Enabling Enterprise Blockchain Interoperability

Bernardo Carreira dos Santos
Instituto Superior Técnico
bsantostecnico@gmail.com

***Abstract*—In recent years, with the advances made in technology overall, there is a need to modernize the processes currently in place, especially in governmental organizations. This increases the number of different blockchain infrastructures that need to interact with each other, this is the reason that blockchain interoperability is so important and is a problem that needs to be solved for adoption to continue. In this document, we will study blockchain technology, as a possible fit as a solution for existing issues in these systems. A comparison between available blockchain infrastructures is shown to evaluate its applicability in real-world scenarios. We also have a comparison between the different interoperability solutions available, to be able to come to an appropriate solution regarding blockchain interoperability. The solution proposed in this thesis leverages blockchain technology to replace the current paper support model in place for Bills of Exchange with a digital model focused on the integrity and security of the data handled by the system, which is guaranteed by blockchain technology. Our solution is deployed using a blockchain infrastructure, Hyperledger Fabric, with nodes representing the multiple institutions involved. Interoperability between these systems is handled with resort to Hyperledger Cactus creating a network of relayers between the multiple blockchain infrastructures interacting. During our evaluation, we achieved a throughput of 114 TPS with an average latency of 0.15 seconds which shows that our solution would be capable of supporting the use of the current bills of exchange system. Regarding our interoperability solution, we have concluded that Cactus offers great flexibility for diverse interoperability use cases, and provides a direct way to further improve its compatibility, while in some niche use cases there might be better solutions that is the trade-off we are committing to by using Cactus.**

***Index Terms*—Blockchain, Interoperability, Hyperledger Fabric, Hyperledger Cactus, Bills of Exchange**

I. Introduction

Blockchain technology has seen a massive increase in adoption and exposure partly because of the financial incentives associated with it, but also because of the promise of providing decentralization and distribution of trust to systems where a single point of failure is the norm. This has caught the attention of organizations, especially governmental organizations. The focus of this paper and the use case provided by INCM was to create a digital platform for bills of exchange using blockchain technology to first work alongside the current paper support model eventually completely replacing it, to explore the interoperability required for such a system. With this in mind, we introduce a bill of exchange blockchain solution implemented using Hyperledger Fabric [1], a permissioned blockchain designed for enterprise solutions, accompanied by an interoperability solution that takes advantage of Hyperledger Cactus [2] to provide the flexibility and compatibility required associated with such a system. Our Fabric solution handles all of the life cycle associated with a bill of exchange asset, taking into account the organizations involved in this respective life cycle and permissions associated with each organization and member, this solution serves as a way to explore the multiple interoperability use cases associated with bills of exchange. As there can be several different implementations of bills of exchange systems, there is a requirement that our solution is able to seamlessly interact with those systems, this is provided by leveraging Cactus flexibility and compatibility as an interoperability solution.

To provide an initial evaluation of our bills of exchange blockchain solution we are going to be using Hyperledger Caliper that provides a framework for load-testing blockchain infrastructures. Regarding the evaluation of our interoperability solution, we are going to be discussing the complexity, flexibility, and compatibility that Cactus is able to offer.

II. Background and Related Work

A. Hyperledger Fabric

Hyperledger Fabric is an open-source blockchain platform, which serves as a distributed operating system for the creation of customized permissioned blockchains [1] focused on enterprise solutions. Fabric is designed with a modular architecture that allows the use of pluggable implementations of several functions [3] offering a high degree of flexibility, confidentiality, scalability, and resiliency.

A distributed application in Fabric consists of 2 core components, the *chaincode* which is a smart contract that is program code implementing the application logic and runs during the execution phase, Fabric also has a system chaincode, which is run in the configuration channel. The *endorsement policy* is evaluated during the validation phase and is used for transaction validation. It can specify the endorsers for a certain transaction as a subset of *peers*.

The configuration channel stores the definition of the Membership Service Provider, consensus configurations, ordering service parameters, and rules for altering the channel's configuration. Each channel has a different ledger, as each channel enforces chaincode and data isolation. Channels allow the creation of a communication network between participants, giving these participants access to the transactions they have permission to visualize.

As Fabric follows an *execute-order-validate* architecture, consensus in Fabric is broken into 3 phases, *endorsement*, *ordering*, and *validation*. A transaction proposal is created by a blockchain client, representing a member of an organization, and sent to endorsement peers, as defined in the endorsement policy. During the endorsement phase, the endorsers simulate the transaction proposal, producing a write-set, with the modified values and correspondent keys, and a read-set. This simulation is run in an isolated environment. With the endorsement created, it is sent back to the client which upon receiving enough endorsements creates the transaction and sends it to the ordering service. In the ordering phase, orderers check if the client who submitted the transaction proposal has the required permissions and produced blocks that contain the endorsed transaction ordered. The ordering allows the network to achieve consensus. The ordering service then broadcasts the blocks to peers who maintain the state of the ledger, through the ordering service or the peer gossip protocol. After this, we get into the validation phase where peers firstly check if the transactions follow the endorsement policy. Then, for each transaction, the versions of the keys in the read-set are compared with the keys currently in the shared ledger. Transactions with non-matching versions are discarded from the block and the block is then appended to the head of the ledger [4].

Fabric supports pluggable consensus for all 3 phases. This allows applications with different requirements to use different endorsement, ordering, and validation plugins to satisfy those needs.

The Fabric network is maintained by a group of *peers*, as Fabric is a permissioned blockchain, these are provided an identity by the modular *membership service provider*.

Peers can take up 3 different roles, *Clients*, *Endorsers*, *Orderers*. *Clients* are those who submit transactions for execution, help in the execution phase, and broadcast transactions for the ordering phase, while also keeping a snapshot of the current state of the ledger. *Client* peers are not able to invoke chaincode functions. *Endorser* peers have access to chaincode, and when they receive a transaction proposal, they simulate the execution of the transaction in an isolated environment, and based on that simulation's results they prepare a transaction endorsement to send to *Orderer* peers. *Orderer* peers receive endorsed transactions and assemble them into blocks while establishing the total order of all transactions, as each transaction contains state updates and dependencies computed during the execution phase. *Orderers* do not participate in the execution of validation of transactions, they propagate such blocks to *Client* peers, where the blocks are validated and committed to the shared ledger.

Fabric makes use of a key protocol for communication between peers, known as the *Peer Gossip*, which is responsible for broadcasting the results of the ordering phase for unsynched peers.

B. Hyperledger Cactus

Hyperledger Cactus [2] is a project, in the domain of trusted relays, trying to provide a framework for future interoperability solutions. Cactus has 4 core components, the connectors which allow establishing a connection between the blockchain infrastructures. The validators are effectively a node in both of the involved blockchains with permissions to publish transactions in both networks. The business logic plugin defines the logic of the interoperability use case between both infrastructures, such as the transaction flow and asset changes. We also have the verifier which is responsible for verifying and accepting results from validators.

Cactus offers a core framework for achieving interoperability between multiple distributed ledger technologies (DLT), which include blockchains and other more traditional DLT. To this core framework, by using the mentioned components cactus offers a pluggable way to achieve interoperability between heterogeneous system architectures. The plugins in cactus are effectively an abstraction layer on top of Cactus core source code. For each specific ledger Cactus consortium associates a group of validators, which effectively act as a secondary network that actively monitors the state of the underlying ledger network. Validators run a consensus algorithm separate from the connected ledgers to agree on the state of the underlying network. Upon agreement on the state, the proof of state is produced and signed by several validator nodes according to the consensus in use.

Validator nodes depend on ledger-specific plugins, consequently, a smart contract on the connected blockchain can enable the ledger-specific functionalities necessary for a validator node to observe the ledger state to finalize a proof of state. When providing the results, validators associate their digital signature, by using their key, with the results in order for verifiers to be able to certify the produced results.

These ledger-specific plugins are the Cactus connectors, which are composed of validators and verifiers, to provide communication between the business logic plugin and each involved ledger. By using validators and verifiers, connectors provide a way for the business logic plugin to operate and monitor the ledger behind them.

Verifiers are responsible for verifying the results and produced proof of state provided by validators. Verifier nodes can request and register the public keys of the validator nodes of a blockchain network that they want to connect to. Therefore, they can verify the signed proofs of the state of the blockchain since they have the public keys of the validator nodes. This implies that the verifier nodes trust the validator nodes and consequently they trust the Cactus consortium operating the validator nodes.

The business logic plugin executes business logic and provides integration services that are connected with multiple blockchains. It is composed of web applications or smart contracts on a blockchain. It is a single plugin and is required for executing Hyperledger Cactus applications.

It should be developed with a specific use case in mind, by implementing the business logic associated with such use case to interact with ledger plugins respective to each involved ledger.

It offers multiple interactions with the ledger plugins, such as submitting a transaction request at the targeted ledger, querying the targeted ledger, or receiving event messages associated with those transactions requests and queries.

C. Bills of Exchange

Bills of Exchange are a paper-written contract that involves 3 parties, the drawer which is the party that is in debt, the payee to whom the drawer is in debt, and the drawee which accepts the payment of the drawer's debt. Once the bill of exchange is approved by all parties, the drawee is legally bound to pay the drawer's debt to the payee on behalf of the drawer within a set deadline, so the debt between the drawer and the payee is extinguished [5]. This paper-based model doesn't work in the current situation of a global market where suppliers and customers aren't next-door neighbors, rendering Bills of Exchange unpractical to use. As creating a digital model for such a process has certain requirements, such as ensuring that such a document has legal validity and a non-tampering warranty, distributed ledger technologies such as blockchain technology is seen as a good fit.

As blockchain has been seen improvements in recent years, different solutions regarding a digital bill of exchange model have been in development. These solutions, such as Billex [6] and DigiBoE [5] resort to blockchain technology and more specifically smart contracts to implement the whole life cycle of bills of exchange. The core operations regarding the bill of exchange life cycle include the issuing of the bill by the drawer, the acceptance of such bill by the drawee, the payee receiving the bill, and redeeming it receiving the payment from the drawee. The current life cycle of a bill of exchange varies between countries, that is one of the reasons that blockchain interoperability is a requirement for such a system.

By using smart contracts it facilitates the creation of a digital platform that handles the life cycle of the bills, as the platform will be able to keep a record of issuing the smart contract and its current owner, the issue can also fill the details of the bill such as the payee's public key, the amount to be paid, its maturity date and then sign it with his private key providing the required non-tampering warranty. For certain platforms such as Billex redeeming the bill can even be automated by the account associated with the smart contract if its balance allows.

III. Bills of Exchange' Blockchain Solution Implementation

A. Requirements

The solution was developed and implemented as a database system capable of issuing, signing, registering, and executing all transactions and operations contemplated by the law for parties who rely on bills of exchange to obtain and provide funds. The solution should be able to interact with solutions from other countries, regardless of the infrastructure and technologies used in different implementations of this system. INCM has several non-functional requirements for this system:

1) *Availability:* The solution developed should be available 99.9% of the time.
2) *Scalability:* The solution should provide support for the possible increase of demand for use of the system without compromising with performance. (i.e., an increasing number of nodes participating in the platform should not affect its performance).
3) *Security:* The solution should provide a robust system for safely issuing, signing, and execute transactions or operations throughout the bills of exchange life cycle.
4) *Testability:* The solution should provide the possibility of being tested in a non-production environment. (i.e., an environment where operations such as issuing, signing, and execution of transactions can be simulated using similar characteristics to the production system).
5) *Privacy:* Information regarding bills of exchange should only be accessible to those who have the proper privilege rights to access that information.
6) *Auditability and Traceability:* The solution should provide ways for authorities (such as the Tax Authority, the Central Bank, and the Justice Sector) to have access to logs of the system to trace, prevent or act upon malicious or illegal activity.
7) *Interoperability:* The solution requires interaction with other bills of exchange systems to be able to support the current operations.

As for functional requirements, the solution is supposed to be integrated within the domain of INCM's projects and should facilitate all of the features currently allowed by the original model in use. This includes several operations regarding bills of exchange, such as:

- Issuance of bills of exchange
- Signature of bills of exchange
- Registration of bills of exchange
- Execution of all transactions and operations regarding bills of exchange
- Offer interoperability between other blockchain implementations handling bills of exchange

The system ingests data generated by each transaction executed by the users in the *Portal de Letras e Créditos Digitais* (PLCD) platform, which is the platform that gives access to the user to all operations contemplated by the applicable legislation in the life cycle of bills of exchange.

B. Preliminaries

By exploring the existing solution that is currently in use in Portugal we define the conditions and environment in which the bills of exchange system operates, this allows us to leverage this knowledge to be able to develop and implement a solution that works in these same conditions and still meets the mentioned requirements.

The use case discussed in this paper presents several characteristics:

1) Participants are willing to cooperate but have limited trust in each other
2) Bill's of Exchange assets is a responsibility of all stakeholders (this includes INCM, ATA, BP, and the participant banks)

3) Multiple organizations can have an administrative role in the network (such as INCM, ATA, and BP)
4) End users only have permission to access bill's related to them (where they are one of the entities represented in the bill)

A Blockchain participant represents an entity that participates in the blockchain. As mentioned, we assume that participants have limited trust in each other and participate in the same channel. Participants control peer nodes that maintain the ledger and may endorse transactions. We also assume that at least one participant is capable of running a blockchain client to commit transactions to the blockchain-based on the information present in the PLCD platform. There are 3 actors who take part in the network:

- *Network Administrator*. Network Administrators are responsible for managing all of the blockchain configurations. They also create identities and manage the participants in the network. The organization responsible for the network is in most cases the Administrator (in this case INCM), if there are several Administrators, a quorum will make the decisions instead.
- *Forwarder*. Forwarders are oracles responsible for interpreting operations related to bills of exchange executed in the PLCD platform and committing the respective transactions associated with those operations to the blockchain. These act as a blockchain client and oracle.
- *Auditor*. Auditors audit the life cycle of bills of exchange, such as the operations executed throughout its lifespan. In this use case, ATA and BP would be the auditors.

C. Architecture

The assets being stored in the blockchain are the bills of exchange generated by the PLCD platform, which is the information system that allows the end-user to issue operations on their bills. This proposed solution provides scalability in terms of the participating organizations. The information system mentioned and the blockchain for the bills of exchange are independent, this is possible due to the modularity offered by Hyperledger Fabric. The blockchain component includes the ledger, permission management, transaction validation and definition, and the data model. This architecture is represented in Figure 1.

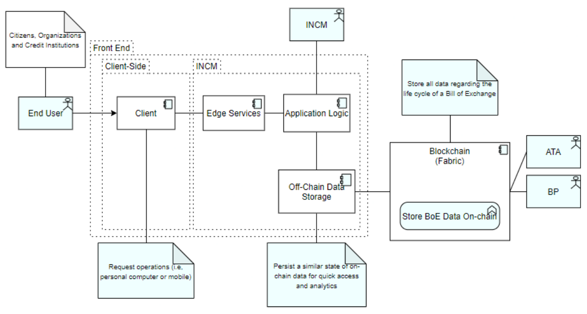
Citizens, Organizations and Credit Institutions
INCM
Front End
Client-Side
INCM
End User
Client
Edge Services
Application Logic
Store all data regarding the life cycle of a Bill of Exchange
Blockchain (Fabric)
ATA
BP
Off-Chain Data Storage
Store BoE Data On-chain
Request operations (i.e, personal computer or mobile)
Persist a similar state of on-chain data for quick access and analytics

Fig. 1. Architecture of the Blockchain Platform for Bills of Exchange

Blockchain clients allow organizations to request operations to the blockchain, for different needs we should have different clients as each different client is able to request different operations depending on their permissions and roles.

In our use case, we require 3 different blockchain clients, as we have INCM acting as a network administrator and forwarder between the PLCD platform and the blockchain, we have ATA and BP acting as auditors requiring access to data respective to bills of exchange, and lastly, we have banks which directly interact with a specific operation in the bills of exchange life cycle. These 3 blockchain clients are defined as follow:

1) Forwarder Client: After receiving transaction requests from the PLCD platform, those are processed and verified before a blockchain transaction being committed to the network, triggering the updates necessary on the respective asset.
2) Audit Client: Responsible for committing transactions on behalf of the auditors, ATA, and BP, these transactions are in fact requests from the auditors to query the blockchain ledger regarding data from bills of exchange assets in the blockchain network.
3) Bank Client: This gives banks the capability to commit transactions in respect to the Discount operation in which banks are directly involved, to make the necessary changes to the bills of exchange asset.

Fabric's blockchain component requires 3 different types of peers: Orderer peers (orderers), Endorser peers (endorsers), Committing peers (peers). Orderers are assigned in the configuration file which can be deployed by the network administrators, and provide delivery guarantees of blocks containing transactions while assuring atomic communication, consensus. A node can be both an endorser and a peer node at the same time, endorsers host and execute instances of the chaincode to run simulations of transactions, peers host instances of the ledger.

It is important to note, that in Fabric, there is the possibility to attribute different levels of trust to each individual peer using roles. Fabric also offers a modular system that decouples the trust system from the consensus algorithm, this is done using endorsement policies that can assign a higher or lower degree of trust to specific subsets of endorsing peers in the network.

We have a single instance of chaincode which defines all of the life cycles of the bills of exchange, both endorsing peers and committing peers are required to have this chaincode installed.

Regarding the authentication of participants, Hyperledger Fabric offers a built-in CA to issue identities to each participant, but a custom CA could be used if INCM chooses to do so. These certificates are important as they are used by participants to sign transactions, ensuring non-repudiation of transactions.

With respect to data privacy between different organizations, we decided to tune the ABAC [7] system through chaincode where we can define which organizations have permissions to access data, as it offers the best performance and ease of implementation while being able to address the restrictions in our use case.

The architecture we end up with has a single channel where we have the 3 main organizations, being INCM, BP, and ATA, where in the future Banks should be represented as they are crucial for the bills of exchange life cycle. We also have a single applicational chaincode that implements the logic behind all of the operations regarding bills of exchange. Lastly, to address the privacy concerns we end up tuning an ABAC system to meet our needs as mentioned before.

D. Implementation

As mentioned Fabric uses a certificate authority to generate the necessary cryptographic information, such as the key pairs, to enroll different participants in the network. The CA server can be configured to define what implementations will be used for the generation of the certificates, we have opted to use Fabric's default implementation.

We have implemented the solution with 3 default organizations which represent INCM, ATA, and BP. Each node representing a different organization has different specific permissions regarding which data can be accessed and which operations can be requested, using the ABAC system. Nodes from INCM, the network administrator, have a higher level of permissions allowing them to validate and execute smart contracts and transactions, ATA and BP nodes can validate transactions and access the ledger as an auditor. These configurations and permissions allow enforcing the necessary privacy and confidentiality of data.

The blockchain client is written in NodeJS, it allows communication with the blockchain, providing an entry point to the developed chaincode mentioned before. Taking into account the permissions defined for the authenticated client, the chaincode will then access the distributed ledger and execute the requested operations: such as register a bill of exchange, protest a bill of exchange or audit the network.

While possible to develop a graphical user interface, the clients made were used through the command line. A script was used to simulate the population of the blockchain with bills of exchange assets.

IV. Interoperability Solution Implementation

In this section, the goal is to explore interoperability solutions between heterogeneous blockchains such as a permissioned blockchain like Fabric which we used for our bill of exchange solution as it was outlined in the previous section, and a permissionless blockchain which in this case we have chosen Ethereum as shown in 2. Apart from these networks required to explore the interoperability use case, we are using a validator to monitor both networks and commit transactions associated with the different interoperability use cases being explored which are implemented using a business logic plugin, for this the connectors respective to each network are required.

When it comes to bills of exchange, there are several interoperability use cases that can be explored with Hyperledger Cactus. These include replicating the assets between multiple implementations of bills of exchange, replicating transactions based on a specific trigger such as having a bill of exchange being registered in the Portuguese network for bills of exchange that involves an entity from Spain, you can replicate the transactions associated with this specific asset on the Spanish network for bills of exchange. We could also use a foreign currency or even a cryptocurrency to liquidate a bill of exchange, which would require interoperability between our solution and that cryptocurrency blockchain.

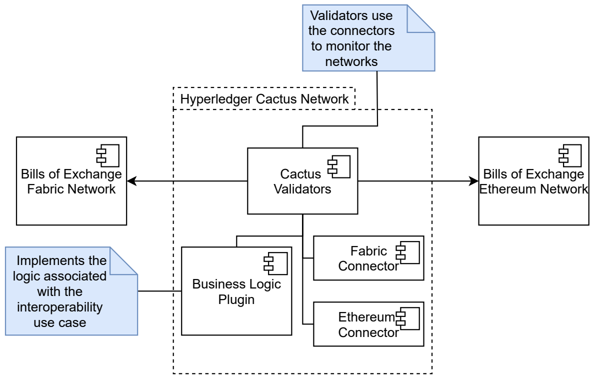
Validators use the connectors to monitor the networks
Hyperledger Cactus Network
Bills of Exchange Fabric Network
Cactus Validators
Bills of Exchange Ethereum Network
Implements the logic associated with the interoperability use case
Business Logic Plugin
Fabric Connector
Ethereum Connector

Fig. 2. Architecture of the Cactus interoperability solution

While there is no standard for how to implement a bill of exchange blockchain solution, every bill of exchange implementation, regardless of the country it is associated with, requires a core set of operations such as registering or paying the bill. Taking this into account we define the required transaction flow associated with each interoperability use case to achieve the needs for that specific use case, which we will go more in-depth throughout this Chapter.

Most of the work being done is associated with designing business logic plugins that define the transaction flow for each interoperability use case. We will outline that work for each interoperability use case explored, while also mentioning the difficulties and where it could have been improved.

A. Network Replication

The goal with this specific use case is to replicate every transaction requested in one of the involved networks, we have our solution developed on Hyperledger Fabric and a sample solution using Ethereum, whenever a transaction is requested on one of the networks it should trigger Cactus to request the corresponding transaction on the other network.

For this to happen, on the business logic plugin we need to associate transactions from the solution which uses Fabric and the solution using Ethereum, for the business logic plugin there is complete abstraction from how these blockchains work, it is not required for them to behave similarly. While both solutions might have different operations for Bills of Exchange, both should have the core operations required for their life cycle, in this case, if a register operation is requested in Fabric's solution, the respective register operation should be triggered by cactus on Ethereum's solution.

As mentioned cactus has nodes monitoring the blockchain's state, using this we can react to updates to this state, such as new transactions being committed, which in this case helps us replicate the exact same state of each asset on the blockchain to other solutions, working similarly to an oracle.

A user commits a transaction to Fabric's network, in this case, registers a bill of exchange, the transaction gets processed in Fabric's network which triggers a new state of the network. Cactus validators network by monitoring the network are aware of this new state and publish the respective transaction to the Ethereum network which then similarly to Fabric ends up creating a new state of the network. We have to be aware of this because new states are what triggers Cactus validators, so we need to have a condition where transactions committed by Validators do not trigger a transaction request on Cactus end.

For this specific use case, there are simpler solutions if the goal is only the replication of the network for the purpose of redundancy which we went over before. One of the problems of solutions that focus on replication only is that it is hard to achieve the transaction history that led to that specific replicated state of the network, Cactus offers a way to accomplish that by actually replicating each transaction requested individually.

B. *Cross-country Asset Replication*

this use case works somewhat similarly to the previous use case explored in the sense that we are effectively reproducing certain transactions between the involved networks. Similarly, we have our own solution previously outlined using Fabric, and an Ethereum sample solution, both running implementations of the bills of exchange use case.

The goal in this use case is to replicate transactions for assets that involve entities from different countries, lets say our Fabric solution and the Ethereum sample solution are running implementations for Portugal's and Spain's respective bills of exchange systems, if a certain bill involves entities from Portugal and Spain, we want to reproduce transactions associated with this specific asset in both networks.

Comparably to the previous solution, we still have to associate corresponding transactions between both networks. The difference in this use case is that instead of replicating every transaction that produces a new state in the ledger, we are being more restrictive by only reproducing transactions for specific assets. In addition to cactus validators nodes to react to changes in the blockchain state, this state update will trigger a verification process to assess if the assets being changed involve entities from each network, in this case, if there are entities from Portugal and Spain associated with this asset. If this is the case then the validators will request a transaction in the opposite network to replicate the state of this asset.

In the case where the transaction does not involve entities from both networks' respective countries, the validator still detects a state update, since the validator network is always monitoring the networks, but after verifying the entities associated with the asset it will not trigger any transaction request on the opposite network.

When the transaction does update an asset that has entities from multiple countries associated with, after detecting an updated state and verifying the entities involved in the transaction the validators will then request the respective transaction in the opposite network, effectively replicating the exact same life cycle for this specific asset in both networks.

Similarly to the previous use case, cactus offers a way to replicate the exact transactions that led to a specific asset current state, keeping its transaction history, in this case, we are not going for full replication of the network but focusing on single assets that respect certain constraints, which in this case Cactus offers a very good solution as the verification for those constraints occurs in the validator nodes without any need for changes in the blockchains interoperating.

C. *Cross-blockchain Payment*

With countries starting to accept cryptocurrencies as legal tender, such as El Salvador, and allowing cryptocurrencies to be seen as a country's currency, this use case becomes more interesting to explore as this could be a potential need in the future. The goal of this use case is to use an existing cryptocurrency to liquidate a bill of exchange asset.

As far as the architecture used, similarly to the previous use cases we are still using our Fabric solution, but in this use case the Ethereum network used is not running a sample implementation for bills of exchange, it is simply being used to run Ethereum transactions as a payment option for bills of exchange to show that if in the future more countries start to accept cryptocurrencies as a currency, this is a possibility.

For this use case instead of having to match the transactions representing operations respective to bills of exchange, we have to find a way to associate, in this case, an Ethereum address to a certain bill of exchange asset or an entity related to this specific bill. The simplest way for this to be done is in the case where the bill of exchange solution is developed with this in mind, and the asset itself can have an Ethereum address associated with it for payment purposes.

Another problem with this use case is the need for an escrow account, as the payment operation on the Fabric network can only be processed after confirmation of payment, but the payment can only be sent after the payment operation has been processed. In this case, Cactus validators can act as escrow where they receive the payment from the Ethereum network and hold it until the payment operation in Fabric's network has been processed, finally, they send the payment to the final address associated with the bill.

If for any reason the transaction for the payment of the bill fails in Fabric's network, the Ethereum that had already been provided as payment is returned to the original address it was sent from instead of being sent as payment to the Fabric's user address associated with the bill being liquidated. This keeps the funds safe until there is confirmation that the bill's payment succeeded, it also prevents the bill from being paid without the funds being secured for the owner of the bill.

It is important to explore such a use case as the future seems to be approaching a state where cryptocurrencies might be seen as an actual currency and start being adopted as such, this also shows that Cactus can be quite flexible in terms of the problems it can solve in a somewhat simple way. It also shows how future-proof Cactus is. While we explored this use case using Ethereum as the cryptocurrency being used, multiple other ones could have been used as long as Cactus already has a connector developed for it, otherwise, a connector could also be developed.

V. Evaluation

A. *Bills of Exchange Blockchain Solution*

In order to evaluate our Bills of Exchange blockchain solution we are going to use Hyperledger Caliper as our load-generating client, Calipers framework goal is to facilitate the evaluation of multiple blockchains solutions built on different infrastructures such as Hyperledger Technologies blockchains, which include Fabric, and others such as Ethereum. Caliper serves as a load-generating client by running tests based on configuration files, which helps to replicate the architecture of the solution's network, to run tests.

To emulate a real production environment where we have several distributed nodes, we are using a GCE machine set up in Amsterdam, Netherlands with a 16vCPU and 256GB of memory. This helps to keep the hardware used for the test environment used to run the tests consistent across every round of tests.

As far as configuring the test environment, we are using an Hyperledger Fabric version 2.2.1 running a simplified network with a single channel with 3 organizations representing INCM, ATA, and BP each with one peer and 1 CA. The consensus algorithm being used is solo orderer which is designed for testing purposes where it effectively bypasses the consensus process. We are also using the default network configuration provided, with a maximum block size of 128MB, a batch timeout of 250ms and the number of transactions per block is 10.

The peers, orderers, and CA are being run on top of Docker containers running Docker version 20.10.8, running the base image of Hyperledger Fabric. The state database used is the database provided by Fabric, LevelDB.

While there is not enough data regarding the use of the system that is currently in place, according to the small amount of data we have the system achieves a peak use of 7000 operations monthly, which is a really small amount of transactions per month for such a system. Taking this into account if we can achieve an average TPS of 5, we can cover much more than the current use of the system that we currently see.

In the following tests, we are going to issue transactions at a constant rate of 5 TPS. We are also going to vary the number of transactions issued in total between 1000, 2000, 4000, and 8000 transactions. Furthermore, we are going to vary the number of blockchain clients between 1, 2, 4, 8, and 16 clients submitting transactions, in our context these clients act as our *forwarders*.

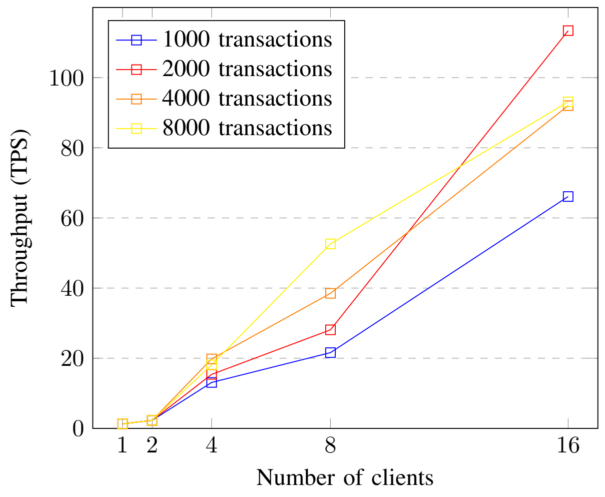

Fig. 3. Throughput variation with different number of clients, using a fixed-feedback controller

The above graphic (Figure 3) shows the variation of the throughput with the number of clients, which in our tests we can see that for a number of clients of 1 or 2 the throughput is equal independent of the number of transactions, but we see more variation with increasing numbers of clients which is a more likely scenario in regular use of the system.

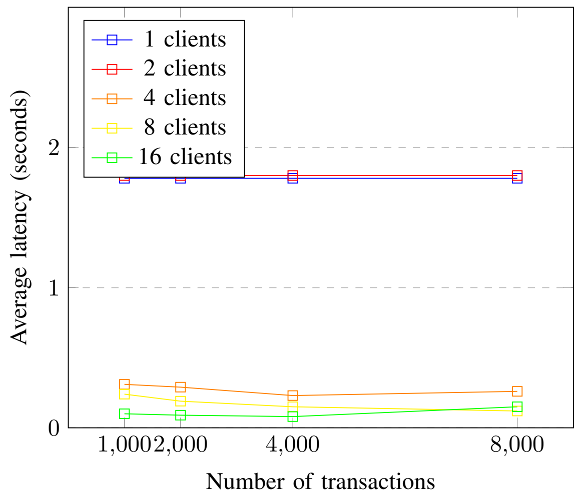

Fig. 4. Average latency variation with different number of transactions, using a fixed-feedback controller

In the above graphic (Figure 4) the variation of the average latency per number of clients is shown, with transactions being submitted at a constant rate of 5 TPS similarly to the previous graph. Usually the more clients the bigger the latency as the peer nodes have to return answers to more clients before a transaction is considered valid, in our testing that is not what we have verified as we have 16 clients with the lowest average latency.

During these tests, something seems to have occurred when we were testing with 1 and 2 clients submitting transactions as the results from those cases seem to be too far apart from what is expected in both throughput and average latency.

Regarding our performance evaluation, we have achieved a peak TPS of 113.4 with a latency of 0.15 seconds in the same test, this peak occurred for a test where 2000 transactions were submitted and 16 clients were being used, while it is likely that fewer clients will be used on a real scenario, during our tests we can see that for any number of clients above 4, we can achieve decent performance without compromising latency. Tests were run multiple times to assure that the results were correct.

One of the places where there are improvements to be made is Fabric's blockchain configuration, this includes several performance improvements in terms of TPS. We could have tested several parameters such as the batch size which includes the size of the block in terms of data and also includes the number of transactions per block, which if we used a higher value in both parameters we could possibly achieve higher values of TPS. Besides this we could also experiment with different values for the batch timeout which is the time for a block to be deemed invalid if not processed in a certain time frame, a higher value would give more time for blocks to be processed. We could also experiment different with consensus algorithms.

B. *Cactus Interoperability Solution*

In this section, we are going to evaluate Cactus compatibility, flexibility, and complexity. It is important to understand that Cactus is still in development and is not production-ready at this moment, it should be production-ready in version 1.0, and it is currently in version 0.4.2. This means that there are still developments being made in the core framework, and there are still connectors to be released, this will improve the usability of Cactus as an interoperability solution.

Regarding the **compatibility** that Cactus offers, it is first important to understand how Cactus can implement interoperability solutions between different blockchains, this includes both permissioned and permissionless blockchains.

Cactus relies on validators to perform transactions on multiple blockchains to facilitate interoperability between those respective blockchains. These validators in turn require a connection to the respective blockchain to submit transactions, this is where Cactus connectors come into play. As each blockchain infrastructure implements a different consensus algorithm, Cactus requires a different connector for each different blockchain infrastructure.

In our specific use cases, we are using connectors that Cactus already has developed, these being connectors for Hyperledger Fabric and Ethereum. Cactus already has other connectors available such as connectors for most Hyperledger Distributed Ledger technologies such as Hyperledger Besu and Hyperledger Sawtooth with existing business logic plugins samples that show these connectors working.

In terms of compatibility the only development necessary to be done in Cactus is effectively the connectors, and with the current number of different blockchain infrastructures available it is hard to develop connectors for every existing one. However, Cactus should be able to support interoperability between most existing blockchain technologies both permissioned and permissionless, as shown in our use cases where we are using both with that being Hyperledger Fabric and Ethereum respectively. Effectively, Cactus offers backward compatibility, by not requiring extra development to the blockchain technologies being used.

With respect to **complexity**, we are not going to evaluate if the implementations steps themselves are complex, we are instead going to evaluate the number of steps required to implement an interoperability solution using Cactus in the different possible scenarios. It is hard to evaluate if the implementation steps taken to achieve a certain solution are or not complex as that is subjective to whoever is implementing them, and how familiar the framework is to them.

There are effectively 2 scenarios when it comes to using Cactus for implementing an interoperability solution, something common in these scenarios is that you will always have to develop a BLP as that is specific to each interoperability use case being implemented. In addition to this, your solution may require the development of a new connector as Cactus does not have connectors available for all existing blockchain infrastructures.

In the worst-case scenario, a connector will have to be developed and tested before actually being available to use in the interoperability solution being implemented. This can be simpler if the respective blockchain infrastructure uses a similar consensus to current blockchain infrastructures supported by Cactus which can be used as an example.

Concerning the **flexibility** of the solution, what we are trying to evaluate here is if the solution can be used to implement a wide variety of use cases, in our case we explored 3 different scenarios each with its own goal. This was done effectively by developing 3 different BLP, as explained before the BLP is what allows the logic behind a specific interoperability use case to be implemented. They define the transaction flow associated with the interoperability use case being explored, respective to each blockchain infrastructure involved in the use case defined by the BLP.

As a consequence of requiring a BLP for each different use case, it means there will be development associated with each new interoperability use case being implemented. Cactus unlike other interoperability solutions requires that whoever is implementing a specific use case has knowledge regarding the multiple applications involved.

This is required because when developing the business logic plugin you are effectively defining what triggers a certain interoperability function, what transactions will the validator submit to each associated distributed ledger depending on the function triggered, and in what order should those transactions be processed. In order for this to be possible, you need to be familiar with the applications involved in the interoperability solution, and Cactus.

The advantage that this provides is that by using Cactus as an interoperability solution you are not required to develop your blockchain solution taking into account the possible interoperability use cases it will require in the future. As all of the work regarding the necessary interoperability can be done when using Cactus to implement those use cases by developing their respective BLP.

One of the focuses of this paper was to explore the interoperability between permissioned and permissionless blockchains, this is a common problem because the implementation of permissioned and permissionless blockchains diverge in multiple points, which in turn hampers the development of interoperability solutions using heterogeneous blockchains. The way Cactus provides interoperability is by using an auxiliary network of Validators to respond to triggers and submit transactions on all of the involved blockchains, them being permissioned or permissionless. So, validators effectively act as participants in those networks, which solves the problem.

As mentioned, the biggest problem with providing blockchain interoperability is that there is an increasing number of new blockchains being developed, and there is no existing standard that these blockchains follow, both at the root of the blockchain that is being developed, as well as the applications which work on top of them.

If we take the 3 scenarios we implemented to show Cactus flexibility as an example, the main concern is that you are required to have knowledge of all of the applications involved in the interoperability use case being implemented, taking our replication scenario as an example, you are required to know which operation has the same result in all applications. Taking an example where multiple organizations are going to develop new applications that will only require interoperability between each other, using something like Cactus might not be the best solution, something like Cosmos or Polkadot which work like an ecosystem where every application built on top of it is already able to interoperate between each other might be a better solution.

The biggest advantage of using Cactus is that it presents itself as a great interoperability solution for cases where your application requires interoperability with an already existing blockchain application. In general Cactus provides a good solution for interoperability, but for very niche use cases, there might certainly be better solutions, such as the case mentioned previously where something like Cosmos or Polkadot would work better.

VI. Conclusion

This paper presents a solution that aims to replace the current Bills of Exchange system which resorts to paper support by a digital solution using the Hyperledger Fabric blockchain technology, these solutions will require to be able to interoperate with solutions implemented in other countries, that is where we propose an interoperability solution using Hyperledger Cactus framework.

During our evaluation of our Bills of Exchange blockchain solution, we managed to achieve a peak of 113 bills of exchange operations per second with an average latency of 0.15 seconds, using 16 clients which ends up being a slightly more costly solution. By using a lower amount of clients such as 4 we achieve an average bills of exchange operations per second of 16 to 20 with an average latency of 0.21 to 0.30 seconds. As far as storage, since our solution uses a very simple implementation of the bills of exchange assets, we can not provide a proper estimate of how this would impact the costs, but in general, we can conclude that the storage costs will increase proportionally to the number of peers in the network. There is a clear trade-off between decentralization and trust which such blockchain-based solutions offer, and the performance and storage requirements that a traditional system provides.

Regarding the interoperability solution, we implemented 3 different scenarios based on our Bills of Exchange blockchain solution to explore what Hyperledger Cactus can offer. This allowed us to explore how flexible such a solution can be by implementing 3 use cases with different goals and different implementation requirements, and we also explored how Cactus is able to provide compatibility with already existing and to-exist blockchain solutions as that is a crucial focus point with interoperability solutions. We concluded that Cactus can be used for a wide variety of use cases, and while for very niche use cases there are better solutions, Cactus is able to provide backward compatibility by not requiring for there to be extra development to already existing blockchain infrastructures as all the work is done on Cactus side. This can be done as all the logic associated with the interoperability use case is implemented as a Cactus plugin. Concerning the compatibility, as Cactus uses a network of validators to issue transactions on all of the involved ledgers to achieve the required interoperability, which depends on Cactus connectors. These connectors can be developed for each different blockchain technology being used.

References

[1] Hyperledger fabric: a distributed operating system for permissioned blockchains. Androulaki, Elli and Barger, Artem and Bortnikov, Vita and Cachin, Christian and Christidis, Konstantinos and De Caro, Angelo and Enyeart, David and Ferris, Christopher and Laventman, Gennady and Manevich, Yacov and others.

[2] Hyperledger Cactus Whitepaper. Hart Montgomery and Hugo Borne-Pons and Jonathan Hamilton and Mic Bowman and Peter Somogyvari and Shingo Fujimoto and Takuma Takeuchi and Tracy Kuhrt and Rafael Belchior.

[3] Architecture of the hyperledger blockchain fabric. Cachin, Christian and others.

[4] Hyperledger Architecture Volume 1: Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus. Hyperledger Architecture Working Group and others.

[5] Envisioning the Digital Transformation of Financial Documents: A Blockchain-Based Bill of Exchange. Ponza, Andrea and Scannapieco, Simone and Simone, Anna and Tomazzoli, Claudio.

[6] Blockchain-Based Discount Bill of Exchange – BILLEX

[7] Attribute-based access control. Hu, Vincent C and Kuhn, D Richard and Ferraiolo, David F and Voas, Jeffrey.