoperability technique described in this section enables communication between the Hyperledger Fabric network and the Corda network through cross-network transactions that perform read and write operations on both ledgers while maintaining ledger consistency. Although the blockchain oracle interoperability technique is explained in the context of Hyperledger Fabric and R3 Corda, it can be applied to any permissioned blockchain platform. The following assumptions are made for the proposed solution:

1. Both networks involved in the oracle interoperability system recognize and form a common understanding of the cross-network transactions performed on both networks.
2. Participating blockchain networks' security is dependent on each network's design and settings.
3. The users involved in the cross-network transactions trust each other and accept each other's validity proofs.

Generally, a blockchain platform is built following a five-layer architecture designed to deliver the basic specifications of a blockchain network. Hyperledger Fabric and Corda both follow the five-layer architecture, although they have different concepts of application and vision. Similarly, the Oracle-based interoperability system is designed based on the same five-layer architecture to ensure that the specifications

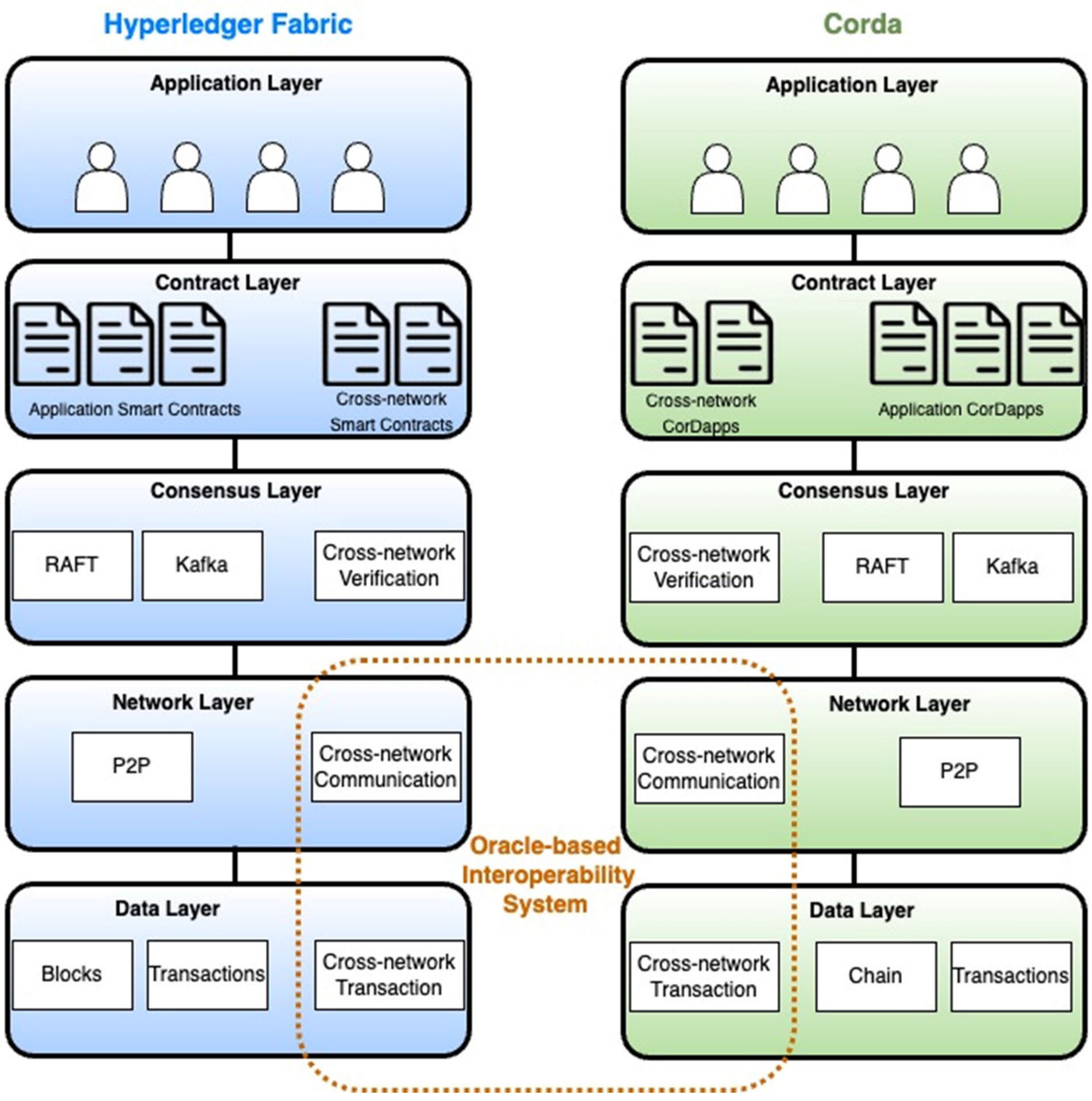

Hyperledger Fabric
Corda
Application Layer
Contract Layer
Application Smart Contracts
Cross-network Smart Contracts
Cross-network CorDapps
Application CorDapps
Consensus Layer
RAFT
Kafka
Cross-network Verification
Network Layer
P2P
Cross-network Communication
Oracle-based Interoperability System
Data Layer
Blocks
Transactions
Cross-network Transaction
Chain

FIGURE 1. Layered architecture of Hyperledger Fabric and Corda platforms with the components of the Oracle-based Interoperability system.

of both blockchain networks are delivered, as shown in figure 1. The basic operations performed in the oracle-based interoperability technique are as follows:

1. Listen for events on both Corda and Hyperledger Fabric.
2. Identify cross-network events emitted from both networks.
3. Handle cross-network events and create transactions that perform read or write operations on both networks.
4. Validate transactions and ensure executed transactions are stored in both networks.

As shown in figure 1, The Oracle-based interoperability technique is divided into two main parts:

- First, cross-network transactions need to be defined and implemented in the Contract layer and the rules for verifying and validating them are included in the Consensus layer of both the Hyperledger Fabric and Corda platforms.
- Second, the Oracle Service node includes two main components that interact with the Network and Data layers of both blockchain platforms. The Cross-network Communication and Cross-network Transaction components are responsible for establishing connections with Hyperledger Fabric and Corda peer nodes and for handling requests between the two platforms.

The Cross-network Transaction and Cross-network Communication components in the Oracle Service node are shown in more detail in the conceptual diagram in figure 2,

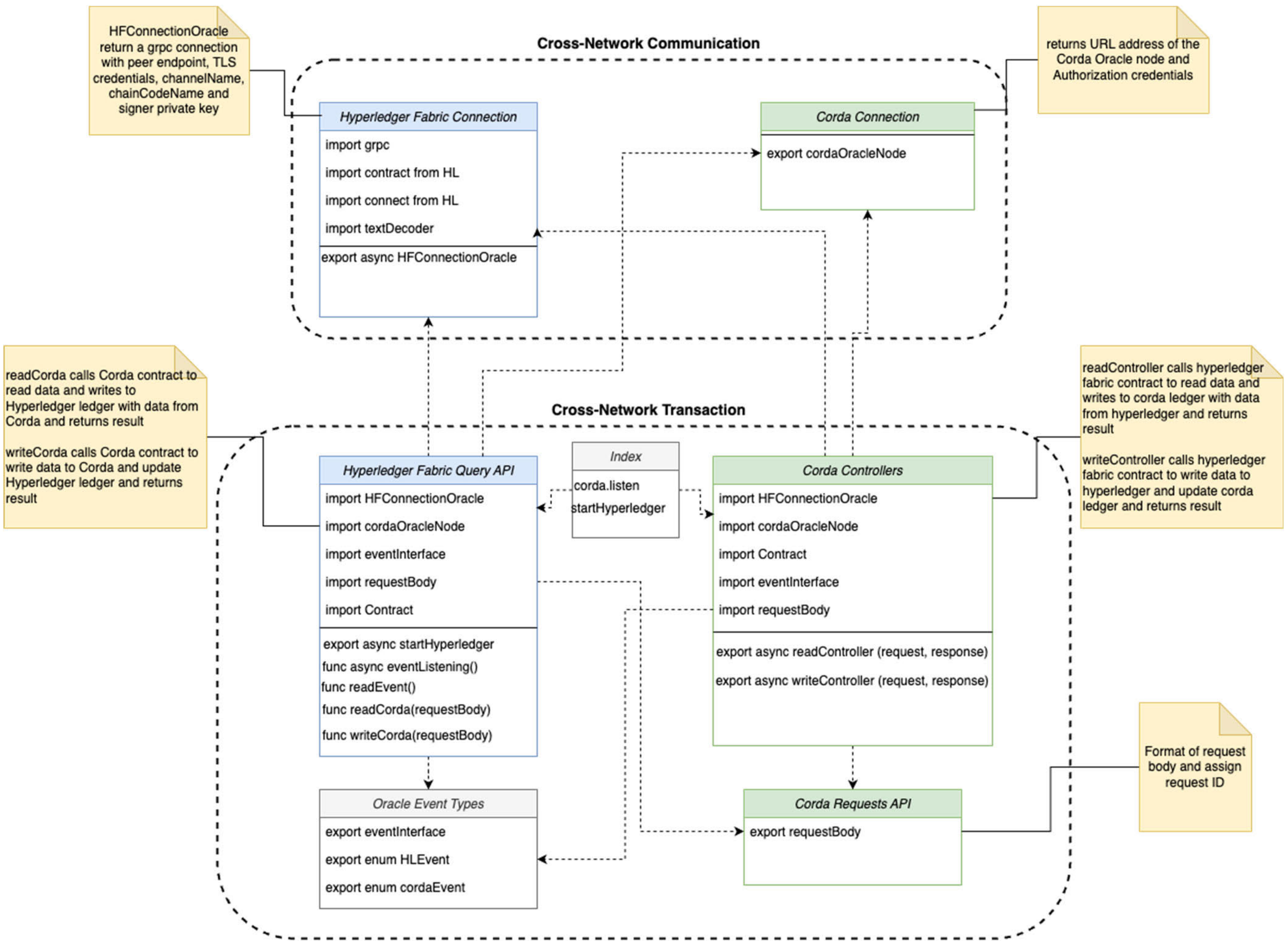

HFConnectionOracle return a grpc connection with peer endpoint, TLS credentials, channelName, chainCodeName and signer private key
Cross-Network Communication
returns URL address of the Corda Oracle node and Authorization credentials
Hyperledger Fabric Connection
import grpc
import contract from HL
import connect from HL
import textDecoder
export async HFConnectionOracle
Corda Connection
export cordaOracleNode
readCorda calls Corda contract to read data and writes to Hyperledger ledger with data from Corda and returns result
writeCorda calls Corda contract to write data to Corda and update Hyperledger ledger and returns result
Cross-Network Transaction
readController calls hyperledger fabric contract to read data and writes to corda ledger with data from hyperledger and returns result
writeController calls hyperledger fabric contract to write data to hyperledger and update corda ledger and returns result
Hyperledger Fabric Query API
import HFConnectionOracle
import cordaOracleNode
import eventInterface
import requestBody
import Contract
export async startHyperledger
func async eventListening()
func readEvent()
func readCorda(requestBody)
func writeCorda(requestBody)
Index
corda.listen
startHyperledger
Corda Controllers
import HFConnectionOracle
import cordaOracleNode
import Contract
import eventInterface
import requestBody
export async readController (request, response)
export async writeController (request, response)
Oracle Event Types
export eventInterface
export enum HLEvent
export enum cordaEvent
Corda Requests API
export requestBody
Format of request body and assign request ID

FIGURE 2. **Conceptual diagram showing the modules implemented in the oracle service node of the oracle-based interoperability system.**

which demonstrates how these components are connected to perform cross-network transactions.

The following subsections provide more details about the modules in Hyperledger Fabric, Corda, and Oracle Service nodes of the Oracle-based Interoperability system, and how they interact to achieve read and write operations between a Hyperledger Fabric network and a Corda network.

A. HYPERLEDGER FABRIC

Smart contracts in Hyperledger Fabric are written to implement business logic and enforce rules for executing transactions on the ledger. When there is a need to interact with Corda to fetch data or update the ledger, the smart contract for executing that transaction should be implemented with respect to the following:

- Smart contracts that require cross-network read or write operation are identified and implemented to emit cross-network events.
- If the smart contract requires a cross-network transaction, then the required cross-network transaction type is passed as an argument.
- The event is then added to the transaction response to be recorded in the blockchain and handled by the oracle node listener.
- Chaincode-level endorsement policies should be defined and approved by channel members to establish validation rules for endorsing a transaction submitted and signed by an oracle service node.

B. CORDA

Corda transactions are processed through workflows initiated by a user to execute an operation on a state. For each operation to be executed on a state, a workflow is implemented to handle the transaction until it is completed and recorded. When a cross-network transaction is required, it should be specified in each workflow that handles the transaction, in addition to implementing an ''Oracle Service'' module that operates as follows:

- Receive the state on which the transaction is executed, the type of cross-network event, and the owner or initiator of the workflow.
- Create HTTP client.

- Check the type of cross-network event, and generate the request body accordingly.
- Identify the oracle node that handles the type of event and call the corresponding API to handle the cross-network event.
- Generate the request using HTTP request and send it to the oracle service node.

When implementing workflows that include cross-network transaction, the following must be considered in the implementation:

- Obtain member information of the node that initiated the flow.
- If the initiator of the flow is the Oracle node, then the transaction is formed with the Oracle signature in addition to the signatures of the participating nodes, and the transaction is submitted to the Notary node/s.
- If the initiator is any other member of the network and the arguments of the flow contain a call to a smart contract on Hyperledger, then the "Oracle Service" module is called along with the state, initiator, and cross-network transaction type.

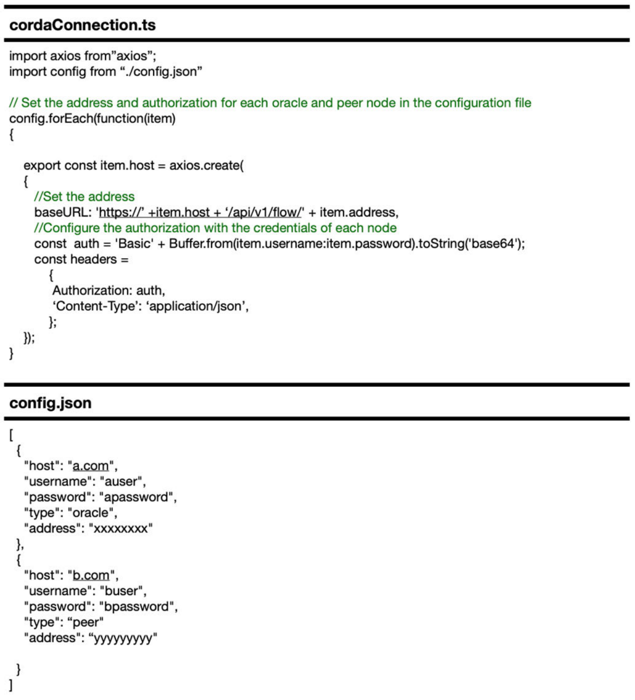
C. ORACLE SERVICE NODE

The conceptual diagram illustrated in figure 2 shows the modules in the Oracle Service Node. The Cross-network Communication component includes a Corda Connection module and a Hyperledger Fabric Connection module. Although the implementations of both modules are different, they both establish connections with the oracle/peer nodes in each blockchain network. The Cross-network Transaction component includes modules that handle formatting and submitting requests and responses for incoming and outgoing transactions in both networks. The following subsections explain each module in detail.

1) CROSS-NETWORK COMMUNICATION

The Cross-network Communication component includes two connection modules: one for handling the Corda connection and the other is responsible for the Hyperledger Fabric connection. Both connection modules interact with the Cross-network Transaction modules to allow incoming and outgoing transactions from the network. The implementation of the modules is shown in figures 3 and 4, and described as Follows:

- *Corda Connection:* communication with Corda oracle nodes and peers is through HTTPS connection and requests and responses are handled by the Javascript library "Axios." In a configuration file, the URL addresses of the oracle nodes and all peers whose states are shared with the oracle node and hence are accessible via the oracle are stored. The *cordaConnection* module then imports the configuration file and uses it to create, for each oracle node and peers that share a state with the oracle node, an instance of Axios, and configures the URL address and peer credentials for authorization.



cordaConnection.ts

```
import axios from"axios";
import config from “./config.json”

// Set the address and authorization for each oracle and peer node in the configuration file
config.forEach(function(item)
{

   export const item.host = axios.create(
   {
     //Set the address
     baseURL: 'https://’ +item.host + ‘/api/v1/flow/' + item.address,
     //Configure the authorization with the credentials of each node
     const  auth = 'Basic' + Buffer.from(item.username:item.password).toString('base64');
     const headers =
        {
         Authorization: auth,
         ‘Content-Type’: ‘application/json’,
        };
   });
}
```

config.json

```
[
 {
  "host": "a.com",
  "username": "auser",
  "password": "apassword",
  "type": "oracle",
  "address": "xxxxxxxx"
 },
 {
  "host": "b.com",
  "username": "buser",
  "password": "bpassword",
  "type": “peer"
  "address": “yyyyyyyy"

 }
]
```

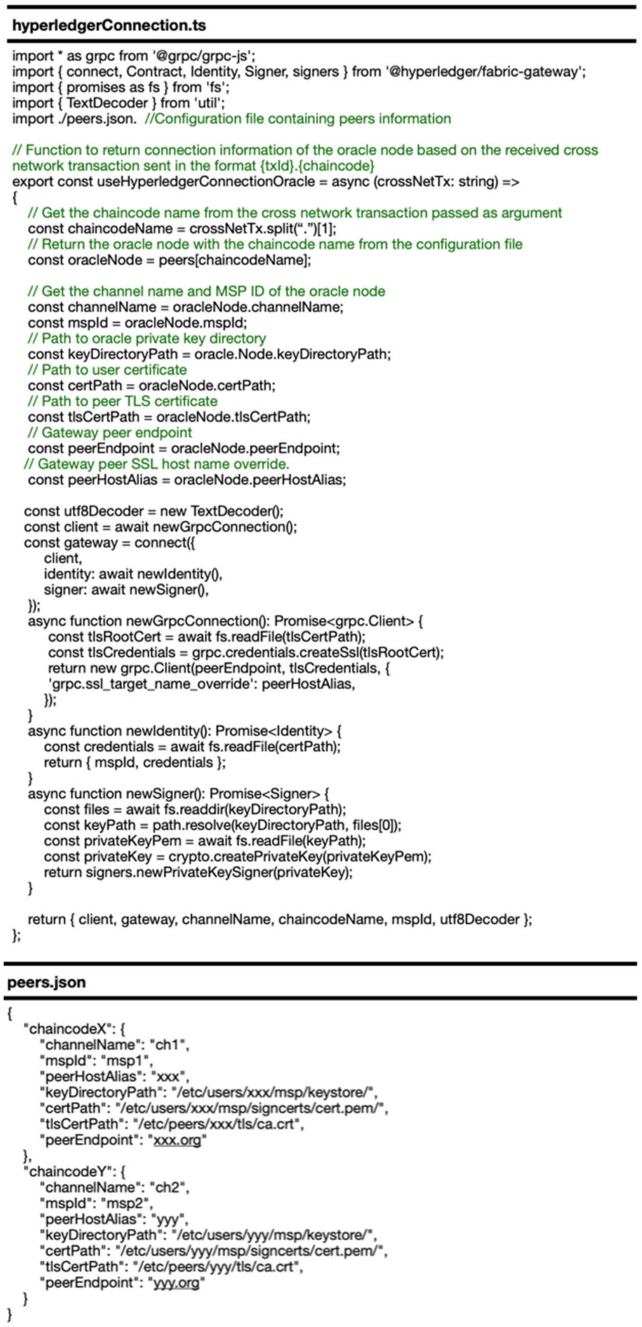
FIGURE 3. Corda connection module.

- *Hyperledger Fabric Connection:* Connection to Hyperledger Fabric is achieved by establishing a gRPC connection. For Hyperledger Fabric, the connection must include the channel name and the chaincode to which the cross-network transactions belong. In addition to the MSP ID, the private key, user certificate, and peer endpoint of the oracle node. The connection also returns the gateway connection information. The peer information is stored in a configuration file and later fetched by the *hyperledgerConnection* according to the cross-network transaction received as an argument. The cross-network transaction includes the transaction id and the name of the chaincode to fetch the corresponding peer node from the configuration file.

2) CROSS-NETWORK TRANSACTIONS

To handle the formatting, submission, and validation of cross-network transactions, the following modules are implemented:

- *Oracle Event Types:* This module stores the interfaces of the cross-network transaction services offered in both Corda and Hyperledger Fabric. It is accessed by the *Hyperledger Fabric Query API* and the *Corda Controllers* to retrieve the interface of a requested cross-network transaction.
- *Corda Requests API:* Also used by both *Hyperledger Fabric Query API*, the *Corda Controllers* and the *Utils* modules. This module creates a request Id and returns the request body from the corresponding Corda workflow for each Corda cross-network request.



hyperledgerConnection.ts

```
import * as grpc from '@grpc/grpc-js';
import { connect, Contract, Identity, Signer, signers } from '@hyperledger/fabric-gateway';
import { promises as fs } from 'fs';
import { TextDecoder } from 'util';
import ./peers.json.  //Configuration file containing peers information

// Function to return connection information of the oracle node based on the received cross
network transaction sent in the format {txId}.{chaincode}
export const useHyperledgerConnectionOracle = async (crossNetTx: string) =>
{
   // Get the chaincode name from the cross network transaction passed as argument
   const chaincodeName = crossNetTx.split(".")[1];
   // Return the oracle node with the chaincode name from the configuration file
   const oracleNode = peers[chaincodeName];

   // Get the channel name and MSP ID of the oracle node
   const channelName = oracleNode.channelName;
   const mspId = oracleNode.mspId;
   // Path to oracle private key directory
   const keyDirectoryPath = oracle.Node.keyDirectoryPath;
   // Path to user certificate
   const certPath = oracleNode.certPath;
   // Path to peer TLS certificate
   const tlsCertPath = oracleNode.tlsCertPath;
   // Gateway peer endpoint
   const peerEndpoint = oracleNode.peerEndpoint;
  // Gateway peer SSL host name override.
   const peerHostAlias = oracleNode.peerHostAlias;

  const utf8Decoder = new TextDecoder();
  const client = await newGrpcConnection();
  const gateway = connect({
      client,
      identity: await newIdentity(),
      signer: await newSigner(),
   });
   async function newGrpcConnection(): Promise<grpc.Client> {
       const tlsRootCert = await fs.readFile(tlsCertPath);
       const tlsCredentials = grpc.credentials.createSsl(tlsRootCert);
       return new grpc.Client(peerEndpoint, tlsCredentials, {
       'grpc.ssl_target_name_override': peerHostAlias,
      });
   }
   async function newIdentity(): Promise<Identity> {
      const credentials = await fs.readFile(certPath);
      return { mspId, credentials };
   }
   async function newSigner(): Promise<Signer> {
      const files = await fs.readdir(keyDirectoryPath);
      const keyPath = path.resolve(keyDirectoryPath, files[0]);
      const privateKeyPem = await fs.readFile(keyPath);
      const privateKey = crypto.createPrivateKey(privateKeyPem);
      return signers.newPrivateKeySigner(privateKey);
   }

   return { client, gateway, channelName, chaincodeName, mspId, utf8Decoder };
};
```

peers.json

```
{
  "chaincodeX": {
    "channelName": "ch1",
    "mspId": "msp1",
    "peerHostAlias": "xxx",
    "keyDirectoryPath": "/etc/users/xxx/msp/keystore/",
    "certPath": "/etc/users/xxx/msp/signcerts/cert.pem/",
    "tlsCertPath": "/etc/peers/xxx/tls/ca.crt",
    "peerEndpoint": "xxx.org"
  },
  "chaincodeY": {
    "channelName": "ch2",
    "mspId": "msp2",
    "peerHostAlias": "yyy",
    "keyDirectoryPath": "/etc/users/yyy/msp/keystore/",
    "certPath": "/etc/users/yyy/msp/signcerts/cert.pem/",
    "tlsCertPath": "/etc/peers/yyy/tls/ca.crt",
    "peerEndpoint": "yyy.org"
  }
}
```

FIGURE 4. Hyperledger connection module.

- *Utils:* This module uses the *Oracle Event Types* module, the *connection* model of both Corda and Hyperledger Fabric, and the *Corda Request API* module. It is responsible for asynchronously calling requests and waiting for the result of a transaction execution to return the results, whether successful or not.
- *Hyperledger Fabric Query API:* This module uses the *Corda connection* and *Hyperledger Fabric connection* modules. It also has access to the smart contracts defined in the chaincode to which the Oracle node has access. This module is the event listener for events emitted Hyperledger Fabric. If an event is detected, the type of event is determined to call the corresponding cross-network transaction from the *Utils* module. If the transaction reads data from Corda and is executed successfully, then a transaction is created with the retrieved data and Oracle's signature, and submitted to be recorded on the Hyperledger Fabric ledger. If the transaction writes to the Corda ledger, then the Corda Oracle node is called to retrieve information about the state on which the transaction is executed. The transaction is then created with Oracle's signature and submitted to Corda for execution and to be recorded on the ledger. Simultaneously, a transaction representing the changes made on Corda is created and submitted to Hyperledger Fabric to be executed and recorded on the ledger. If the transaction execution in Corda is unsuccessful, a transaction is submitted to Hyperledger Fabric to overwrite the previous transaction (rollback) and maintain consistency between the two ledgers. The implementation of the Hyperledger Fabric Query API is shown in figure 5.
  Corda Controllers: For each cross-network event emitted in Corda, a controller is implemented to handle it. An observer API is used to detect the cross- network events, determine the event type and call the appropriate controller. The controller uses the *Corda connection* and *Hyperledger Fabric connection* modules. It also has access to both the *Corda Request API* and *Oracle Event Types* in order to create the request body and retrieve the interface of cross-network services provided in Hyperledger Fabric. If the controller performs a read operation and retrieves data from the Hyperledger Fabric, then using the
- appropriate service interface from the *Oracle Event Types* module a transaction is created, signed by Oracle, and submitted to Hyperledger Fabric. If the read transaction is executed successfully, then another transaction is created using the data retrieved from Hyperledger Fabric and signed by Oracle to be submitted to Corda. If the controller performs a write operation on the Hyperledger Fabric, then a request to fetch the data record that will be modified is submitted to the Hyperledger Fabric. Once the matching data record is found, a transaction is created with the input data from Corda and Oracle's signature and then submitted to be executed and recorded on the Hyperledger Fabric ledger. Another transaction reflecting the transaction that occurred on Hyperledger Fabric is created and signed by Oracle to be executed and recorded on Corda. A rollback function is called when transaction execution is unsuccessful. To further explain this module, the *readController* and *writeController* implementations are shown in figure 6.
- Index: This is the main module that uses both the *Corda Controllers* and the *Hyperledger Fabric Query API*. This module defines the endpoints for Corda to perform read

TABLE 1. Transaction latency measurements in seconds obtained from the evaluation with ten different test data.

Transactions	Test Input #1	Test Input #2	Test Input #3	Test Input #4	Test Input #5	Test Input #6	Test Input #7	Test Input #8	Test Input #9	Test Input #10	AVG	Standard Deviation
	Corda Transaction Latency in Seconds											
	Local Transactions											
Create Policy	26.30	24.3	22.3	22.26	30.37	28.31	28.30	26.29	22.26	22.23	25.3	3.05
Update Policy	34.20	32.41	38.42	28.33	30.41	34.39	26.38	28.33	38.41	36.48	32.78	4.33
	Cross-network Transactions											
Create & Read data from HF	28.44	30.31	30.36	26.32	24.25	32.33	28.32	30.30	24.26	28.36	28.33	2.7
Update C & HF	30.24	32.35	28.23	34.28	38.39	24.22	36.37	32.36	36.34	38.35	33.12	4.6
	Rollback Transactions											
Create Rollback	30.29	30.30	26.25	26.25	30.31	26.28	28.31	28.33	30.35	22.22	27.89	2.67
Update Rollback	32.35	30.30	28.29	30.31	26.28	28.33	30.29	26.26	30.27	30.42	29.31	1.97
	Hyperledger Fabric Transaction Latency in Seconds											
	Local Transactions											
Create Bill	2.15	2.03	2.03	2.03	2.03	2.03	2.02	2.03	2.04	2.03	2.05	0.04
Update Bill	4.07	4.05	4.05	4.06	4.05	4.05	4.06	4.05	5.06	4.06	4.16	0.32
	Cross-network Transactions											
Create & Read from Corda	36.46	26.33	26.32	28.32	32.33	24.35	28.33	28.41	28.42	28.36	28.77	3.41
Update HF & C	30.32	30.32	26.36	26.33	30.34	26.29	26.28	30.37	30.43	28.35	28.54	2.01
	Rollback Transactions											
Create Rollback	2.03	2.12	1.98	2.07	1.97	2.06	2.04	2.04	2.06	2.01	2.04	0.05
Update Rollback	2.25	2.12	4.13	2.15	4.05	4.11	3.98	4.00	4.00	4.03	3.49	0.91

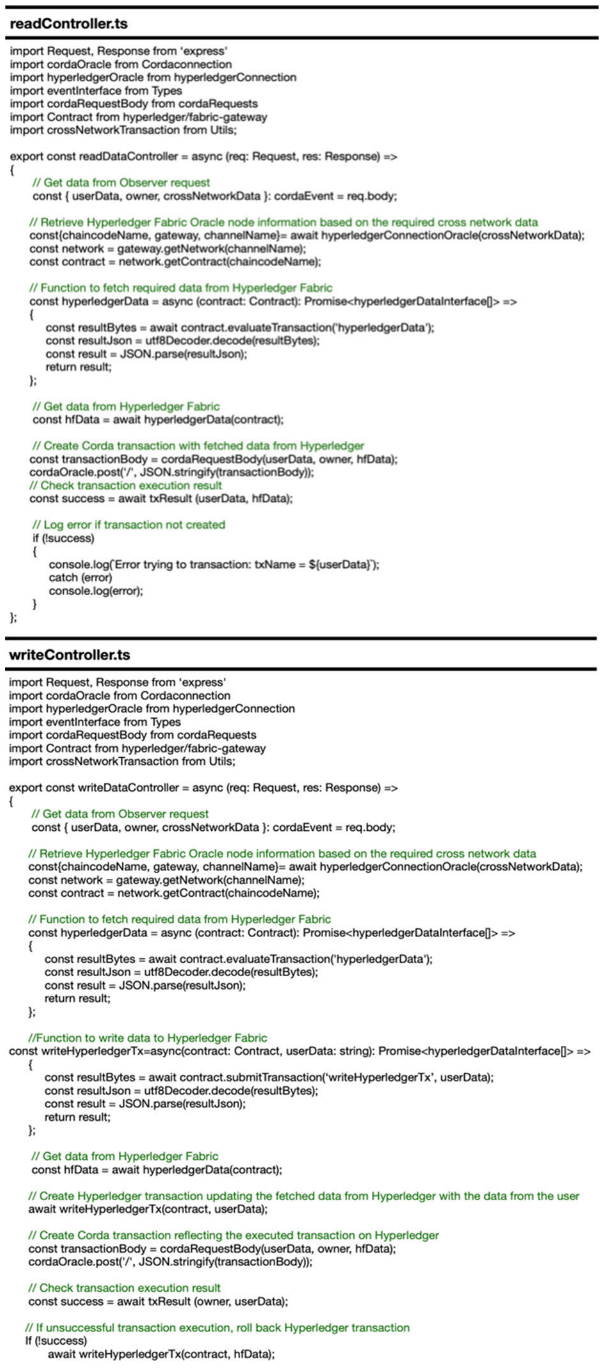
TABLE 2. Mean and standard deviation of transaction latency in seconds.

Blockchain Network	Create New State / Data Record				Update Existing State / Data Record				Rollback Transactions			
	Local Transaction		Cross-network Transaction (Read)		Local Transaction		Cross-network Transaction (Write)		Create New State / Data Record		Update Existing State / Data Record	
	M	SD	M	SD	M	SD	M	SD	M	SD	M	SD
Corda	25.3	3.05	28.33	2.7	32.78	4.33	33.12	4.6	27.89	2.69	29.31	1.97
Hyperledger Fabric	2.05	0.04	28.77	3.41	4.16	0.32	28.54	2.01	2.04	0.05	3.49	0.91

and write operations and then starts event listening on both Corda and Hyperledger Fabric.

VII. EVALUATION OF THE BLOCKCHAIN ORACLE INTEROPERABILITY SYSTEM

To demonstrate the applicability of the Oracle Interoperability technique described in the previous section, we implemented a prototype system consisting of two blockchain networks with oracle nodes to perform cross-network transactions. A permissioned blockchain for a hospital billing system was built atop the Hyperledger Fabric platform using a minimal setup that contained two peer organizations, one of which was the dedicated Oracle node, a Raft ordering service node, and one channel with the Payment System chaincode deployed on. Smart contracts containing the business logic and cross-network transaction



readController.ts

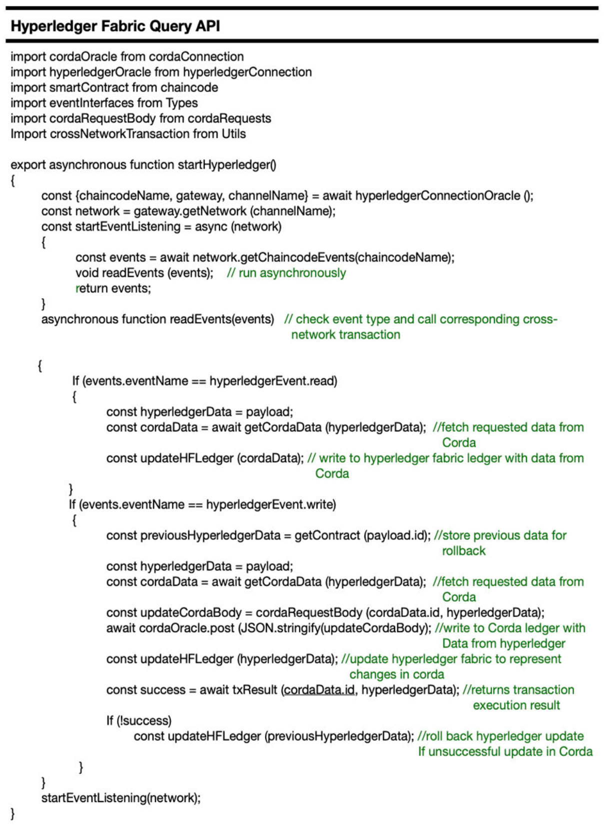
```
import Request, Response from 'express'
import cordaOracle from Cordaconnection
import hyperledgerOracle from hyperledgerConnection
import eventInterface from Types
import cordaRequestBody from cordaRequests
import Contract from hyperledger/fabric-gateway
import crossNetworkTransaction from Utils;

export const readDataController = async (req: Request, res: Response) =>
{
    // Get data from Observer request
    const { userData, owner, crossNetworkData }: cordaEvent = req.body;

    // Retrieve Hyperledger Fabric Oracle node information based on the required cross network data
    const{chaincodeName, gateway, channelName}= await hyperledgerConnectionOracle(crossNetworkData);
    const network = gateway.getNetwork(channelName);
    const contract = network.getContract(chaincodeName);

    // Function to fetch required data from Hyperledger Fabric
    const hyperledgerData = async (contract: Contract): Promise<hyperledgerDataInterface[]> =>
    {
        const resultBytes = await contract.evaluateTransaction('hyperledgerData');
        const resultJson = utf8Decoder.decode(resultBytes);
        const result = JSON.parse(resultJson);
        return result;
    };

    // Get data from Hyperledger Fabric
    const hfData = await hyperledgerData(contract);

    // Create Corda transaction with fetched data from Hyperledger
    const transactionBody = cordaRequestBody(userData, owner, hfData);
    cordaOracle.post('/', JSON.stringify(transactionBody));
    // Check transaction execution result
    const success = await txResult (userData, hfData);

    // Log error if transaction not created
    if (!success)
    {
        console.log(`Error trying to transaction: txName = ${userData}`);
        catch (error)
        console.log(error);
    }
};
```

writeController.ts

```
import Request, Response from 'express'
import cordaOracle from Cordaconnection
import hyperledgerOracle from hyperledgerConnection
import eventInterface from Types
import cordaRequestBody from cordaRequests
import Contract from hyperledger/fabric-gateway
import crossNetworkTransaction from Utils;

export const writeDataController = async (req: Request, res: Response) =>
{
    // Get data from Observer request
    const { userData, owner, crossNetworkData }: cordaEvent = req.body;

    // Retrieve Hyperledger Fabric Oracle node information based on the required cross network data
    const{chaincodeName, gateway, channelName}= await hyperledgerConnectionOracle(crossNetworkData);
    const network = gateway.getNetwork(channelName);
    const contract = network.getContract(chaincodeName);

    // Function to fetch required data from Hyperledger Fabric
    const hyperledgerData = async (contract: Contract): Promise<hyperledgerDataInterface[]> =>
    {
        const resultBytes = await contract.evaluateTransaction('hyperledgerData');
        const resultJson = utf8Decoder.decode(resultBytes);
        const result = JSON.parse(resultJson);
        return result;
    };

    //Function to write data to Hyperledger Fabric
const writeHyperledgerTx=async(contract: Contract, userData: string): Promise<hyperledgerDataInterface[]> =>
    {
        const resultBytes = await contract.submitTransaction('writeHyperledgerTx', userData);
        const resultJson = utf8Decoder.decode(resultBytes);
        const result = JSON.parse(resultJson);
        return result;
    };

    // Get data from Hyperledger Fabric
    const hfData = await hyperledgerData(contract);

    // Create Hyperledger transaction updating the fetched data from Hyperledger with the data from the user
    await writeHyperledgerTx(contract, userData);

    // Create Corda transaction reflecting the executed transaction on Hyperledger
    const transactionBody = cordaRequestBody(userData, owner, hfData);
    cordaOracle.post('/', JSON.stringify(transactionBody));

    // Check transaction execution result
    const success = await txResult (owner, userData);

    // If unsuccessful transaction execution, roll back Hyperledger transaction
    If (!success)
        await writeHyperledgerTx(contract, hfData);
```

FIGURE 5. Implementation of a read and write controller in Corda.

calls were programmed using JavaScript. On Corda, a health-insurance policy permissioned blockchain network was built consisting of two nodes, one of which was the oracle node. Contracts, states, and workflows containing business logic and cross-network transaction calls were written in the Kotlin programming language. The components of the Oracle Interoperability Service system, as described in the previous section, were implemented in the Typescript programming language and deployed on the oracle nodes in both networks.



Hyperledger Fabric Query API

```
import cordaOracle from cordaConnection
import hyperledgerOracle from hyperledgerConnection
import smartContract from chaincode
import eventInterfaces from Types
import cordaRequestBody from cordaRequests
Import crossNetworkTransaction from Utils

export asynchronous function startHyperledger()
{
        const {chaincodeName, gateway, channelName} = await hyperledgerConnectionOracle ();
        const network = gateway.getNetwork (channelName);
        const startEventListening = async (network)
        {
                const events = await network.getChaincodeEvents(chaincodeName);
                void readEvents (events);    // run asynchronously
                return events;
        }
        asynchronous function readEvents(events)   // check event type and call corresponding cross-
                                                     network transaction

       {
                If (events.eventName == hyperledgerEvent.read)
                {
                        const hyperledgerData = payload;
                        const cordaData = await getCordaData (hyperledgerData);  //fetch requested data from
                                                                                   Corda
                        const updateHFLedger (cordaData); // write to hyperledger fabric ledger with data from
                                                           Corda
                }
               If (events.eventName == hyperledgerEvent.write)
                {
                        const previousHyperledgerData = getContract (payload.id); //store previous data for
                                                                                   rollback
                        const hyperledgerData = payload;
                        const cordaData = await getCordaData (hyperledgerData);  //fetch requested data from
                                                                                   Corda
                        const updateCordaBody = cordaRequestBody (cordaData.id, hyperledgerData);
                        await cordaOracle.post (JSON.stringify(updateCordaBody)); //write to Corda ledger with
                                                                                   Data from hyperledger
                        const updateHFLedger (hyperledgerData); //update hyperledger fabric to represent
                                                                changes in corda
                        const success = await txResult (cordaData.id, hyperledgerData); //returns transaction
                                                                                         execution result
                        If (!success)
                              const updateHFLedger (previousHyperledgerData); //roll back hyperledger update
                                                                              If unsuccessful update in Corda
                 }
        }
        startEventListening(network);
}
```

FIGURE 6. Hyperledger fabric query API module.

On Hyperledger Fabric, the billing system creates and manages patient bills for two types of patients: self-paying patients and insured patients. For self-pay patients, all bill-related transactions are performed without the need to interact with the health insurance system in Corda. Transactions for issuing and managing insured patients' bills interact with the health insurance policy system to retrieve and modify the patients' insurance policy information.

On Corda, a new insurance policy is created for a person either by using user input or by retrieving information from the hospital billing system. Likewise, insurance policies are updated by executing transactions locally without the need to interact with Hyperledger Fabric, or by issuing cross-network transactions to update the person's hospital bill according to the changes made to the insurance policy. In cross-network transactions that write to and modify ledgers, rollback transactions are executed when cross-network transaction execution fails to maintain ledger consistency.

The prototypical interoperability system was evaluated using 12 test cases designed to verify the cross-network transaction functionalities and measure their transaction latency on both networks. The test cases were designed to assess the following system functionalities:

- Local transactions to create and update data records/states in both networks.
- Cross-network transactions to read data from Corda and Hyperledger Fabric.

- Cross-network transactions to write data to the ledger in Corda and Hyperledger Fabric.
- Rollback transactions to maintain ledger consistency in case cross-transactions failure in either network.

We benchmarked the measurements of the cross-network transaction latency against the local transaction latency measurements to assess the performance of the Oracle-based interoperability system. The networks were setup in Docker V. 24.0.5 and the measurements were performed on an Ubuntu Pop!_OS 22.04 LTS running on an Intel-Core i7-8700K CPU @ 3.70 GHz x 12 with 32 GB RAM. Each test case was executed 10 times with different inputs to calculate the average transaction latency and standard deviation, which helps to understand the stability of the blockchain networks involved in the cross-network transaction. Table 1 shows the results obtained for both the Corda network and Hyperledger Fabric (Abbreviated as C and HF, respectively). These measurements are summarized in Table 2 to focus on the calculated transaction latency average and standard deviation. A comparison of the average cross-network transaction latency results with the local transaction latency results show that the resulting increase in transaction latency is in the order of seconds, which is a reasonable increase for the end-to-end execution of cross-network transactions. The mean transaction latency for creating a new state locally in Corda is 25.3 seconds, whereas the same transaction when performing a cross-network read operation requires an average time of 28.3 seconds with only a 3 second increase for performing the cross-network read. The state update transactions are slightly higher than the create transactions, which is expected because of the increased complexity of the transaction workflow. The local update transaction latency mean time is 32.8 seconds and the cross-network transaction latency mean time is 33.12, which is a slight and acceptable increase. The mean transaction latency for the rollback transactions of the read and write cross-network operations are in the same range as the local transactions and cross-network transactions in Corda.

As for the Hyperledger Fabric network, there is a noticeable difference in the transaction latency between local transactions and cross-network transactions. However, when compared with the Corda transaction latency time, the increase in cross-network transaction latency is understandable because transactions in Corda generally incur a higher transaction latency than Hyperledger Fabric. Local transaction for creating a new data record in Hyperledger Fabric is performed with a mean transaction latency of 2.05 seconds, whereas the cross-network (read) transaction is executed with an average of 28.8 seconds which is close to the cross-network read transaction performed in Corda. The local update transaction in the Hyperledger fabric requires 4.16 seconds, whereas the cross-network write transaction incurs an increase of approximately 24.3 seconds which is also reasonable because of the interaction with the Corda network. The mean time of rollback transaction latency in Hyperledger Fabric is in the same range as the local transactions latency time of Hyperledger Fabric because the cross-network transaction with Corda failed, and there was no increase in the transaction latency.

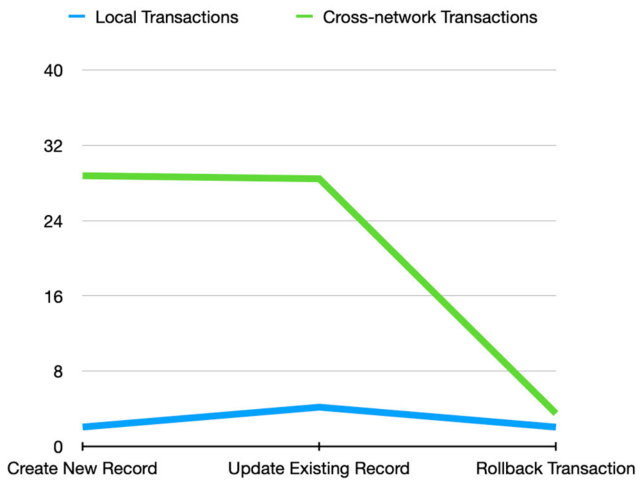


FIGURE 7. Line graph depicting local transaction latency and cross-network transaction latency of the Hyperledger fabric network.

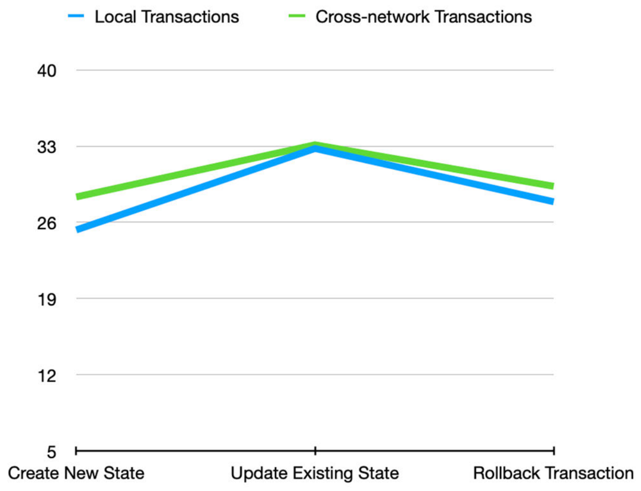


FIGURE 8. Line graph depicting local transaction latency and cross-network transaction latency of the Corda network.

The line charts shown in figures 7. and 8. depict the transaction latency difference between local transactions and cross-network transactions in both the Hyperledger Fabric and Corda networks. For the Corda network, the average transaction latency of cross-network transactions (in green) is slightly higher than that of local transactions (in blue); however, they are still in the same range and are considerably close. In Hyperledger Fabric, the graph demonstrates how the cross-network transactions that read and write to Corda are in the same range as the Corda network transaction latency, but are significantly higher than the local Hyperledger Fabric transaction. However, the cross-network transaction line drops to a range closer to the Hyperledger Fabric local transactions line when the cross-network transaction fails, and rollback transactions are executed.

TABLE 3. Comparison between our oracle-based technique and other techniques in the literature.

Study reference number	Proof of concept	Performance evaluation	Fully decentralized	Existing BC frameworks unmodified	Blockchain agnostic	Cryptocurrency-based interoperability	Data-based interoperability	Smart contract invocation
[23]	✓	✓	X	✓	✓	X	✓	X
[24]	✓	✓	X	✓	X	X	✓	X
[25]	X	X	✓	✓	✓	X	✓	✓
[26]	✓	X	X	✓	X	X	✓	X
[27]	✓	X	X	✓	✓	X	✓	X
[28]	✓	X	✓	✓	X	✓	X	✓
[29]	✓	X	X	✓	✓	✓	X	✓
This Paper	✓	✓	✓	✓	✓	X	✓	✓

The standard deviation value obtained from the transaction latency of the transactions in Corda, whether local or cross-network, is higher than the standard deviation of the Hyperledger Fabric transactions. This indicates that the transaction latency in Corda is not steady and has varying values, whereas Hyperledger Fabric has more consistent values. This is due to the difference in the way the networks handle transaction execution, where Hyperledger Fabric broadcasts the transaction to all members of the channel, whereas Corda does this on a peer-to-peer basis, resulting in a higher management overhead.

It is worth mentioning that various factors affect the transaction latency such as the block size, the number of nodes and organizations participating in the network, the chosen consensus algorithm, the applied encryption method and the method of transaction execution, which can be either batch or sequential. The evaluation performed in this study was sufficient to prove the feasibility of the Oracle-based Interoperability technique for achieving cross-network transactions between two permissioned networks with an acceptable transaction latency time. However, further evaluation is required to measure the scalability of this interoperability solution and the performance measurements with a higher number of participating nodes to measure the transaction throughput.

VIII. COMPARATIVE ANALYSIS

In this section we compare the Oracle-based interoperability method with the interoperability solutions described in the ‘‘Related work’’ section. The comparison is done using 6 features identified in a recent survey paper [5] to assess interoperability solutions for blockchain systems. We added a seventh feature, ‘‘Performance Evaluation’’, since it is essential to measure the technique’s performance to ensure acceptable performance of the system and identify weaknesses and possible bottlenecks. We also considered ‘‘Smart contract invocation’’ as important interoperability feature to expand the capabilities of the interoperation process. Table 3 shows the comparison between the Oracle-based system described in this paper with the studies described in the ‘‘Related Work’’ section. It is clear that the Oracle-based technique has advantage over other recently developed techniques as it satisfies most of the features used for assessing blockchain interoperability techniques. It is also evident that majority of the proposed techniques are centralized and lack performance evaluation measurements that would be helpful to properly assess their suitability as interoperation techniques for permissioned blockchains.

IX. LIMITATIONS AND FUTURE WORK

The performance evaluation showed acceptable latency measurements compared with the local transaction measurements for each blockchain platform. However, Hyperledger Fabric experienced high transaction latency when interoperating with Corda, it is worth exploring methods to lower the transaction latency between them and experiment with other blockchain platforms to gain more understanding on the performance of the system. Moreover, Further evaluation is required to gain precise performance measurements under varying network setups and configurations to accurately assess the efficiency of our Oracle-based Interoperability technique. In future work, we aim to conduct an extensive performance evaluation to obtain important performance measurements, such as transaction throughput and scalability, using multiple oracle nodes and a higher number of peer nodes with the use of a network benchmarking tool. We also seek to perform a comparative study between the performance of our interoperability solution and existing interoperability solutions. Additionally, we plan to study how to integrate multiple permissioned blockchain platforms using an oracle-based interoperability framework with connectors that allow easier integration and require less developer effort.

X. CONCLUSION

Currently, various private blockchain platforms exist that provide tools for enterprise organizations to build their private blockchain networks and implement their business processes. Thus, the design of an interoperability technique that fits the specific requirements of private permissioned blockchain networks is necessary for the advancement of private blockchains. The literature review of existing private blockchain interoperability solutions revealed the limitations and current challenges. To the best of our knowledge, blockchain oracle has not been studied as an interoperability technique that allows smart contract invocation between permissioned blockchain networks.

In this paper, we presented an Oracle-based Interoperability technique between two of the most widely used permissioned networks, Hyperledger Fabric and Corda. The components of the system are described in detail, and a prototypical system is developed to evaluate the feasibility of the proposed interoperability solution. The prototype system is evaluated to measure the transaction latency of cross-network transactions and obtain the average latency and standard deviation. The cross-network transaction latency was compared with the local transaction latency, showing a slight and acceptable increase in Corda. However, the difference between local and cross-network transaction latency in Hyperledger Fabric was significant but understandable because of the interaction with Corda, which has a higher transaction latency and affects the overall cross-network transaction latency. Our proposed Oracle-based Interoperability technique is relatively straightforward to implement and integrate with any permissioned blockchain system. It allows permissioned blockchain networks to perform cross-network transactions to exchange data and execute business transactions.

REFERENCES

[1] A. Gorkhali, L. Li, and A. Shrestha, "Blockchain: A literature review," *J. Manage. Analytics*, vol. 7, no. 3, pp. 321–343, Jul. 2020, doi: 10.1080/23270012.2020.1801529.

[2] B. K. Mohanta, D. Jena, S. S. Panda, and S. Sobhanayak, "Blockchain technology: A survey on applications and security privacy challenges," *Internet Things*, vol. 8, Dec. 2019, Art. no. 100107, doi: 10.1016/j.iot.2019.100107.

[3] C. Zhang and Y. Chen, "A review of research relevant to the emerging industry trends: Industry 4.0, IoT, blockchain, and Bus. Analytics," *J. Ind. Integr. Manage.*, vol. 05, no. 01, pp. 165–180, Mar. 2020, doi: 10.1142/s2424862219500192.

[4] S. Khan, M. B. Amin, A. T. Azar, and S. Aslam, "Towards interoperable blockchains: A survey on the role of smart contracts in blockchain interoperability," *IEEE Access*, vol. 9, pp. 116672–116691, 2021, doi: 10.1109/ACCESS.2021.3106384. https://doi.org/10.1109/access.2021.3106384

[5] S. D. Kotey, E. T. Tchao, A. Ahmed, A. S. Agbemenu, H. Nunoo-Mensah, A. Sikora, D. Welte, and E. Keelson, "Blockchain interoperability: The state of heterogenous blockchain-to-blockchain communication," *IET Commun.*, vol. 17, no. 8, pp. 891–914, Mar. 2023, doi: 10.1049/cmu2.12594.

[6] A. Pasdar, Y. C. Lee, and Z. Dong, "Connect API with blockchain: A survey on blockchain Oracle implementation," *ACM Comput. Surveys*, vol. 55, no. 10, pp. 1–39, Oct. 2023.

[7] S. K. Ezzat, Y. N. M. Saleh, and A. A. Abdel-Hamid, "Blockchain oracles: State-of-the-Art and research directions," *IEEE Access*, vol. 10, pp. 67551–67572, 2022, doi: 10.1109/ACCESS.2022.3184726.

[8] Y. Lu, "Blockchain: A survey on functions, applications and open issues," *J. Ind. Integr. Manage.*, vol. 3, no. 4, Nov. 2018, Art. no. 1850015, doi: 10.1142/s242486221850015x.

[9] Y. Lu, "The blockchain: State-of-the-art and research challenges," *J. Ind. Inf. Integr.*, vol. 15, pp. 80–90, Sep. 2019, doi: 10.1016/j.jii.2019.04.002.

[10] K. K. Vaigandla, R. Karne, M. Siluveru, and M. Kesoju, "Review on blockchain technology : Architecture, characteristics, benefits, algorithms, challenges and applications," *Mesopotamian J. Cyber Secur.*, vol. 1, pp. 73–85, Mar. 2023.

[11] I. Priyadarshini, "Introduction to blockchain technology," *Cyber Secur. Parallel Distrib. Comput.*, vol. 1, pp. 91–107, Mar. 2019.

[12] E. Abebe, D. Behl, C. Govindarajan, Y. Hu, D. Karunamoorthy, P. Novotny, V. Pandit, V. Ramakrishna, and C. Vecchiola, "Enabling enterprise blockchain interoperability with trusted data transfer (Industry Track)," in *Proc. 20th Int. Middleware Conf. Ind. Track*. NY, USA: Assoc. for Comput. Machinery, Dec. 2019, pp. 29–35, doi: 10.1145/3366626.3368129.

[13] T. Hardjono, A. Lipton, and A. Pentland, "Toward an interoperability architecture for blockchain autonomous systems," *IEEE Trans. Eng. Manag.*, vol. 67, no. 4, pp. 1298–1309, Nov. 2020, doi: 10.1109/TEM.2019.2920154. https://doi.org/10.1109/tem.2019.2920154

[14] Y. Pang, "A new consensus protocol for blockchain interoperability architecture," *IEEE Access*, vol. 8, pp. 153719–153730, 2020, doi: 10.1109/ACCESS.2020.3017549. https://doi.org/10.2139/ssrn.3632084

[15] R. Ramadoss, "Blockchain technology: An overview," *IEEE Potentials*, vol. 41, no. 6, pp. 6–12, Nov. 2022, doi: 10.1109/MPOT.2022.3208395. https://doi.org/10.6028/nist.ir.8202

[16] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Comput. Surveys*, vol. 54, no. 8, pp. 1–41, Nov. 2022, doi: 10.1145/3471140.

[17] N. Hewett, M. van Gogh, and L. Pawczuk, "Inclusive deployment of blockchain for supply chains: Part 6 -A framework for blockchain interoperability," in *Proc. World Economic Forum*, 2020, pp. 1–11.

[18] Y. Kayıkcı and N. Subramanian, "Blockchain interoperability issues in supply chain: Exploration of mass adoption procedures," *Stud. Big Data*, vol. 1, pp. 309–328, Feb. 2022.

[19] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-enabled smart contracts: Architecture, applications, and future trends," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 49, no. 11, pp. 2266–2277, Nov. 2019, doi: 10.1109/TSMC.2019.2895123. https://doi.org/10.1109/tsmc.2019.2895123

[20] Chainlink. (2019). *Enterprise Smart Contracts: Trusted Computation Framework & Chainlink*. Accessed: Dec. 22, 2023. [Online]. Available: https://blog.chain.link/driving-demand-for-enterprise-smart-contracts-

[21] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, and S. Ellis, "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," Chainlink Labs, 2021, pp. 1–136, vol. 1.

[22] A. Beniiche, "A study of blockchain oracles," 2020, *arXiv:2004.07140*.

[23] S. Ghaemi, S. Rouhani, R. Belchior, R. S. Cruz, H. Khazaei, and P. Musilek, "A pub-sub architecture to promote blockchain interoperability," 2021, *arXiv:2101.12331*.

[24] T. T. A. Dinh, A. Datta, and B. C. Ooi, "A blueprint for interoperable blockchains," 2019, *arXiv:1910.00985*.

[25] S. Bayraktar and S. Gören, "Design principles for interoperability of private blockchains," in *The International Conference on Deep Learning, Big Data and Blockchain (DBB 2022)*. Cham, Switzerland: Springer, 2022, pp. 15–26.

[26] B. Bradach, J. Nogueira, G. Llambías, L. Gonzalez, and R. Ruggia, "A gateway-based interoperability solution for permissioned blockchains," in *Proc. 18th Latin Amer. Comput. Conf. (CLEI)*. IEEE, 2022, pp. 1–10.

[27] G. Llambias, B. Bradach, J. Nogueira, L. González, and R. Ruggia, "Gateway-based interoperability for distributed ledger technology," *CLEI Electron. J.*, vol. 26, no. 2, pp. 1–17, Sep. 2023.

[28] C. Pedreira, R. Belchior, M. Matos, and A. Vasconcelos, "Trustable blockchain interoperability: Securing asset transfers on permissioned blockchains," Authorea Preprints, 2023.

[29] D. Reijsbergen, A. Maw, J. Zhang, T. Tuan, and A. Datta, "Demo: PIEChain—A practical blockchain interoperability framework," in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*. IEEE, 2023, pp. 1021–1024.

[30] K. Mammadzada, M. Iqbal, F. Milani, L. Garcia-Bañuelos, and R. Matulevičius, "Blockchain oracles: A framework for blockchain-based applications," in *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18* (Lecture Notes in Business Information Processing). Springer, 2020, pp. 19–34.

[31] A. S. de Pedro, D. Levi, and L. I. Cuende, "WitNet: A decentralized oracle network protocol," 2017, *arXiv:1711.09756*.

[32] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, "Trustworthy blockchain oracles: Review, comparison, and open research challenges," *IEEE Access*, vol. 8, pp. 85675–85685, 2020, doi: 10.1109/ACCESS.2020.2992698. https://doi.org/10.1109/access.2020.2992698

[33] R. Mühlberger, S. Bachhofner, E. C. Ferrer, C. Di Ciccio, I. Weber, M. Wöhrer, and U. Zdun, "Foundational Oracle patterns: Connecting blockchain to the off-chain world," in *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18* (Lecture Notes in Business Information Processing). Springer, 2020, pp. 35–51.

[34] X. Liu and J. Feng, "Trusted blockchain Oracle scheme based on aggregate signature," *J. Comput. Commun.*, vol. 9, no. 3, pp. 95–109, 2021, doi: 10.4236/jcc.2021.93007.

[35] A. Pupyshev, D. Gubanov, E. Dzhafarov, I. Sapranidi, I. Kardanov, V. Zhuravlev, S. Khalilov, M. Jansen, S. Laureyssens, I. Pavlov, and S. Ivanov, "Gravity: A blockchain-agnostic cross-chain communication and data oracles protocol," 2020, *arXiv:2007.00966*.

[36] E. Androulaki et al., "Hyperledger fabric," in *Proc. 13th EuroSys Conf.*, Apr. 2018, pp. 1–18, doi: 10.1145/3190508.3190538.

[37] R. Brown, J. Carlyle, I. Grigg, and M. Hearn. (2016). *Corda: An Introduction*. [Online]. Available: https://www.smallake.kr/wp-content/uploads/2016/10/corda-introductory-whitepaper-final.pdf

ASMA A. ALHUSSAYEN received the B.Sc. degree in computer science and the M.Sc. degree in software engineering from the College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia, in 2006 and 2017, respectively. She is currently pursuing the Doctor of Philosophy (Ph.D.) degree in computer science (CS) with the Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia. She is also a Teaching Assistant with the Department of Software Engineering, College of Computer and Information Sciences, King Saud University. Her research interests include blockchain, distributed systems, software engineering, cloud computing, tangible user interface, and human–computer interaction.

KAMAL JAMBI received the M.S. degree in computer science from Michigan State University, East Lansing, in 1986, and the Ph.D. degree in computer science from Illinois Institute of Technology, Chicago, in 1991. He was a PI in many research projects from KACST and KAU. He was a Former Vice Dean at Graduate Studies and Scientific Research, FCIT. He was the Chairperson at the Department of Computer Science. He is currently a Professor of computer science with King Abdulaziz University, Jeddah, Saudi Arabia. His research interests include AI, deep learning, blockchain, resilience, and bigdata.

MAHER KHEMAKHEM received the Master of Science and Ph.D. degrees from the University of Paris Sud (Orsay), France, in 1984 and 1987, respectively, and the Habilitation (Accreditation) degree from the University of Sfax, Tunisia, in 2008. He is currently a Full Professor of computer science with the Faculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia. His research interests include distributed systems, HPC, performance analysis, networks security, and pattern recognition.

FATHY E. EASSA received the B.Sc. degree in electronics and electrical communication engineering from Cairo University, Egypt, in 1978, and the M.Sc. and Ph.D. degrees in computers and systems engineering from Al Azhar University, Cairo, Egypt, in 1984 and 1989, respectively. He was a joint supervision at the University of Colorado, USA. He is currently a Full Professor with the Department of Computer Science, Faculty of Computing and Information technology, King Abdulaziz University, Saudi Arabia. His research interests include agent based software engineering, cloud computing, software engineering, big data, distributed systems, exascale system testing.