Interoperability between DLT following a Gateway-based approach: the Case of Ethereum and Hyperledger Fabric

Guzman Llambias [1], Sebastián Pandolfi [2], Laura González [2], Raúl Ruggia [2], Emiliano González [2], and Mathias Castro [2]

[1]Universidad de la República
[2]Affiliation not available

October 31, 2023

Abstract

Distributed ledger technologies (DLT) usage is currently limited to a single platform as they don't have design-based interoperability capabilities. In general, it's challenging for a DLT to communicate with another one. Although several DLT solutions have been proposed and applied in specific application areas, building a general purpose interoperability solution for any DLT remains a challenge. In previous work, we proposed a tailor-made interoperability solution between Hyperledger Fabric and Corda. This paper extends that work to enable interoperability between Hyperledger Fabric and Ethereum. The main contributions of this paper are proposals for a new Ethereum connector, a new request-response interaction model and the introduction of future payments to enable the payments of services. A prototype was developed and evaluated through a case scenario, performance tests and cost analysis. Performance tests showed bottlenecks under heavy load scenarios due to Ethereum's design. Costs analysis showed that the approach is suitable for purchasing high-priced services. These promising results constitute a step forward in developing a general-purpose solution for DLT interoperability.

Interoperability between DLT following a Gateway-based approach: the Case of Ethereum and Hyperledger Fabric

Sebastián Pandolfi
Facultad de ingeniería
Universidad de la República
Montevideo, Uruguay
sebastian.pandolfi@fing.edu.uy

Emiliano González
Facultad de ingeniería
Universidad de la República
Montevideo, Uruguay
emiliano.gonzalez.martinez@fing.edu.uy

Mathias Castro
Facultad de ingeniería
Universidad de la República
Montevideo, Uruguay
mathias.castro@fing.edu.uy

Guzmán Llambías
Facultad de ingeniería
Universidad de la República
Montevideo, Uruguay
gllambi@fing.edu.uy

Guzmán Llambías
Pyxis
Montevideo, Uruguay
guzman.llambias@pyxis.tech

Laura González
Facultad de ingeniería
Universidad de la República
Montevideo, Uruguay
lauragon@fing.edu.uy

Raúl Ruggia
Facultad de ingeniería
Universidad de la República
Montevideo, Uruguay
ruggia@fing.edu.uy

***Abstract*—Distributed ledger technologies (DLT) usage is currently limited to a single platform as they don't have design-based interoperability capabilities. In general, it's challenging for a DLT to communicate with another one. Although several DLT solutions have been proposed and applied in specific application areas, building a general purpose interoperability solution for any DLT remains a challenge. In previous work, we proposed a tailor-made interoperability solution between Hyperledger Fabric and Corda. This paper extends that work to enable interoperability between Hyperledger Fabric and Ethereum. The main contributions of this paper are proposals for a new Ethereum connector, a new request-response interaction model and the introduction of future payments to enable the payments of services. A prototype was developed and evaluated through a case scenario, performance tests and cost analysis. Performance tests showed bottlenecks under heavy load scenarios due to Ethereum's design. Costs analysis showed that the approach is suitable for purchasing high-priced services. These promising results constitute a step forward in developing a general-purpose solution for DLT interoperability.**

***Index Terms*—distributed ledger technology, blockchain, interoperability, cross-chain transactions, ethereum, hyperledger fabric**

I. Introduction

Nowadays, distributed ledger technology (DLT) is applied in different domains like finance, art, health, supply chain and IoT among others. Bitcoin [1], Ethereum [2] and Hyperledger Fabric [3] are some examples of DLT with different characteristics and purposes. However, the usage of this technology is limited by its nature. DLT are usually designed as silos of information and interoperability is not part of their design. It is a challenge for a DLT to communicate with an external software system or another DLT [4]. In addition, registering external data into a DLT is a challenge, as data verification needs to be performed. In some cases, DLT can trust the data source, but in other scenarios like DLT interoperability, data verification must conform to the consensus protocol rules defined by the source DLT. In this scenarios there is the need to achieve a consensus between both DLT about data reliability, which is a challenge.

In recent years, several interoperability solutions were proposed for a particular type of DLT: blockchain platforms [5] [6]. A blockchain platform is a distributed ledger technology whose ledger structure is an ordered linked list of blocks, each containing a set of transactions. Bitcoin and Ethereum are some of the most popular blockchain platforms. These blockchain interoperability solutions were applied with quite success but are for specific domains or between specific blockchains. One example is Polygon PoS Bridge [7] that enables asset transfer (e.g. cryptocurrencies) between Ethereum and Polygon blockchains. Another example is Hyperledger Cactus [8], an initiative that enables interoperability between business applications and DLT following an orchestrated architecture. With this approach, business applications send messages to Hyperledger Cactus, which registers transactions on one or more DLT in a coordinated (orchestrated) way. Weaver [9] is a recent DLT interoperability solution following a choreography approach. With this approach a source DLT can reliably send messages to a target DLT through Weaver. However, not every DLT is supported. More specifically, it is still a challenge to enable interoperability between some blockchain platforms and DLT. In particular, with Ethereum, one of the most popular blockchain platforms nowadays.

In our previous work [10], we proposed a gateway-based DLT interoperability solution that enabled interoperability between two DLT (Hyperledger Fabric and Corda [11]) with a similar approach to Weaver. In this work, a source DLT (i.e. Hyperledger Fabric) sent messages to the gateway, which

transformed and routed them to the target DLT (i.e. Corda). The proposal was designed not to change the DLT source code and to provide an interoperability solution based on the DLT native building blocks. In particular, we used events triggered by the source DLT and listened by the gateway. Smart contracts were used as interfaces of the target DLT to receive the messages. This method seems suitable to enable interoperability between DLT and blockchain platforms, in particular, between Hyperledger Fabric and Ethereum.

Given this context, this work proposed the following research questions: How can the gateway solution be applied to enable interoperability between Ethereum and Hyperledger Fabric? How does the proposed solution impact the performance while interoperating Ethereum and Hyperledger Fabric? Which costs are required by the proposed solution when interoperating Ethereum with Hyperledger Fabric?

This paper presents preliminary results to these research questions. In particular, it was possible to extend the gateway and build a new Ethereum connector that enabled connectivity with Ethereum. Furthermore, an interaction model based on Enterprise Integration Patterns [12] between these two DLT was proposed to enable request-response interactions. In addition, this work introduces the concept of future payments that allow the payment of services provided by Hyperledger Fabric with Ethereum cryptocurrency (Ethers), enabling a use case for Hyperledger Fabric and Ethereum interoperability. A cost assessment showed that the proposed extension is suitable for paying services with medium to high costs but not for the payment of services with low costs. Performance assessments showed limitations in the proposal's throughput, being the Ethereum connector a bottleneck in heavy load scenarios. The main reason for this issue was the design of Ethereum to prevent replay attacks that limited the number of concurrent transactions the connector could send. Finally, the proposed approach provides a generic purpose interoperability solution between Ethereum and Hyperledger Fabric that can be used in different use case scenarios.

The rest of the paper is organised as follows. Section II provides background concepts. Section III describes an electric vehicle scenario where DLT interoperability is required. Section IV presents the detailed design of the proposal. Section V describes a reference implementation of the proposal. Section VI presents the assessment performed to the gateway. Section VII presents a discussion of this work answering the research questions and limitations of the proposal. Section VIII analyses related work. Finally, section IX presents conclusions and future work.

II. Background

This section presents the background concepts required for the comprehension of this work.

A. DLT & Blockchain concepts

A distributed ledger is an append-only database distributed across a network of machines called DLT Nodes [13]. Once a transaction is registered into the ledger, it cannot be updated or deleted. DLT Nodes use consensus mechanisms that define rules and procedures to synchronise the ledger between them.

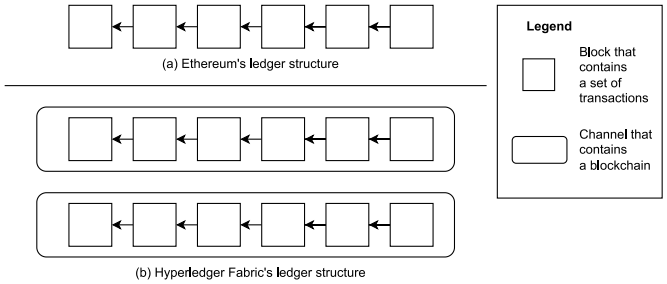

Fig. 1. Ethereum and Hyperledger Fabric ledger structure (adapted from [5])

Blockchain is a particular type of distributed ledger whose ledger structure is based on an ordered linked list of blocks, each containing a set of transactions [14]. Blocks are linked by cryptographic hashes that assure the security of the link. If data are changed, the link breaks, giving users a proof that the block was tampered. As participants reject tampered blocks, this gives data immutability to the blockchain in practice.

The operation and usage of distributed ledgers are enabled by Distributed Ledger Technologies (DLT) [13]. Blockchain platforms are a specialisation of DLT that enable the usage and operation of blockchain ledgers [14]. Both technologies provide the software and hardware required to run a node or any other required software to access the ledger. Ethereum is one of the most popular blockchain platforms, while Hyperledger Fabric is a DLT provided by the Hyperledger Foundation. Fig. 1 depicts the ledger structure of both technologies. Ethereum follows a blockchain ledger structure, while Hyperledger Fabric defines the channel concept, where each channel has its blockchain ledger. In Hyperledger Fabric, channels (and blockchains) are restricted and only authenticated and authorised users can read and write into the ledger. Users authorised to read on one channel, may not be authorised to read or write on another channel. These properties differ from Ethereum where users do not require authentication (they use pseudonyms using public keys), and every user may have access to read or write the ledger. A DLT that requires access to the ledger is usually called a permissioned DLT, while a permissionless DLT give users unrestricted access to the ledger [13]. Permissioned DLT are usually used by a consortium of enterprises to improve their business process and reduce costs, while permissionless DLT are used on decentralised use cases where data transparency and unrestricted access to the ledger is required [15].

On the other hand, smart contracts are software scripts deployed on the DLT, whose execution is registered on the distributed ledger [13]. Smart contracts execution is triggered by DLT transactions autonomously by any DLT Node [14]. They can hold and transfer digital assets or invoke other smart contracts stored on the DLT. Smart contract execution may use data stored in the distributed ledger or gathered outside of the DLT using special entities called Oracles. Once deployed, the code of a smart contract is deterministic and immutable [14].

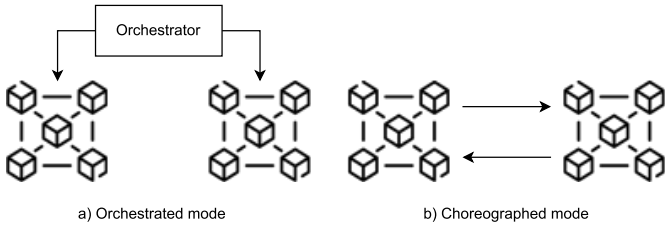

Fig. 2. Orchestrated vs choreographed architecture

B. DLT Interoperability

According to Wegner [16], "interoperability is the ability of two or more systems to exchange information despite their differences in language, interfaces and execution platform". Usually, information systems provide mechanisms that enable interoperability (e.g. Web Services SOAP [17]). However, these mechanisms cannot be used for DLT interoperability. It is a challenge for a DLT to communicate with external software systems or other DLT, as they are usually built as information silos without interoperability mechanisms [4]. Furthermore, registering data from an external data source is also a challenge for a DLT, as every transaction registered on the ledger must conform to the defined consensus protocol rules. In some scenarios, the DLT may trust the data source. However, in other cases like DLT interoperability, this is not possible, so there is the need to achieve a consensus between the involved DLT, and this is a challenge [4].

DLT interoperability involves a source DLT and a target DLT. The source DLT usually initiates a local transaction in its ledger that triggers a cross-chain transaction targeted to the target DLT. A cross-chain transaction is a transaction that spans the domain of a DLT into another DLT and involves a local transaction on the source DLT and a local transaction on the target DLT. The target DLT must verify the cross-chain transaction using the source DLT consensus protocol rules to reliably register the transaction in its ledger.

DLT interoperability solutions enable the delivery of cross-chain transactions. In the last few years, several DLT interoperability solutions were proposed by the industry and academic community [5], [6]. Notary Scheme involves a trusted third party that monitors the source blockchain and triggers cross-chain transactions on the target blockchain. Sidechain/Relays requires a primary blockchain (called main chain) and a secondary blockchain (also called Sidechain) which is an extension of the main chain. Usually, the sidechain offers additional capabilities not offered by the main chain, like improved performance, reduced costs or smart contract execution. Cross-chain transactions are verified by components on the target blockchain called relays using Simple Payment Verification (SPV) mechanisms. Atomic Swaps is a protocol followed by two users on two different blockchains that enable the exchange of assets hosted on each blockchain[1].

[1]The reader may have noted the term blockchain was used here, and that is because Notary Scheme, Sidechains, Atomic Swaps and Blockchain of blockchains are specific interoperability solutions for DLT with a blockchain ledger structure

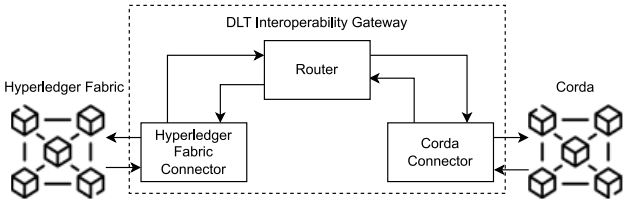

Fig. 3. Blockchain interoperability gateway

Gateways are a basic type of middleware that enables the communication between two DLT, adapting message format and communication protocols. DLT API Gateways (previously known as Generic Interface) are a type of middleware that enables external systems to communicate with one or more DLT using a common interface. Enterprise Relays are sophisticated gateways that relies on proofs to perform cross-chain transaction verification and provides policy rules to access the ledger. Finally, blockchain of blockchains involves a main chain and several Sidechains that use the main chain to exchange cross-chain transactions reliably.

DLT interoperability solutions use different interaction architectures. Some use an orchestrated architecture, while others use a choreographed architecture. Fig. 2 depicts both approaches. On orchestrated architectures, an entity exists that manages the cross-chain transaction and is responsible for registering the local transactions on DLT in a coordinated way. The orchestrator initiates the cross-chain transaction. Notary Scheme and DLT API Gateway use this architecture. On the other hand, on a choreographed architecture, the cross-chain transaction is performed by the exchange of messages between two DLT, without the participation of an external third party that coordinates the execution. In this architecture, the cross-chain transaction is initiated by a source DLT. Gateways, Sidechains and Blockchain of Blockhains use this approach.

C. DLT Interoperability Gateway

In our previous work [15], we presented a DLT interoperability solution based on gateways that enabled interoperability between two DLT following a choreography architecture: Hyperleger Fabric and Corda. Fig. 3 depicts the DLT interoperability gateway.

The gateway was composed of two connectors and a router component. The connectors enabled connectivity with the DLT and allowed to send cross-chain messages to the DLT and receive cross-chain messages from the DLT. On the other hand, the Router received cross-chain messages from a source connector and routed messages to the target connector. Each DLT triggered events that were listened by to the DLT connector. Those events were transformed into a canonical data format and sent to the Router component. The Router component inspected the message and routed it to the target DLT connector. The target DLT connector transformed the canonical message to the native target DLT data format and sent it to the DLT. Smart contracts were used by the DLT connectors to send messages to the blockchain.

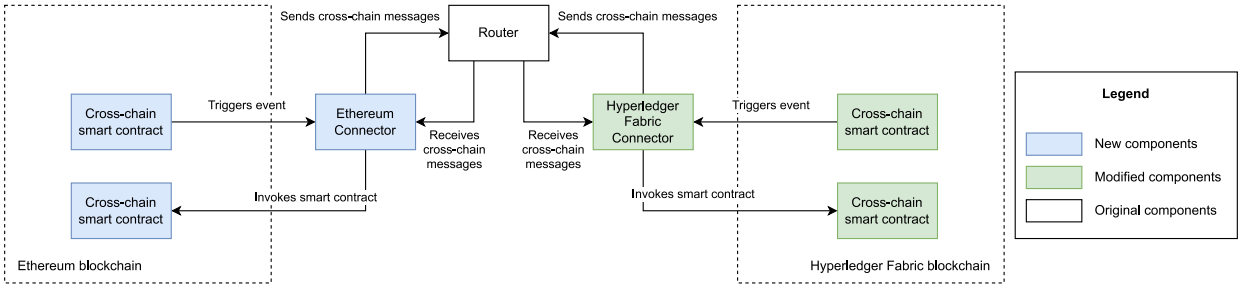

Fig. 4. Gateway architecture

III. Interoperability motivational scenario

This section presents a motivation scenario to illustrate DLT interoperability between Ethereum and Hyperledger Fabric.

The proposed motivational scenario was inspired by the work of Castro [18] that proposed the use of a DLT by a consortium to register the electricity consumption for charging electrical vehicles. The participants of the consortium were the electric pump stations, the national electricity company of Uruguay (UTE, Administración Nacional de Usinas y Trasmisiones Eléctricas) and the national energy regulator of Uruguay (URSEA, Unidad Reguladora de Servicio de Energía y Agua). These participants used a DLT to have a reliable source of truth regarding pump and electricity usages among all participants [18]. This work extended Castro's scenario and allowed users to pay electricity with Ethereum's cryptocurrency (Ether). In this extended scenario, users used a Web application and check if the pump has the requested amount of energy to charge their vehicles. After this, the user may select the amount to charge and pays with Ethereum. As a consequence, the amount of Ethers are transferred from the user's account to the pump's account and a cross-chain transaction is triggered from Ethereum to the DLT to release the energy from the pump. For this scenario Hyperledger Fabric was used as the DLT.

IV. DLT interoperability gateway

This section describes the design principles defined for this work, a general description of the improved gateway, the canonical data format used, the proposed interaction model and the concept of future payments that enabled interoperability between Ethereum and Hyperledger Fabric to purchase services provided by a consortium.

A. Design principles

This works posed the following design hypothesis to extend the gateway:

- Choreography interactions: the gateway must maintain the original design and keep the choreography architecture. It must avoid being an orchestrator of the interactions with the DLT.
- Not invasive: The extension must not require to change the source code of Ethereum and its design must use its native building blocks.
- Trusted and centralised: Considering the nature of some DLT such as Hyperledger Fabric (they may run on a semi-trusted environment), it is acceptable to trust in the gateway. Furthermore, it is also acceptable to provide a centralised design.

B. General description

Fig. 4 depicts the architecture of the gateway, locating Ethereum and Hyperledger Fabric connectors and the smart contracts that enabled the solution. New components are depicted in blue, modified components in green and white components are kept unmodified from the original proposal.

The Ethereum connector requires the existence of a cross-chain smart contract on Ethereum that triggers events that are listened by the Ethereum Connector. These events are used to send cross-chain messages to other DLT. After receiving these events, the Ethereum connector transforms them to the canonical data format expected by the Router component. On the other hand, when the Router receives a message that needs to be delivered to Ethereum blockchain, it sends it to the Ethereum connector. The Ethereum component inspects the message and invokes the target cross-chain smart contract on Ethereum. The Ethereum connector uses the passive mode to integrate with Ethereum. Passive mode implies that the connector monitors the blockchain an triggers events to other blockchain platforms [19]. As every transaction sent to Ethereum needs to be signed using a private key and has costs attached, the Ethereum connector has its own wallet to register transactions (and invoke smart contracts) on Ethereum.

The Hyperledger Fabric connector follows a similar approach. There must exist cross-chain smart contracts (called chaincode on Hyperledger Fabric) that triggers events that are listened by the Hyperledger Fabric connector and transformed to the canonical data format. When the connector receives messages from the Router, it invokes a smart contract on Hyperledger Fabric based on the message content. As every transaction sent to Hyperledger Fabric needs to be authenticated and authorised, the connector has the credentials of an authorised user to write into the ledger. This way, the Hyperledger Fabric connector writes into the ledger on behalf of the Ethereum blockchain. Changes were made to the original connector to provide a general purpose Hyperledger Fabric connector and be agnostic to the target smart contract or the listened event.

C. Canonical message data format

The Ethereum and Hyperledger Fabric connectors exchange messages with the Router component using a canonical message data format. This format was inspired by the format described by Hohpe and Wolf [12] and was composed of a message header and a message body (in this case, data). The message header contains all the information used by the gateway to deliver the message to the target DLT and includes the message origin, destination, timestamp, message identification, among others. On the other hand, the message data element includes business information that needs to be delivered to the final destination. Table I presents the message header attributes and their description.

TABLE I
MESSAGE HEADER ATTRIBUTES

Attribute	Description
messageId	Global unique identifier of the message.
correlationIdentifier	This attribute only applies to response messages. It is a unique identifier that identifies the request message of a request-response interaction.
timestamp	The creation timestamp of the message.
source	The blockchain that created the message.
target	Contains information about the destination of the cross-chain message. The blockchain attribute indicates the name of the target DLT and it is used by the Router component to route the message. The contract and operation attributes defines the target smart contract and operation in this smart contract, that the connector must invoke.
replyTo	Specifies how to reply to the message. It specifies a blockchain, contract and operation attribute. The blockchain attribute specifies to which DLT the receiver must send the reply message to. The smart contract and operation attribute specifies which operation must be invoked of the specified smart contract hosted by the DLT.

D. Interaction model

The gateway provides a request-response interaction model based on the Request-Reply, Return Address and Correlation Identifier patterns [12] to deliver cross-chain messages. Fig. 5 depicts the interaction model using Ethereum as the source DLT and Hyperledger Fabric as the target DLT. Nevertheless, the behaviour is the same in case Hyperledger Fabric is the source DLT and Ethereum the target DLT.

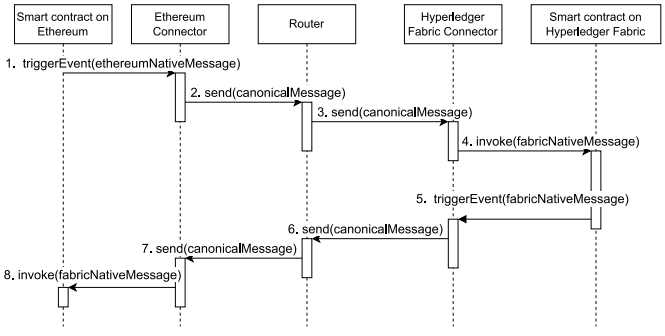

Fig. 5. Request-response interaction model

When creating a cross-chain message, the source cross-chain smart contract must set the messageId, timestamp, source, target and replyTo data (step 1). The source connector transforms the message to the canonical data format and sends it to the Router (step 2). The Router uses the blockchain element on the target header attribute to identify the blockchain connector to route the message (step 3). The target connector identifies the target smart contract and operation to invoke using the target attribute (step 4). In particular, the contract and operation elements. To generate a response, the target smart contract triggers an event that is listened by the target connector (step 5). This event must specify the replyTo attribute on the request cross-chain message to build the response cross-chain message. In particular, the target blockchain, target smart contract and target operation must use the values defined in the replyTo attribute of the request message. The event must also specify the correlationIdentifier and must be equal to the messageId of the request message. The target connector transforms the message to the canonical data format and sends it to the Router (step 6). The Router inspects the message and delivers it to the source connector (step 7). The source connector inspects the message searching for the target contract and operation, transforms the message to the native data format and invokes the source smart contract (step 8). In this example the source smart contract creates the request message and process the response. Nevertheless, this smart contracts may be different, and one smart contract may create the request and other smart contract may process the response.

The proposed request-response interaction models implies that a cross-chain transaction is composed of at least three local transactions. Two on the source DLT and one on the target DLT. In the case of Ethereum and Hyperledger Fabric, this implies that the final execution of a cross-chain transaction must wait for at least two blocks to be committed on Ethereum and one block to be committed on Hyperledger Fabric. Note that, delays in transaction processing may cause that the cross-blockchain transaction may not be committed on consecutive blocks.

E. Future payments

Registering a transaction on Ethereum implies costs and signing the transaction with the user's private key. When the gateway registers a cross-chain transaction on Ethereum it must pay the transaction's fee and sign the transaction on behalf of the user. To avoid having the user's private key on the gateway and pay for the user, this work introduces the concept of future payments. A future payment is a cryptocurrency transfer created by the user but not executed yet. Its execution is locked until the gateway confirms its execution in the future.

Future payments implies the existence of a new smart contract on Ethereum created and owned by the gateway. This means, there are restrictions on who can invoke its operations. This smart contract has four operations: register, check, release and expire. The register operation allows the user to register a future payment. This operation receives an amount of Ethers, the user's account, a target account and transfers the amount of Ethers from the user's account to the

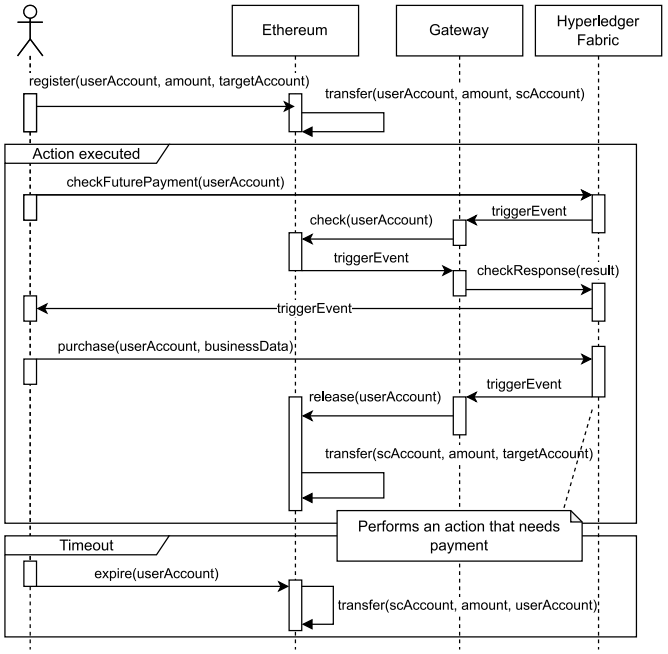

Fig. 6. Future payments

smart contract account. This future payment is locked until it is released by the release operation. The release operation receives an account and releases the future payment related to the specified account. This operation is restricted and can only be invoked by the gateway. In case a timeout period expires and the release operation is never called, the user can invoke the expire operation to release the funds back to his account. Finally, the check operation verifies if a future payment for a specified account is registered.

Fig. 6 depicts the behaviour of future payments used for cross-chain transaction between Ethereum and Hyperledger Fabric. In this example, the user needs to purchase a service provided by the DLT implemented with Hyperledger Fabric. To do so, the user invokes the register operation and creates the future payment that transfers an amount of Ethers from the user's account to the service providers' account. This operation temporary transfers the amount to the smart contract account. Later on, the user invokes a smart contract on Hyperledger Fabric to verify the future payment. This operation triggers a request-response interaction from Hyperledger Fabric to Ethereum for this purpose. After the future payment is verified, the user buys the service by invoking another smart contract that triggers a cross-chain transaction to Ethereum to release the future payment and transfer the Ethers to the service provider account. In this case, a transfer from the smart contract account to the service provider's account.

V. Reference Implementation

This section describes the reference implementation of the Ethereum connector and future payments. Further details and the source code are available online[2].

[2]Code will be online after response of the reviewers

A. Ethereum connector

The Ethereum connector was developed with Node.js and the Web3 library[3]. The generalisation of the connector was enabled by loading all the contract ABI (Application Binary Interface) files from a specific folder[4]. At startup, the Ethereum connector processes all these files and loads them so it has all the necessary information to invoke the configured smart contracts or listen to events they may trigger. Listing 1 presents how the connector load this files and starts listening to the events that may be triggered by these smart contracts.

```
var contracts = [];

async loadABI(){
    glob.sync( './ABI/*.json' ).forEach( function( file ) {
        var contract = require( path.resolve( file ) );
        contracts.push(contract);
      });
    await this.listenBlockchainEvents();
    await this.startRouterEndpoint();
}

async listenBlockchainEvents() {
    const web3 = new Web3(config.blockchain.ethereumHost)
    const networkId = await web3.eth.net.getId();
    contracts.forEach((contract) => {
      var deployedNetwork = contract.networks[networkId];
       var myContract = new web3.eth.Contract(
          contract.abi,
          deployedNetwork && deployedNetwork.address,
      );

    myContract.events.allEvents()
       .on('data', event => this.eventReceivedFromEthereum(
            event))
       .on('changed', changed => console.log(changed))
       .on('error', err => { throw err })
       .on('connected', str => console.log(str))
    });
}
```

Listing 1. Load smart contracts on Ethereum connector

Listing 2 presents how the connector uses the ABI files to invoke a smart contract when the Router component sends a cross-chain message. The connector checks for the contract header of the message and tries to match it to a smart contract previously loaded. If it exists, the connector invokes the smart contract on Ethereum with the corresponding parameters found on the data message element. If the smart contract was not found, the message is ignored. It is expected that data elements parameters are ordered in the same way as smart contract parameters, otherwise, an error occurs.

B. Future payments

Future payments were implemented as a structure in Solidity[5] (Accion) with two states: open (ABIERTO), invalid (INVALIDO). Future payments are composed of an amount (monto) and an account (billSur). There were needed two auxiliary mappings to map an address to states (saldoEstado) and map address to future payments (saldo). Listing 3 presents

[3]https://web3js.readthedocs.io/en/v1.8.2/

[4]A contract ABI file is a specification that describes the interface of a smart contract in Ethereum and how the caller can interact with it. Each smart contract has an ABI file.

[5]Solidity is the programming language offered by Ethereum to write smart contracts

```
async sendEventToEthereumBlockchain(event) {
    lock.acquire("send", async function(done) {
        const web3 = new Web3(config.blockchain.
            ethereumHost)
        const networkId = await web3.eth.net.getId();
        contracts.forEach(async (contract) => {
            if (contract.contractName == event.header.
                target.contract){
                var deployedNetwork = contract.networks[
                    networkId];
                var myContract = new web3.eth.Contract(
                    contract.abi,
                    deployedNetwork && deployedNetwork.
                        address);
                var params = [];
                for (var param in event.data)
                    params.push(event.data[param]);
                var nonce = await web3.eth.
                    getTransactionCount(config.blockchain.
                    bigWalletAddress);
                var rawTransaction = {
                    nonce: web3.utils.toHex(nonce),
                    from: config.blockchain.
                        bigWalletAddress,
                    to: deployedNetwork.address,
                    data: myContract.methods[event.header.
                        target.operation].apply(null,params
                        ).encodeABI()
                };
                await web3.eth.sendTransaction(
                    rawTransaction);
                done();
            }
        });
    });
}
```

Listing 2. Invoke smart contract on Ethereum.

the register and release operations. As shown in the code, the release function has the *onlyOwner* modifier that restricts its invocation only to the Ethereum connector. It also has the *chequeoAbierto* modifier that checks it can only be invoked if the state of the future payment is open. One limitation of the implementation is that it does not support the expire operation yet.

```
function register(address payable _billSur) public payable
    {

  if (saldoEstado[msg.sender] == Estados.INVALIDO){
    Accion memory accion = Accion({
      monto: msg.value,
      billSur: _billSur});
    saldo[msg.sender] = accion;
    saldoEstado[msg.sender] = Estados.ABIERTO;
  }
}

function release(address billCli) public onlyOwner()
    chequeoAbierto(billCli) payable {
  Accion memory accion = saldo[billCli];
  accion.billSur.transfer(accion.monto);
  saldoEstado[billCli] = Estados.INVALIDO;
}
```

Listing 3. Register and release operations

VI. Gateway Assessment

The gateway assessment was performed through the development of a case scenario, performance tests and a cost analysis.

A. Development of the case scenario

The development of the case scenario involved the development of two smart contracts on Ethereum and one smart contract on Hyperledger Fabric. Future payments were used to enable the user to pay for the service provided by Hyperledger Fabric. In this case, electricity to charge the vehicle.

The smart contract CheckEnergy on Ethereum provided two operations: checkEnergy, checkEnergyResponse. The first operation allowed the user to check the energy of a pump and trigger a query to Hyperledger Fabric. The second operation was required to receive Hyperledger Fabric's response to the query following the request-response interaction model presented on Section IV-D. The smart contract FuturePaymentsForEnergy provided the implementation of future payments in this scenario and provided the three operations required and described in Section IV-E: register, check and release.

The smart contract on Hyperledger Fabric provided three operations: checkEnergy, checkFuturePayment and purchaseEnergy. The first operation triggers an event with the available energy of a specified pump. This event was listened by the Hyperledger Fabric connector and sent as a response to Ethereum. The checkFuturePayment operation triggered a cross-chain transaction from Hyperledger Fabric to Ethereum through the Gateway and checked if a future payment was opened. Finally, the purchaseEnergy operation received an amount and pump, and released an amount of energy from the specified pump equal to the specified amount. This operation also triggered an event to release the future payment.

B. Performance assessment

Performance tests were performed to evaluate the gateway performance. In particular, the performance tests were performed to only one operation of Hyperledger Fabric: checkFuturePayment. Ganache and version 1.4.12 of Hyperledger Fabric was used to simulate both DLT. jMeter was used to invoked the operation and simulate concurrent users.

The performance tests were performed using one notebook with Windows 10 operating system, using the Windows Subsystem for Linux 2 (WSL 2). This subsystem enabled the installation of a Linux kernel on Windows operating system without using Virtual Machines. The notebook's hardware was composed of an Intel Core I7-10750H 6 core/12 threads 2.6Ghz (Maximum frequency 5 Ghz), 32Gb DDRA 3200 Mhz of memory and 1Tb SSD NVMe of storage. jMeter, Ganache and the gateway were hosted on this hardware.

The initial results were unsatisfactory as all the tests failed. The reason for this error was that creating a raw transaction on Ethereum required to obtain the account's nonce, which is done using the web3.eth.getTransactionCount method of the Web3 library. This method worked correctly when no concurrency was involved, but when multiple concurrent threads invoked this method, the same nonce value was returned for every thread, generating a reply attack. Under this attack, Ethereum rejected all other transactions besides the first one. To cope with this issue, a thread queue was added to the Ethereum connector by using the lock.acquire method (see

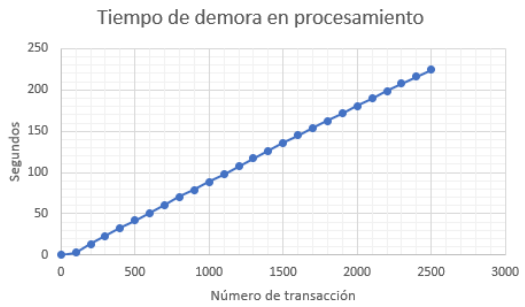

Fig. 7. Execution time of the performance tests

Listing 2). This method enabled serial processing of cross-chain transaction on the Ethereum connector.

Table II presents the results obtained by the performance tests after this performance fix. The execution time was the time between the first request executed by jMeter until the last response received by jMeter. With twenty concurrent users, all requests were successfully executed. However, with forty concurrent users, connection errors occurred between the Ethereum connector and Ganache and only seventy-nine percent of the requests were successfully executed and replied. On the other hand, Fig. 7 depicts the response time of each request. In this figure it is clearly depicted the impact of the thread queue fix performed to the Ethereum connector and the impact it had on the overall performance. The last request was completely executed after 229 seconds. The latency between the Router and the connectors did not show significant delays.

TABLE II
PERFORMANCE TESTS RESULTS

#Users	#Requests	#Success	% Success	Execution time (mm:ss)
20	2,548	2,548	100%	08:44
40	5,139	4,073	79%	13:46

C. Costs assessment

Table III presents the cost evaluation of the gateway applied to the proposed case scenario. The evaluation showed that registering a future payment had minimum costs to the client[6]. On the other hand, the verification and the release of the future payment involved zero cost to the client, but had costs to be afforded by the gateway. Finally, the pump check had minimum costs for the gateway and client. The evaluation considered a gas price of 14.31 Gwei (1 Gwei is equal to 0.,000000001 Ether) and a cotization of 1.625 USD per Ether[7].

VII. Discussion

This section discusses the proposed research questions, performs an analysis of the obtained results and presents the limitations of the proposal.

[6]All costs applied to the client did not include the cost of electricity to charge the vehicle

[7]This price was gathered the 5th of November 2022.

TABLE III
COSTS ANALYSIS

Operations	Cost in gas (thousands)	Total cost (USD)	Total costs gateway (USD)	Total costs client (USD)
Register future payment	53.7	1.248	0	1.248
Check pump	51	1.186	0.580	0.606
Check future payment	29.3	0.681	0.681	0
Release future payment	23.9	0.555	0.555	0

How is it possible to apply the gateway solution to enable interoperability between Ethereum and Hyperledger Fabric? This work proposed a detailed design and a reference implementation on how to apply the gateway solution to enable interoperability between Ethereum and Hyperledger Fabric. The proposed design and implementation leverages existing building blocks and native features of both DLT to build a new Ethereum connector and reuse the existing Hyperledger Fabric connector following a choreography approach. Furthermore, a new concept of Future payments was introduced that enabled the purchasing of services provided by Hyperledger Fabric and be paid with Ethereum.

How does the proposed solution impacts on the performance while trying to interoperate Ethereum and Hyperledger Fabric? The performance assessment showed that there are challenges to achieve interoperability between Ethereum and Hyperledger Fabric under heavy load. The design of Ethereum to prevent replay attacks was the main obstacle to achieve this task.

Which are the costs required by the proposed solution when trying to interoperate Ethereum with Hyperledger Fabric? The cost assessment performed to the gateway on the proposed scenario showed that the gateway had minimum costs to the user that needed cross-chain transactions. Furthermore, some costs were applied to the gateway on some operations (check pump, check future payment, release future payment) that need to be considered when using this solution. After this evaluation, we consider that the proposed solution can be applied by a consortium to provide services that can absorb this costs in the overall price of the service. We do not think that this solution may be applied for small price services that can have a considerable impact on its final cost.

Besides the above comments, the experiment had some limitations. The proposed principles restricted the problem leaving out of scope cross-chain transaction verification and cross-chain transaction authentication. A fully reliable solution, requires the target DLT to validate any incoming transactions following the consensus protocol rules defined by the source DLT. Our approach trusted in the gateway and did not validate cross-chain transactions on the target DLT. Furthermore, Hyperledger Fabric requires to authenticate any incoming transaction. This means Hyperledger Fabric needs to verify the user and permissions of all cross-chain

transactions initiated from Ethereum. This task is a challenge, as Ethereum users use pseudonyms that hinders this task. Our approach provided a simplified solution to this challenge and the gateway acted on behalf of all of the users that initiated cross-chain transactions on Ethereum. Furthermore, all authenticated users were authorised to write on Hyperledger Fabric's ledger.

VIII. Related work

Ghosh et al. [20] proposed a gateway-based architecture that enabled interoperability between Ethereum and Hyperledger Fabric. They also proposed a cross-chain transaction verification based on Simplified Payment Verification (SPV) for Ethereum and signature validation for Hyperledger Fabric transactions. They proved liveness and safety properties for the verification process. This is the most related work we found, however, there is little information regarding the gateway design, implementation or performance. In particular, the authors did not mention performance issues while integrating the gateway with Ethereum as we had in our experiments. Their performance test were focused on Hyperledger and Ethereum, rather than on the integration solution. Considering this, our approach complements this work as it is focused on the gateway design, implementation and performance details.

André et al. [21] proposed a middleware based on the Secure Asset Transfer Protocol (SATP) [22] to bridge Central Bank Digital Currencies (CBDC) between Hyperledger Fabric and EVM-based[8] permissioned blockchains. The authors proposed the usage of smart contracts and connectors to build the middleware and shares similarities with our approach. André et al. went a step further than our work and proposed a mapping between EVM accounts and Hyperledger Fabric certificates to achieve cross-chain transaction authentication. The main differences with our approach is that the authors' middleware follows an orchestration architecture, while our work follows a choreography architecture. Furthermore, our approach focuses on permissionless EVM-based blockchains, like Ethereum, while the authors used a permissioned EVM-based blockchain. In addition, their work was focused on a specific asset (like CBDC), while our proposal is for general purpose cross-chain messages. Despite these differences, our work confirms the results obtained by the authors that is possible to enable interoperability between Ethereum and Hyperledger Fabric using either a choreography or orchestrated approach. The authors did not report any performance issues as we had. This different result may be explained as the authors performed latency tests independently from each other, while our performance tests incurred in twenty concurrent users.

Franzoni [23] proposed a decentralised application for the fashion industry to enable interoperability between Hyperledger Fabric and Ethereum. The proposal followed an orchestrated architecture, where a decentralised application interacts with both DLT. Franzoni proposed a component called Fab3 Proxy that acted as a proxy between the decentralised application and Hyperledger Fabric. The Fab3 Proxy provided cross-chain transaction authentication and mapped the identity on Ethereum (accounts) to the identities on Hyperledger Fabric (X.509 certificate). Our proposal differs from this work following a choreography architecture. Furthermore, we propose a general purpose interoperability solution that can be applied to the fashion industry or any other use case scenario. Franzoni performed an annual cost analysis to support the usage of both DLTs. This includes the hosting of Hyperledger Fabric on Amazon and costs of executing transactions on Ethereum for an estimated number of users. Our approach considered the cost that each transaction may have to the service provider and user (client), but did not estimate the annual cost of the proposal.

YUI [24] enables interoperability between DLT by implementing the Inter Blockchain Communication (IBC) protocol [25]. Each DLT must have an IBC Module that enables communication with other blockchains. The IBC Module implements the IBC protocol and verifies the incoming messages following the source DLT consensus protocol rules. YUI has similarities with our work as it follows a choreography architecture and enables interoperability between Hyperledger Fabric and Hyperledger Besu[9]. However, the support for the IBC protocol requires changes in the source code of the blockchain, in contrast with our approach. We could not find any reports regarding performance or costs applied by YUI.

Weaver [9] proposed a trusted relay approach to enable reliable data exchange between blockchains. Weaver is composed of an IOP Module, a Relay and an Identity Service. The Relay acts as an ingress and egress endpoint to the blockchain, providing service discovery and routing capabilities. The IOP Module was implemented as a set of smart contracts on the blockchain to provide access control, proof generation, proof verification, asset locks, claims, pledges and reclaims capabilities. The Identity Service provides the identity capabilities required to support authentication on permissionless blockchains like Hyperledger Fabric. Nowadays, Weaver supports interoperability between Hyperledger Fabric and Hyperledger Besu. The Weaver's approach is similar to our work as it follows a choreography architecture. Furthermore, it provides mechanisms for cross-chain transaction authentication and cross-chain transaction verification that can serve as input for our work. However, it does not yet support interoperability with Ethereum.

Hyperledger Cactus [8] is an interoperability framework that follows an orchestrated architecture and enables interoperability between business applications and different DLT. It provides validators that enable cross-chain transaction verification and a business logic plugin to allow developers to orchestrate cross-chain transactions between DLT in a coordinated way. Nowadays, Hyperledger Cactus enables interoperability between Hyperledger Fabric and Hyperledger Besu, but following a different approach than our proposal.

[8]Ethereum Virtual Machines or EVM-based blockchains are blockchains based on Ethereum source code.

[9]Hyperledger Besu is an Ethereum client.

Finally, Optimism bridge [26] and PoS Bridge [7] are two Sidechain solutions that enabled interoperability between Ethereum and Optimism and Polygon (two EVM-based blockchains). These bridges enable asset transfer between Ethereum and Optimism/Polygon and vice-versa by defining a communication protocol between them. This work shares similarities with our proposal as it follows choreography architecture. However, these two solutions are specific to asset transfer scenarios between these two blockchains and do not provide a general purpose interoperability solution.

IX. Conclusions and future work

This paper extends our previous work that used a gateway-based approach to enable interoperability between Corda and Hyperledger Fabric. The proposed extension enabled interoperability between Ethereum and Hyperledger Fabric. This approach was composed of a new Ethereum connector, a request-response interaction model based on the Enterprise Integration Patterns and the introduction of future payments, which allowed paying services provided by Hyperledger Fabric with Ethereum's cryptocurrency. A prototype was developed and evaluated through a case scenario, performance test and cost analysis. The evaluation assessed the technical feasibility of the proposal. Performance tests showed bottlenecks may limit its usage, where the design of Ethereum to deny reply attacks was one of the observed reasons. Costs analysis showed the approach fits purchasing services offered by a consortium for medium to large amounts, but it is not suitable for purchasing services with low amounts. These results are the paper's main contributions and constitute a step forward in developing general-purpose solutions for DLT interoperability.

Future work includes design improvements to enable better performance, improve the interaction model to support queries without costs and improve future payments to support more than one payment per account. Furthermore, the design evolution is expected to support cross-chain transaction authentication and verification.

Acknowledgment

Guzmán Llambías was supported by Pyxis. The research that gives rise to the results presented in this publication received funds from the Agencia Nacional de Investigación e Innovación under the code POS_NAC_2022_4_174476. Blockchain icons made by Freepik from www.flaticon.com.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009, (accessed Apr. 2023). [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014, (accessed Apr. 2022). [Online]. Available: https://ethereum.org/en/whitepaper/

[3] E. e. a. Androulaki, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, Porto, Portugal, Apr. 2018, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[4] B. e. a. Pillai, "Cross-blockchain technology: Integration framework and security assumptions," *IEEE Access*, vol. 10, pp. 41 239–41 259, Apr. 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3167172

[5] G. Llambías, L. González, and R. Ruggia, "Blockchain interoperability: a feature-based classification framework and challenges ahead," *CLEI Electron. J.*, vol. 25, no. 3, pp. 4–1, 2022.

[6] R. e. a. Belchior, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1–41, Oct. 2021. [Online]. Available: https://doi.org/10.1145/3471140

[7] Polygon Team, "Introduction to polygon pos," (accessed Apr. 2023). [Online]. Available: https://wiki.polygon.technology/docs/develop/getting-started

[8] H. Montgomery *et al.*, "Hyperledger cactus whitepaper," 2022, (accessed Apr. 2023). [Online]. Available: https://github.com/hyperledger/cactus/blob/main/whitepaper/whitepaper.md

[9] Weaver Team, "Weaver: Dlt interoperability framework," (accessed Apr. 2023). [Online]. Available: https://labs.hyperledger.org/weaver-dlt-interoperability/

[10] B. e. a. Bradach, "A gateway-based interoperability solution for permissioned blockchains," in *2022 XVLIII Latin American Computer Conference (CLEI)*, Armenia, Colombia, Oct. 2022, pp. 1–10. [Online]. Available: https://doi.org/10.1109/CLEI56649.2022.9959907

[11] R. G. e. a. Brown, "Corda: An introduction," 2016, (accessed Apr. 2023). [Online]. Available: https://docs.r3.com/en/pdf/corda-introductory-whitepaper.pdf

[12] G. Hohpe and B. Woolf, *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Boston, MA, USA: Addison-Wesley Professional, 2003.

[13] "Iso 22739:2020. blockchain and distributed ledger technologies — vocabulary," (accessed Feb. 2023). [Online]. Available: https://www.iso.org/standard/73771.html

[14] X. Xu, I. Weber, and M. Staples, *Architecture for blockchain applications*. New York, NY, USA: Springer Cham., 2019.

[15] G. L. et al., "Gateway-based Interoperability for DLT," Feb. 2023. [Online]. Available: https://www.techrxiv.org/articles/preprint/Gateway-based_Interoperability_for_DLT/22120520

[16] P. Wegner, "Interoperability," *ACM Comput. Surv.*, vol. 28, no. 1, p. 285–287, mar 1996. [Online]. Available: https://doi.org/10.1145/234313.234424

[17] D. e. a. Box, "Simple object access protocol (soap) 1.1," May 2000, (accessed Feb. 2023). [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[18] D. I. Castro Santestevan, "Estudio para la aplicación de la tecnología blockchain y la gestión inteligente a la red eléctrica uruguaya. caso de estudio : recarga de vehículos eléctricos," 2022, (accessed Apr. 2023). [Online]. Available: https://www.colibri.udelar.edu.uy/jspui/handle/20.500.12008/34204

[19] H. Jin, X. Dai, and J. Xiao, "Towards a novel architecture for enabling interoperability amongst multiple blockchains," in *2018 IEEE 38th Int. Conf. Dist. Comp. Sys.*, Vienna, Austria, Jul. 2018, pp. 1203–1211. [Online]. Available: https://doi.org/10.1109/ICDCS.2018.00120

[20] B. C. e. a. Ghosh, "Leveraging public-private blockchain interoperability for closed consortium interfacing," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, Vancouver, BC, Canada, May 2021, pp. 1–10.

[21] A. A. et al., "CBDC bridging between Hyperledger Fabric and permissioned EVM-based blockchains," Mar. 2023, (accessed Apr. 2023). [Online]. Available: https://www.techrxiv.org/articles/preprint/CBDC_bridging_between_Hyperledger_Fabri based_blockchains/21809430

[22] M. Hargreaves, T. Hardjono, and R. Belchior, "Secure asset transfer protocol (satp)," Mar. 2023, (accessed Apr. 2023). [Online]. Available: https://datatracker.ietf.org/wg/satp/about/

[23] S. Franzoni, "Blockchain and smart contracts in the Fashion industry," Jul. 2020, (accessed Apr. 2023). [Online]. Available: https://webthesis.biblio.polito.it/15336/

[24] Datachain, "Yui," (accessed Apr. 2023). [Online]. Available: https://www.datachain.jp/products/yui

[25] C. Goes, "The interblockchain communication protocol: An overview," Jun. 2020, (accessed Apr. 2023). [Online]. Available: https://arxiv.org/abs/2006.15918

[26] Optimism Team, "Bridging basics," (accessed Apr. 2023). [Online]. Available: https://community.optimism.io/docs/developers/bridge/basics/