Automated Gateways: A Smart Contract-Powered Solution for Interoperability Across Blockchains

***Abstract*—This study introduces Automated Gateways, a new framework designed to facilitate the sharing of data and services across various blockchain networks through smart contracts. These contracts are integral and foundational, akin to the core infrastructure of the blockchain, and enhance existing systems by adding essential built-in interoperability features. This approach reduces the need to adopt new technologies or to depend on external services. Additionally, smart contracts handle accessibility and authorization policies for cross-chain transactions by defining fine-grained access control mechanisms to enable the selective sharing of methods between blockchains. The evaluation results indicated that Automated Gateways efficiently facilitate cross-chain transactions, decrease operational complexities, and maintain transactional integrity and security across diverse blockchain networks. With its focus on user-friendliness, self-managed permissions, and independence from external platforms, this framework is designed to encourage broader adoption within the blockchain community.**

***Index Terms*—Blockchain, Interoperability, Smart contract, Cross-chain, Access control**

I. Introduction

Blockchain technology has received significant attention from the academic and industrial sectors and is declared to disrupt various industries, such as healthcare [1], [2], finance [3], [4], and agriculture [5], [6]. This is because blockchain represents a paradigm shift from a centralized server-based network to a decentralized, transparent, and secure network by realizing traceable, tamperproof, and shareable transactions.

Outstanding attention toward blockchain technology has fostered the development of diverse blockchain platforms, each of which demonstrates remarkable potential. However, these platforms do not incorporate interoperability into their architectural design. Consequently, these blockchain networks lack support for direct service interworking, leading them to operate in isolation, which causes data and assets to be in silos and presents efficiency [7], [8] and trustworthiness [9], [10] challenges. Blockchain interoperability refers to the ability of different blockchain systems, each with its own unique distributed ledger, to work together. This interoperability allows transactions to be executed across multiple varied blockchain systems. Additionally, it ensures that the data recorded in one blockchain can be accessed, verified, and linked to by a transaction on a different blockchain [11], [12]. Such interoperability provides the necessary infrastructure for efficiency and innovation by expanding the capabilities of decentralized applications across blockchain platforms.

Although various interoperability solutions like Axelar [13], Polkadot [14], and Cosmos [15] have been developed, their lack of widespread adoption still leaves data and service silos as an unresolved issue. One major barrier to the practical application of these solutions is the extensive time and effort required for development teams to integrate them into existing platforms. This integration process involves a steep learning curve because developers must understand each platform's unique architecture, APIs, and design principles. For some solutions, there is also a need to construct and maintain specialized bridges [16], leading to extra development and maintenance costs. Moreover, reliance on external parties for support and infrastructure can lead to risks such as service discontinuation, further complicating the adoption of these interoperability platforms. However, most existing solutions focus primarily on public blockchain networks and tend to overlook the complexities of authentication and access management across permissioned blockchains, a gap that has not been addressed in the existing literature.

This study introduces an automated gateway framework, a novel blockchain interoperability solution designed to address challenges in data and service sharing across different blockchain platforms. Our solution utilizes smart contracts within existing blockchain platforms to determine and control authorization policies. This minimizes the need to learn new technologies and reduces the reliance on third-party services. The framework is user-friendly, featuring easy-to-use APIs that allow users to connect their blockchains with minimal coding, thus reducing the need to build bridges, gateways, or additional modules. Furthermore, our infrastructure relies solely on existing blockchain platforms, eliminating dependence on external interoperability services. We also implemented fine-grained access control mechanisms integrated through smart contracts to authorize and selectively share specific methods between blockchains.

To implement and evaluate our solution, we developed a distributed production network instead of a local test network. This method surpasses local simulation setups and test networks commonly used in similar studies [17], [18]. By focusing on ease of use and self-sufficiency, the Automated Gateways framework aims to achieve broad adoption in blockchain communities.

The remainder of this paper is organized as follows. Section II provides a summary of related studies on blockchain interoperability and existing frameworks. Section III presents the architecture, security overview, and process flow of the proposed framework. Section IV discusses the implementation and evaluation of the Automated Gateways framework. Finally, Section V draws conclusions from our findings and outlines the potential future directions for research and development.

II. Related Works

Blockchain interoperability, a multifaceted concept, has been explored using various lenses. Researchers have studied interoperability solutions for blockchains based on various criteria, such as functionality and design [19], [20], the impact of smart contracts on interoperability challenges, and the role of programming languages in smart contract development [21]. Moreover, these solutions can be classified based on system characteristics, including decentralization, locking mechanisms, verification, and communication processes, along with safety considerations such as trust, liveness, safety, and atomicity [22].

Blockchain interoperability has also been studied based on network architecture and consensus methods. Ghaemi et al. [23] propose a publish-subscribe architecture for facilitating inter-blockchain communications. In terms of consensus methods, Liu et al. [24] and Kammuller et al. [25] explored the aggregation and sequential ordering of blocks from different blockchains into 'meta-blocks,' and investigated the validation and management of cross-blockchain transactions.

Belchior et al. [19] introduce three primary categories for developing blockchain interoperability solutions: Blockchain of Blockchains (BoB), Public Connectors, and Hybrid solutions.

BoBs aim to create customized interoperable blockchains by providing reusable structures in various layers, such as data, networks, consensus, and smart contracts. Polkadot [14] and Cosmos [15] are well-known platforms that provide protocols to establish an interface for different state machines to connect and communicate with each other.

Current solutions only enable interoperability between public blockchains. Public connectors provide interoperability between public blockchains, focusing on trading digital assets and cryptocurrencies [26], [27]. They applied various methods, such as sidechains [28], notary schemes [7], [17], and hash locks [29] to develop a secure and reliable interoperability solution. The issue of trading digital assets and cryptocurrencies across blockchain networks has been widely investigated [30], [31], whereas this study centers on data and service sharing, that is, the transfer of data and methods of one blockchain network to another blockchain network [9].

The last category of blockchain interoperability solutions known as hybrid connectors focuses on integrating public and permissioned blockchain networks and platforms with heterogeneous structures [32]. Trusted relays, blockchain agnostic protocols, and blockchain migrators are the three main approaches applied in hybrid implementations.

III. System Design

This section provides an overview of the key design principles employed in the development of Automated Gateways. It also includes an overview of the overall architecture and various components that are integral to the proposed solution. Additionally, a detailed security overview and process flow are presented to explain the functioning of the solution.

A. Design Principals

The Automated Gateways solution is grounded in three fundamental principles, serving as its cornerstone:

1) **Decentralization:** Embracing the core tenet of Web3, our proposed solution prioritizes decentralization to mitigate the risks associated with a single point of failure and the reliance on trusted third parties. By adhering to this principle, the solution facilitates the seamless sharing of data and services across different blockchains.
2) **Transparency:** Given the objective of sharing services and data, transparency is considered crucial. This necessity is rooted in the requirements of trust-building, auditability, accountability, and verifiability. The incorporation of transparency into the interoperability solution is highlighted through the use of smart contracts and on-chain transactions to handle policies. This guarantees that any changes are recorded in the involved blockchains as immutable transactions.
3) **Self-reliance:** Standalone interoperability solutions introduce added complexity and challenges when adapting to different platforms. Learning and integrating these with various systems requires additional effort. Hence, the proposed smart contract-based gateway emphasizes self-reliance. This involves maximizing the use of technologies already present in the host blockchain platform, minimizing the need for new technologies, decreasing reliance on solutions managed by third-party companies, and mitigating alterations to the platform's architecture.

B. Architecture

To align with our design principles, the proposed solution is organized into two key components: Policy Management and Communication Management. As illustrated in Figure 1, the Policy Management component operates through a set of smart contracts, referred to as 'policy smart contracts'. They are deployed on the host blockchain to ensure distribution, transparency, and self-reliance. Distribution is ensured by deploying smart contracts across various nodes in the blockchain network. Transparency is achieved through transactions recorded on a shared ledger, and self-reliance is maintained by leveraging existing blockchain systems, rather than developing external services.

The Communication Management component, referred to as the relay, serves as a messenger for transferring data and services between different blockchains using the gRPC protocol. To achieve distribution within a blockchain, a relay must be deployed for each peer-hosting policy smart contract. To streamline the development and enhance adaptability, a relay was developed and released as a library. This allows easy integration into any source code used by blockchain owners, thereby minimizing the time and effort required for implementation. The relay components are divided into two parts. The first part is network-neutral, which means that it is blockchain-agnostic. This part of the relay is designed to operate independently of the host blockchain network type.

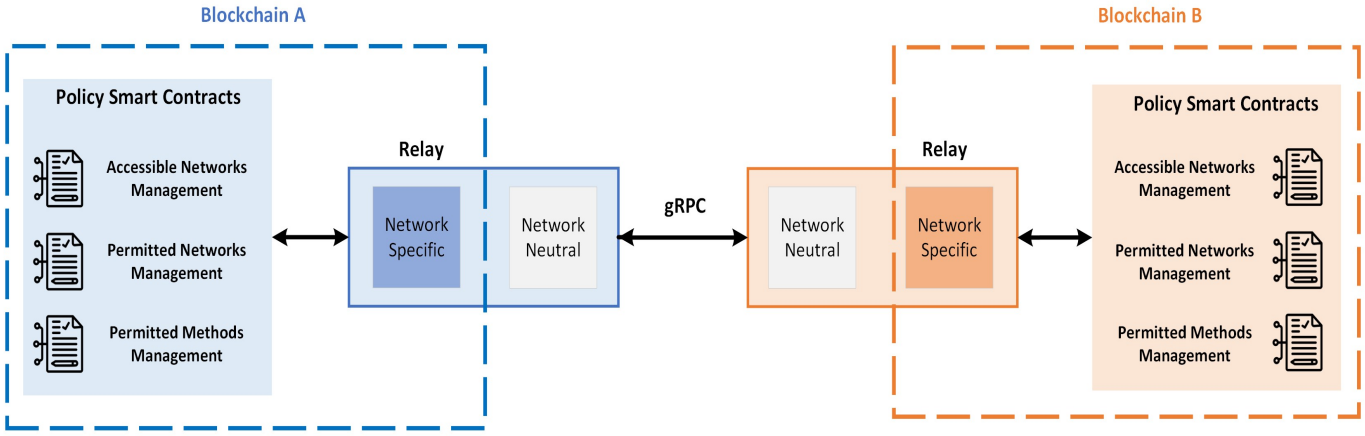

Fig. 1. The high-level architecture diagram of Automated Gateways

It manages the communication between blockchains and executes commands received from the user. The second part of the relay is network-specific and is responsible for establishing a connection between the relay and the host blockchain. Network-specific relays utilize features of the host blockchain, including its software development kit (SDK), to create a connection between the host blockchain and other blockchains. In the following sections, we provide a detailed exploration of both Policy Management and Communication Management components (relay).

1) Policy Management Component: This section comprises three smart contracts designed to establish accessibility and authorization policies. These three smart contracts include the following.

- **Accessible Networks Smart Contract:** This smart contract manages the blockchain networks accessible to the host blockchain. It maintains crucial information about these accessible networks, including IP addresses, URLs, network IDs, network names, and the names of managing companies.
- **Permitted Networks Smart Contract:** The Permitted Networks Smart Contract is responsible for managing information about blockchains that have been granted access to the host blockchain data or services, allowing them to proceed with requests for desired services and data on the host blockchain. The data maintained by this smart contract for permitted blockchains includes the IP address, URL address, company name, network ID, and network name.
- **Permitted Methods Smart Contract:** In addition to the Permitted Networks Smart Contract, the Permitted Methods Smart Contract introduces an additional layer of authorization to our system. While the Permitted Networks Smart Contract grants authorization to networks accessing data and services on the source blockchain, the Permitted Methods Smart Contract specifies the precise services and methods within the host blockchain that other networks can invoke and access. This smart contract maintains information about permitted methods through which permitted blockchains can receive service or data. This information includes the permitted method ID, permitted method name, the name of the smart contract where the smart contract is deployed, and the input and output types of the method.

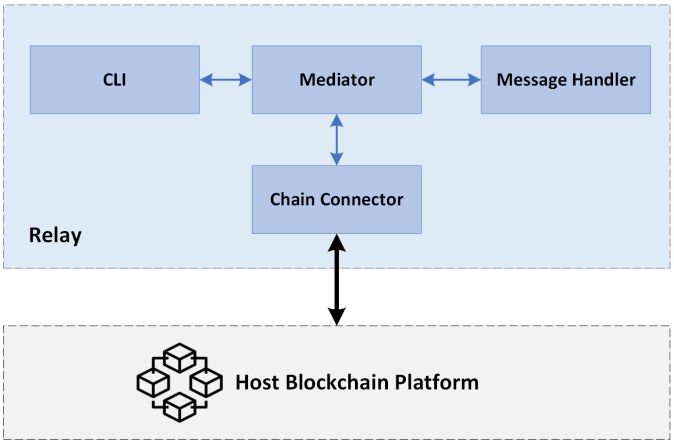

Fig. 2. Overview of the Relay Component

2) Communication Management Component: The Communication component is pivotal for establishing connections between the host and permitted blockchains, enabling the exchange of data and services. As depicted in Figure 2, this component includes four main components: a Chain Connector, the network-specific part of the relay, and a Mediator, Message Handler, and Command Line Interface (CLI), which constitute the network-neutral part of the relay.

The first component is the **Chain Connector**, which is responsible for establishing the connection between the relay and host blockchain. This component manages all interactions between the relay and host blockchain, including interactions with policy smart contracts, data retrieval, and service invocation on the host blockchain. Distinct Chain Connectors must

be implemented to support various host blockchain network types. The implementation of each Chain Connector is based on the software development kits provided by the creators of the respective host blockchain.

The second component of the relay library is **Mediator**. This component primarily focuses on connecting different elements of the relay. All requests for sending or receiving data or services are routed through the Mediator component, and the corresponding responses are sent back to the Mediator.For example, when a command is initiated from the Command Line Interface (CLI), it passes through the Mediator to interact with the host blockchain. Additionally, this component offers APIs to users, enabling them to utilize relay services. Users can leverage these APIs to initiate the interoperability process and send or receive data from the host or accessible blockchains.

The **Message Handler** component serves as the hub for all communication between the host blockchain and accessible blockchains. We opted for gRPC [33] as our connection protocol, given its advantages such as a simple and well-defined interface, seamless integration with the cloud ecosystem, and support for multiple programming languages. This component consists of two parts: the gRPC server, responsible for authentication and receiving requests, and the gRPC client, tasked with sending requests to the accessible blockchain.

The final component is the **Command Line Interface (CLI)**, designed to offer owners of the host blockchain a user-friendly way to interact with the interoperability platform without requiring coding skills. Through the CLI, blockchain owners can manage information about shared services and data, oversee accessible and permitted networks, and administer authorization policies. Moreover, they can invoke services that are accessible to the host blockchain. As illustrated in Figure 3, once the CLI is initiated, the user can choose to interact with either the host blockchain or an accessible blockchain external to the host infrastructure. If the user intends to engage with an externally accessible blockchain, they should initially select the desired accessible network from the available options in the accessible list. Subsequently, they can interact with the chosen network by listing the information of the host blockchain stored on the accessible blockchain, obtaining a list of accessible methods, or invoking accessible methods to receive data or services. Moreover, users can interact with the host blockchain through the CLI for policy management. The user is required to first select the policy smart contract they wish to work with, after which they can create, retrieve, remove, or update entities of the smart contract. At each state, the user is presented with a list of options, allowing them to provide input or receive the appropriate output. Additionally, they have the option to return to their previous state.

C. Security Overview

To enable the secure exchange of data and services, it is necessary to design an approach that ensures the security and confidentiality of interactions, both between two relays and between the relay and the host blockchain.

To establish a secure flow of data between two relays, we implemented certificate-based authentication [34]. Certificate-based authentication is a process wherein a digital certificate is employed to verify the identity of a user, device, or system. In this context, the owner of a blockchain generates certificates for its relay server and issues certificates for the relay clients of blockchains seeking to connect to the host blockchain (permitted blockchain). The host blockchain owner can utilize its own certificate authority or any valid certificate authority of its choice. After registering an external blockchain as a permitted blockchain, the blockchain owner should provide the permitted blockchain owner with a client certificate. The permitted blockchain owner can then introduce this certificate to the relay through relay configuration. This certificate plays a crucial role in the authentication process, as described in Algorithm 1, which is applied to all the requests received by the gRPC server.

Algorithm 1 Certificate authentication algorithm

```
1: procedure AUTHENTICATION(CertInfo)
2:     cn ← extarctCommonName(CertInfo)
3:     pn ← GetPermittedNetwroks(cn) ▷ get from chain
4:     if pn exists then
5:         return true                ▷ authentication succeeded
6:     else
7:         return false               ▷ authentication failed
```

As depicted in Algorithm 1, the authentication process between two relays initiates by extracting the common name, denoting the URL for which the certificate is issued, from the certificate data of the relay of a permitted blockchain. Then, a verification, in the host blockchain, is conducted against the Permitted Networks Management Contract to check whether this URL is registered. If the URL is registered as a permitted network, the request proceeds for further processing; otherwise, an error is returned.

To ensure secure data transfer between the relay and the host blockchain, the relay adheres to the protocol established by the respective blockchain network. As an illustration, Hyperledger Fabric employs certificate-based authentication [35], and our connection security aligns with this established protocol to connect relay to the Hyperledger Fabric blockchain network.

D. Process Flow

The process of data and service sharing in Automated Gateways comprises two distinct steps. The first step involves registration, during which the necessary configurations are established to enable desired access between blockchains. The second step involves the invocation of methods that enable blockchains to share data and services. In the rest of this section, we will discuss further details regarding these two steps through a detailed example. Consider a scenario with two blockchains, Blockchain(1) and Blockchain(2), where we aim

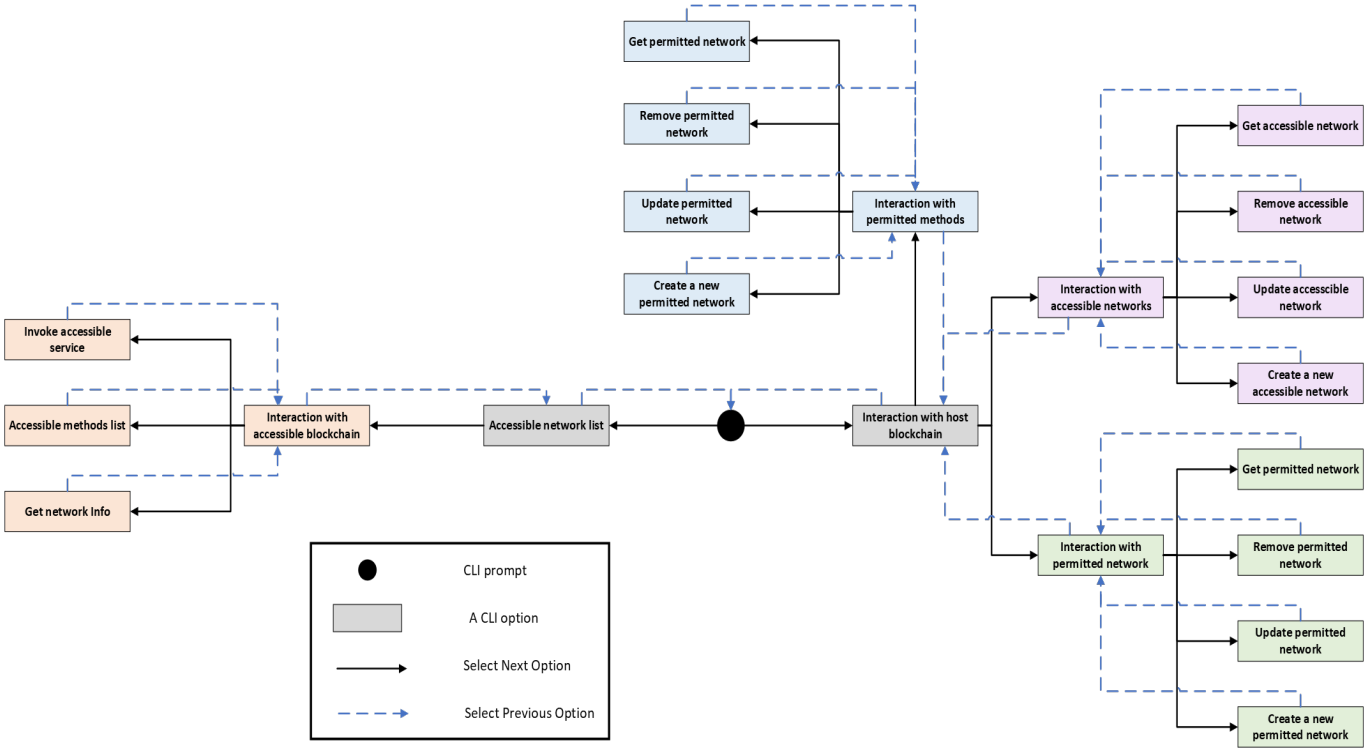

Fig. 3. A schematic view of Command Line Interface (CLI) navigation map.

to outline a flow allowing Blockchain(2) to access a service provided by Blockchain (1).

To successfully complete the registration process, the following steps must be accomplished:

1) **Request for Access:** This step initiates outside the domain of computer systems, where the owner of Blockchain(2) engages in negotiations with the owners of Blockchain(1) to obtain access to a specific service provided by one of the smart contracts on Blockchain(1).
2) **Certificate Issuance:** In case of getting permission, the owners of Blockchain(1) must issue certificates to the owner of Blockchain(2) specifically for the purpose of certificate-based authentication.
3) **On-chain Registration:** Following the issuance of certificates, the owner of Blockchain(1) is required to register Blockchain(2) as a permitted network, while the owner of Blockchain(2) should reciprocate by registering Blockchain(1) as an accessible network. Moreover, to enable access to the desired service, the owner of blockchain(1) should grant permission to the desired service for blockchain(2). This is done by registering the information of the method, alongside the Permitted Network Id of the blockchain, as a permitted method entity in the Permitted Methods smart contract.

In regard to the method invocation process, as it is illustrated in Figure 4, the owner of the blockchain(2) should follow these steps:

1) **Get Accessible Network Information:** For any communication between two blockchain relays, a connection must be established through the Message Handler component. To configure the Message Handler component of the relay, specific information about the destination accessible network, including its IP address, needs to be obtained from the blockchain. The owner of Blockchain(2) obtains information about the desired accessible network by interacting with the relay through the Command Line Interface (CLI). The CLI receives the owner's request and transmits it to the Mediator, which in turn sends it to the Chain Connector. Chain connectors get the response by calling GetAccessibleNetworksByAddress from the Accessible Networks Contract. The response from the chain follows the reverse stream back to the blockchain owner.
2) **Request for Permitted Network Information:** In this step, Blockchain (2) sends a request to Blockchain(1) to obtain its unique identifier stored on Blockchain(1). This identifier will be used in the next step to retrieve the list of methods accessible to Blockchain(2). When the relay of Blockchain(1) receives the request for the desired information, it verifies the certificate information of the relay from Blockchain(2) with Blockchain(1) to authenticate the process, following the steps outlined in algorithm 1. Upon successful authentication, relay(1) retrieves information from Blockchain(1) using the GetPermittedNetworksByAddress method from the Permitted Networks Contract and shares it with relay(2).

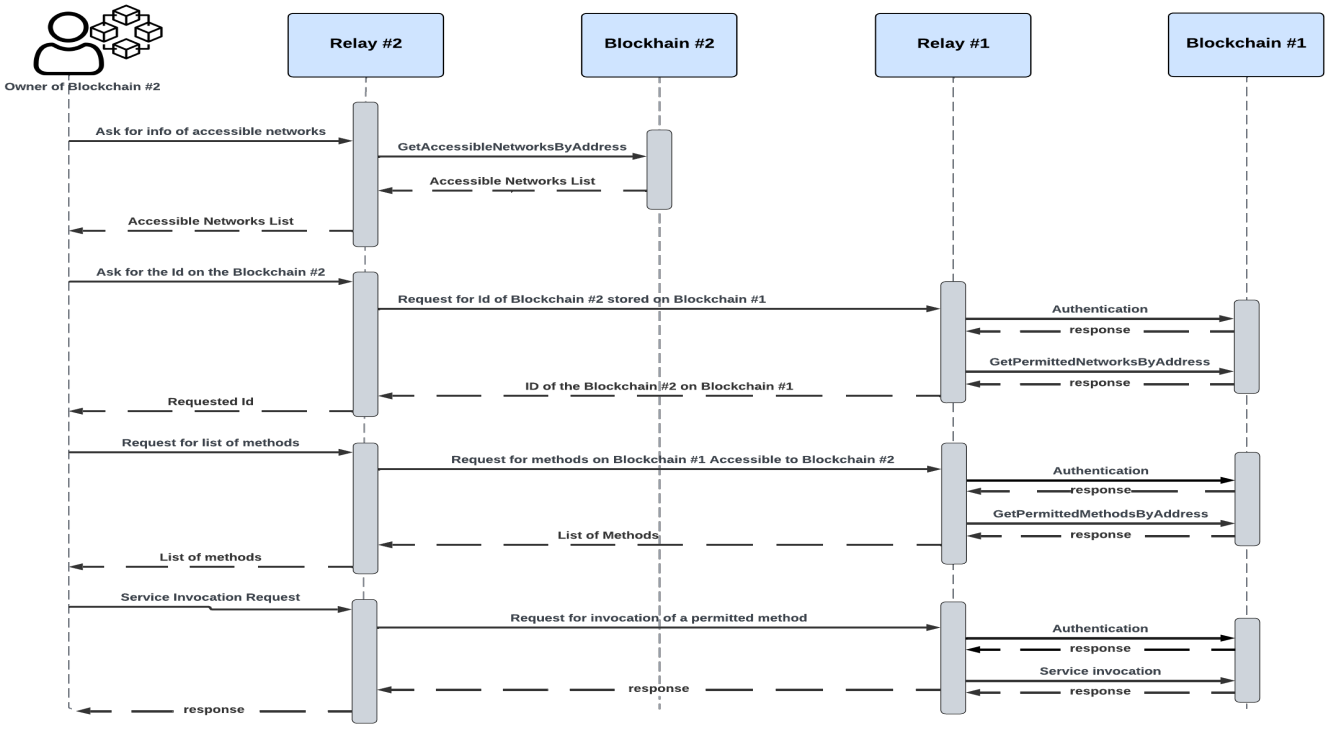

Fig. 4. The sequence diagram of cross-platform service invocation

3) **Request for Permitted Methods Information:** To utilize a service or obtain data, the owner of Blockchain(2) needs information about the method on Blockchain(1) through which this data or service is accessible. To achieve this, the owner of Blockchain(2) sends a request to relay(1) to obtain a list of methods using relays(2). After successful authentication, relay(1) acquires the list of permitted methods for Blockchain(2) using its unique identifier. The Chain Connector of relay(1) invokes the GetPermittedMethodsByNetworkId method from the Permitted Methods Smart Contract. Upon receiving the response from the chain, relay(1) forwards the response to relay(2). Subsequently, the owner of Blockchain(2) can retrieve the list of available methods.
4) **Service Invocation:** Upon obtaining the list of methods and their respective information, the owner of Blockchain(2) can request the invocation of specific methods to access the data or service provided. After authentication, relay(1) forwards the information to the Chain Connector through the Mediator component. The Chain Connector invokes the desired method and returns the result to the Mediator, which then forwards it to the Message Handler. The Message Handler subsequently sends the response back to relay(2), enabling the owner of Blockchain(2) to retrieve and utilize the data or service result.

IV. Implementation and Evaluation

In this section, we elaborate on the implementation of the proposed architecture, detailing the technologies used for development and the infrastructure where our solution was deployed. Additionally, we present the technologies and infrastructure used in evaluating Automated Gateways, along with the results of the performance analysis derived from this evaluation.

A. Implementation

For the development of Automated Gateways, the Go programming language was chosen for its efficiency and suitability in creating the relay component, which was then released as a Go module. This modular design simplifies the integration process, allowing any Go program that includes Automated Gateways to initiate the relay with a few commands, covering both the Message Handler and CLI components. The relay module automatically manages the underlying processes.

Furthermore, Hyperledger Fabric, as a mature permissioned blockchain platform, was selected for this study. Its widespread industry adoption, as detailed in the Hyperledger Fabric Annual Report 2020 [36], along with its customizable permission settings, made it an ideal choice for implementing and testing our solution. This selection was aimed at acquiring practical insights into the applicability of our architecture in real-world industry settings. Within this framework, the smart contracts for Hyperledger Fabric were also developed using the Go programming language, ensuring consistency and

streamlined interaction between the relay and the blockchain network. Both the relay module and the smart contracts are available on a GitHub repository[1].

B. Evaluation

In this section, we detail the performance evaluation results for the Automated Gateways framework. The framework's effectiveness hinges on three key methods within the policy smart contracts, which are crucial to the system's overall workflow. Therefore, our experiments focused intensively on these essential methods. Specifically, these methods are 'GetAccessibleNetworksByAddress' from the Accessible Networks Contract, 'GetPermittedNetworksByAddress' from the Permitted Networks Contract, and 'GetPermittedMethodsByNetworkId' from the Permitted Methods Contract.

To attain results that more accurately reflect real-world scenarios, we deployed two Hyperledger Fabric networks in production mode, and we increased the rate of concurrent requests that the gateway of a peer could handle from 500 to 2500. For this deployment, we allocated a Virtual Private Server (VPS) with 16 gigabytes of RAM, 8 vCPUs, and 500 gigabytes of storage to each instance of Hyperledger Fabric. Within each of our Hyperledger Fabric networks, we defined three organizations—two peer organizations and one orderer organization. To simulate real-life networking scenarios between organizations, each organization was launched on a distinct Docker engine, thereby preventing connections between these organizations through Docker's local network. For benchmarking purposes, we employed Hyperledger Caliper [37]. Hyperledger Caliper was deployed on a separate VPS with the same configuration as the Hyperledger Fabric VPSs, ensuring that the resource consumption of Caliper did not impact the performance of Hyperledger Fabric.

To begin assessing the performance of the specified methods, we initiated our evaluation by adjusting the transaction input rate. During this phase, we maintained a consistent setup with 10 concurrent workers operating on Caliper, sending a total number of 30K transactions in each round of testing different transaction input rates, aiming to analyze the system's performance under varying loads. After identifying the system's saturation point, we fixed the transaction input rate at this level and shifted our focus to modifying the number of concurrent workers. In each round of testing with a specific number of worker, we send a total number of 30K transactions. This second phase of evaluation, transmitting transactions from Caliper to Hyperledger Fabric, was designed to determine the most effective number of workers for optimal system performance.

As depicted in Figure 5, the GetAccessibleNetworksByAddress method at the top-left indicates an initial linear correlation between input transaction rate and throughput. The throughput commenced at 100 tps and progressively increased to 1600 tps as the input transaction rate reached 1600 tps. After this point, a fluctuation will start. This trend shows that

[1]https://github.com/tcdt-lab/Automated-Gateways

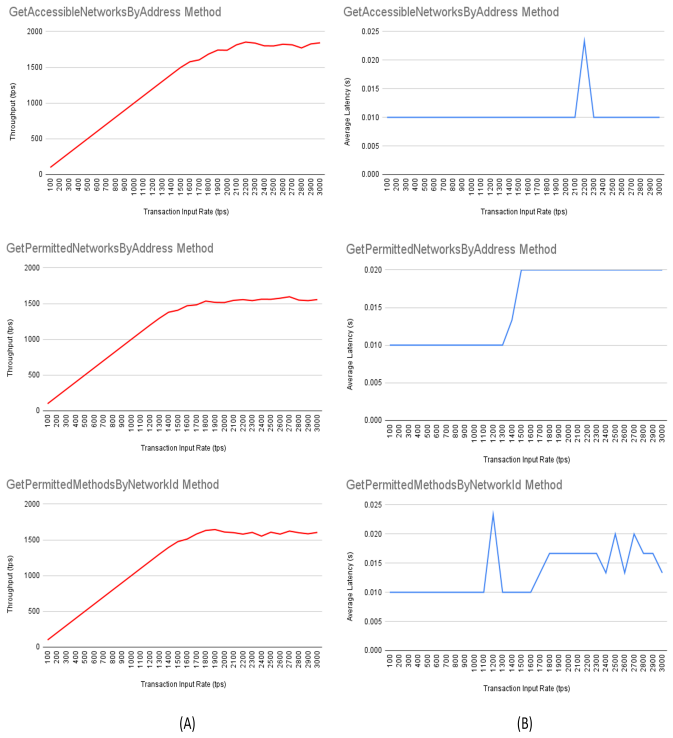

Fig. 5. The trends of system throughput (A) and average latency (B) across different transaction send rates. These trends are shown for three methods: GetAccessibleNetworksByAddress, GetPermittedNetworksByAddress, and GetPermittedMethodsByNetworkId.

we reach the saturation point at approximately 1600 tps as the transaction input rate. This trend persisted consistently for both 'GetPermittedNetworksByAddress' and 'GetPermittedMethodsByNetworkId,' with their respective saturation points identified at 1400 tps and 1500 tps, signifying a similar behavior in the performance characteristics of these methods. Examining the Average Latency in Figure 5 indicates that while the latency may not consistently remain constant and some spikes may occur, particularly noticeable for the method 'GetPermittedMethodsByNetworkId', the range between the maximum and minimum average latency values during various send rates does not exceed fifteen milliseconds for any of the methods, which is normal for a system with transaction input rate more than 1000tps. This observation illustrates an acceptable level of average latency across the evaluated methods. In our second evaluation approach, we maintained a fixed transaction input rate at the saturation point for each method and proceeded to increase the number of workers. As depicted in Figure 6, with the incremental addition of workers, an initial surge in throughput rate is observable, peaking at a certain point. Beyond this peak, the throughput plateaus. The onset of this fluctuation marks the saturation point, identified as 8 workers for 'GetAccessibleNetworksByAddress,' 8 workers for 'GetPermittedNetworksByAddress' and 7 workers for 'GetPermittedMethodsByNetworkId'. Considering the Process Time, which is indicative of the time required for all workers

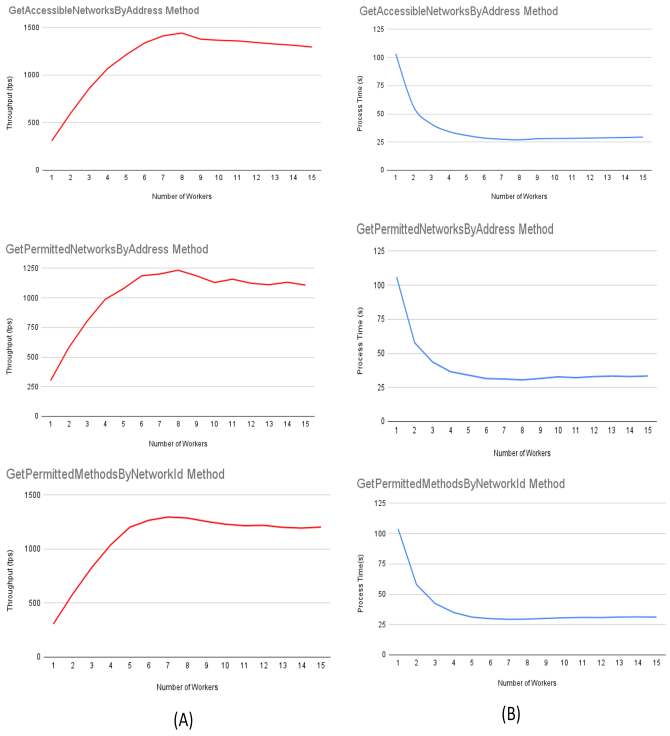

Fig. 6. The trends of system throughput (A) and process time (B) across different numbers of workers. These trends are shown for three methods: GetAccessibleNetworksByAddress, GetPermittedNetworksByAddress, and GetPermittedMethodsByNetworkId.

to complete the test, an initial substantial decrease is noted from approximately 100 to 25 seconds before reaching the saturation point. Subsequently, a marginal fluctuation in process time is observed.

The capability of the system to handle a high number of transactions per second concurrently, coupled with maintaining an average latency under 25 milliseconds, is indicative of its robustness under industrial pressure. This performance demonstrates that our system can effectively meet the objectives outlined in our research, including goals such as easy maintenance and cost-effective technology adaptation for connecting diverse blockchain platforms with notable stability.

V. Conclusion

In this paper, we introduced Automated Gateways as an easy-to-use, smart contract-powered interoperability solution for facilitating data and service sharing among blockchains. The objective is to eliminate reliance on external solutions and reduce the time and effort required to establish communication between blockchains. Evaluation results have confirmed the system's ability to efficiently handle a high volume of concurrent transactions with low latency, demonstrating robustness under heavy load. By deploying a production environment and carefully tuning the network and resource configurations, we achieved realistic testing conditions. Our results, showing a linear correlation between transaction rate and throughput and identifying optimal worker configurations, highlight the system's scalability and efficiency. This performance aligns with our goals of easy maintenance and cost-effective integration for diverse blockchain platforms, indicating strong potential for wider application in blockchain technology.

For the future direction, a feasible expansion of the current solution could involve integrating functionalities like atomic swap and token swapping among multiple blockchains. This extension needs to uphold the design principles and intrinsic values of Automated Gateways, thereby augmenting its capabilities and broadening its applicability across diverse blockchain scenarios.

References

[1] F. Reegu, S. M. Daud, and S. Alam, "Interoperability challenges in healthcare blockchain system-a systematic review," *Annals of the Romanian Society for Cell Biology*, pp. 15 487–15 499, 2021.

[2] W. J. Gordon and C. Catalini, "Blockchain technology for healthcare: facilitating the transition to patient-driven interoperability," *Computational and structural biotechnology journal*, vol. 16, pp. 224–230, 2018.

[3] H. Wang, C. Guo, and S. Cheng, "Loc—a new financial loan management system based on smart contracts," *Future Generation Computer Systems*, vol. 100, pp. 648–655, 2019.

[4] R. Wang, Z. Lin, and H. Luo, "Blockchain, bank credit and sme financing," *Quality & Quantity*, vol. 53, pp. 1127–1140, 2019.

[5] G. S. Sajja, K. P. Rane, K. Phasinam, T. Kassanuk, E. Okoronkwo, and P. Prabhu, "Towards applicability of blockchain in agriculture sector," *Materials Today: Proceedings*, vol. 80, pp. 3705–3708, 2023.

[6] S. A. Bhat, N.-F. Huang, I. B. Sofi, and M. Sultan, "Agriculture-food supply chain management based on blockchain and iot: a narrative on enterprise blockchain interoperability," *Agriculture*, vol. 12, no. 1, p. 40, 2021.

[7] S. Karumba, R. Jurdak, S. S. Kanhere, and S. Sethuvenkatraman, "Bailif: A blockchain agnostic interoperability framework," in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2023, pp. 1–9.

[8] G. Wang, "Sok: Exploring blockchains interoperability," *Cryptology ePrint Archive*, 2021.

[9] 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*, 2019, pp. 29–35.

[10] S. Schulte, M. Sigwart, P. Frauenthaler, and M. Borkowski, "Towards blockchain interoperability," in *Business Process Management: Blockchain and Central and Eastern Europe Forum: BPM 2019 Blockchain and CEE Forum, Vienna, Austria, September 1–6, 2019, Proceedings 17*. Springer, 2019, pp. 3–10.

[11] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," *arXiv preprint arXiv:1906.11078*, 2019.

[12] G. Wang, Q. Wang, and S. Chen, "Exploring blockchains interoperability: A systematic survey," *ACM Computing Surveys*, 2023.

[13] "Axelar network: Connecting applications with blockchain ecosystems," 2021.

[14] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[15] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, vol. 27, 2019.

[16] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3003–3017.

[17] E. J. Scheid, T. Hegnauer, B. Rodrigues, and B. Stiller, "Bifröst: a modular blockchain interoperability api," in *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. IEEE, 2019, pp. 332–339.

[18] M. S. Rahman, M. Chamikara, I. Khalil, and A. Bouras, "Blockchain-of-blockchains: An interoperable blockchain platform for ensuring iot data integrity in smart city," *Journal of Industrial Information Integration*, vol. 30, p. 100408, 2022.

[19] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–41, 2021.

[20] P. Han, Z. Yan, W. Ding, S. Fei, and Z. Wan, "A survey on cross-chain technologies," *Distributed Ledger Technologies: Research and Practice*, vol. 2, no. 2, pp. 1–30, 2023.

[21] 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. 116 672–116 691, 2021.

[22] K. Ren, N.-M. Ho, D. Loghin, T.-T. Nguyen, B. C. Ooi, Q.-T. Ta, and F. Zhu, "Interoperability in blockchain: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

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

[24] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, and Y.-C. Hu, "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 549–566.

[25] F. Kammüller and U. Nestmann, "Inter-blockchain protocols with the isabelle infrastructure framework," 2020.

[26] Y. Sun, L. Yi, L. Duan, and W. Wang, "A decentralized cross-chain service protocol based on notary schemes and hash-locking," in *2022 IEEE International Conference on Services Computing (SCC)*. IEEE, 2022, pp. 152–157.

[27] B. Pillai, K. Biswas, and V. Muthukkumarasamy, "Blockchain interoperable digital objects," in *Blockchain–ICBC 2019: Second International Conference, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 2*. Springer, 2019, pp. 80–94.

[28] S. Johnson, P. Robinson, and J. Brainard, "Sidechains and interoperability," *arXiv preprint arXiv:1903.04077*, 2019.

[29] B. Dai, S. Jiang, M. Zhu, M. Lu, D. Li, and C. Li, "Research and implementation of cross-chain transaction model based on improved hash-locking," in *Blockchain and Trustworthy Systems: Second International Conference, BlockSys 2020, Dali, China, August 6–7, 2020, Revised Selected Papers 2*. Springer, 2020, pp. 218–230.

[30] H. Tian, K. Xue, X. Luo, S. Li, J. Xu, J. Liu, J. Zhao, and D. S. Wei, "Enabling cross-chain transactions: A decentralized cryptocurrency exchange protocol," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3928–3941, 2021.

[31] W. Liu, H. Wu, T. Meng, R. Wang, Y. Wang, and C.-Z. Xu, "Aucswap: A vickrey auction modeled decentralized cross-blockchain asset transfer protocol," *Journal of Systems Architecture*, vol. 117, p. 102102, 2021.

[32] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov, "Smart contract invocation protocol (scip): A protocol for the uniform integration of heterogeneous blockchain smart contracts," in *Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32*. Springer, 2020, pp. 134–149.

[33] K. Indrasiri and D. Kuruppu, *gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes*. O'Reilly Media, 2020.

[34] N. A. Lal, S. Prasad, and M. Farik, "A review of authentication methods," *International journal of scientific & technology research*, vol. 5, no. 11, pp. 246–249, 2016.

[35] Hyperledger Fabric Organisation. (2023) Running a fabric application. Accessed:2023-12-18. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/write_first_app.html

[36] ——. (2020) Hyperledger annual report. Accessed: 2023-12-18. [Online]. Available: https://www.hyperledger.org/hubfs/Hyperledger/Printables/HL_AnnualReport2020_Print_121820.pdf

[37] ——. (2023) Hyperledger caliper. Accessed: 2023-12-18. [Online]. Available: https://www.hyperledger.org/projects/caliper