

A Reference Implementation of Blockchain Interoperability Architecture Framework

Harish V, Swathi R, Satyanarayana N

e-Security Department, Centre for Development of Advanced Computing, Hyderabad, India

vharish@cdac.in, rswathi@cdac.in, nanduris@cdac.in

Abstract—In this paper we describe interoperability architecture between blockchain networks. A blockchain network can serve multiple organizations that form as a consortium for dealing with transactions among themselves. However, when multiple blockchain networks are deployed how do we support interoperability and what are the major concerns in this direction have to be studied. Towards this direction, we developed a reference architecture for smooth transmission of transactions between the blockchain networks using custom designed transaction flow mechanism on top of Cactus framework from Linux Foundation.

Keywords—blockchain, interoperability, Hyperledger fabric, Hyperledger sawtooth

I. INTRODUCTION

There is no single blockchain platform which caters to the needs of each and every application domain. Certain blockchain platforms have been designed keeping a particular application domain in view (for e.g., corda is considered to be well suited for financial related applications). Hence there is a need for exploring opportunities and technical feasibilities for providing cross chain support between different domains.

Suppose when existing blockchain based applications prefer to switch to a new or an upgraded blockchain platform there is a need for interoperating with legacy blockchain platforms. In such cases, cross chain or multichain is an option where multiple chains coexist and interoperate with each other corresponding to an application domain. Further when a blockchain network managed by a consortium of organizations want to interoperate with another blockchain administered by a separate consortium of organizations we need effective interoperable blockchain solutions.

Scheme/Platform	Hash Time Lock Contracts	Oracles	Notary Scheme	Relays/Side Chains
Polkadot [7]	No	No	No	Yes
Cactus	Yes	No	Yes	No

In literature survey, we can find many different interoperability approaches suggested by researchers [1], [2], [3] and some of them have been implemented also. Each proposed technique of interoperability has its own purpose. For example, Hash Time Lock contracts can be used for confirmation of

transaction upon confirming payment between the respective parties.

In our approach, we have taken Cactus [4] an interoperability framework from Hyperledger community as a base code and built up on it a transaction flow model in the context of interacting applications running on different blockchain platforms. Without the transaction flow mechanism in place with the available source code the logic to deal with exchange of transaction information across the blockchain had to be dealt by the cactus middleware. This results in a bottleneck situation as the cactus middleware code has to be redesigned as and when more networks are to be added by integrating necessary logic to deal with transaction information exchange between them. We studied this aspect and have come out with a unique strategy of making cactus middleware responsible for information exchange only based on the pre-define transaction flow information. The transaction flow information can be updated on the fly as and when new networks or applications contexts in which different blockchain networks are to be operated.

II. PROPOSED BLOCKCHAIN INTEROPERABILITY FRAMEWORK

A. Architecture

In the proposed architecture, we support two different types of communication mechanism across the blockchain platforms that needs to be interoperate with each other as follows.

Types of communication:

- Immediate Confirmation:** In this type of communication, a client submits a transaction to one of the Blockchain networks which gets confirmed on that particular Blockchain network. Later, the transaction particulars will be forwarded to another network through middleware. The second blockchain network upon receiving the transaction particulars from the middleware, processes it and confirms the same on its side.
- Deferred Confirmation:** In this type of communication, a client submits a transaction to one of the Blockchain networks which will be forwarded to the other Blockchain network may be after initial processing through a middleware. The second blockchain network upon receiving the transaction particulars from the middleware, processes it and confirms the same on its side and sends back the transaction to its initiator through the same middleware. The first Blockchain network upon

receiving the transaction particulars from the second blockchain network, it processes the transaction on its side.

The block level diagram of the Interoperability architecture in the context of National Blockchain Framework (NBF) is as shown in figure 1 below.

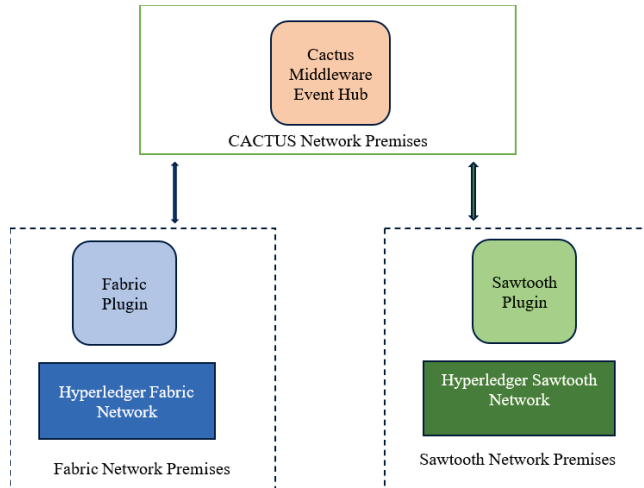


Figure 1: Block level diagram of NBF Interoperability Architecture

Fig 1., depicts the block level diagram of NBF Interoperability solution. In this architecture, a fabric network [5] administrator deploys the Fabric plugin network component on their side and a sawtooth network administrator deploys the Sawtooth plugin network component on their side. The cactus-based middleware event hub can be installed on mutually agreed premises.

The respective plugins listen for transaction block events from corresponding blockchain networks and shares them with each other in accordance with the transaction flow logic controlled by the cactus middleware event hub. The transaction block event comprises details about all the transactions hold by a particular block that is generated by the corresponding blockchain network.

The transaction flow logic that is handed over to the cactus middleware event hub is the core component that determines above mentioned types of communication. We will delve in to each component in details in later sections.

Fig 2., below depicts an example deployment scenario of NBF's Interoperability solution between Hyperledger Fabric and Hyperledger Sawtooth [6] networks.

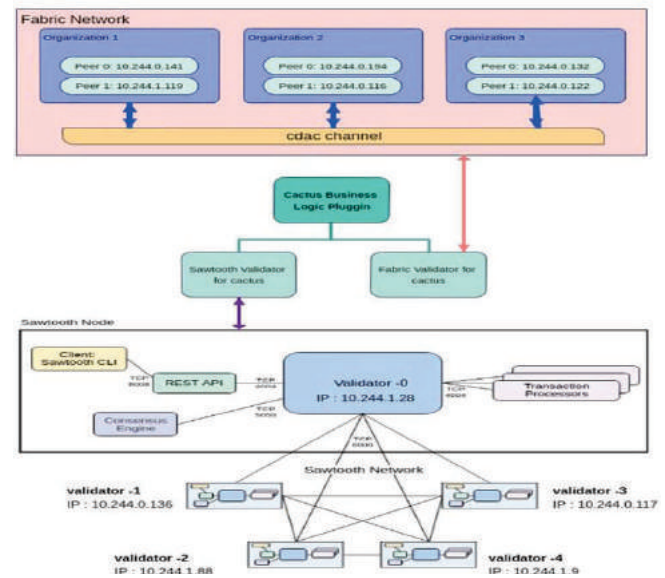


Figure 2.: Example Deployment Scenario of Interoperability Solution

B. Essential Requirements of Operating Proposed Framework

Network Perspective:

The administrators of the Fabric, Sawtooth and Cactus Middleware Event Hub components must be knowing about YAML (Yet Another Markup Language) syntax. A detailed tutorial about various directives written in YAML in different configuration files can be found in the developed software.

1) Fabric Network

A Hyperledger Fabric network must be in running state and its admin identity credentials from its wallet must be shared with the Fabric plugin administrator by external way.

The fabric plugin component must be configured with fabric network details hence its details must be made available to the Fabric plugin administrator. Ideally, the plugin would be installed on Fabric network side only hence the fabric network administrator only can administer plugin configuration also.

2) Sawtooth Network

A Hyperledger Sawtooth network must be in running state and its admin identity credentials from its wallet must be shared with the Sawtooth plugin administrator by external way.

The sawtooth plugin component must be configured with sawtooth network details hence its details must be made available to the Sawtooth plugin administrator. Ideally, the plugin would be installed on sawtooth network side only hence the sawtooth network administrator only can administer plugin configuration also.

3) Cactus Middleware Event Hub

The fabric network administrator and sawtooth administrator have to share their respective plugin details with the Cactus Middleware Event Hub administrator. Apart from

that the cactus middleware event hub administrator also should get transaction flow details between the applications deployed on Fabric and Sawtooth networks in consultation with the respective network's smart contract developers.

Developer Perspective

1) Chain code / Smart code Developer

The chain code developer has to upgrade their current chain code/smart contract in order to support interoperability as per the tutorial provided in developed software. The tutorial is made available for JavaScript developers.

A sample code snippet in the tutorial being provided through the software is described below.

Let's assume that we have a smart contract with the name electricityAsset.js in which updateMeter(ctx, meterID, owner, usage) is one of the functions. It's code looks like the below.

```
async UpdateMeter(ctx, meterID, usage) {
  let updatedMeter = {};
  updatedMeter.meterID = meterID;
  updatedMeter.usage = usage;
  ctx.stub.putState(meterID,
    Buffer.from(JSON.stringify(updatedMeter)));
  return;
}
```

The code written above is self-explanatory hence no detailed description is provided here. Now, in case, if the application owner would like to interoperate with any other Blockchain network, he/she has to revise the code of UpdateMeter. In the below example, it is envisaged that as and when client application sends its request to the Hyperledger Fabric network specifying the name of chaincode function to be invoked i.e., UpdateMeter_db and an object with request parameter property as a mandatory property and dbserver, nextSeq property as an optional property.

For example,

```
contract.submitTransaction("UpdateMeter_db",{requestparam
:{meterID:101, usage:32}});
```

Here, the request parameters that the client used to send in legacy code have been embedded into an object as requestparam. Apart from that two optional properties (dbserver and nextSeq) also can be set to convey to the chaincode that mongo dbserver is required to be used for fetching additional information and nextSeq value that indicates the cactus middleware event hub about its next course of action as defined in the transaction-flow.yaml configuration.

2) User Roles

Hyperledger Fabric Network Administrator – Responsible for monitoring Hyperledger Fabric network and its configuration as well as fabric plugin node monitoring and its configuration.

Hyperledger Sawtooth Network Administrator – Responsible for monitoring and managing Hyperledger Sawtooth network and its configuration as well as Sawtooth Plugin Node monitoring and its configuration.

Cactus Middleware Event Hub Tx Flow Configurator – Responsible for defining tx flow configuration in consultation with Fabric and Sawtooth network's smart contract developers in a specific application context.

Hyperledger Sawtooth Transaction Processor Developer – Responsible for Transaction processor development and deployment of the same on Sawtooth network in a specific application context.

Hyperledger Fabric Chaincode Developer – Responsible for smart contracts development and deployment of the same on Fabric network in a specific application context.

C. Connecting Blockchain Networks

The Interoperability solution has been developed for NBF using upgraded Cactus framework mechanism.

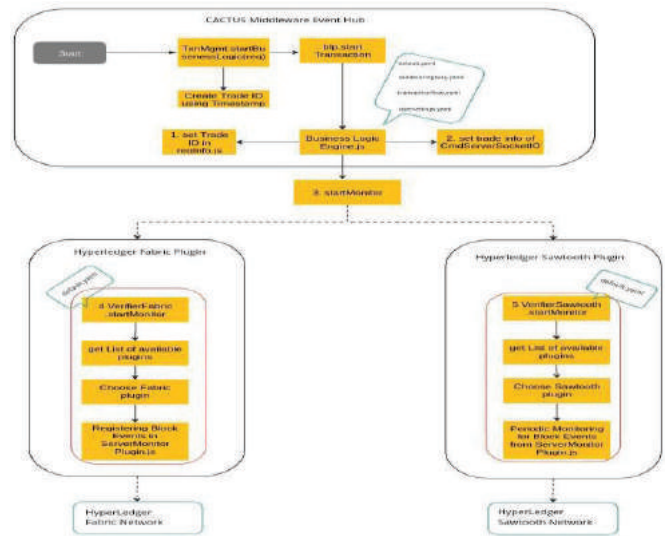


Figure 3: Bootstrapping of Fabric and Sawtooth along with Cactus middleware hub

Figure 4., below depicts the initiation of client request from respective Blockchain platform network and their request processing paradigm.

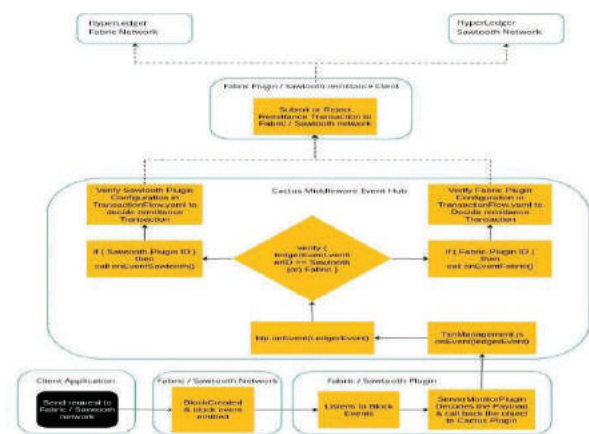


Figure 4. Client request handling by Blockchain networks

D. Understanding Tx Flow Configuration between Blockchain platforms

The transaction-flow.yaml file contains various sections that is defined by the Tx-Flow-Configurator in the following manner.

The below section describes about various transaction flows that can be defined between different blockchain networks

```
Transaction Flows #this section defines tx flows specific to a different application contexts.
- Application Tx Flow Name (1) # name of the tx flow of an application context
  o PluginID [Sawtooth/Fabric] #Multiple plugin sections can be defined
    ▪ action name #name that indicates which action has occurred
    ▪ <action name 1>
      ▪ cond # determines flow control conditions
        o <1>
          ▪ forward: true/false
          ▪ targetblockchain: <fabric/sawtooth>
          ▪ remittance action #name of action to be initiated on target blockchain
          ▪ remittance params #parameters to be passed
          ▪ toPluginID #target blockchain's plugin id
        o <2>
          ▪ Second flow control section starts here
    ▪ <action name 2>
```

This section contains details about name of the blockchain networks, application (smart contracts) context and transaction flow to be followed

```
fabric #name of the blockchain network
  <PluginID name1>
    <Channel name> # channel name in the fabric network to which the application
    is bound
    <Chaincodes> #application context
    <Tx flow> #transaction flow to be used
  <PluginID name2>
  .....
sawtooth #name of the blockchain network
  <PluginID name1>
    <Tx Processor Name1> #Name of the transaction processor
    <Tx Flow> # transaction flow to be used
  <PluginID name2>
```

In order to support Interoperability between Hyperledger Fabric and Hyperledger Sawtooth networks (please be noted that the solution supports both homogenous and heterogenous network combinations), there is a need to upgrade existing chaincode or transaction processors (a.k.a smart contracts) in the respective networks. Hence, the smart contract developers have to stick to few suggestions as mentioned below.

At present the solution has been tested with extended fabric's chaincode version. As per the Hyperledger Fabric's chaincode syntactical requirements, the chaincode functions can be invoked by passing desired number of request parameters from the client application. However, the chaincode function signature does not allow passing of variable request parameters.

If we look at the transaction-flow.yaml directives mentioned above there is a need for passing additional parameters other than the one submitted by the client applications. Since the present chaincode function signature does not permit or useful in fulfilling this requirement there is a need for upgrading existing chaincodes.

The chaincodes have to be upgraded in such a manner that the function signature in it accepts the JSON object instead of individual parameters. The JSON object format is as follows and the client applications have to pass the JSON object in the prescribed format only.

```
obj:= {
  requestparam : {key1:val1, key2:val2,...},
  dbobj: {dbserver: 'mongodb', data: {<database reply
in json format>}}
  nextSeq: <number [1...]>
}
```

In the above object format, nextSeq property is optional. Usually when the client sends his/her request while initiating transaction, this property would not be included. The chaincode developer need not handle presence or absence of this property in the chaincode functions. However, the chaincode developer MUST handle the other two properties viz., requestparam and dbobj in the chaincode functions. All functions in chaincode MUST adhere to above requirement in order to support Interoperability.

III. USE CASE SCENARIO

This section describes about a fictitious use case scenario related to electricity units consumption and bill generation by two different blockchain networks.

A. Application Context - Electricity Trade

In this use case, the fabric network is used for recording client requests about consumed electricity units and corresponding bill details on its side and the sawtooth network is used for recording client requests about electricity units' consumption details on its side. Here, for fabric network client may be end users who is consuming the electricity for domestic or commercial purpose whereas client w.r.t sawtooth network could be the electricity distribution agency.

Fabric clients and Sawtooth clients can hit their respective blockchain networks any time. However, the system should confirm the client request on both networks to meet the specific purpose of information maintained at respective networks.

B. Transaction Initiation

The sequence of events that occurs when client applications initiate transactions on either side of blockchain networks would be as follows.

Scenario 1: The electricity distributing agency has sent its request with details such as meter id, consumed units to the sawtooth blockchain network.

Step 1: Client sends his/her request to the sawtooth network.

Step 2: Sawtooth network receives the request from the client, processes it and confirms the transaction on its side.

Step 3: The sawtooth network generates a block event comprising details about confirmed transaction on its side which comprises data about meter id and consumed units information.

Step 4: The block event will be captured by the sawtooth plugin node that is associated with the sawtooth network, which in turn, read out each transaction information from the block event, and sends the transaction particulars to the cactus middleware event hub node.

Step 5: The cactus middleware event hub node upon receiving transaction particulars from the sawtooth plugin, read out the tx flow information from its configuration settings and decides upon whether the request has to be further forwarded to the fabric network or not.

Step 6: In case, the tx flow definition that is available with the cactus middleware event hub finds that the tx has to be forwarded to the fabric network, the cactus middleware event hub forwards the tx information unaltered to the fabric plugin node which in turn send the information to the fabric network to which it is associated.

Step 7: The fabric network upon receiving the tx information, processes it and generates a chaincode event to fetch the unit cost from external data source. The chaincode event will be captured by the fabric plugin node. The chaincode event contains tx data such as meter id, consumed units and data source name that needs to be contacted for further information.

Step 8: The fabric plugin upon receiving the chaincode event finds information about external data source that needs to be contacted for fetching unit cost of electricity. It then connects to the external data source (for example, mongo db) and fetches the required details. The collection name, data source url, user authentication particulars everything can be fetched from its default.yaml file.

Step 9: Upon receiving the data from the external data source, the fabric plugin sends information to the cactus middleware event hub node. The event hub node determines whether the information has to be sent back to the fabric network based on the defined settings in tx flow file.

Step 10: If cactus middleware event hub node finds it that the information has to be sent back to the fabric node from the tx flow configuration file, it forwards the request to the fabric plugin node which in turn forwards the same to its associated fabric network.

Step 11: The fabric network upon receiving the information from its fabric plugin node, processes it, compute the bill based on unit cost information received and confirms the transaction on its side.

Step 12: The fabric plugin node that is associated with the fabric network receives block events with transaction details that are confirmed by the fabric network. The fabric plugin node, then verifies each and every transaction that is found in the block event and forwards the tx particulars to the cactus middleware event hub node.

Step 13: The cactus middleware event hub node decides whether to end the transaction or next course of action based on tx flow configuration settings. In this scenario, it ends further transaction processing as the objective of recording transaction particulars on both the networks is achieved.

Scenario 2: The electricity distribution agency has sent its request with details such as meter id, consumed units to the fabric blockchain network.

All the steps defined in Scenario 1 will be followed in the same manner whenever the client sends its request to the fabric network initially. However, in this case, the transaction will not be confirmed on fabric network initially as it needs to fetch the unit cost before confirming it on its side. Once it receives the unit cost from the external data source i.e., mongo db server, it confirms it on its side and send the transaction particulars to the sawtooth network through cactus middleware event hub.

Scenario 3: Both scenario 1 & 2 have defined immediate transaction confirmation-based model, meaning, in both cases the transaction particulars are forwarded to another block chain network only after they are confirmed on respective blockchain network first.

In scenario 3, we shall brief the steps when deferred tx confirmation approach has to be followed assuming that client had sent his/her request to the Hyperledger Fabric Network initially.

Step 1: Client sends his/her request containing consumed units and meter id to the Hyperledger fabric network.

Step 2: Fabric network sends a chaincode event to the Fabric plugin node along with units consumed and meter id and data source name. In this case, the tx is not yet confirmed on peer ledger.

Step 3: Cactus middleware event hub accesses the tx flow settings, finds that it has to send the information to the Sawtooth network.

Step 4: Sawtooth plugin receives data from cactus middleware event hub node and forwards the same to the associated sawtooth network.

Step 5: Sawtooth network records the transaction event on its network and sends block event about confirmed transaction to the associated sawtooth plugin node.

Step 6: Sawtooth plugin node sends the information received from its associated sawtooth network in turn to the cactus middleware event hub node.

Step 7: Cactus middleware event hub finds from the Tx flow settings that it has to forward the confirmed Tx details to the fabric network through the associated fabric plugin node.

Step 8: Fabric plugin receives the unit cost from the mongo db server and forwards the transaction data along with unit cost to the cactus middleware event hub node.

Step 9: Cactus middleware event hub finds from the tx flow settings that it has to forward the confirmed tx details to the fabric network through the associated fabric plugin node.

Step 10: Fabric network upon receiving data from the associated fabric plugin node process it by generating the bill and confirms the transaction on its side. The fabric plugin will get the block event about the confirmed transaction subsequently from its associated fabric network and forwards the tx detail to the cactus middleware event hub node. At this point of time, the cactus middleware event hub node completes transaction processing when it sees that there is no next action to be performed from the tx flow settings file.

Fig 5, 6 and 7 demonstrate how to configure various parameters that help fabric and sawtooth plugins interconnect with their respective blockchain networks as well as with the cactus middleware hub.

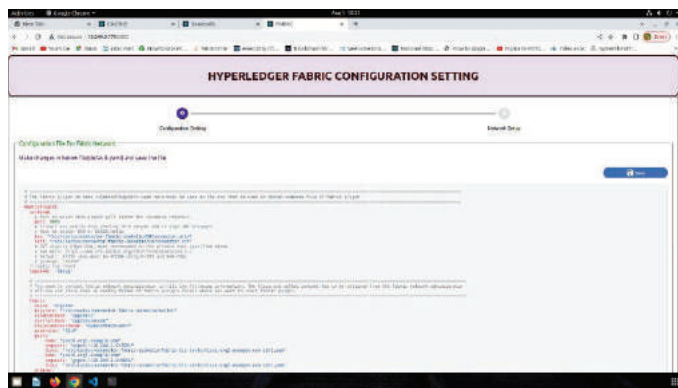


Figure 5: Hyperledger Fabric Plugin Configuration Settings

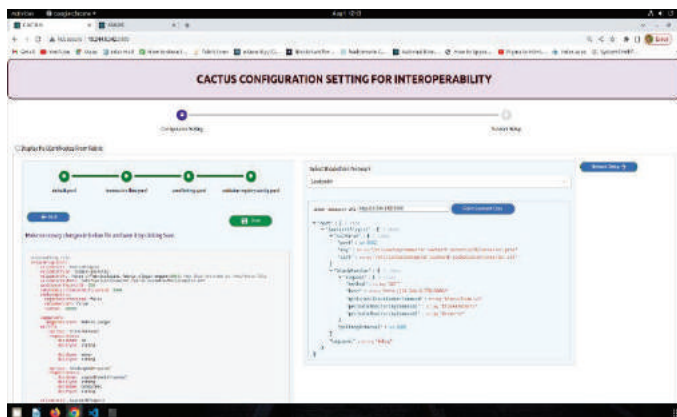


Figure 6: Sawtooth Plugin Configuration

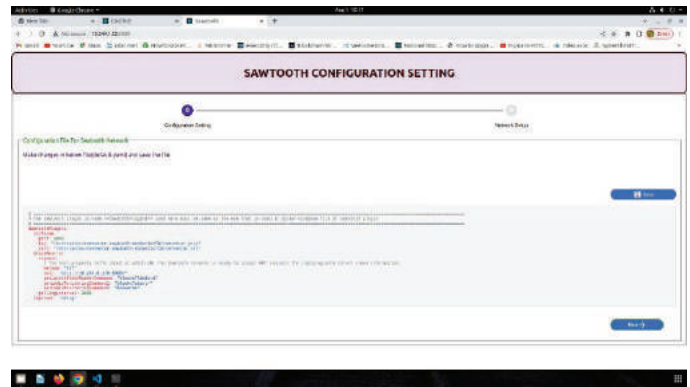


Figure 7: Cactus Plugin Configuration

The plugin configurations can be done respective administrators. The cactus plugin information include tx flow configuration also as it needs to take care of routing decision about incoming transaction events from the respective blockchain network

CONCLUSION

In this paper, we highlighted research approaches that are proposed by the researchers and explained about how the Blockchain interoperability solution is supported across multiple blockchain platforms. In our work, we focused more upon intricacies related to the implementation of these approached by creating a layer of transaction flow mechanism as a configurable option based on which cactus middleware takes a decision about how to deal with transactions across the platforms. The proposed architecture has been tested in the LAN environment using sample use case scenarios.

ACKNOWLEDGMENT

The work in this paper has been carried out as part of the National Blockchain Framework initiative supported by the Ministry of Electronics and Information Technology, Government of India.

REFERENCES

- [1] Belchior, R., Vasconcelos, A., Guerreiro, S., & Correia, M. (2020). A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *ArXiv*. /abs/2005.14282
- [2] Monika and R. Bhatia, "Interoperability Solutions for Blockchain," 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE), Bengaluru, India, 2020, pp. 381-385, doi: 10.1109/ICSTCEE49637.2020.9277054.
- [3] S. Zhu, C. Chi and Y. Liu, "A study on the challenges and solutions of blockchain interoperability," in *China Communications*, vol. 20, no. 6, pp. 148-165, June 2023, doi: 10.23919/JCC.2023.00.026.
- [4] Hyperledger Cactus Documentation, Retrieved from "Welcome to Hyperledger Cactus documentation! — Hyperledger Cactus 2.0.0-alpha.2 documentation (hyperledger-cactus.readthedocs.io)" on 23-11-2023
- [5] Hyperledger Transaction Flow, Retrieved from "Transaction Flow — hyperledger-fabricdocs main documentation" on 23-11-2023.
- [6] Creating a Hyperledger Sawtooth Network, Retrieved on "Creating a Sawtooth Test Network (hyperledger.org)" on 23-11-2023.
- [7] Polkadot Blockchain Network, Retrieved from [Polkadot: Web3 Interoperability | Decentralized Blockchain](https://polkadot.network/) on 23-11-2023