A Pub-Sub Architecture to Promote Blockchain Interoperability

Sara Ghaemi*, Sara Rouhani†, Rafael Belchior‡, Rui S. Cruz‡, Hamzeh Khazaei§, Petr Musilek*

*University of Alberta, Edmonton, Canada, {sghaemi, petr.musilek}@ualberta.ca

†University of Saskatchewan, Saskatoon, Canada, sara.rouhani@usask.ca

‡Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal, {rafael.belchior, rui.s.cruz}@tecnico.ulisboa.pt

§York University, Toronto, Canada, hkh@yorku.ca

Abstract—The maturing of blockchain technology leads to heterogeneity, where multiple solutions specialize in a particular use case. While the development of different blockchain networks shows great potential for blockchains, the isolated networks have led to data and asset silos, limiting the applications of this technology. Blockchain interoperability solutions are essential to enable distributed ledgers to reach their full potential. Such solutions allow blockchains to support asset and data transfer, resulting in the development of innovative applications.

This paper proposes a novel blockchain interoperability solution for permissioned blockchains based on the publish/subscribe architecture. We implemented a prototype of this platform to show the feasibility of our design. We evaluate our solution by implementing examples of the different publisher and subscriber networks, such as Hyperledger Besu, which is an Ethereum client, and two different versions of Hyperledger Fabric. We present a performance analysis of the whole network that indicates its limits and bottlenecks. Finally, we discuss the extensibility and scalability of the platform in different scenarios. Our evaluation shows that our system can handle a throughput in the order of the hundreds of transactions per second.

Index Terms—Distributed Ledger Technology, Blockchain, Interoperability, Publish/Subscribe

I. INTRODUCTION

The distributed ledger technology (DLT) enables a set of independent untrusted nodes to establish an agreement on the state of a shared ledger. Blockchain, a type of DLT, is mostly known for its use cases in cryptocurrencies such as Bitcoin [1], Ethereum [2], and XRP [3], among others. However, the technology can be used for more diverse applications and industries. Some examples are biomedical and health care [4], [5], Internet of Things (IoT) [6], [7], public administration [8], [9], and cloud computing [10], [11]. Since each industry has its own unique sets of requirements, many isolated permissioned and permissionless blockchains have been introduced, posing limits regarding its interoperability.

Currently, developers commit to a single blockchain solution, and they cannot use the capabilities of more than one blockchain (vendor lock-in). These isolated, incompatible networks have resulted in silos of data and assets, which cannot be used from other networks. Blockchain interoperability solutions are needed to enable asset and information transfer

from one blockchain to another. However, interoperability for blockchains encounters challenges that make it different from interoperability for other software networks.

First, each interoperability solution should take into account the differences in the architecture of blockchain networks. Although all blockchains have an immutable ledger that stores the history of assets, they usually reach a consensus on the order of transactions using different algorithms. As a result, their underlying network and their validation mechanisms can be different from other blockchains. Each blockchain network that participates in the interoperation is independent and in full control of their assets and information. Moreover, the interoperability solutions should not require significant changes in the underlying blockchain networks, and it should be usable with minimal effort for existing blockchains.

This paper aims to tackle this problem by proposing a blockchain interoperability solution based on the publish/subscribe architecture across permissioned blockchains. We have implemented a broker blockchain that acts as a middleman in the interoperability process between the source and the destination networks. It is worth noting that since the broker is itself a blockchain network, it is not a central authority and peers from the source and destination blockchains can also participate in the governance of this network. The *broker* blockchain keeps a copy of the information that needs to be shared in the form of a *topic*. A topic has a name, message, publisher, and a set of subscribers. The *publisher* is the source blockchain network that wants to share the information. It is responsible for creating the topic on the broker and publishing it to the corresponding topic whenever the information needs an update. The *subscribers* are the destination networks that need some information from the source network. As soon as the subscriber network subscribes to a topic, the broker network notifies it whenever a change happens. This solution enables interoperability between blockchains with minimal effort.

We used a performance benchmark tool to analyze the performance of the implemented prototype of the platform. The throughput and average latency for different functionalities of the broker network were investigated. The results indicate that our network can handle hundreds of transactions per second. Moreover, the evaluations identified the *PublishToTopic* functionality to be the broker network's bottleneck.

The rest of this paper is organized as follows. Section II gives a summary of the related work on blockchain interoperability and blockchain-based publish/subscribe protocols. Section III introduces the system design details for the proposed interoperability solution. Section IV demonstrates the implementation and deployment details of the platform, while section V presents its performance evaluation. Section VI outlines some discussions on the design and evaluation of the platform and section VII concludes the paper.

II. RELATED WORK

In this section, we discuss the related work in the field of blockchain interoperability, as well as blockchain-based publish/subscribe protocols.

A. Blockchain Interoperability

The emergence of the blockchain interoperability research area has brought attention from both the industry and academia. Blockchain interoperability projects have been tackled as early as in 2016 [12].

A recent survey classifies blockchain interoperability studies in three categories: Cryptocurrency-directed interoperability approaches, Blockchain Engines, and Blockchain Connectors [12]. Cryptocurrency-directed approaches are mostly industry solutions that provide interoperability across public blockchains. This category has a focus on asset interoperability and is divided into sidechains, hash lock time contracts, notary schemes, and hybrid solutions. Sidechains allow for offloading transactions to a secondary chain, enhance performance, and provide features that the main chain would not provide. Sidechains also enable the representation of a token from the mainchain at the secondary chain. Some sidechain solutions include the BTC Relay [13], Zendoo [14], and RSK [15]. Hash lock time contract solutions enable cross-chain atomic operations using smart contracts. Wanchain uses this scheme and provides loan services with cryptocurrencies [16]. Notary schemes are centralized or decentralized entities that mediate token exchange (e.g., cryptocurrency exchanges). Finally, hybrid solutions combine characteristics from previous approaches. The cryptocurrency-directed approaches only work for transferring different types of cryptocurrencies between blockchain network. As a result, these approaches cannot be used for permissioned blockchains with arbitrary assets and smart contracts, which are the focus of this paper.

The second category is blockchain engines, which enable creating customized blockchains that can interoperate by providing reusable data, network, consensus, and contract layers. Platforms like Polkadot [17] and Cosmos [18] provide such infrastructure, with "free interoperability" among the instances they allow to create. These approaches are fundamentally different from what has been proposed in this paper. Instead of enabling blockchain interoperability for currently running blockchains, blockchain engines propose blockchain networks that are interoperable by design. As a result, these solutions cannot be used for currently running permissioned blockchains.

The Blockchain Connector category is composed of interoperability solutions that are not cryptocurrency-directed or blockchain engines. They include trusted relays, blockchain agnostic protocols, blockchain of blockchains solutions, and blockchain migrators. Each of these categories responds to a particular set of use cases. Trusted relays allow discovering the target blockchains, often appearing in a permissioned blockchain environment where trusted escrow parties route cross-blockchain transactions. An interesting trusted relay approach is Hyperledger Cactus [19], the most recent Hyperledger project aiming to connect a client to several blockchains, whereby transactions are endorsed by trusted validators. Cactus focuses on providing multiple use case scenarios via a trusted consortium. Another trusted relay is proposed by Abebe et al. [20], which is an interoperability solution between Fabric networks using Hyperledger Fabric chaincode and a protocol-buffer based communication protocol. Blockchain agnostic protocols enable cross-blockchain communication between arbitrary distributed ledger technologies, including refactoring and making changes to each blockchain. Blockchain of blockchains are approaches that allow users to build decentralized applications using multiple blockchains. Finally, blockchain migrators enable the state of one blockchain to be migrated to another blockchain. Simple blockchain migrations have already been performed, paving the direction for complex blockchain migrations [12].

Our pub-sub solution is considered a trusted relay (although decentralized), as it contains a blockchain mediating communication across heterogeneous blockchains [12]. While trusted relays can provide a straightforward way of achieving interoperability, most of them are not trustless (e.g., contain a centralization point). Our solution is a decentralized trusted relay that implements a pub/sub system, anchored on the trust that underlying blockchains offer.

B. Blockchain-based Publish/Subscribe Protocol

The blockchain technology has been applied to the pub/sub paradigm in a few previous studies. However, those studies adopt blockchain to address the existing problems in other areas, such as IoT [21], supply chain [22], multi-tenant edge cloud [23], and digital trading [24].

Huang et al. [23] exploit blockchain technology to enhance the security of pub/sub communications in multi-tenant edge clouds. Most topic-based and broker-enabled pub/sub streaming systems use centralized cloud servers to store sensitive metadata and access control lists (ACL), which can compromise the confidentiality, anonymity, and integrity of tenants' data. Alternatively, critical data such as ACL and identity information, as well as the hash of raw messages, and operation logs, can be stored on the blockchain to guarantee data security and integrity. Their smart contracts implement access control mechanisms to authorize requests from publishers and subscribers.

Trinity [22] proposes a blockchain-based distributed publish/subscribe broker to solve the existing flaws of centralized brokers in IoT and supply chain monitoring applications.

Trinity has three main components: blockchain network, broker, and clients. The blockchain network is responsible for consensus in the system and persistent storage. The broker handles the communications between the blockchain network and clients. The clients are the publishers and subscribers of the topics. The blockchain network is pluggable, and the authors have used Tendermint, Hyperledger Fabric, Ethereum, and IOTA. The broker has used the Mosquitto MQTT broker, which provides a single point of failure.

Zhao et al. [25] have proposed Secure Pub-Sub (SPS), which provides fair payment based on a reputation for publishers and subscribers in cyber-physical systems. They use the Bitcoin's network to enable payments between the entities, and they propose a reputation mechanism that helps calculate the price of data.

Lv et al. [21] presents a decentralized privacy-preserving pub/sub model for IoT systems to solve centralized brokers' problems such as single point of failure, data tampering due to corrupter brokers, and heavy encryption algorithms. The presented model applies public-key encryption with equality test [26] and ElGamal [27] to protect participants' (both publishers and subscribers) privacy. A system prototype is implemented and evaluated against the feasibility of the proposed model.

Bu et al. [24] and Zupan et al. [28] have proposed blockchain-based pub/sub brokers to address the drawbacks of traditional pub/sub systems such as privacy and accountability. However, they have not explained their implementation and evaluation in their studies.

All these studies adopt blockchain to improve the centralized predicaments in traditional pub/sub systems in distinct application domains, whereas our paper focuses on establishing robust and practical interoperability between permissioned blockchains with different architectures and infrastructures. To the best of our knowledge, our paper is the first study that enhances blockchain interoperability utilizing the pub/sub communication model across heterogeneous blockchains (Hyperledger Fabric/Hyperledger Besu).

III. System Design

In this section, we first discuss the design principles that a blockchain interoperability solution should follow. We then propose our interoperability solution and explain its components and message flow.

A. Design Principles

Blockchain interoperability aims to enable applications to use the assets and information available on blockchains other than their main blockchain network [12]. This allows for a greater range of applications. A blockchain interoperability solution should take into account the following design principles:

- The blockchain networks are independent, and they may have different architectures.
- The blockchain networks are in full control of their assets and information.
- The transfer protocol should be technology agnostic.

- The interoperability solution should not require significant changes in the source and destination networks.
- The blockchain networks should be able to incorporate the solution with minimal effort.

Following the mentioned principles, we present our solution, which allows interoperability based on a publish/subscribe architecture.

B. Components

The platform proposed in this paper aims to solve the interoperability problem of permissioned blockchains using the publish/subscribe pattern. When a blockchain wants to use the data from another blockchain, there needs to be a way to fetch and transfer this data between the networks securely. Moreover, if the data changes in the source network, the destination network should be notified of the change. Figure 1 shows the architecture of the platform and the message flow.

The *publisher blockchain* is the blockchain network that sends the data, also referred to as the source network. For a publisher to participate in this platform, it needs to run the appropriate connector smart contract on its blockchain and enroll as a publisher in the broker. The publisher can then create as many topics as they want and use the connector smart contract to publish the changes to the topic.

The *subscriber blockchain* is the blockchain network that received the data, also referred to as the destination network. Similar to the publisher, the subscriber also needs to run the appropriate connector smart contract and enroll as a subscriber. It can then subscribe to any topic available on the broker blockchain. Every time the topic changes, the broker notifies the subscriber by invoking the connector smart contract. There can exist many subscribers for a topic.

The *broker blockchain* is the core component of the platform. It is a blockchain network that stores all the information about the topics and the blockchains that participate in the interoperability process. Since the broker is a blockchain, operations regarding interoperation are immutable and traceable, providing a robust basis for accountability. The broker has two different smart contracts, the *connector smart contract* and the *topics smart contract*. The connector smart contract stores the information about the participating blockchain networks and how the broker can interact with them. The topics smart contract is responsible for storing the information about the topics such as their publisher, subscribers, and the latest message.

C. Message Flow

The complete interoperation process in the platform is shown in Figure 1. For simplicity, only one publisher and one subscriber blockchain are shown in this figure. However, for each topic, there can be an unlimited number of subscribers and, in general, there is no limit on the number of publisher and subscriber blockchains. A detailed explanation of each step in Figure 1 follows:

1) For any blockchain network to interact with the broker blockchain, it must enroll in the connector smart
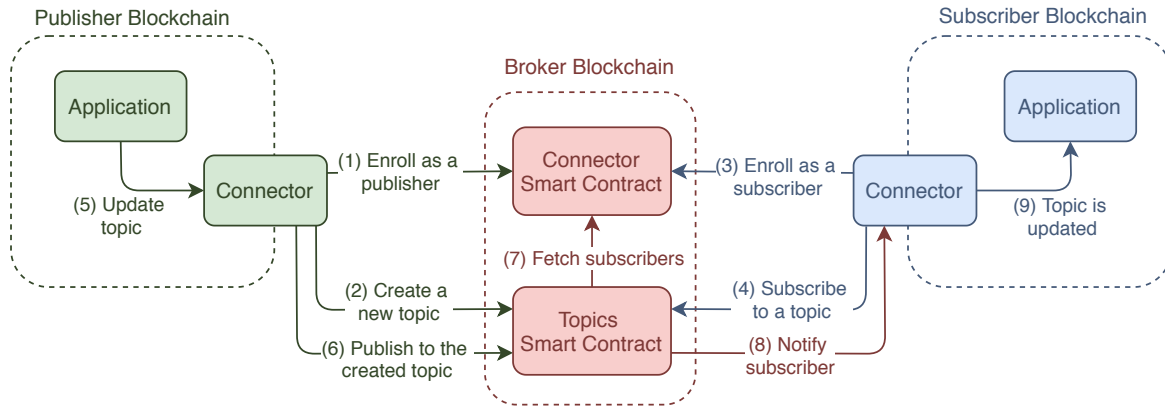
Fig. 1. Architecture of the platform and the message flow.

contract. During this process, the information that the broker needs to be able to interact with the blockchain is registered in the connector smart contract. As a result, the publisher is required to enroll in the connector smart contract as a publisher. This step only needs to be performed once, when the publisher wants to create its first topic. It can then create topics or publish to existing ones without the need to be enrolled again.

2) A publisher that is registered in the connector smart contract can create a new topic. In this step, the publisher needs to specify a name for the topic and the initial topic message. This step only needs to be executed once for each topic.

3) Similar to the publisher blockchain, the subscriber blockchain should also enroll in the connector smart contract. This step only needs to be done once, when the subscriber wants to subscribe to a topic for the first time.

4) To receive a notification when a topic has been changed, the subscriber should subscribe to the topic by sending a request to the topics smart contract. This results in the subscriber being added to the list of all subscribers for the topic. This step only needs to be performed once for each topic.

5) Whenever needed, the application in the publisher blockchain can update the topic by sending a request to the connector smart contract.

6) The connector smart contract sends a publish request to the topics smart contract for the existing topic.

7) As soon as a publish request is received by the topics smart contract, the smart contract fetches the information about all the subscribers of the topic from the connector smart contract. It includes information on how the broker can interact with each of the subscribers.

8) The topics smart contract then uses the data fetched from the connector smart contract to notify all the subscribers of the change in the topic. This happens by sending an update request to the connector smart contract in each of the subscriber networks.

9) In each subscriber network, the connector smart contract receives the update for the topic and notifies the application.

IV. IMPLEMENTATION AND DEPLOYMENT

A prototype of the proposed platform has been implemented as a proof-of-concept to demonstrate the feasibility of the design. This project is a Hyperledger Labs open-source project[1], under the Apache License, Version 2.0. To show the interoperability capabilities of the platform, we implemented two example subscriber networks, as well as an example publisher network. The broker and the example publisher are implemented using two different Hyperledger Fabric V2 [29] networks. The two example subscribers are implemented using Hyperledger Fabric V1.4 [30], [31], and Hyperledger Besu [32]. In this section, the implementation and deployment details of the broker and the example networks are discussed.

A. Broker Blockchain

The broker blockchain acts as a messaging broker between other blockchains to enable interoperability. When choosing the blockchain solution to implement the broker network, we had to ensure that the solution fits well with the needs of this platform. First, since we aim to address interoperability in permissioned blockchains, the broker also needs to be permissioned. Moreover, many permissioned blockchains are enterprise-level, and they may have privacy and governance concerns, which our broker aims to addresses using blockchain that enables immutability and traceability. We need a blockchain solution for the broker network that considers these needs. Finally, the smart contracts that need to be implemented on the broker blockchain are complicated, and the blockchain needs to support this kind of smart contract.

Hyperledger Fabric is an open-source permissioned blockchain that has been designed for enterprise use cases. Unlike the open permissionless blockchains that have scalability issues, Fabric enables high transaction throughput

---
[1]https://github.com/hyperledger-labs/pubsub-interop

and low transaction confirmation latency. The architecture of Hyperledger Fabric is highly modular and configurable, which enables customization for each specific use case. Many blockchains only support smart contracts written in non-standard and domain-specific programming languages, making it impossible to implement smart contracts that require a Turing-complete language. On the other hand, Hyperledger Fabric supports smart contracts in general-purpose programming languages such as Go, Node.js, and Java [29].

In the broker network, we leverage the capabilities of Hyperledger Fabric V2.2 to implement a messaging broker. We implement two chaincodes called the topics and the connector to support the features needed for the broker. The topics chaincode is responsible for keeping all the topics and their corresponding details. In Hyperledger Fabric, everything is stored as a key-value pair. In the topics smart contract, the key is a unique topic ID. The value is an object that includes the following properties: name, publisher, subscribers, and message. The name of a topic is a string value set by the publisher when creating the topic. Each topic has one publisher, the blockchain network that has registered the topic on the broker, which is the only blockchain that can make changes to the topic. The subscribers' property stores a list of all the blockchains that have subscribed to the topic. It is worth mentioning that the publisher and the subscribers' properties only accept objects stored on the connector blockchain.

The connector chaincode is responsible for storing the connection details of other blockchain networks. Similar to the topics chaincode, the key in the key-value pair used in this chaincode is a unique ID for each blockchain. The value is an object that has the following properties: name, type, server IP, port, extra information. The name is a string value that can be selected when enrolling in the network. Type shows what kind of blockchain technology this network is using. Currently, support for Fabric and Besu has been implemented, and other blockchains will be added in future versions. The server IP and port are used by the broker blockchain to access the publisher or subscriber using an HTTP request. The extra information property stores network-specific details that may be needed when interacting with the blockchains. For instance, for a Hyperledger Besu network, this includes the private key, address, and the contract application binary interface (ABI) that the broker should use to send a request to the Besu network. This kind of implementation allows our solution to be extendable to other blockchains, independent of their underlying network.

To better understand how the topics and connector chaincodes work, we need to discuss their implemented functionalities. Figure 2 shows the UML class diagram of the implemented chaincodes. The Hyperledger Fabric contract API provides an interface for developing smart contracts and applications. Each developed chaincode should extend the contract class from this API and then implement the required logic. In each smart contract, the *InitLedger* function is used to create a set of initial assets on the ledger when the chaincode is deployed. In the topics chaincode, the *CreateTopic* function

is used to create a new asset of type topic. The *QueryTopic* and the *QueryAllTopics* functions can be used to query one specific topic and all the existing topics, respectively. The connector chaincode implements the same initialize, create, and query functionalities but for assets of type blockchain.
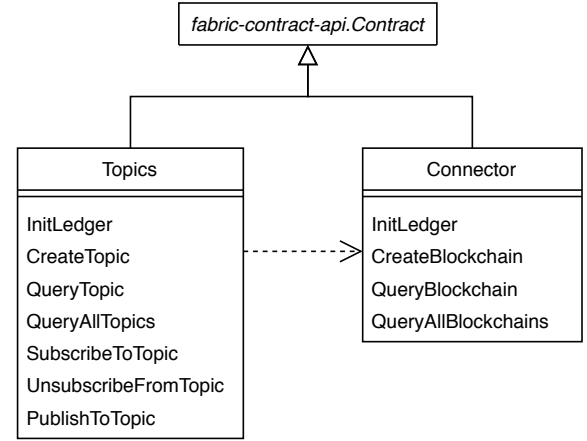

Fig. 2. UML class diagram of the implemented chaincodes.

Other than the mentioned functions, the topics blockchain also implements *SubscribeToTopic*, *UnsubscribeFromTopic*, and *PublishToTopic* functionalities. When a destination blockchain wants to get notified of a topic's change, it subscribes to that topic by invoking the *SubscribeToTopic* function. The subscriber can also unsubscribe from the topic at any time by invoking the *UnsubscribeFromTopic* function. Finally, the *PublishToTopic* function is used by the source blockchain network when they want to update the topic's message. An invoke request to this function triggers update requests to all the subscribers of the topic. Algorithm 1 shows the detailed implementation of the *PublishToTopic* method. First, the broker retrieves the latest version of the topic from the ledger. In the case that no topic was found, it immediately throws an error. Next, the topic's message is updated with the new message value and the topic's state is put to the ledger. The next step is for the broker to notify all the subscribers. For each of the subscribers of the topic, the blockchain object is queried from the connector contract. This inter-chaincode communication is also shown in Figure 2. Then, given the type of subscriber blockchain, the steps to invoke the remote network are followed.

B. Subscriber Blockchains

The subscriber, or destination blockchain, is the blockchain that requires information from another blockchain to run a task. For the subscriber to be able to participate in the platform, it needs to have the appropriate connector smart contract deployed on it. We have implemented subscriber connector contracts for Hyperledger Fabric V1.4 and Hyperledger Besu. However, the connector is a simple smart contract that can also be developed by the owners of the subscriber blockchain. This smart contract needs to keep track of the topics that the subscriber has subscribed to and store their latest version for

---

Algorithm 1: PublishToTopic Method

Input: topicID, newMessage
Result: Subscribers are notified of the new message
1   topicState ← getState(topicID)
2   **if** *!topicState* **then**
3 | throw error
4   **end**
5   topicState.message ← newMessage
6   putState(topicID, topicState)
7   **for** *subID* **in** *topicState.subscribers* **do**
8 | subState ← query *subID* from connector contract
9 | **if** *subState.type = Fabric* **then**
10 | | follow steps to invoke a remote Fabric network
11 | **else if** *subState.type = Besu* **then**
12 | | follow steps to invoke a remote Besu network
13 | **end**
14   **end**

---

other smart contracts to access at any time. Two example subscriber networks have been implemented to demonstrate the interoperability capabilities of the platform.

The first example subscriber is implemented using Hyperledger Fabric V1.4. The second example subscriber is implemented using Hyperledger Besu, an open-source Ethereum client that supports private and permissioned blockchains. Besu can create networks that work based on a proof of work (PoW) or a proof of authority (PoA) consensus algorithm. In this work, we implemented a PoW network using Besu, which can be thought of as a private Ethereum network. We then implemented a connector smart contract in Solidity to keep a record of the subscribed topics.

C. Publisher Blockchains

The publisher, or the source blockchain, is the blockchain network that needs to send information to other blockchains. Similar to what we have in the subscriber blockchain, a connector smart contract is also required for the publishers. However, the connector is slightly different in the publisher. The publisher connector should not only keep track of the topics, but it should also connect to the broker blockchain to publish the topics. We implemented an example publisher network using Hyperledger Fabric V2.2.

V. Experimental Evaluation

In this section, we focus on evaluating the performance of the implemented prototype of the broker blockchain. The goal is to see how the throughput and latency of the system changes in different scenarios. We have conducted two series of experiments to achieve this goal. The first set of experiments aims to show the performance metrics of different functionalities in the broker blockchain. The second set of experiments focuses on the publish function, which is the most important and time-consuming component of the broker blockchain.

We have used Hyperledger Caliper [33] to run the experiments. Hyperledger Caliper is an open-source blockchain performance benchmark tool that allows performance measurement for different blockchains, such as Hyperledger Fabric, Ethereum, Hyperledger Besu. In Hyperledger Caliper, the

workloads or benchmarks are responsible for generating the content of each transaction that is sent to the blockchain network. Given the network and benchmark configurations, Caliper uses a set of independent workers to send scheduled requests to the blockchain network and monitor the response. When the tests are finished, Caliper generates a performance report consisting of the average throughput and minimum, maximum, and average latency throughout the test. The throughput shows the number of transactions that were processed in the system in a given time. The latency shows the amount of time it takes for a transaction to be finished and added to the ledger.

Table I summarizes the specifications of each component in the experimental evaluation. We have set up Hyperledger Caliper on a separate machine to ensure that its process does not affect the performance of the broker network. We use five workers, a fixed rate controller, and a test duration of 60 seconds for each benchmark round. The broker network is implemented using Hyperledger Fabric V2.2 with two peer organizations and an orderer organization, each with an independent certificate authority. Each of the peer organizations hosts one peer node, and the orderer uses Raft implementation. Two chaincodes have been implemented that run on one channel. The Fabric subscriber, implemented using Fabric V1.4, has two organizations, each hosting two peers. The whole subscriber network uses one Solo orderer and one Fabric certificate authority. The Besu subscriber implements a private Ethereum network with PoW consensus algorithm. The publisher has been implemented using Hyperledger Fabric V2.2 with the same configurations as the broker network.

TABLE I
EXPERIMENTAL EVALUATION SETUP

Component	Type	CPU	RAM	Disk
Caliper Benchmark	N/A	8 vCPU	30 GB	288 GB
Broker	Fabric V2.2	8 vCPU	30 GB	288 GB
Fabric Subscriber	Fabric V1.4	2 vCPU	7.5 GB	36 GB
Besu Subscriber	Besu	2 vCPU	7.5 GB	36 GB
Publisher	Fabric V2.2	2 vCPU	7.5 GB	36 GB

The first set of experiments focuses on the performance evaluation of broker blockchain. In these experiments, we conduct a series of tests using Hyperledger Caliper for each functionality that broker blockchain offers. Figure 2 summarizes all these functionalities. Each type of transaction goes through a specific set of steps in Hyperledger Fabric, which highly influences the response time for that transaction. For instance, an invoke transaction goes through endorse, order and commit steps. On the other hand, a query transaction is not transferred to the orderer, and the response is immediately sent back by the peer. The create actions in the connector and topics smart contract are invoke actions that have very similar implementations. The same goes for the query actions in the two smart contracts. As a result, it would be repetitive to run
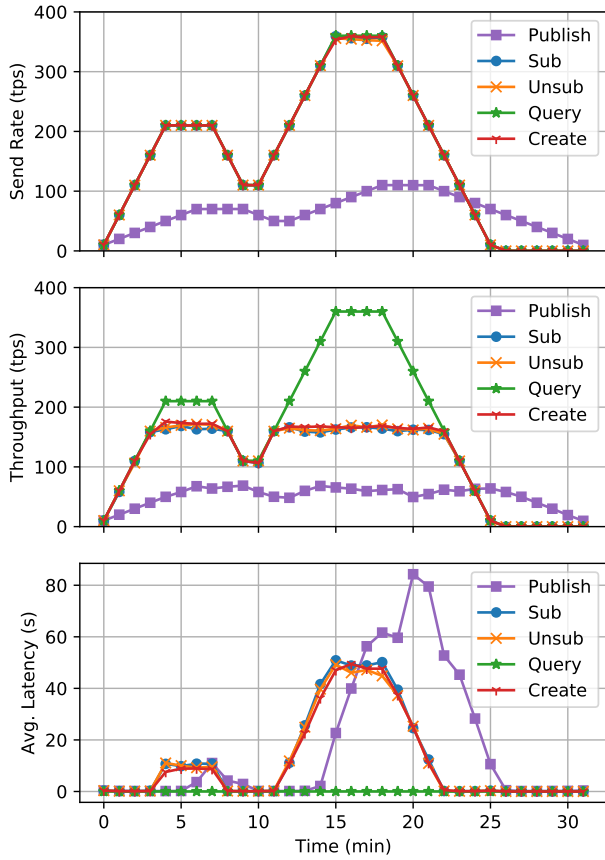
Fig. 3. The trend of system throughput and average latency for various functionalities throughout time with the change of request send rate. The words publish, sub, unsub, query, and create in the plots stand for PublishToTopic, SubscribeToTopic, UnsubscribeFromTopic, QueryTopic, and CreateTopic functions, respectively.

performance evaluation experiments for both smart contracts. Therefore, we run the experiments on the topics smart contract.

The topics smart contract has five important functionalities: create a topic, query a topic, publish to a topic, subscribe to a topic, and unsubscribe from a topic. For each of these actions, we run a set of experiments by changing the transaction send rate in the Hyperledger Caliper benchmark. The goal is to see how the system's throughput and average latency changes when the send rate is changed. Figure 3 shows the details of these experiments. It can be seen that the send rate follows the same pattern for all the actions except for *PublishToTopic*. The reason for this difference is that the *PublishToTopic* action takes more time and needs more resources to run compared to other actions. Consequently, broker blockchain's hardware limits are reached when the network receives more than roughly 100 publish transactions in each second. We discuss the behaviour of the network with different *PublishToTopic* requests in the second set of experiments shown in Figure 4. As a result of this limitation, we lowered the send rate for the *PublishToTopic* action in our experiments.

It can be seen in Figure 3 that the *SubscribeToTopic*, *Un-*

subscribeFromTopic, and *CreateTopic* have similar behaviours under the same send rate. These three actions are of type invoke. Since an invoke transaction proposes a change in the blockchain, it needs to go through the consensus algorithm, which can be time-consuming. Since the three actions are of the same type, and none need heavy computations in execution, the system throughput and latency for all of them are similar. As shown in the experimentation results, when the send rate is lower than a threshold (around 160 TPS in this case), the throughput is the same as the send rate, and the average latency is only a few hundred milliseconds (around 100 to 300 milliseconds). This shows that with send rates below the threshold, all transactions are processed immediately. When the number of create, subscribe, or unsubscribe transactions sent in each second is more than the threshold, the broker network's processing limit is reached. The throughput is limited to the broker's maximum capacity (around 160 TPS), and the transactions are queued before being processed, which results in an increase in the latency. Figure 3 shows that when the send rate for the create, subscribe, or unsubscribe transactions is around 210 TPS, the average latency increases to about 11 seconds. The latency keeps increasing with higher send rates and reaches approximately 50 seconds with a send rate of 360 TPS.

The *QueryTopic* action is different from the previous ones. Since a query transaction does not go through the consensus protocol, its process is much faster. The send rate pattern used for the query is similar to that of create, subscribe, and unsubscribe. However, the throughput and average latency act very differently. The throughput follows the same pattern as the send rate, and the average latency is around 10 milliseconds throughout the whole experiment. These results show that this experiment does not reach the process limit for *QueryTopic*.

Finally, the *PublishToTopic* is similar to create, subscribe, and unsubscribe because they are all invoke transactions. However, the publish action requires stronger computations. As mentioned earlier, since the publish action needs more time and computational resources, we use a different send rate pattern. If we were to use the same send rate, the broker blockchain's hardware limits would be reached, resulting in the experiments being halted. We discuss this in more detail in the second set of experiments shown in Figure 4. To ensure that the performance of the remote source and destination networks do not influence the performance evaluation of the broker network, we only send dummy requests to the subscriber networks during the experiments. It can be observed from Figure 3 that the publish action reaches the processing limit of the broker network much faster than the other invoke transactions. With send rates of about 70 TPS and more, the throughput is limited to 65 TPS. The average latency for the publish action has more fluctuations compared to other invoke actions. The main reason for this fluctuation is that in the publish method, depending on the number of subscribers that the topic has, the processing time can vary. In this experiment, the average latency gets as high as 80 seconds, with the send rate of 110 TPS.
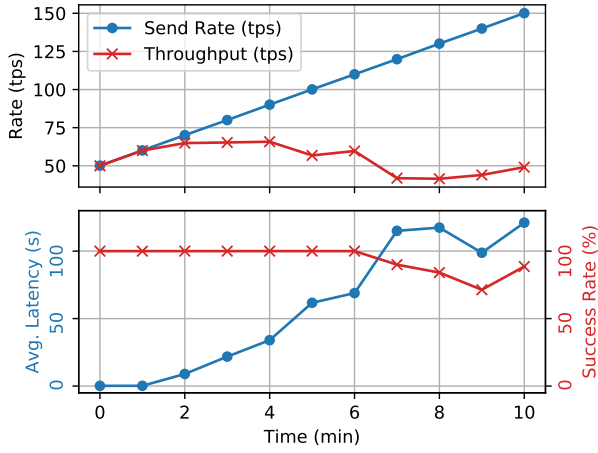
Fig. 4. The trend of system throughput, average latency, and request success rate throughout time with the change of send rate.

Given the limits of the *PublishToTopic* action, we decided to run some additional experiments on this type of transaction. This experiment aims to find the broker network's limits and discover what happens when the limit is reached. In the previous experiment, we discovered that the processing limit for the publish transactions is reached at the sent rate of around 70 TPS. We also observed that the latency increases and the throughput is limited for send rates above 70 TPS and below 110 TPS. However, we would like to know what happens if the send rate is more than 110 TPS. In this experiment, we linearly increase the send rate from 50 to 150 TPS and observe the throughput, latency, and transaction success rate. Figure 4 shows the results of this experiment. Similar to the previous experiment, we see that the throughput is limited, and the latency is increased when the send rate reaches 70 TPS. Nevertheless, the interesting change happens at the 120 TPS send rate. At this point, a significant drop in the throughput and a significant rise of latency are observed. Moreover, the transaction success rate is not 100% anymore. From this point on, a portion of the transactions fail since the broker network has reached its hardware limits.

VI. Discussion and Future Work

To enable blockchain interoperability, we have proposed the use of a broker blockchain as a middleman. The broker blockchain acts as a decentralized trusted relay between the source and destination network. Using a relay enables the interoperating networks to transfer data with minimal effort. Verifying the data and handling the communications between different blockchain networks can be delegated to the relay. As a result, there is no need for source and destination networks to make fundamental changes to their underlying structure. The relay network is also a blockchain network; while exploiting all desirable features offered by blockchain, it runs smart contracts implementing the interoperability functional-

ity. Therefore, the broker blockchain allows the interoperation to be seamless, transparent, and secure.

The platform proposed in this paper stores the destination and source blockchains as assets on the distributed ledger. As a result, a large number of blockchains can be supported as there are no limits on the number of assets. A study on the performance of Hyperledger Fabric V1.1 shows that the network can scale to 100+ peers [34], [35]. As the broker network has been implemented using Fabric V2.2, we expect this number to be even higher in our network. Therefore, at least 100 peers can participate in the governance of the broker blockchain.

In the current prototype of our platform, every participant can subscribe to all existing topics, and there is no access control mechanism implemented. The private data feature presented by Hyperledger Fabric can be utilized to establish private channels in the broker blockchain and keep data topics separate from other organizations. Furthermore, an access control mechanism can be added to our pub/sub system to control the flow of data at a more granular level, for instance, the decentralized access control proposed by Rouhani et al. [36].

It is also possible to conduct authorization processes with minimal information disclosure if one uses the novel self-sovereign based access control model [37]. This way, we could model each blockchain as an agent to prove that they own specific credentials needed for the access control process. Moreover, the publisher network may choose only to make a topic available to a subset of subscribers. Access control can be used to manage the blockchains that can access each topic.

VII. Conclusion

With blockchain technology gaining popularity in academia and industry, many blockchain networks are being introduced worldwide. These networks are highly isolated and incompatible with each other, resulting in silos of data and assets. Blockchain interoperability solutions can revolutionize this technology by enabling data and asset transfers between homogeneous and heterogeneous blockchains. In this paper, we proposed a blockchain interoperability solution based on the publish/subscribe architecture. Our solution consists of a broker blockchain that keeps a record of the data being transferred between blockchain networks. The blockchains that aim to participate in the interoperability can connect to the broker network as publishers or subscribers, depending on their role. A prototype of the broker blockchain has been implemented using Hyperledger Fabric. Moreover, an example publisher and two example subscribers have been implemented using Hyperledger Besu and two versions of Hyperledger Fabric to show that the design works for heterogeneous blockchains. The network's performance has been analyzed using a benchmark tool to identify the platform's limits and bottlenecks. The implementation and evaluations indicate the feasibility of the idea with satisfactory performance, and the bottleneck is identified to be the process of publishing a new message to a topic. Finally, a discussion on the extensibility, scalability, and possible improvements of the system is presented.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." https://bitcoin.org/bitcoin.pdf, 2008. Last accessed 2020-07-17.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] B. Chase and E. MacBrough, "Analysis of the xrp ledger consensus protocol," *arXiv preprint arXiv:1802.07242*, 2018.

[4] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *Journal of the American Medical Informatics Association*, vol. 24, no. 6, pp. 1211–1220, 2017.

[5] S. Rouhani, L. Butterworth, A. D. Simmons, D. G. Humphery, and R. Deters, "Medichain tm: a secure decentralized medical data asset management system," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1533–1538, IEEE, 2018.

[6] T. M. Fernández-Caramés and P. Fraga-Lamas, "A review on the use of blockchain for the internet of things," *IEEE Access*, vol. 6, pp. 32979–33001, 2018.

[7] C. Fan, H. Khazaei, Y. Chen, and P. Musilek, "Towards a scalable dag-based distributed ledger for smart communities," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pp. 177–182, IEEE, 2019.

[8] R. Belchior, M. Correia, and A. Vasconcelos, "JusticeChain: Using Blockchain To Protect Justice Logs," in *CoopIS 2019: 27th International Conference on Cooperative Information Systems*, 2019.

[9] R. Belchior, A. Vasconcelos, and M. Correia, "Towards Secure, Decentralized, and Automatic Audits with Blockchain," in *European Conference on Information Systems*, 2020.

[10] S. Ghaemi, H. Khazaei, and P. Musilek, "Chainfaas: An open blockchain-based serverless platform," *IEEE Access*, vol. 8, pp. 131760–131778, 2020.

[11] J. H. Park and J. H. Park, "Blockchain security in cloud computing: Use cases, challenges, and solutions," *Symmetry*, vol. 9, no. 8, p. 164, 2017.

[12] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *arXiv preprint arXiv:2005.14282*, 2020.

[13] Ethereum Foundation and Consensys, "BTC-relay: Ethereum contract for Bitcoin SPV," 2015.

[14] A. Garoffolo, D. Kaidalov, and R. Oliynykov, "Zendoo: a zk-SNARK Verifiable Cross-Chain Transfer Protocol Enabling Decoupled and Decentralized Sidechains," tech. rep., 2020.

[15] S. Lerner tech. rep., RSK, 2015.

[16] J. Lu, B. Yang, Z. Liang, Y. Zhang, S. Demmon, E. Swartz, and L. Lu, 2017.

[17] G. Wood, "Polkadot: Vision for a Heterogeneous Multi-Chain Framework," *Whitepaper*, 2017.

[18] J. Kwon and E. Buchman, "Cosmos Whitepaper," tech. rep., 2016.

[19] H. Montgomery, H. Borne-Pons, J. Hamilton, M. Bowman, P. Somogyvari, S. Fujimoto, T. Takeuchi, T. Kuhrt, and R. Belchior, "Hyperledger Cactus Whitepaper." https://github.com/hyperledger/cactus/blob/master/whitepaper/whitepaper.md, 2020. Last accessed 2020-09-28.

[20] 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 *Proceedings of the 20th International Middleware Conference Industrial Track*, pp. 29–35, 2019.

[21] P. Lv, L. Wang, H. Zhu, W. Deng, and L. Gu, "An iot-oriented privacy-preserving publish/subscribe model over blockchains," *IEEE Access*, vol. 7, pp. 41309–41314, 2019.

[22] G. S. Ramachandran, K.-L. Wright, L. Zheng, P. Navaney, M. Naveed, B. Krishnamachari, and J. Dhaliwal, "Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 227–235, IEEE, 2019.

[23] B. Huang, R. Zhang, Z. Lu, Y. Zhang, J. Wu, L. Zhan, and P. C. Hung, "Bps: A reliable and efficient pub/sub communication model with blockchain-enhanced paradigm in multi-tenant edge cloud," *Journal of Parallel and Distributed Computing*, 2020.

[24] G. Bu, T. S. L. Nguyen, M. P. Butucaru, and K. L. Thai, "Hyperpubsub: Blockchain based publish/subscribe," in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pp. 366–3662, IEEE, 2019.

[25] Y. Zhao, Y. Li, Q. Mu, B. Yang, and Y. Yu, "Secure pub-sub: Blockchain-based fair payment with reputation for reliable cyber physical systems," *IEEE Access*, vol. 6, pp. 12295–12303, 2018.

[26] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong, "Probabilistic public key encryption with equality test," in *Cryptographers' Track at the RSA Conference*, pp. 119–131, Springer, 2010.

[27] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[28] N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: a decentralized, permissioned, publish/subscribe service using blockchains," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, pp. 15–16, 2017.

[29] Hyperledger, "Hyperledger fabric v2.2 documentation." https://hyperledger-fabric.readthedocs.io/en/release-2.2/, 2020. Last accessed 2020-10-22.

[30] Hyperledger, "Hyperledger fabric v1.4 documentation." https://hyperledger-fabric.readthedocs.io/en/release-1.4/, 2019. Last accessed 2020-10-22.

[31] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.

[32] Hyperledger, "Hyperledger besu documentation." https://besu.hyperledger.org, 2020. Last accessed 2020-10-22.

[33] T. L. Foundation, "Hyperledger Caliper." https://www.hyperledger.org/use/caliper, 2020. Last accessed 2020-11-5.

[34] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126927–126950, 2020.

[35] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[36] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, "Distributed Attribute-Based Access Control System Using a Permissioned Blockchain," *arXiv preprint*, 2020.

[37] R. Belchior, B. Putz, G. Pernul, M. Correia, A. Vasconcelos, and S. Guerreiro, "SSIBAC : Self-Sovereign Identity Based Access Control," in *The 3rd International Workshop on Blockchain Systems and Applications*, IEEE, 2020.