Peer to Peer Microlending

A Charitable Donation Management Platform on Blockchain

by

Sourabh Siddharth

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2020 by the
Graduate Supervisory Committee:

Dragan Boscovic, Co-Chair
Srividya Bansal, Co-Chair
Javier Gonzalez Sanchez

ARIZONA STATE UNIVERSITY

August 2020

ABSTRACT

Microlending aims at providing low-barrier loans to small to medium scaled family run businesses that are financially disincluded historically. These borrowers might be in third world countries where traditional financing is not accessible. Lenders can be individual investors or institutions making risky investments or willing to help people who cannot access traditional banks or do not have the credibility to get loans from traditional sources. Microlending involves a charitable cause as well where lenders are not really concerned about what and how they are paid.

This thesis aims at building a platform that will support both commercial microlending as well as charitable donation to support the real cause of microlending. The platform is expected to ensure privacy and transparency to the users in order to attract more users to use the system. Microlending involves monetary transactions, hence possible security threats to the system are discussed.

Blockchain is one of the technologies which has revolutionized financial transactions and microlending involves monetary transactions. Therefore, blockchain is viable option for microlending platform. Permissioned blockchain restricts the user admission to the platform and provides with identity management feature. This feature is required to ensure the security and privacy of various types of participants on the microlending platform.

ACKNOWLEDGMENTS

I express my sincere gratitude towards Dr. Dragan Boscovic who gave me the opportunity to work under him and continuously guided me throughout my entire research work. He continuously supported me and also provided me with knowledge that would be helpful to me in the real-world industry. His experience and knowledge have been the helping hands all through my thesis.

I would like to thank Dr. Srividya Bansal for agreeing to co-chair my thesis and taking the time to provide me with constructive feedback for writing my thesis. I would also like to thank Dr. Javier Gonzalez Sanchez for agreeing to be a member of my thesis committee and reviewing my work and providing me with valuable feedback for successful completion of my thesis.

I would like to thank Sasa Pesic from Vizlore LLC for helping me with understanding ChainRider and providing continuous hands-on support with the tool.

I would also like to thank the members of the Blockchain Research Lab at ASU especially Raj Sadaye for helping me get started with hyperledger fabric development, Kaushal Mhalgi and Manish Vishnoi for supporting me throughout the research.

Lastly, I would like to thank my family and friends who have been a constant support for me throughout the course work and were my pillars of support through thick and thin.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Introduction to Blockchain

Blockchain is a chain of immutable records of transactions. Blocks store the information about the transaction and also store a hash code to distinguish each block from another. With the help of blocks, a ledger of transactions is created. This ledger of transactions is not centrally stored but distributed among all the participating entities. These entities run and maintain physical nodes for the blockchain and have copies of the ledger. The integrity and reliability of this distributed ledger is ensured via standard consensus protocols. Since the data is distributed, it makes this technology highly secure and scalable. [1]

Blockchain consists of 3 major components:

- **Blocks**: Every chain consists of blocks. The data is stored in the blocks. A 32-bit whole number is called nonce is generated when a new block is created. A 256- bit hash is also combined with the nonce. Hash helps to distinguish the blocks.
- **Miners**: Miners create new blocks on the chain. For mining new blocks i.e. adding new information to the ledger, there has to be consensus from all other miners as well. Every block contains the reference to the hash of the previous block. So, for mining new block this information is important. Making any changes to a block not only requires remining the block but also all the blocks after it. Hence block chain is considered to be very secure.

- **Nodes**: Blockchain is distributed via the nodes in the chain. Every node has a copy of the blockchain. Since every node has its own copy of the data, if any changes are made all other nodes will be notified making it impossible to change any information. This enables immutability in blockchain. Each node validates each of the transactions and only if the new transaction is verified it is added to the chain.

Figure 1 gives an overview of the flow of a transaction in any blockchain application.

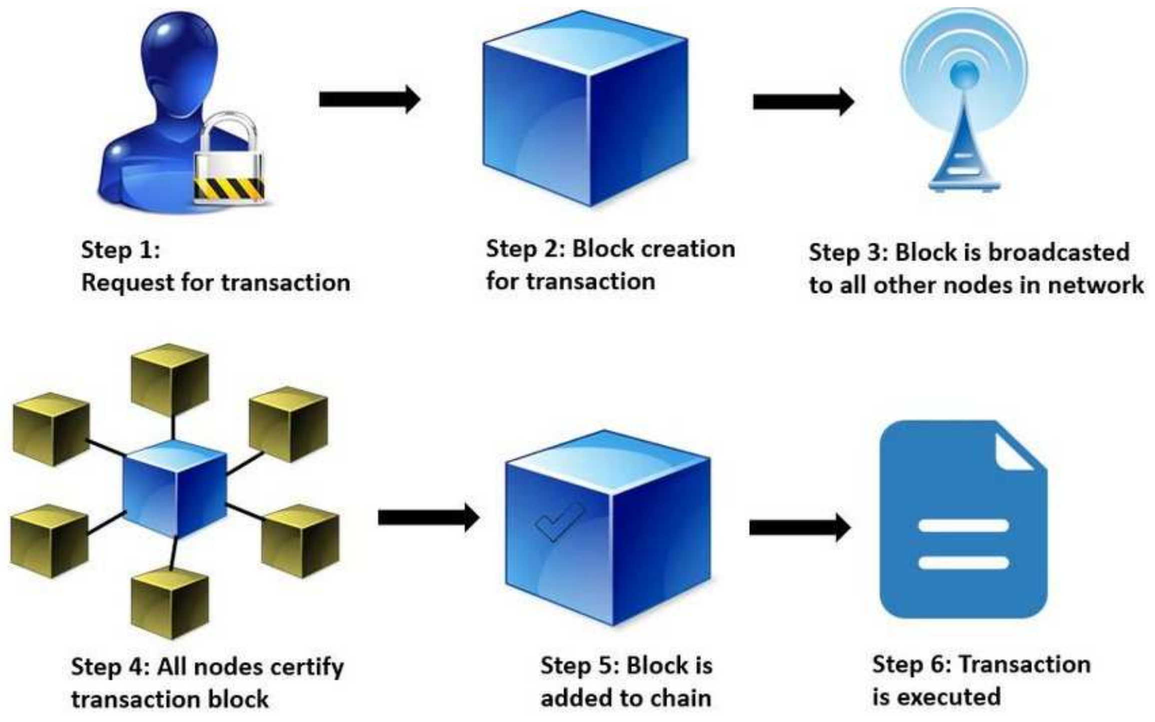


Figure 1. Transaction Flow in Blockchain. *Source:* Nadeem, Sara & Rizwan, Muhammad & Manzoor, Jaweria & Ahamd, Fahad. (2019). Securing Cognitive Radio Vehicular Ad Hoc Network with Fog Node based Distributed Blockchain Cloud Architecture. International Journal of Advanced Computer Science and Applications. 10. 10.14569/IJACSA.2019.0100138.

1.2 Types of Blockchain

Public Blockchain: A public blockchain network is completely open and anyone can join the network. There are no permissions or roles involved. Since there are no permissions involved, the data is accessible by all the participants. No particular participant has control over the data. It is completely decentralized. There are a couple of issues with the public blockchain. Since the network is open, the computational power required to manage the ledger is huge. Every node has to solve a complex, computationally intensive problem such as proof of work (PoW) to join the network. Also, there is no privacy in transactions and hence it is considered to be less secure. Bitcoin [4] and Ethereum [5] are examples of public blockchain. [2]

Private Blockchain: A private blockchain is also known as permissioned blockchain. In this we can set up access control and restrict who can participate in the network. Participants need to have an invitation or permission to join the network. There is some central management involved in private blockchain. Only the participants involved in a transaction are aware about the details of the transaction. It is useful for commercial businesses where it is important to keep the transaction information private. It is not completely decentralized but is considered to ensure more privacy and security. Since the number of participants in the network is limited, it is more scalable. Hyperledger fabric [6] is an example of a private blockchain. [2]

1.3 Introduction to Microlending

Microlending is a type of financing that provides small amounts of money to very poor fledgling entrepreneurs to encourage self-sufficiency and to end destitution, particularly in developing countries. It is similar to small business loans, but also different based on the motivation behind it, size of the loan and the people involved. Customary lenders exclusively center on procuring a benefit by charging interest. Microlenders have a greater amount of enthusiasm for improvement. Some of them unquestionably want to make profit, but the fundamental objective is to help create organizations for small entrepreneurs who might not have the available opportunity.

Microlending began in 1974 by one man, Muhammed Yunus, who gave a small loan to a Bangladeshi lady. That lady used the money to make and sell bamboo stools to take care of her family. The idea then spread globally. Numerous banks and non-bank institutions presently offer small scale loaning services. It has reformed aid efforts in third-world countries. [3]

Microlending in its purity is viewed as a charitable cause. The lenders don’t concern themselves with the money being repaid. The primary motivation is the opportunity to help people who want to work hard but don’t have access to affordable business capital. But in the present world, microlending has also become a business idea. Some individuals consider it as a means to earn more money by charging high interest rates from the borrowers.

1.4 Problems with Existing Microlending Platforms

With the existing microlending platforms most people take loan to fulfill their basic necessities and cannot repay the loan later due to missing sources of returns. Microlending is criticized for creating debt by complex schemes for poor borrowers. The loan amount is small and the overhead to manage the process is huge. Trust requirements and corruption represent numerous issues in the present lending scenario. Below mentioned are some of the issues associated with existing microlending platforms:

- No transparency of information and transactions involved in the lending process
- Missing legit donors who support the cause of microlending
- High interest rate imposed by donors
- No verifiability of information provided by the borrowers or the lenders making the process prone to corruption
- Vulnerable to security threats

1.5 Why Private Blockchain is Useful for Microlending Application?

Different types of users are a part of a microlending application, and they have different roles. Various transactions are involved in the lending process. We want to build an application which can support the following requirements:

- Identify and manage different user roles such as borrowers, donors etc.
- Maintain the privacy of data being shared with different users
- Provide transparency of transactions to the users

- Security of data

Private/Permissioned blockchain provide us with features which will helps in achieving these requirements. Following an invite only approach, only approved users can join the network. This helps in determining the identity of the users. Separate rules for transactions can be defined based on the user roles. Borrowers and Donors will have separate rules governing their transactions. Self-executing contracts which is the terms and conditions for a transaction to take place between the users are stored across the blockchain network in the form of lines of code. These are known as smart contracts, one of the critical components of blockchain technology. Smart contracts allow trusted transactions to be carried out between the participants. It works on the principle IFTTT i.e. if this then that. If the terms and conditions are met, then the transaction is completed. The smart contracts cannot be changed or deployed without all the participants of the contract being aware about it. We can also limit the amount of information given to the user based on their roles. This helps in ensuring privacy.

Each of the participants in the blockchain network has accessibility to the blocks of information. These blocks contain all the transactional information. With the help of this donors will always have an update about how and where their money is being used. Traceability of transactions is possible by using the hash address of the block.

Having the ability to add users on demand basis restricts the number of users that can use the network, improving the efficiency of transactions. Hyperledger Fabric is a permissioned blockchain framework which we will be using for our application. Hyperledger Fabric is explained in detail in the section 1.6.

1.6 Introduction to Hyperledger Fabric

Hyperledger Fabric is an open source permissioned framework used to develop enterprise-grade blockchain applications. It has a modular and highly configurable architecture. It follows plug and play components architecture which enables the platform to be customized effectively and support the business needs of different applications. The charitable microlending application is designed using this technology. [6]

The various features of hyperledger fabric are:

Modularity: Hyperledger fabric has a modular architecture. It ensures separation of transaction processing into three different phases: [7]

- Distributed logic processing and ordering of system
- Transaction ordering
- Transaction validation and commitment

This separation improves the network scalability and helps in increasing the overall performance of the network.

Privacy and Confidentiality: Hyperledger fabric supports privacy with the help of channels. Participants in the fabric network can establish channels between subsets of participants. The visibility of transactions can be restricted using these channels. Only the participants involved in the channel will have access to the smart contract and the data involved in the transaction. [8]

Performance and Scalability: Hyperledger being a permissioned blockchain framework, it is easier to restrict the number of participants involved in the network which helps in better performance of the network. As discussed above, the transaction processing is split into phases which ensure fewer levels of trust and validation across the nodes reducing any overhead cost. [9]

1.7 Components of Hyperledger Fabric

Organization: Organizations are basically the participating members in the network. Organizations are invited to join the blockchain network. The membership service provider (MSP) manages the members of the organization. A collection of organizations is known as consortium. [10]

Membership Service Provider: This component defines the rules by which members are allowed access to the network. It provides the credentials, manages the user ids and authenticates the users as well. Clients use the credentials for authentication of transactions and peers use for authentication of transaction processing results. [11]

Ordering Service: All the transactions are written on the shared ledger in a consistent manner. The order of transactions has to be established for ensuring that only valid transactions are entered into the ledger and bad transactions can be rejected by the network. [12] Hyperledger fabric allows choosing the organization mechanism which is most suited for the network. Three different ordering mechanisms are provided by HLF which are SOLO, Kafka, and Simplified Byzantine Fault Tolerance (SBFT).

- **SOLO** - Transactions are ordered in chronological order. It involves a single ordering node.
- **Kafka** - This uses apache kafka. It provides a crash fault tolerant ordering service.

- **Simplified Byzantine Fault Tolerance** - This ordering mechanism is both crash fault-tolerant and byzantine fault-tolerant. [13]

Peers: A peer is a type of node. It maintains the ledger and also performs the read/write operations on the ledger. Organizations interact via the peers hosted on their systems. Peers can also have a special role i.e. the endorsing peer. They execute the transactions and return responses to the client application. Endorsing peers are specified in the endorsement policies corresponding to the chaincode implementation. [14]

Clients: Clients are the nodes that submit the transaction invocation to the endorsers, and also broadcast transaction-proposals to ordering service. They communicate with both the peer and ordering service.

Channels: Channels help in partitioning the network to enable data insolation and confidentiality. They segregate interactions amongst organizations in the network. Each channel has its own ledger space and hosted smart contracts that operate on its ledger. This ensures business rules and data access limitations across channels. Channels can only get a read only access to other channel's ledger via smart contracts with proper credentials. Same chaincode logic can be used by multiple channels and also single user can participate in multiple channels. [15]

Chiancode (Smart Contracts): In hyperledger fabric ecosystem, smart contracts are known as chaincodes. Chaincode provides the business logic for the hyperledger network. It is written using either one of these programming languages Go, NodeJS or Java. It is installed on the peer nodes and instantiated onto one or more channels. It contains the business rules for the application with consent from the organizations involved. Chaincode interacts with the world state and updates it if the transactions are valid. [16]

Ledger: Ledger stores the information about the validated transactions. The information is stored in the form of key value pairs. This information is immutable, once added to the ledger it cannot be modified. Each peer has its own copy of the ledger. [17]

1.8 Transaction Flow and Ordering in Hyperledger Fabric

The transaction process is divided into below 3 phases:

1. **Transaction proposal:** In this step the client application sends a transaction proposal to the endorsing peers. This proposal consists of the client id, payload, timestamp, and client signature. Each endorsing peer then executes an individual chaincode on the proposal and generates a proposal response. The transaction is not written in the ledger yet. It is returned to the application along with the response.
2. **Ordering and block formation:** All the transaction proposal responses from phase1 are sent to the ordering service. The ordering service is responsible for packaging the responses into blocks. The blocks are stored on the orderer's ledger and also distributed among the peers of the channel. The transactions are put in a strict order making hyperledger fabric's blocks to be final. Peers will use these blocks in the next phase to validate and commit.

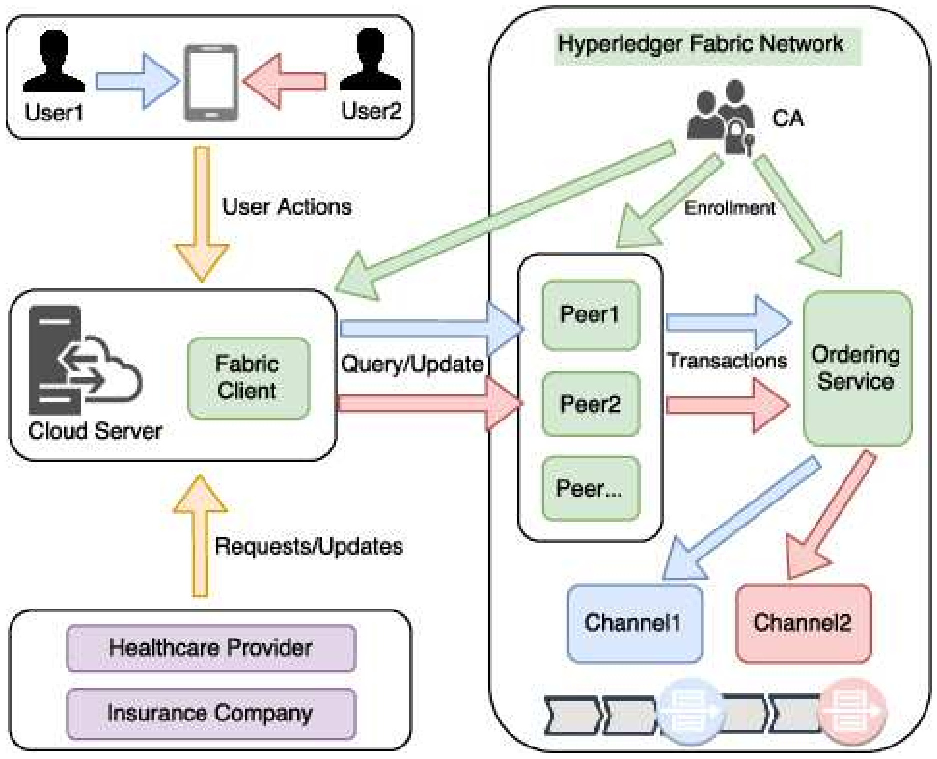


Figure 2 Example of component interaction for transaction execution. *Source:* Liang, Xueping, Juan Zhao, Sachin Shetty, Jihong Liu and Danyi Li. "Integrating blockchain for data sharing and collaboration in mobile healthcare applications." 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC) (2017): 1-5.

3. **Validate and commit:** The orderer distributes the blocks generated in phase2 to all the peers connected to it. Each peer validates the blocks independently maintaining the consistency of the ledger as well. The peer validates that the transactions are endorsed by the required organizations using the endorsement policy defined in the chaincode. If the transaction is valid, then the peer will try writing it to the ledger. Once written to the ledger, the transaction is final.

1.9 Hypothesis

The motivation behind traditional microlending is to support the financially deprived. But these days microlending has become more of a business venture. With the advancement in technology and blockchain there is an opportunity to revisit economies of microlending by combining charitable and commercial financing. Hence, is it possible to develop a platform that will support both commercial microlending as well as charitable donation to support the real cause of microlending. The model proposed and implemented supports this capability. Both individuals as well as financial institutions can participate in the platform. The objective of the model is to:

1. Increase the efficiency of the funding process by reducing the time taken for borrowers to receive funds.
2. Enhancing the transparency of how funds are disbursed along with providing visibility of project progress and how the funds are being utilized.
3. Ensuring privacy of information shared by the participants.
4. Identifying potential security threats and ensuring that our model addresses those threats.

1.10 Organization of Thesis

The remaining section of the thesis is organized as follows:

Chapter 2 contains the related work, problems with existing technologies and our solution to overcome those problems. Chapter 3 describes the design and the implementation of the application using hyperledger fabric. It explains how the design supports the objectives of our platform. It describes the various components in our system. Different scenarios to explain the flow of transactions is also stated in this chapter. It also discusses about ChainRider[20], an important service used for the blockchain network setup. Chapter 4 explains the experimental setup of the platform and then discusses the results for transparency and privacy. It then explains a detailed survey of security analysis for our system. Chapter 5 concludes the thesis along with mentioning some possible future work scenarios.

CHAPTER 2

RELATED WORK

2.1 Blockchain for Financial Transactions

Blockchain has revolutionized the process of financial transactions. Blockchain increases the transparency of transactions and has helped in making systems much more secure. Since blockchain has a decentralized architecture, it eliminates the risk of a vulnerable centralized database. The immutable ledger makes it unhackable and hence it is difficult to fake a transaction which makes it safer.

In finance, all transactions have to follow some contract. These contracts define the rights and the duties involved. Traditional contracts written on paper are time consuming and can be forged. Smart contracts help in automating the process. With no intermediaries involved, transactions become faster and more reliable. All the transactions can be tracked and verified. Different parties involved in the business can validate the transactions. There is no centralized authority so there is no single point for failure. Banks are also able to reduce any transaction fees or any overhead costs since no middleman is involved. This also increases the efficiency of the system. The entire transactional process is automated reducing any delays. [18]

With so many benefits of this technology, IBM partnered with we.trade [19] to create a platform based on IBM Blockchain platform to provide trading solutions. It simplifies cross border trading and provides a platform to perform smooth international transactions. we.trade is a joint venture between twelve major European banks - CaixaBank, Deutsche

Bank, Erste Group, HSBC, KBC, Natixis, Nordea, Rabobank, Santander, Société Générale, UBS and UniCredit. Companies are brought to the platform by the banks for the purpose of trading which ensures trust between the users. Since the platform is based on blockchain, there is no single point of failure or a single controlling entity. All the relevant parties have the same information regarding the trade deals which is stored on the ledger. If some company attempts to make any changes to the terms and conditions of the trade, all other companies involved in the trade will be informed about the changes. Smart contracts ensure that if one company has fulfilled the requirements of a transaction then the payment process will be executed immediately. This avoids any delay of payment for the companies. These contracts are built into the system and hence no third-party legal systems are involved. [19]

The users of this platform are mostly big financial institutions who want to trade internationally. It is not meant for peer to peer transactions and no charitable cause is involved.

2.2 Peer to Peer Transfer Using Crypto Cash by Everex

Blockchain capital transfer using crypto cash: Everex, a financial technology company supports peer to peer money transfers, payments using blockchain. They use Ethereum blockchain technology to support financial transactions. Ethereum is an open source, decentralized, public blockchain software platform.

Everex supports financial inclusion of unbanked individuals using crypto cash. Crypto cash is a cryptocurrency, where each unit has its value pegged to, and a name based on, the fiat currency it represents. Users convert their local currency to crypto cash at their banks and then transfer this crypto cash using blockchain. This process supports cross border remittance, online payments and microlending as well. But all this is peer to peer. They use a credit scoring algorithm to determine the borrower's credit worthiness. Then based on the result of the algorithm borrower and lender negotiate the terms and conditions for the contract. On agreement the lender transfers the money to the borrower in the form of crypto cash. The loan should be returned in the form of cryptocurrency as well.

Their system consists of four major components, the Account management component, Everex services component, the Ethereum parser application and the database component. The account management component is responsible for user registration, account creation and crypto cash wallet setup. The Everex services component comprises all the services involved in the money transfer process. The lending API is responsible for the micro lending service. The Ethereum parser application is responsible for tracking all the crypto cash transactions involved in the lending process. The database component stores all the user data as well as the transaction data. [20]

One of the major issues with the Everex platform is that it is based on public blockchain. Any number of users can use the platform and there is no privacy involved in transactions. With the increase in number of users, the load on the network also increases reducing the efficiency of the network. Moreover, we cannot add any third-party institutions like lender institutions and law bodies while securely ensuring privacy of data. This platform is business oriented which does not support the moral cause behind microlending.

2.3 Platform for Microlending – Kiva

Numerous platforms have been developed for microlending offering some or other features. Kiva was among the first few platforms developed for microlending. Kiva supports only person to person lending. Kiva provides a platform where borrowers can share their story about why they need money and lenders browse through different borrower profiles to choose whom they want to lend money. Borrowers include various types of information in their profile such as: why they need money, how much amount they need and for how long. Kiva works with local organizations known as field partners to monitor the lending process. The field partners look out for borrowers who are in need of money. [21]

The unique feature of this platform is that the field partners disburse loans to the borrowers as soon as they find one. Most of the funds are disbursed even before the borrower's story is posted on Kiva's platform. Lenders browse through these stories and choose borrowers they want to support. They can lend as small as $25. All these small amounts are aggregated to backfill the already disbursed loan by the field partners. Field partners are responsible for collecting the loan repayments from the borrowers. They might also charge some interest from borrowers to cover their operation costs. The field partner then gives back this money to Kiva and Kiva in turn returns it to the lenders. All the transactions are done using wire transfers.

One of the major problems with this platform is transparency. Lenders are not aware where their money is actually going. The field partners provide the money to borrower, so it defies the idea of person to person lending. If the field partners are capable of lending the money to borrowers, then there is no actual requirement for a platform like Kiva. The loans are disbursed to the users even before there are donors available. There is no defined way for transactions to take place and hence making the platform not reliable.

2.4 Our Solution for Problems with Existing Applications

To solve the problems with existing applications we have tried to incorporate the features of blockchain to make a secure and reliable platform for microlending. Our application supports both charitable donation as well as commercial funding. The individual donors of our application are motivated to help the underprivileged borrowers without expecting any monetary returns. We are using hyperledger fabric to build our application which is a private blockchain framework. Being a permissioned framework, the users are added to the application only on invite basis. With the help of smart contracts, we can clearly define the set of rules and regulations for any transaction between the borrower and donor. The transactions are stored on the ledger and donors have the copy of this ledger. With the help of this ledger donors can track where their money is actually being invested providing complete transparency to the users. We are also using multiple channels in our platform for making the interaction between the actors much more secure. Only the actors who are a part of the channel will be aware about the assets and transactions on that channel. By incorporating the above-mentioned features our application accounts for the cause of microlending.

CHAPTER 3
DESIGN AND IMPLEMENTATION

A peer to peer approach

We introduce a peer to peer application where we have both individuals as well as financial institutions as lenders. The platform is operated by charity organizations who are motivated to extend the application's outreach to third world economically deprived small sized businesses. These NGOs should actively search and verify the identities of such entities and genuineness of their business models either through physical visits or sources like credit history and valid business registrations (if available). These verification steps are out of scope of our implementation and handled externally by the charity.

The individual donors who wish to participate in this charitable cause should be motivated to help the borrowers without expecting any monetary returns. They get to have a detailed view of a project plan and the loan requirements and accordingly decide to contribute. Their donations may provide the initial seed money to start a project. The repayments from the borrowers are then saved into a pool of funds managed by the charity which can be reused to loan other borrowers. We aim to provide these loans at no interest to the borrowers but a very small fraction of the loan as operational fees for the platform.

There might be times when the contributions by the donors are not sufficient to fund a project and even the pool of funds (donation pool) falls short of funds. Financial institutions such as banks come into play in such scenarios. The charity can enroll banks that can provide loans at preferably low to reasonable interest rates on separate channels. Channels are discussed in section 1.7. This ensures restricted access to user and loan data. The charity deducts an enrollment fee and small transaction fees per loan as its operation costs.

The charity may ensure transaction traceability for donors and banks by motivating the borrowers to actively update fund utilizations and progress on project milestones on the application platform. Donors and banks can have a view of the transactions on the project asset and trace their tokens by their hash addresses at any point of time. This provides them a sense of trust towards the charitable organization and the platform while gratifying their willingness of benevolence as they can see their contributions to provide essential resources to those in need. This in turn might motivate them to contribute more to the cause.

3.1 Components of our System

3.1.1 Actors

Monetary transactions are involved in our application. Hence, it is difficult to find legit borrowers or donors. We need to have some central authority governing the addition of the users to the system as wells as monitoring the transactions involved. I have used four different actors for the application.:

1. **Borrower**: The borrowers are the beneficiaries of our platform. They are the economically underserved small to medium scale family run businesses who propose their business plans to the charity in order to obtain micro loans. The provide the details of the project along with the project milestones to Charity.

2. **Charity:** Charity is the central authority of our platform. They add the all the other 3 actors to the platform. They provide the infrastructure to the borrowers and the donors to perform transaction in the platform. They govern the transactions of our platform. They are also responsible for maintaining a pool fund. This pool fund is used when premium donors vote to donate for a project.
3. **Donor:** Donors are the benevolent actors of the system who wish to donate to this social cause and are ensured that the effects of their efforts are visible. They contribute by evaluating the project proposal provided by the borrowers. There also exists a class of donors called premium donors. These are the donors who are given a special advantaged because of their good donation history. When normal donors refuse to fund a project, the premium donors are given an opportunity to vote and fund a project.
4. **Bank:** Banks are third party institutions of the platform. In case the donations fall short of the required loan amount, or no individual donors want to fund a project the bank can authorize a loan and provide funds to the borrower. The banks do charge some additional interest from the borrowers.

3.1.2 Assets in our system

We have the following assets in our application:

1. **Project**: The project asset records the basic information about the project and its loan needs. It is frequently updated in our system to record all the donation

transactions, repayment info and milestone progress. The key attributes of a project record are listed below:

- Project Id
- Project Name
- Fund available
- Fund deadline
- Fund required
- Owner
- State description
- Timeline
- Unutilized funds
- Vote Yes Count
- Voting deadline
- Loan Balance

2. **Pool:** Pool asset records the information and transactions on the fund pool. It also holds the records of individual donors and their transactions. The key attributes of a pool object are:

   - Pool Id
   - State Description
   - Donors
   - Fund Available
   - Owner

- Pool Records
- Premium Donors
- Unutilized Funds
- User Records

3. **Token**: Token represents real world currency on the platform. Registered users on this platform use token to donate and repay funds. Each token is associated with a user id. Only the owner of the token can make the transfer making it highly secure. The main elements of a token are listed as follows:
    - Balances
    - Name
    - Symbol
    - Total Supply
    - Value

3.1.3 Endorsement policies

Endorsement policies are a set of rules for a chaincode that allow peers from various organizations to endorse a transaction proposal. If an endorser peer from an organization is required by an endorsement policy of a chaincode, the peer needs to have that chaincode installed on it. The peer can then simulate the proposal and send the proposal to the orderer with its signature. In case a member with a malicious intent tries to inconsistently modify the state of an asset, the ordering node can identify the conflict in the Read-Write data set and reject the transaction proposal. We have set the endorsing policies to require one endorsement form each organization from that channel. Table 1 provides the details of endorsement policies for the chaincode in our system.

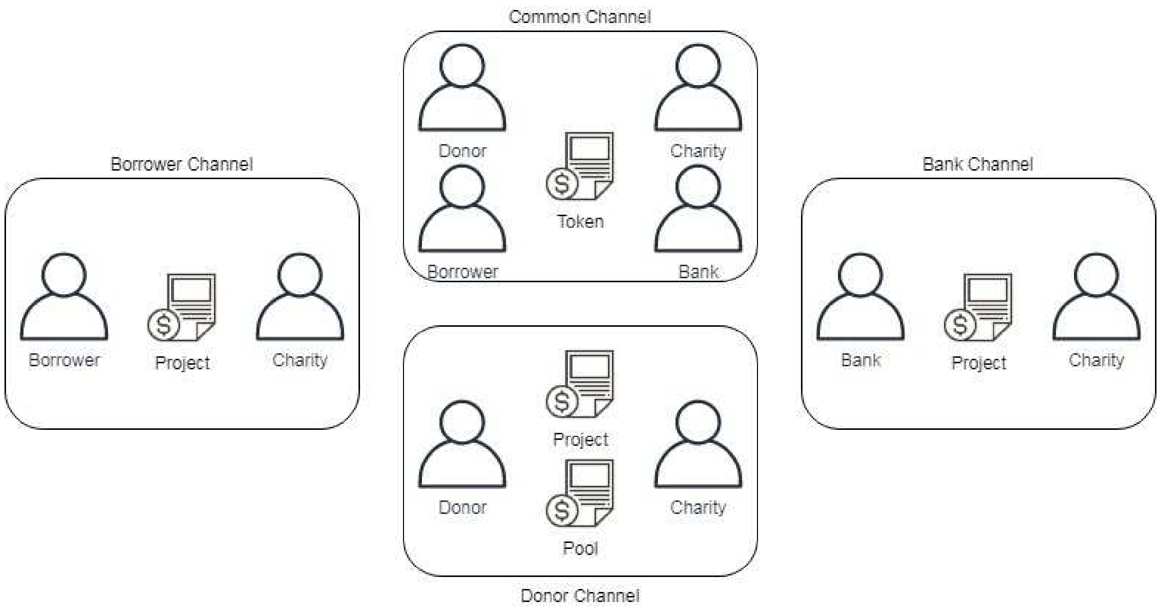


Figure 3 Channelization for organization and asset management

3.1.4 Channels

In an enterprise application like this, actors from different organizations often need to exchange and modify data while isolating data from other organizations. I have used 4 different channels in my platform. Charity being ad admin needs to interact with all other actors. Hence 3 channels one for each, Borrower, Donor and Bank. The common channel is used for monetary transactions. An overview of channel organization for the design is presented in figure 3. This makes sense in a peer to peer permissioned architecture. For example, a borrower doesn't need to access the information related to the fund pool in the donor channel. A channel provides a medium to secure asset states and regulate privacy of transactions by providing a local scope to each chaincode hosted on the channel. Only the members specified in the policy at the time of chaincode instantiation in a channel are authorized to endorse or sign transactions. At times, a transaction data needs to be propagated between members of two different channels, for e.g., a transaction where a bank loans fund to a project proposal, the borrower just needs to know the amount of funds loaned by bank and the rate of interest without knowing the state of the actual pool asset. In this case, the charity who has seen this transaction in a channel with the bank can update the filtered information on the charity-borrower channel. Thus, channels strengthen the confidentiality of data.

3.1.5 Smart Contracts (Chaincodes)

The smart contracts embed the business rules of transactions for assets on our microlending platform. The endorsing peers running a valid version of the chaincode can simulate the correct transaction which can be further validated by each individual peer by matching the signatures on the proposal against the policy before committing the proposal to their copy of the ledger. The endorsing peers belonging to each organization for a chaincode specified in table 1 should have a valid copy of the smart contract for the transactions to be recorded in a consistent manner on their ledgers. The smart contracts on our system are:

- **Borrower Contract**: Specifies transaction rules for borrowers to create a project and repay the loan to charity.

- **Pool Contract**: Donations and pool-based voting and fund cycle management [figure 3 below] must be securely transacted on the donor channel. Functionalities like user activity and project level activity management are essential for our transparency goals.
- **Bank Contract**: Banks are able to provide loans to charity and a set of project details are published for the banks to accept/reject a loan request.
- **Token Contract**: Tokens must be handled very securely amongst all the parties. This contract is critical in the sense that it specifies rules to manage tokens as it represents real world money.

Our setup has 4 channels, 4 members and 4 chaincodes deployed under certain policy restrictions as specified below:

Table1. Chaincode Policies for Actors for Endorsement of Transactions on assets

Channel	**Actors**	**Chaincode**	**Assets**	**Policy**
Bank Channel	Charity, Bank	Bank Contract	Project	AND (bankMSP.member, charityMSP.member)
Borrower Channel	Charity, Borrower	Borrower Contract	Project	AND (bankMSP.member, borrowerMSP.member)
Common Channel	Charity, Bank, Borrower, Donor	Token Contract	Token	AND (bankMSP.member, borrowerMSP.member, charityMSP.member, donorMSP.member)
Donor Channel	Charity, Donor	Pool Contract	Project, Pool	AND (charityMSP.member, donorMSP.member)

To restrict transactions on pool asset to donors and the charity, we have created a separate channel called the Donor Channel. Similarly, there is a Bank Channel to make transactions between the bank and the charity private. Only the actors who are a part of the channel are allowed by the policy to endorse the transactions on assets specified and mapped against the channel name in the table 1. The And() operator in the hyperledger fabric policy specification requires all the actors passed to it to endorse and sign a transaction during the transaction proposal phase mentioned in section 1.8.1.

3.2 Implementation

The objective of our application is to provide funding to the borrowers efficiently at low interest rates. Since the individual donors are not charging any interests from the borrowers, I have given them the opportunity to donate first in my implementation. If there are no donors available, then the project will be passed on to the banks for funding. There are two strict deadlines defined for any project:

Funding deadline: Individual donors have to donate before the funding deadline.

Voting deadline: Premium donors have to decide to vote in favor or against the project before the voting deadline.

If the total amount of fund is not available until these deadlines, then the project is passed on to the banks for funding. Banks will charge some interest for funding the project. We want to ensure that the borrowers are charged with minimal to no interest rates along with providing them funds within a limited time. All the projects will always reach an end state. It will either be funded or dropped by the application. Below section 3.2.2 explain in detail the flow of transactions in our application.

The implementation can be found on *https://github.com/sidd1811/P2PMicrolending.*

3.2.1 Transaction Flow for Assets

The state diagram Figure 4 describes the transactional states for the Project asset. The project asset is a key asset to establish transparency in our system as it holds all the information required to fund a project, its loan information, fund utilization data and progress report. The project asset contains all the information to attract a donor as well as keep them updated. Its end to end transaction flow based on different scenarios can be described as below:

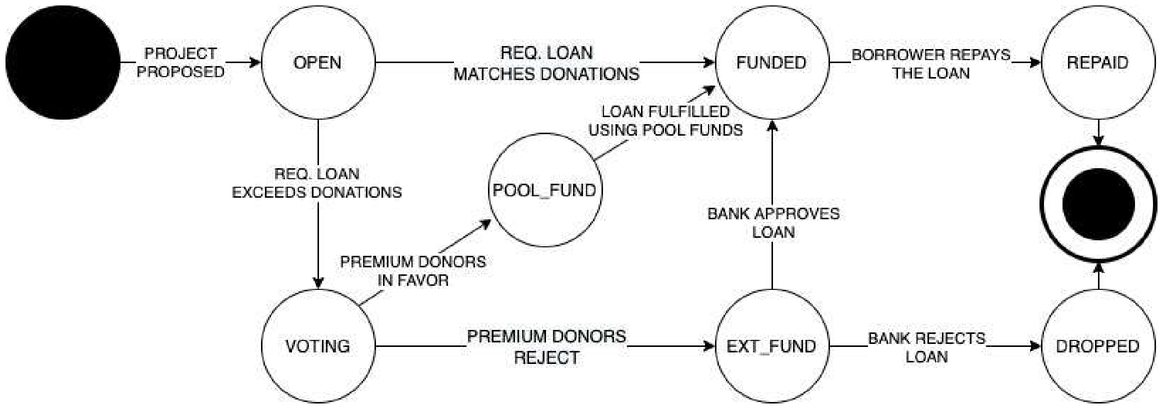


Figure 4 Transactional state diagram for project asset

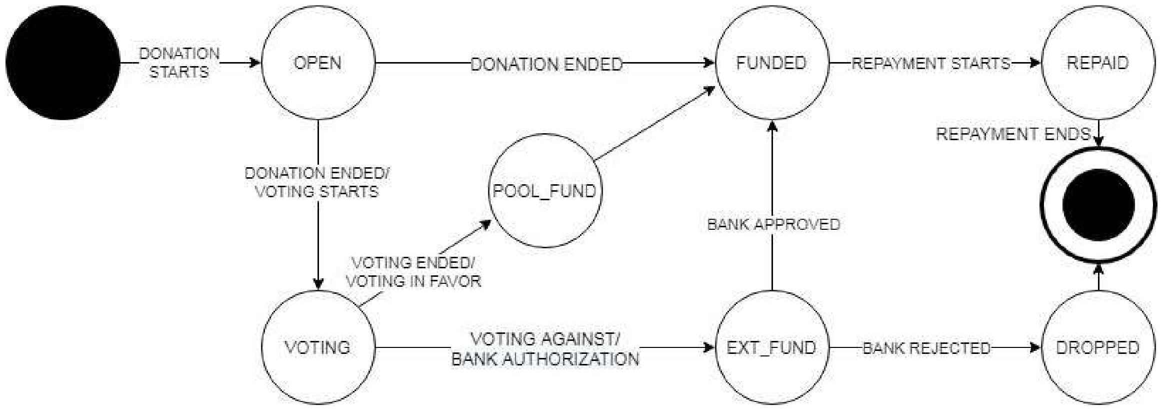


Figure 5 State diagram of cycles for project asset

A project is proposed and is set to ‘open’ state Figure 4, this marks the beginning of the donation cycle Figure 5. Once donation period ends Figure 5, there can be multiple scenarios; these have been described using subsets of Figure 4 in the state transition diagrams below:

Scenario 1: Donors contribute to fulfill loan requirements

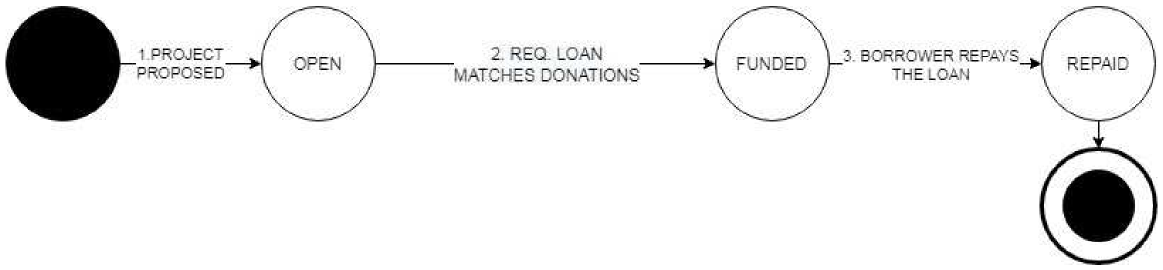


Figure 6 State diagram for scenario 1

1. The donors contribute enough funds that match the proposed loan amount and the project state is set to ‘funded’. As soon as the state is set, corresponding token transfers happen in the common channel and the repayment cycle starts.
2. The borrower pays off the tokens in the common channel and the status of the project is set to ‘repaid’. This is the final state for the project.

Scenario 2: Insufficient funds by donors, successful voting and fund pool contribution.

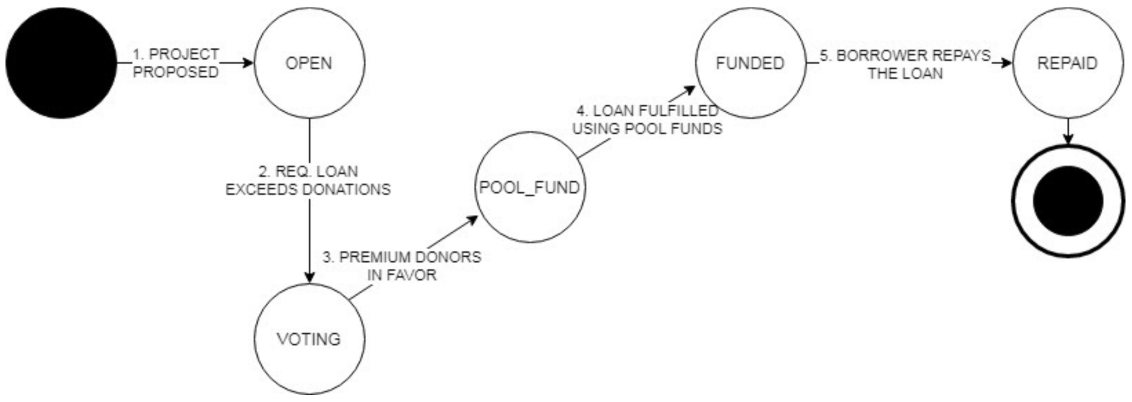

1. PROJECT PROPOSED
OPEN
2. REQ. LOAN EXCEEDS DONATIONS
VOTING
3. PREMIUM DONORS IN FAVOR
POOL_FUND
4. LOAN FULFILLED USING POOL FUNDS
FUNDED
5. BORROWER REPAYS THE LOAN
REPAID

Figure 7 State diagram for scenario 2

1. When the donations are not sufficient to match the loan asked, the voting phase starts as shown in Figure 5. State of the project asset is set to 'voting'. The premium donors of the pool vote against the project proposal.
2. The votes are evaluated at the end of the voting period as shown in Figure 5. If 3/4th of the number of premium members agrees, the proposal is accepted by the pool and the project state is set to 'pool_fund'.
3. Enough pool funds are consumed to suffice the remaining loan amount and the related token transactions are made in the common channel. The state of the project is now 'funded'.
4. The borrower pays off the tokens in the common channel and the status of the project is set to 'repaid'. This is the final state for the project.

Scenario 3: Funds donated by donors not enough, premium donors of pool reject proposal and loan request forwarded to bank.

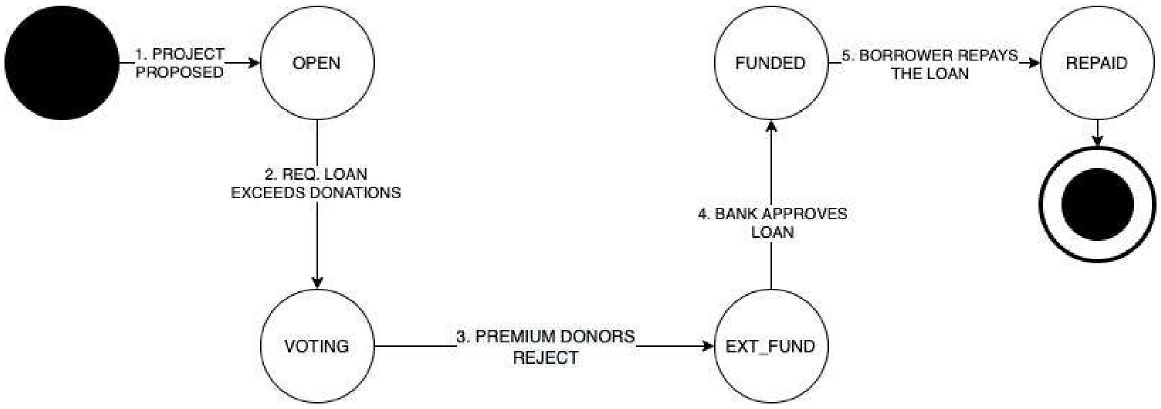


Figure 8 State diagram for scenario 3

1. When the donations are not sufficient to match the loan asked, the voting phase starts as in Figure 5. State of the project asset is set to 'voting'. The premium donors of the pool vote against the project proposal.

2. The votes are evaluated at the end of the voting period as shown in Figure5. If more than 1/4th of the number of premium members disagrees, the proposal is rejected by the pool and the project state is set to ‘ext_fund’.
3. Bank approves the loan proposal and transfers the corresponding tokens for the required loan amount in the common channel.

Scenario 4: Same as scenario 3, except the bank rejects project proposal

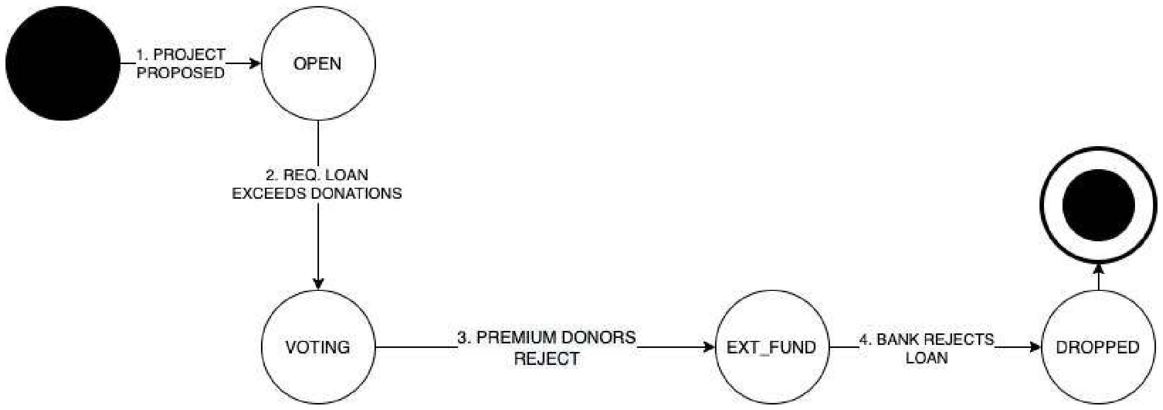


Figure 9 State diagram for scenario 4

1. When the donations are not sufficient to match the loan asked, the voting phase starts as shown in Figure 5. State of the project asset is set to ‘voting’. The premium donors of the pool vote against the project proposal.
2. The votes are evaluated at the end of the voting period as shown in Figure5. If more than 1/4th of the number of premium members disagrees, the proposal is rejected by the pool and the project state is set to ‘ext_fund’.
3. Bank rejects the loan proposal and the project state is set to ‘dropped’ and any donor token contributions are reverted in the common channel.

3.3 Framework Used

ChainRider: ChainRider [22] an online service built upon Hyperledger Framework. It offers different services to build blockchain applications quickly. We are using ChainRider to generate skeleton blockchain network. It provides the options to choose the modules necessary to run a hyperledger network on cloud such as orderer service, certificate authority, MSPs, skeleton rest API and blockchain explorer. We design and provide the necessary network configurations such as channel configuration, choice of consensus protocol, number of peers per actor, endorsement policies for chaincodes. We have built our application using these services provided by chain rider. Below are the services offered by ChainRider.

1. **Blockchain Network Generator:** This service is responsible for setting up a permissioned blockchain network. It supports networks with multiple organizations and channels. This service generates all the configuration files and cryptographic for a fully functional network. It provides a streamlined and well documented mechanism for network deployment and startup process. The extracted network configuration files are structures in such a way that the configuration file of each physical machine is in a separate folder. These machine specific files can then be deployed on physical or virtual machines to create blockchain nodes.
2. **Smart Contract Generator:** This service creates a hyperledger smart contract which can be built and deployed in less than a minute. The smart contracts are written in NodeJS. It provides an efficient way to create basic and advanced

functionalities for managing an asset object such as insert, query and update. After deploying the smart contract on the blockchain network, it can immediately write on its data ledger.

3. **Smart Contract Marketplace:** This service provides advanced smart contracts offering some sophisticated functionalities for the network. These contracts are available for free of charge. There are 10 smart contracts with 4 different categories available on the marketplace.
4. **RestAPI**: The REST API service can run on any machine in the blockchain network. The API is the form of NodeJS application which provides different APIs to interact with the blockchain application. It can be used to perform different tasks such as registering users, installing smart contracts, creating channels, querying smart contracts, invoking transactions and many more.
5. **Blockchain Explorer:** This service provides analytics of the private blockchain network. It can be optionally setup on each of the machines in the form of web application and provide information such as peer information, block information, transaction information, installed smart contracts, available channels.

3.3.1 Advantages of Using ChainRider for our Application

1. **Token authentication for rest API**

   ChainRider provides in-built user authentication mechanism using json web tokens for all requests. This prevents any malicious user that may try to assume other user's identity.

2. **Trusted environment configuration**

   Any system is always prone to human errors. Hyperledger Fabric being a new and complex technology involving a multitude of configurations and components involves tedious network setup especially for beginners. A platform like ChainRider can be a blessing to businesses and result oriented developers to quickly setup and start playing around a fully-fledged network. ChainRider has automated scripts to setup and easily modify the network which ensures avoiding a human error which can eventually lead to compromising the security of the users or the business.

3. **Cloud enabled**

   A hyperledger fabric network generally comprises of actors running their own peers on their physical systems. ChainRider's cloud-based deployment service prevents users from using faulty or improperly configured machines which might lead to a hacked and therefore a compromised network entity.

CHAPTER 4

EXPERIMENTAL SETUP AND RESULTS

4.1 Experimental Setup

4.1.1 Blockchain Infrastructure Setup

The setup for this infrastructure is driven by the transactions performed by the actors (discussed in section 3.1.1) who interact with the application following specific sets of rules as laid out in our requirements. Amongst these four, Charity is one such key organization operating, maintaining and overseeing any transaction that occurs in the system.

Hyperledger Fabric is an open source permissioned blockchain framework. I have used hyperledger fabric sdk for NodeJS for my application. The *fabric-client* [23] package provides API's to interact with the peers and the orderers of the network and the *fabric-shim* [24] package provides API's for the chaincode interface. The hyperledger fabric components (discussed in section 1.7) for the microlending application are configured as follows:

1. **Peers**: Peers hold the ledger data and host chaincodes. The peers are run and maintained by each actor in the system. Peers can be added dynamically and at least one peer should be interacting with the system for each actor. Charity and Banks host their own peers. The infrastructure for hosting peers for the borrowers and

donors is also provided by Charity. Let us say peers are denoted as *Actor.peer0* then Donor.peer0 depicts a peer with peer id peer0 running on a system accessible by a donor.

2. **Clients**: The actors can interact with the system peers using REST API. This interaction is established using REST or GRPC protocol. Each actor has a corresponding REST API client. We denote them using the convention *Actor API.*
3. **Ordering Service**: Kaka ordering mechanism is used to ensure that the transactions from endorsing peers are validated and ordered in a consistent manner before they are committed onto the ledgers.
4. **Channels**: The setup implements segregation of actor roles and hosting of specific chaincodes running on the peers enrolled on 4 channels. The actors in a channel transact over the assets that are managed by each channel and read-written on the channel's ledger. The endorsement policies for these chaincodes defines the actors' ability to view and endorse a transaction. As discussed in the beginning of this section, we have Charity as the common actor enrolled in all the channels as they are a medium of transacting data between channels. Therefore, we have named the channels using the other actor's role that is enrolled in the channel except the common channel that has all the actors.
5. **Ledger Data**: Ledger in the hyperledger framework is comprised of two parts. The *World State* which holds the current key-value relations of assets and the blockchain transaction logs which has the history of all the transactions. The world state is held in a choice of database used by the chaincodes hosted in a channel to fetch the latest values of assets and transact on them rather than following the entire

blockchain transaction logs to get the required information. We are using CouchDB [25] as the *World State* db as it allows data to be stored in JSON format, rich queries to be executed against the world state data, and indexes to support queries.

The figure below depicts the organization of above components for the experiments:

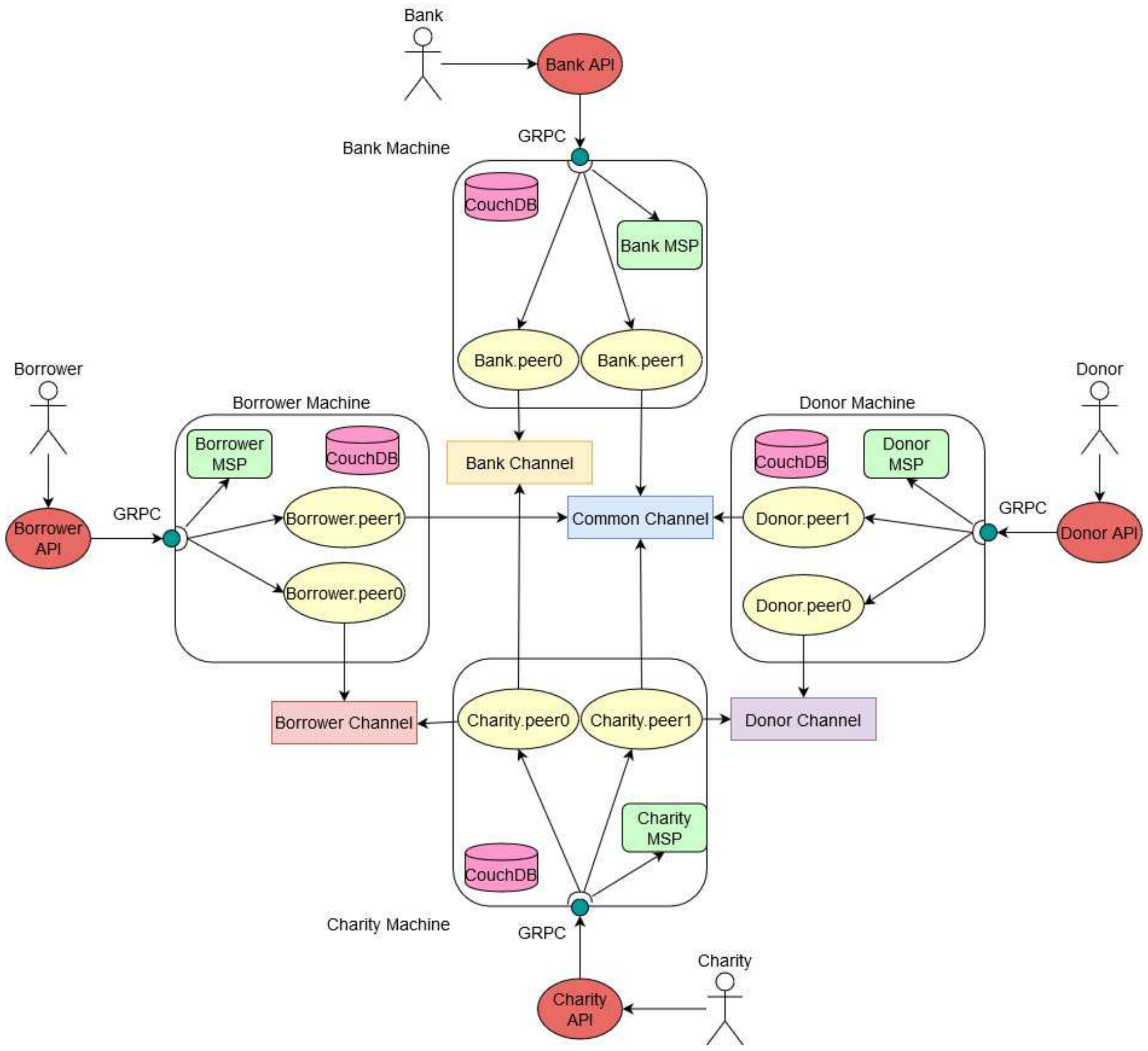


Figure 10 Component diagram for Experimental Setup

4.2 Transparency and Privacy of Data

Transparency and privacy of data are the key aspects for the platform. Project and pool are the two key assets for which transparency needs to be established on this platform.

1. **Project Asset**: Comprises of the key features of a borrower's proposal. It originates from the borrower in the borrower channel and is published on all the key channels. These private copies of the asset are updated at many stages before the accumulated donation or loan is finally paid back to the borrower and it repays the sum back to the system. At the very same time, the privacy of Project asset needs to be ensured and only the required attributes need to be published to all the channels. Maintaining a proper view of this asset for all actors in each channel is extremely crucial bearing in mind that any private information updated or added by an organization is not viewable to other actors. Endorsement feature of hyperledger fabric makes sure that actors view and endorse transactions on assets they are concerned with as shown in table 1 in chapter 3. These policies are bound to chaincodes which are in turn bound to run on designated channels for this platform. Further, this asset also has a section where the borrower updates the status and their progress against the proposed project milestones once the loan is paid out. This is a key data to be kept transparent to the donors for their motivation as described in the hypothesis section of this document.
2. **Pool Asset**: This asset is privately owned and managed by the Charity and only the Donor actor can access it. It comprises of important data related to the fund pool

and donors' activity and is a key aspect to provide a transparent view of these activities to the users concerned actors in the system.

We will go through different states of the project and the pool asset in the lending process and discuss its views for different actors in the system to understand how data privacy and transparency is maintained. Different actors involved are Borrower, Donor, Charity and Bank. Charity actor is common on all the channels. It acts like an operator of the platform.

1. **Open**: Project is in open state when the borrower first proposes a project.

   API endpoints involved: propose and publish

   Borrower calls the *propose* API to give the project details for which funding is needed.

   View for the actors on the borrower channel:

   Actors involved: Borrower and Charity

Table2. Attributes of Project Asset for Borrower in Open State

Project Id	0001		
Project Name	Weave Baskets		
Fund Available	0		
Fund Required	450		
Loan Balance	0		
Owner	user1@borrower.com		
State Description	**OPEN**		
Timeline			
Description	SNO	Available	Required
Get Raw Materials	1	0	100
Machine Setup	2	0	350

Charity calls the *publish* API and the project is published on the donor channel.

View for the actors on the donor channel:

Actors involved: Donor and Charity

Table3. Attributes of Project Asset for Donor in Open State

Project Id	0001				
Project Name	Weave Baskets				
Create Date	Tue Jul 06 2020				
Fund Available	0				
Fund Required	450				
Fund Deadline	Thu Jul 9 2020				
Charity	user1@charity.com				
Owner	user1@borrower.com				
State Description	**OPEN**				
Timeline					
Description	SNO	Donors	Available	Required	Status
Get Raw Materials	1		0	100	OPEN
Machine Setup	2		0	350	OPEN
Voting Deadline	Fri Jul 11 2020				

From table2 and table3 we can see the different views of the open state of the project asset for the borrower and the donor. When borrower proposes a project, he gives the necessary details about the project and its timeline. Charity then modifies this asset and adds new project attributes which is provided to the donor and is not visible to the borrower.

2. **Voting**: If a project is unable to fetch the amount of money required, then that project goes to the voting state. The premium donors will vote and decide whether to fund a project or not.

   API endpoints involved: endCycle

   View of pool asset for the actors on donor channel:

   Actors involved: Donor and Charity

Table4. Attributes of Pool Asset for Donor in Voting State

Create Date	Tue Jul 06 2020
Name	Pool fund
Fund Available	100000
Owner	user1@charity.com
State Description	ACTIVE
Donors	
user1@donor.com	
user2@donor.com	
user3@donor.com	
user4@donor.com	
Pool Records	
user1@charitycom	
Project Id	0003
Fund	610
Milestone Id	2
Premium Donors	
user1@donor.com	
user3#@donor.com	
User Records	
user1@donor.com	
Project Id	0001
Fund	100
Milestone Id	1
Fund	2
Milestone Id	10
user2@donor.com	
Project Id	0003
Fund	45
Milestone Id	1
user3@donor.com	
Project Id	0002
Fund	50
Milestone Id	1
Project Id	0004
Fund	80
Milestone Id	2
user4@donor.com	
Project Id	0003
Fund	50
Milestone Id	2

When funding deadline is reached the *endCycle* API is triggered by a scheduler service maintained by charity and the project asset status changes to *Voting*. This will start the voting phase and premium donors can vote.

View of project asset for the actors on donor channel:

Actors involved: Donor and Charity

Table5. Attributes of Project Asset for Donor in Voting State

Project Id	0001				
Project Name	Weave Baskets				
Create Date	Tue Jul 06 2020				
Fund Available	110				
Fund Required	450				
Fund Deadline	Thu Jul 9 2020				
Charity	user1@charity.com				
Owner	user1@borrower.com				
State Description	**VOTING**				
Timeline					
Description	SNO	Donors	Available	Required	Status
Get Raw Materials	1	user1@donor.com	100	100	FUNDED
Machine Setup	2	user1@donor.com	10	350	OPEN
User Records					
user1@donor.com					
Project Id	0001				
Fund	100				
Milestone Id	1				
Fund	2				
Milestone Id	10				
Vote Yes Count	2				
Voter List					
user1@donor.com					
user3@donor.com					
Voting Deadline	Fri Jul 11 2020				

Pool asset contains the details of the premium donors and the pool record. Pool record attribute provides information about the projects funded by the pool. It stores all the information about the donors and hence pool asset is visible only to the donors and the charity. The information is not shared with any other actors of the system. Charity is responsible for updating the information in the pool asset.

Project did not get the sufficient funding before the deadline and went to voting phase. Donors get a complete view of the project asset and its timeline which will motivate them to whether vote in favor or against. Borrowers and Bank don't have any view of this state of project asset as this information is of no use to the bank or the borrower. The **voterList** and **userRecord** attributes are updated which were blank in the open state as shown in table 5.

3. **Pool fund**: If a project is in voting state and the premium donors agree to provide the funds for the project, it goes in pool fund state.

   When voting deadline is reached *endCycle* API called by charity. This will end the voting period and internally call the *assessVotes* method to check if the voting criteria is met. Currently the criteria for successful voting is that the at least 80% of premium donors should vote in favor. If the voting is successful, then project asset state changes to *pool_fund*.

   API endpoints involved: endCycle, assessVotes

   View of project asset for the actors on donor channel:

   Actors involved: Donor and Charity

Table6. Attributes of Project Asset for Donor in Pool Fund State

Project Id	0002
Project Name	COVID local relief
Create Date	Tue Jul 06 2020
Fund Available	425
Fund Required	930
Fund Deadline	Thu Jul 9 2020
Charity	user1@charity.com
Owner	user1@borrower.com
State Description	**POOL_FUND**

Timeline					
Description	SNO	Donors	Available	Required	Status
Acquire health kits	1	user1@donor.com	380	380	FUNDED
Rent space for medical urgency	2	user1@donor.com	45	550	OPEN

User Records	
user1@donor.com	
Project Id	0002
Fund	200
Milestone Id	1
Project Id	0002
Fund	45
Milestone Id	2
user3@donor.com	
Project id	0002
Milestone id	1
Fund	180
Vote Yes Count	2
Voter List	
user1@donor.com	
user3@donor.com	
Voting Deadline	Fri Jul 11 2020

In table 6 the project was successfully voted by the premium donors. Here the donors have a clear visibility of the fund required by the project, milestones that need to be funded and the other donors who have funded for the project. This transparency of information is helpful for the donors in deciding their vote. Voter List attribute contains the information about the premium donors who successfully voted for the project. Borrowers or banks will not have any view of all this information to ensure the privacy of donors and the fund pool maintained by charity.

4. **External fund**: If a project from the voting state is not selected by the premium donors to be funded, then the state of the project changes to external funds. Banks will decide to whether fund or reject the project.

   API Endpoints involved: publish, loan

   View of project asset for the actors on bank channel:

   Actors involved: Bank and Charity

Table7. Attributes of Project Asset for Bank in Ext_Fund State

Project Id	0001			
Project Name	Weave Baskets			
Create Date	Tue Jul 06 2020			
Fund Available	110			
Fund Required	450			
Fund Deadline	Thu Jul 9 2020			
Charity	user1@charity.com			
State Description	**EXT_FUND**			
Timeline				
Description	SNO	Available	Required	Status
Get Raw Materials	1	100	100	FUNDED
Machine Setup	2	10	350	OPEN
Bank Records				

Charity calls the *publish* API and the project attribute information is published on the bank channel. If the bank agrees to fund the project the *loan* API is called by the bank.

As shown in table 7 the entire donor related information is removed. Bank doesn't need to know about the donors involved in the lending process. Based on the find required bank will decide to fund or not. Bank records field is added and will be updated accordingly.

5. **Funded**: Once the amount of money required by project is completely provided by the donors, the state of the project changes to funded.

   API Endpoints involved: donate, loan, publish

   View for the actors on the borrower channel:

   Actors involved: Borrower and Charity

Table8. Attributes of Project Asset for Borrower in Funded State

Project Id	0001		
Project Name	Weave Baskets		
Fund Available	450		
Fund Required	450		
Loan Balance	340		
Owner	user1@borrower.com		
State Description	**FUNDED**		
Timeline			
Description	SNO	Available	Required
Get Raw Materials	1	100	100
Machine Setup	2	350	350

Project attributes are updated showing that the project is completely funded. Bank or donor related information is not given to the borrower to maintain the privacy of the donors and the bank.

View for the actors on the bank channel:

Actors involved: Bank and Charity

Table9. Attributes of Project Asset for Bank in Funded State

Project Id	0001			
Project Name	Weave Baskets			
Create Date	Tue Jul 06 2020			
Fund Available	450			
Fund Required	450			
Loan Balance	340			
Fund Deadline	Thu Jul 9 2020			
Charity	user1@charity.com			
State Description	**FUNDED**			
Timeline				
Description	SNO	Available	Required	Status
Get Raw Materials	1	100	100	FUNDED
Machine Setup	2	10	350	FUNDED
Bank Records				
user1@bank.com				
Fund		340		

If the bank funds, the project then *loan* API is called. Bank information is updated in the attributes.

The project attributes are updated in table 10 to show that the project is funded. Even though this project was funded by bank as well, that information is not visible to the donor. The donor will be shown the progress of the project milestones. This transparency of data to show the project progress to the donors will help in attracting more donors for the platform. The donors can always track the progress and see where their money is being used.

View for the actors on the donor channel:

Actors involved: Donor and Charity

Table10. Attributes of Project Asset for Donor in Funded State

Project Id	0001				
Project Name	Weave Baskets				
Create Date	Tue Jul 06 2020				
Fund Available	450				
Fund Required	450				
Fund Deadline	Thu Jul 9 2020				
Charity	user1@charity.com				
Owner	user1@borrower.com				
State Description	**FUNDED**				
Timeline					
Description	SNO	Donors	Available	Required	Status
Get Raw Materials	1	user1@donor.com	100	100	FUNDED
Machine Setup	2	user1@donor.com	10	350	FUNDED

4.3 Survey of Smart Contract Security

Ensuring security of chaincode in the system

The key feature of any private blockchain application is the security it brings along with it. Despite this, developing such an application requires manually written code on top of the blockchain platform. The proposed microlending system involves transactions of token asset which represents real world currency on the platform.

Ensuring that the monetary transactions are carried out in a secure and verifiable manner becomes a priority in an enterprise grade application like this.

Chaincodes embed the business rules for the application. This means each transaction is governed by a chaincode. Hence, writing the chaincode in a secure manner and assessing the code against possible breaches in mind are two essential aspects for developing an enterprise application like this.

4.3.1 Security Features Provided by Chaincode

Hyperledger fabric is developed with the focus of enabling enterprises to create and deploy their applications with ease rather than spend effort and money towards ensuring security of the chaincode. As opposed to a public blockchain like Ethereum where an anonymous user can join and access any sensitive data, hyperledger fabric is built from top to bottom to ensure security of data and transactions and to restrict interactions of authenticated users with data (using MSP) based on their roles. As a result, most of the security features are spread out across various components on the entire fabric network as opposed to the focus on deploying highly secure smart contracts as in a public blockchain. Let us discuss some of these key security features provided by hyperledger fabric:

1. **ACL (Access Control List) and endorsement policies**

   In hyperledger fabric, admin of the network can set access control of resources using policies. Each resource policy is associated with certain sets of identities and are simple rules that evaluate to true or false [26]. The resources like user chaincode and events stream source [27] are used by the users to interact with a blockchain network. Resources like system chaincode [6] are run within peer processes and are called in the background. These policies are a fundamental way for hyperledger fabric to ensure that an identity who is trying to access a system resource has the respective rights provided by the admin in order to do so. Endorsement policy set at a chaincode level is one such important policy to ensure that a peer has rights to endorse a transaction.

Using Endorsement Policies for Transactional Security:

When a transaction is proposed by a user on a channel, the transaction is executed by a subset of peers assigned to this channel. These peers have a copy of the chaincode required to run this transaction. This set of peers is called endorser peers. The endorsement capability can be activated by the admin for any peer on the network. The endorsers sign this transaction block with their private key and send this transaction to the ordering service. The transaction is then sent to the ordering service which orders the transactions received into blocks. This block is then broadcasted to other peers of the channel who validate the signatures on the block against the endorsement policy of the chaincode set by the admin before its deployment. This policy specifies the number of endorsements required on a transaction to be deemed valid. Finally, the block is written into their local ledgers only if the transactions sent by each endorser match. This helps us to make sure that each endorser has a valid copy of the chaincode.

The endorsement policy can be used to select these endorsers and can be configured to require validation from a given set of peers from specific organizations. We have set the endorsing policies to require one endorsement form each organization from that channel. This will ensure that each organization has a correct set of rules and agreements for each possible transaction on an asset.

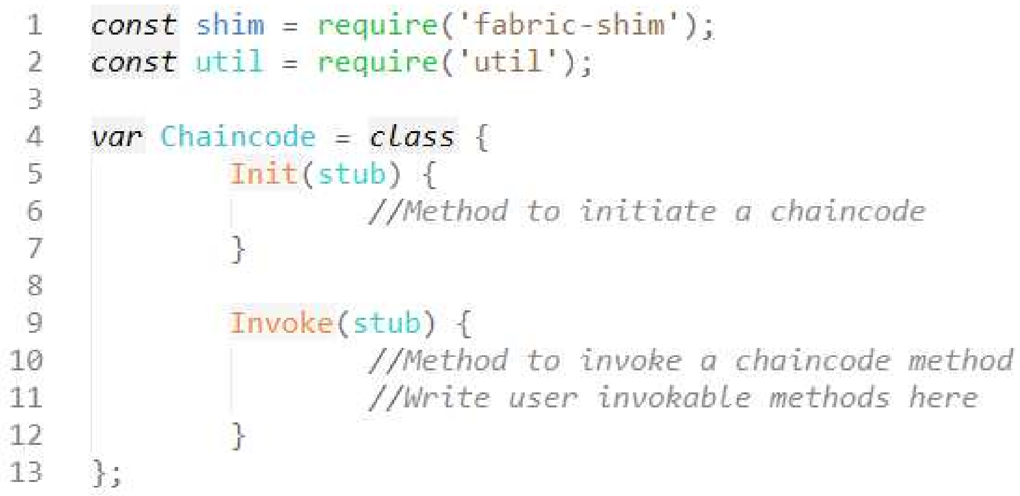
2. **Run in a dockerized environment**

   In hyperledger fabric, chaincode is run inside a docker container. This helps in isolating the chaincode from other chaincodes and the peer process. This also makes the chaincode lifecycle (starting, upgrading and aborting) manageable. The peer and the chaincode dockers communicate via the gRPC protocol. This creates a loose coupling between these processes and results in loose coupling between the chaincode language and the peer.

3. **Chaincode interface is strictly defined and cannot be changed**

   The packages for fabric libraries provide an interface for chaincode developers to implement for writing chaincode in any of the three supported languages. This interface adds to the security by providing a blueprint or a rule for implementing a chaincode. The input and output formats are determined before-hand to ensure deterministic behavior. We will discuss the deterministic behavior in section 4.3.3.

   A sample interface for NodeJS fabric-shim [24] package is shown in figure 11. The *Init* method is called via a client by an admin for the first time after the chaincode is installed. The instantiation call initializes the internal data including initialization of the application state. The *Invoke* method is generally called by the users of the application to perform transactions.



```
const shim = require('fabric-shim');
const util = require('util');

var Chaincode = class {
        Init(stub) {
                //Method to initiate a chaincode
        }

        Invoke(stub) {
                //Method to invoke a chaincode method
                //Write user invokable methods here
        }
};
```

Figure 11 Snapshot of Chaincode Interface for fabric-shim Package

4. Chaincode deployment scenarios

Here we present some common deployment scenarios [28] where the expectations of the channel members are not in line with others. This may cause some serious security issues, however hyperledger fabric handles these scenarios.

- **An organization that disagrees on chaincode definition:** In this scenario, the organization won't install the chaincode on its peers, resulting in the organization not being able to use the chaincode for performing any transaction endorsement. This will in turn lead to the transactions being declined as the endorsement policy will not be met due to our AND policy as described in point 4.3.1.1 above.
- **The channel members do not agree on a chaincode definition:** The chaincode definition will not be installed on any peer of the channel.
- **Organizations install different chaincode packages:** Organizations can install/instantiate a chaincode with a different package id. Fabric allows peers to install different chaincode binaries which will read and write into the same chaincode namespace.

We have enforced the scenario where peers have to download chaincode binaries and install the chaincode with the same id. This will ensure that chaincode binaries maintained by each peer in a channel is the same at a point of time.

4.3.2 Tools for Chaincode Security Assessment

Smart contract on public blockchain Vs Chaincode Security

Smart contracts in public blockchain differ completely from that in private as it is designed to run securely on an open network with anonymous participation. This requires them to be highly secure. However, as mentioned in section 4.3.1, fabric is designed for secure enterprise applications, the responsibility to ensure security is shifted towards the entire infrastructure rather than on the smart contract. The types of risks associated with both the types of smart contracts are different as well [30].

Many DSLs (Domain Specific Language) have been introduced to ensure security of smart contracts on public blockchains. These DSLs aim to minimize risks from user written smart contract codes by restricting instructions or adding specific features on top of the languages they are derived from. Solidiy influenced by C++, python and JavaScript is the most popular smart contract DSL for Ethereum. Vyper, a derivative of python, restricts instructions such as infinite looping and recursive calls. Flint introduced a definition of function access and created asset type to avoid inheritance and overloading. However, private blockchain like hyperledger fabric uses general purpose languages to avoid learning costs. The risks are shifted to other components of the network. [30]

Tools for validating smart contracts security:

Even when developers use such specific languages for smart contracts, vulnerabilities may still exist. There are many smart contract scanners and analyzers available to assess and avoid security risks. There are a variety of tools to assess security for public blockchain. Some of the popular tools for Ethereum smart contracts are listed below.

Table 11 Comparison of a Few Popular Ethereum Smart Contract Analyzers [34]

Tool	**Concept**	**Issues Detectable**	**Drawback**
Mythril	Symbolic execution of EVM bytecode	• Reentrancy • Mishandled exceptions • Transaction order dependence • Timestamp dependence	• Can miss critical violations because of under-approximation
Oyente	Symbolic execution of EVM bytecode	• Reentrancy • Mishandled exceptions • Transaction order dependence • Timestamp dependence	• Can miss critical violations because of under-approximation
Securify	• Define compliance patterns • Define violation patterns, which imply its negation • Analyzes the dependency graph of the smart contract	• Dependency issues • Identify violations of satisfaction of property	• Missing numerical analysis such as overflows • Reachability: Assumes all instructions in the contract to be reachable

However, hyperledger with its inherently guaranteed security as described in section 4.3.1, does not motivate the open source community to create such analyzers. As a result, there does not exist any such tool as of now. [28] Chaincode scanner [31] is one such tool that does some verification for go chaincodes but we confirmed after a proper analysis that the tool doesn't work correctly and moreover only supports chaincode written in Go language. Therefore, developers who write smart contracts using general-purpose languages need to take care of their code. Static analysis of the code is the appropriate way to ensure that coding best practices are being followed.

4.3.3 Static Analysis Assessment Results

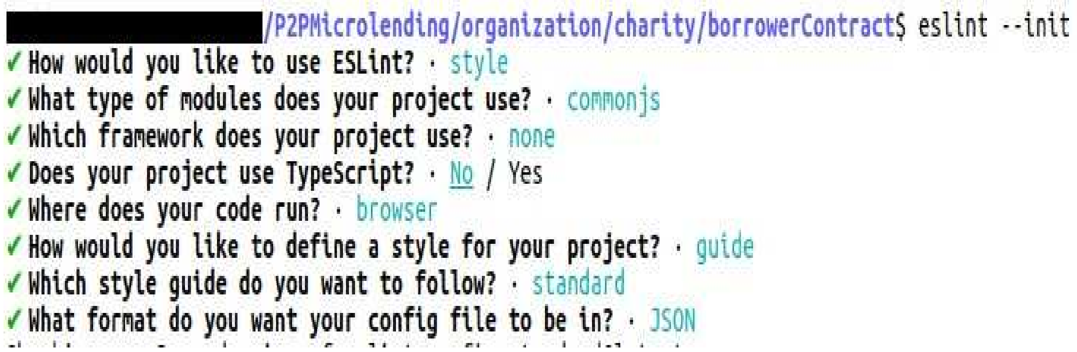
With the understanding that chaincodes are inherently very secure and hence require basic checks for programming errors, we perform static analysis of our chaincode and also look at some hyperledger fabric specific deterministic programming conventions as suggested in an external study [30].

We use static analysis to verify programming conventions and standards against general language level. This will ensure that the contract creator does not introduce any programmatic security vulnerabilities in the code.

Tool used: ESLint [32] A npm package to identify problematic patterns in a given JavaScript code by comparing it against ECMAScript [33] standards.

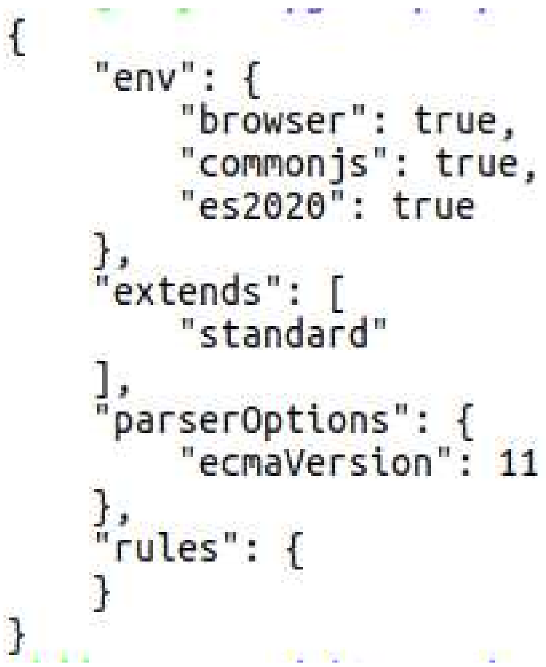
Test Scenario:

We start by creating a config file required for the tool. When we run the command to initialize this configuration, it asks few questions based on the test scenario. A screenshot of the questionnaire and the resultant configuration file for the tool are shown in figure 12 and figure 13 respectively.



```
[redacted]/P2PMicrolending/organization/charity/borrowerContract$ eslint --init
✓ How would you like to use ESLint? · style
✓ What type of modules does your project use? · commonjs
✓ Which framework does your project use? · none
✓ Does your project use TypeScript? · No / Yes
✓ Where does your code run? · browser
✓ How would you like to define a style for your project? · guide
✓ Which style guide do you want to follow? · standard
✓ What format do you want your config file to be in? · JSON
```

Figure 12 Questionnaire with Answers to Generate Lint Configuration File



```
{
    "env": {
        "browser": true,
        "commonjs": true,
        "es2020": true
    },
    "extends": [
        "standard"
    ],
    "parserOptions": {
        "ecmaVersion": 11
    },
    "rules": {
    }
}
```

Figure 13 Generated Configuration File

This configuration file contains the environment related configuration for the files to analyze. The *parserOptions* is an important field and defines the JavaScript version to be used for parsing the smart contract files. The *ecmaVersion* defines the ECMAScript [33] standards to be checked while parsing these files.

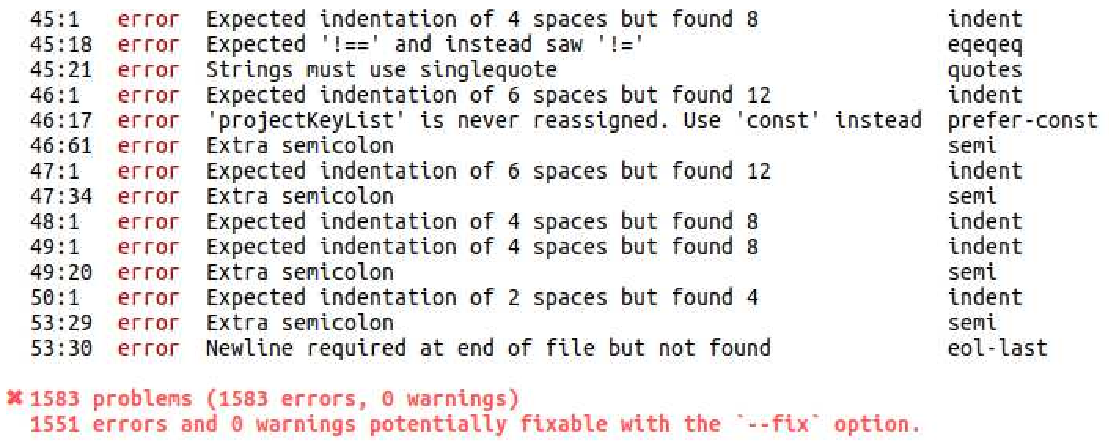
Next, we run the analyzer for each of the 4 smart contracts for our application. We have shown the result of the run for each smart contract in figure 14, 17, 20 and 23.

Now we use the --fix option provided by the tool to fix potentially fixable errors and warnings. These corrections are mostly to enforce language standards and correct any syntax issues. The tool generates a new set of warnings and errors unhandled by the tool. These are shown in figure 15, 18, 21 and 24.

Finally, we fix the remaining semantic errors like the existence of an unused variable or not using camel cases. These errors require programmer intervention as the tool is not smart enough to detect and fix semantic errors. We re-run the analyzer on these files and try to find more errors. If the check passes and no warnings or errors are detected, the tool just exits without any output as shown in figure 16, 19, 22 and 25.

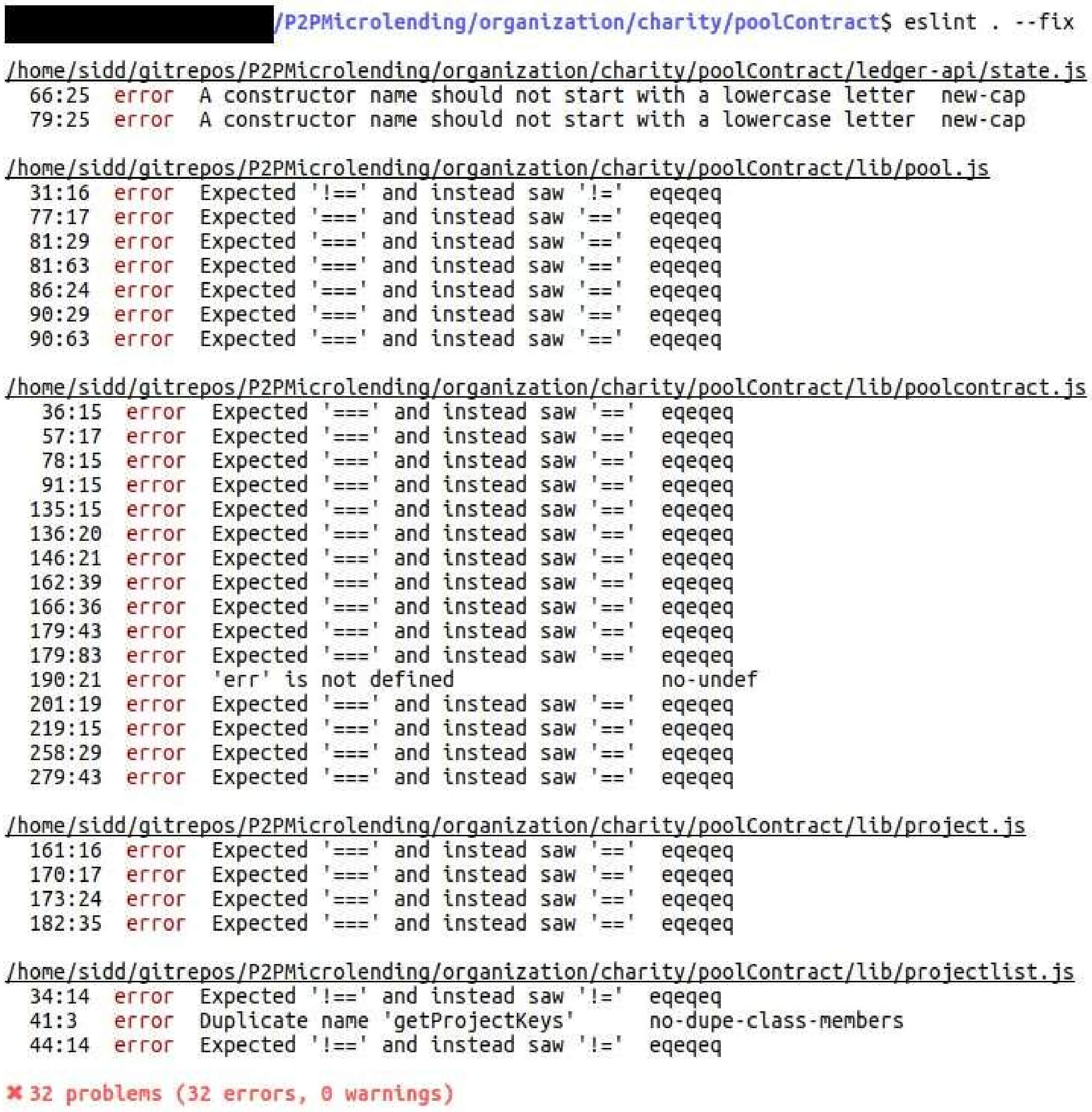
Test Outputs:

1. **Pool Contract**



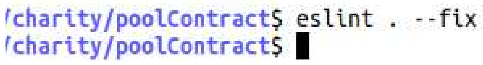
```
  45:1   error  Expected indentation of 4 spaces but found 8            indent
  45:18  error  Expected '!==' and instead saw '!='                     eqeqeq
  45:21  error  Strings must use singlequote                            quotes
  46:1   error  Expected indentation of 6 spaces but found 12           indent
  46:17  error  'projectKeyList' is never reassigned. Use 'const' instead  prefer-const
  46:61  error  Extra semicolon                                         semi
  47:1   error  Expected indentation of 6 spaces but found 12           indent
  47:34  error  Extra semicolon                                         semi
  48:1   error  Expected indentation of 4 spaces but found 8            indent
  49:1   error  Expected indentation of 4 spaces but found 8            indent
  49:20  error  Extra semicolon                                         semi
  50:1   error  Expected indentation of 2 spaces but found 4            indent
  53:29  error  Extra semicolon                                         semi
  53:30  error  Newline required at end of file but not found           eol-last

✖ 1583 problems (1583 errors, 0 warnings)
  1551 errors and 0 warnings potentially fixable with the `--fix` option.
```

Figure 14 Output for Initial Execution of lint for Pool Contract



```
[redacted]/P2PMicrolending/organization/charity/poolContract$ eslint . --fix

/home/sidd/gitrepos/P2PMicrolending/organization/charity/poolContract/ledger-api/state.js
  66:25  error  A constructor name should not start with a lowercase letter  new-cap
  79:25  error  A constructor name should not start with a lowercase letter  new-cap

/home/sidd/gitrepos/P2PMicrolending/organization/charity/poolContract/lib/pool.js
  31:16  error  Expected '!==' and instead saw '!='  eqeqeq
  77:17  error  Expected '===' and instead saw '=='  eqeqeq
  81:29  error  Expected '===' and instead saw '=='  eqeqeq
  81:63  error  Expected '===' and instead saw '=='  eqeqeq
  86:24  error  Expected '===' and instead saw '=='  eqeqeq
  90:29  error  Expected '===' and instead saw '=='  eqeqeq
  90:63  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/poolContract/lib/poolcontract.js
   36:15  error  Expected '===' and instead saw '=='  eqeqeq
   57:17  error  Expected '===' and instead saw '=='  eqeqeq
   78:15  error  Expected '===' and instead saw '=='  eqeqeq
   91:15  error  Expected '===' and instead saw '=='  eqeqeq
  135:15  error  Expected '===' and instead saw '=='  eqeqeq
  136:20  error  Expected '===' and instead saw '=='  eqeqeq
  146:21  error  Expected '===' and instead saw '=='  eqeqeq
  162:39  error  Expected '===' and instead saw '=='  eqeqeq
  166:36  error  Expected '===' and instead saw '=='  eqeqeq
  179:43  error  Expected '===' and instead saw '=='  eqeqeq
  179:83  error  Expected '===' and instead saw '=='  eqeqeq
  190:21  error  'err' is not defined                no-undef
  201:19  error  Expected '===' and instead saw '=='  eqeqeq
  219:15  error  Expected '===' and instead saw '=='  eqeqeq
  258:29  error  Expected '===' and instead saw '=='  eqeqeq
  279:43  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/poolContract/lib/project.js
  161:16  error  Expected '===' and instead saw '=='  eqeqeq
  170:17  error  Expected '===' and instead saw '=='  eqeqeq
  173:24  error  Expected '===' and instead saw '=='  eqeqeq
  182:35  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/poolContract/lib/projectlist.js
  34:14  error  Expected '!==' and instead saw '!='  eqeqeq
  41:3   error  Duplicate name 'getProjectKeys'      no-dupe-class-members
  44:14  error  Expected '!==' and instead saw '!='  eqeqeq

✖ 32 problems (32 errors, 0 warnings)
```

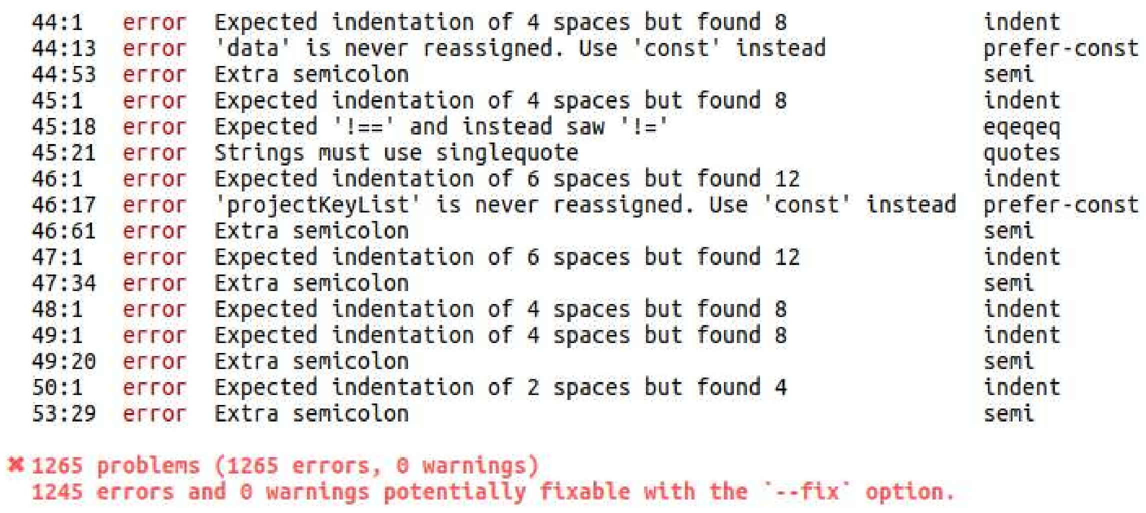
Figure 15 Output After Syntactic Correction by lint for Pool Contract



```
/charity/poolContract$ eslint . --fix
/charity/poolContract$ █
```

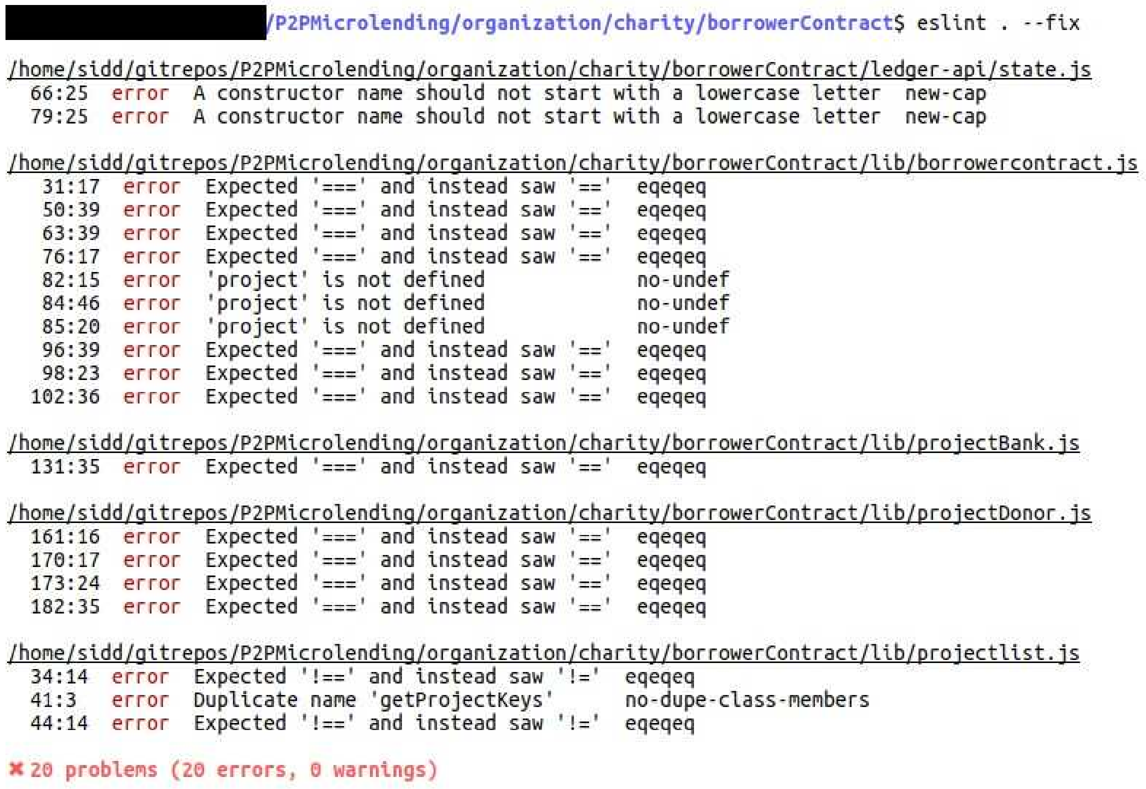
Figure 16 Output After All Corrections for Pool Contract

2. Borrower Contract



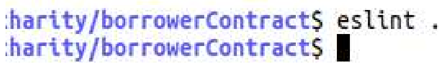
```
  44:1   error  Expected indentation of 4 spaces but found 8                 indent
  44:13  error  'data' is never reassigned. Use 'const' instead              prefer-const
  44:53  error  Extra semicolon                                              semi
  45:1   error  Expected indentation of 4 spaces but found 8                 indent
  45:18  error  Expected '!==' and instead saw '!='                          eqeqeq
  45:21  error  Strings must use singlequote                                 quotes
  46:1   error  Expected indentation of 6 spaces but found 12                indent
  46:17  error  'projectKeyList' is never reassigned. Use 'const' instead    prefer-const
  46:61  error  Extra semicolon                                              semi
  47:1   error  Expected indentation of 6 spaces but found 12                indent
  47:34  error  Extra semicolon                                              semi
  48:1   error  Expected indentation of 4 spaces but found 8                 indent
  49:1   error  Expected indentation of 4 spaces but found 8                 indent
  49:20  error  Extra semicolon                                              semi
  50:1   error  Expected indentation of 2 spaces but found 4                 indent
  53:29  error  Extra semicolon                                              semi

✖ 1265 problems (1265 errors, 0 warnings)
  1245 errors and 0 warnings potentially fixable with the `--fix` option.
```

Figure 17 Output for Initial Execution of lint for Borrower Contract

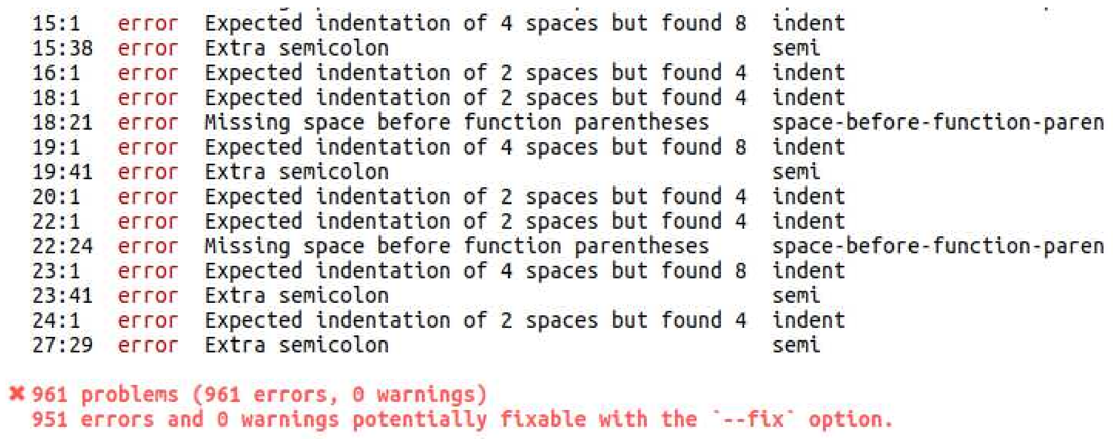


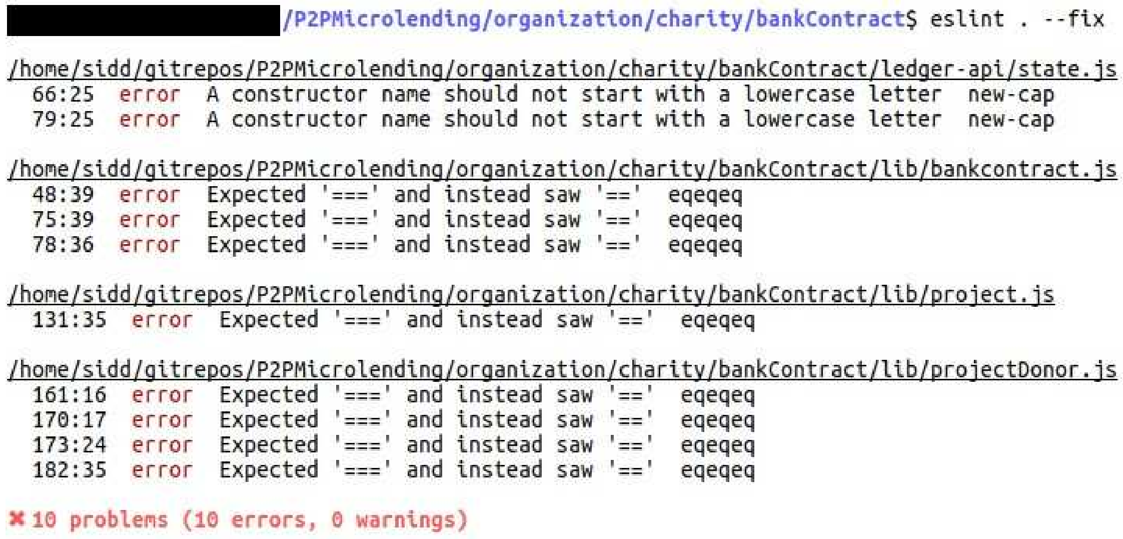
```
[redacted]/P2PMicrolending/organization/charity/borrowerContract$ eslint . --fix

/home/sidd/gitrepos/P2PMicrolending/organization/charity/borrowerContract/ledger-api/state.js
  66:25  error  A constructor name should not start with a lowercase letter  new-cap
  79:25  error  A constructor name should not start with a lowercase letter  new-cap

/home/sidd/gitrepos/P2PMicrolending/organization/charity/borrowerContract/lib/borrowercontract.js
   31:17  error  Expected '===' and instead saw '=='  eqeqeq
   50:39  error  Expected '===' and instead saw '=='  eqeqeq
   63:39  error  Expected '===' and instead saw '=='  eqeqeq
   76:17  error  Expected '===' and instead saw '=='  eqeqeq
   82:15  error  'project' is not defined            no-undef
   84:46  error  'project' is not defined            no-undef
   85:20  error  'project' is not defined            no-undef
   96:39  error  Expected '===' and instead saw '=='  eqeqeq
   98:23  error  Expected '===' and instead saw '=='  eqeqeq
  102:36  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/borrowerContract/lib/projectBank.js
  131:35  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/borrowerContract/lib/projectDonor.js
  161:16  error  Expected '===' and instead saw '=='  eqeqeq
  170:17  error  Expected '===' and instead saw '=='  eqeqeq
  173:24  error  Expected '===' and instead saw '=='  eqeqeq
  182:35  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/borrowerContract/lib/projectlist.js
  34:14  error  Expected '!==' and instead saw '!='  eqeqeq
  41:3   error  Duplicate name 'getProjectKeys'      no-dupe-class-members
  44:14  error  Expected '!==' and instead saw '!='  eqeqeq

✖ 20 problems (20 errors, 0 warnings)
```

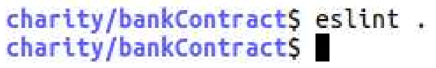
Figure 18 Output After Syntactic Correction by lint for Borrower Contract


```
:harity/borrowerContract$ eslint .
:harity/borrowerContract$ █
```
Figure 19 Output After All Corrections for Borrower Contract

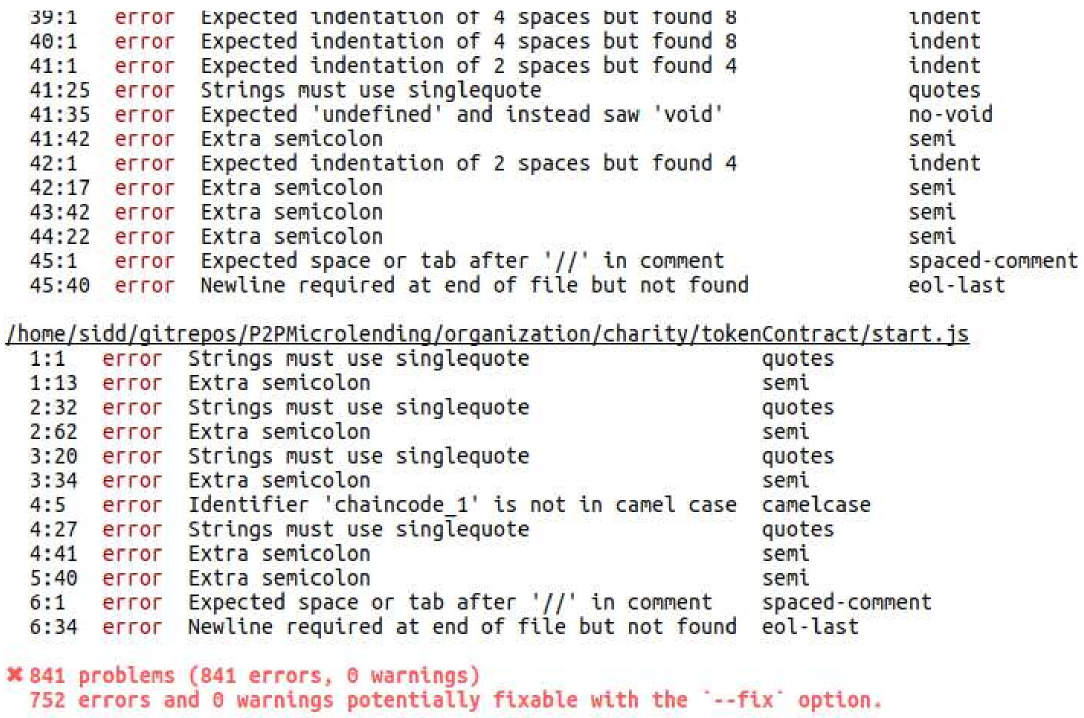
3. **Bank Contract**


```
  15:1   error  Expected indentation of 4 spaces but found 8  indent
  15:38  error  Extra semicolon                               semi
  16:1   error  Expected indentation of 2 spaces but found 4  indent
  18:1   error  Expected indentation of 2 spaces but found 4  indent
  18:21  error  Missing space before function parentheses     space-before-function-paren
  19:1   error  Expected indentation of 4 spaces but found 8  indent
  19:41  error  Extra semicolon                               semi
  20:1   error  Expected indentation of 2 spaces but found 4  indent
  22:1   error  Expected indentation of 2 spaces but found 4  indent
  22:24  error  Missing space before function parentheses     space-before-function-paren
  23:1   error  Expected indentation of 4 spaces but found 8  indent
  23:41  error  Extra semicolon                               semi
  24:1   error  Expected indentation of 2 spaces but found 4  indent
  27:29  error  Extra semicolon                               semi

✖ 961 problems (961 errors, 0 warnings)
  951 errors and 0 warnings potentially fixable with the `--fix` option.
```
Figure 20 Output for Initial Execution of lint for Bank Contract


```
[redacted]/P2PMicrolending/organization/charity/bankContract$ eslint . --fix

/home/sidd/gitrepos/P2PMicrolending/organization/charity/bankContract/ledger-api/state.js
  66:25  error  A constructor name should not start with a lowercase letter  new-cap
  79:25  error  A constructor name should not start with a lowercase letter  new-cap

/home/sidd/gitrepos/P2PMicrolending/organization/charity/bankContract/lib/bankcontract.js
  48:39  error  Expected '===' and instead saw '=='  eqeqeq
  75:39  error  Expected '===' and instead saw '=='  eqeqeq
  78:36  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/bankContract/lib/project.js
  131:35  error  Expected '===' and instead saw '=='  eqeqeq

/home/sidd/gitrepos/P2PMicrolending/organization/charity/bankContract/lib/projectDonor.js
  161:16  error  Expected '===' and instead saw '=='  eqeqeq
  170:17  error  Expected '===' and instead saw '=='  eqeqeq
  173:24  error  Expected '===' and instead saw '=='  eqeqeq
  182:35  error  Expected '===' and instead saw '=='  eqeqeq

✖ 10 problems (10 errors, 0 warnings)
```
Figure 21 Output After Syntactic Correction by lint for Bank Contract

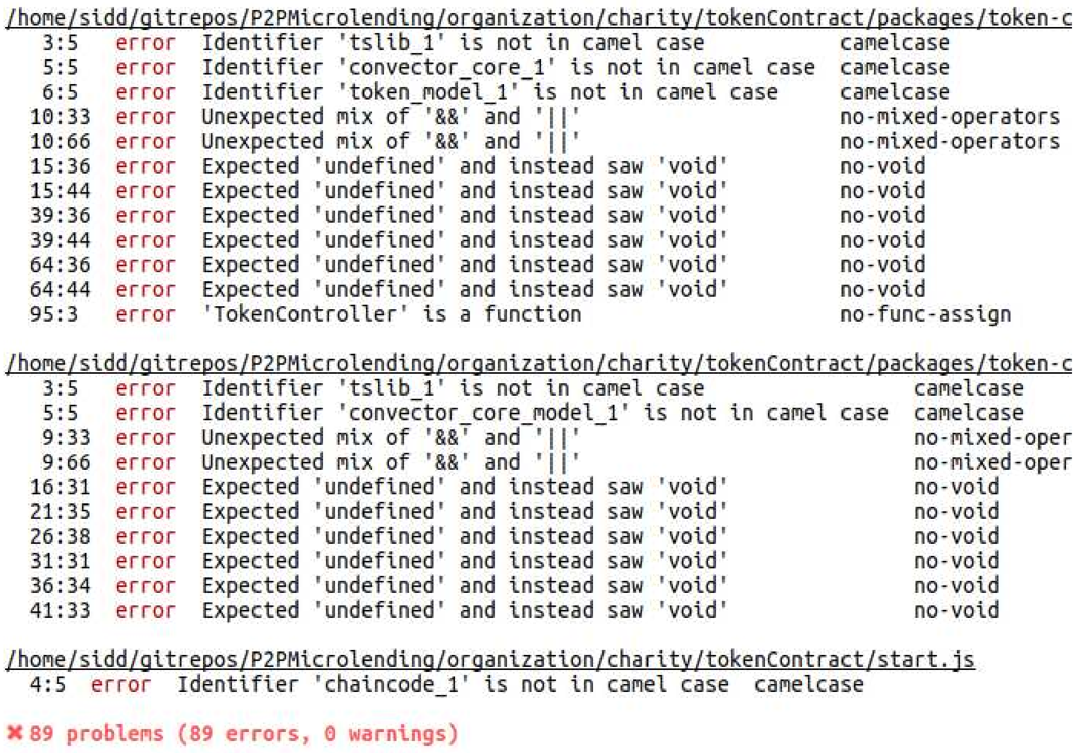

```
charity/bankContract$ eslint .
charity/bankContract$ █
```
Figure 22 Output After All Corrections for Bank Contract

4. Token Contract



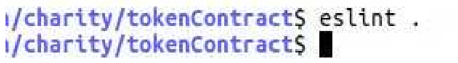
```
  39:1   error  Expected indentation of 4 spaces but found 8           indent
  40:1   error  Expected indentation of 4 spaces but found 8           indent
  41:1   error  Expected indentation of 2 spaces but found 4           indent
  41:25  error  Strings must use singlequote                           quotes
  41:35  error  Expected 'undefined' and instead saw 'void'            no-void
  41:42  error  Extra semicolon                                        semi
  42:1   error  Expected indentation of 2 spaces but found 4           indent
  42:17  error  Extra semicolon                                        semi
  43:42  error  Extra semicolon                                        semi
  44:22  error  Extra semicolon                                        semi
  45:1   error  Expected space or tab after '//' in comment            spaced-comment
  45:40  error  Newline required at end of file but not found          eol-last

/home/sidd/gitrepos/P2PMicrolending/organization/charity/tokenContract/start.js
  1:1   error  Strings must use singlequote                     quotes
  1:13  error  Extra semicolon                                  semi
  2:32  error  Strings must use singlequote                     quotes
  2:62  error  Extra semicolon                                  semi
  3:20  error  Strings must use singlequote                     quotes
  3:34  error  Extra semicolon                                  semi
  4:5   error  Identifier 'chaincode_1' is not in camel case    camelcase
  4:27  error  Strings must use singlequote                     quotes
  4:41  error  Extra semicolon                                  semi
  5:40  error  Extra semicolon                                  semi
  6:1   error  Expected space or tab after '//' in comment      spaced-comment
  6:34  error  Newline required at end of file but not found    eol-last

✖ 841 problems (841 errors, 0 warnings)
  752 errors and 0 warnings potentially fixable with the `--fix` option.
```

Figure 23 Output for Initial Execution of lint for Token Contract



```
/home/sidd/gitrepos/P2PMicrolending/organization/charity/tokenContract/packages/token-c
   3:5   error  Identifier 'tslib_1' is not in camel case           camelcase
   5:5   error  Identifier 'convector_core_1' is not in camel case  camelcase
   6:5   error  Identifier 'token_model_1' is not in camel case     camelcase
  10:33  error  Unexpected mix of '&&' and '||'                     no-mixed-operators
  10:66  error  Unexpected mix of '&&' and '||'                     no-mixed-operators
  15:36  error  Expected 'undefined' and instead saw 'void'         no-void
  15:44  error  Expected 'undefined' and instead saw 'void'         no-void
  39:36  error  Expected 'undefined' and instead saw 'void'         no-void
  39:44  error  Expected 'undefined' and instead saw 'void'         no-void
  64:36  error  Expected 'undefined' and instead saw 'void'         no-void
  64:44  error  Expected 'undefined' and instead saw 'void'         no-void
  95:3   error  'TokenController' is a function                     no-func-assign

/home/sidd/gitrepos/P2PMicrolending/organization/charity/tokenContract/packages/token-c
   3:5   error  Identifier 'tslib_1' is not in camel case                 camelcase
   5:5   error  Identifier 'convector_core_model_1' is not in camel case  camelcase
   9:33  error  Unexpected mix of '&&' and '||'                           no-mixed-oper
   9:66  error  Unexpected mix of '&&' and '||'                           no-mixed-oper
  16:31  error  Expected 'undefined' and instead saw 'void'               no-void
  21:35  error  Expected 'undefined' and instead saw 'void'               no-void
  26:38  error  Expected 'undefined' and instead saw 'void'               no-void
  31:31  error  Expected 'undefined' and instead saw 'void'               no-void
  36:34  error  Expected 'undefined' and instead saw 'void'               no-void
  41:33  error  Expected 'undefined' and instead saw 'void'               no-void

/home/sidd/gitrepos/P2PMicrolending/organization/charity/tokenContract/start.js
  4:5  error  Identifier 'chaincode_1' is not in camel case  camelcase

✖ 89 problems (89 errors, 0 warnings)
```

Figure 24 Output After Syntactic Correction by lint for Token Contract



```
ı/charity/tokenContract$ eslint .
ı/charity/tokenContract$ █
```

Figure 25 Output After All Corrections for Token Contract

Static analysis tools are widely available for all languages and as we can see in the above testing scenarios, we see that it helps in a proper implementation of the language in use. It helps remove syntactic errors completely as well as semantic errors up to a certain extent. This is one way of making sure that we write the smart contract code in a verifiable manner and would be recommended to be used as a handy tool for developers before each deployment iteration of their smart contracts.

Common best practices for writing chaincode with respect to hyperledger fabric environment:

Table 12 Potential Risks for Chaincodes due to Non-Deterministic Programming [30]

Category	**Risks**	**Rationale Behind the Risk**
Non-determinism Arising From Language Instructions	Global Variable	Global variable can be changed inherently
	Key-Value Structure Iteration	Key-value returned in random order
	Reified Object Addresses	Addresses of memory depend on execution environment
	Concurrency of Program	Can lead to non-deterministic behavior e.g. race condition
	Generating Random Number	Peers can obtain different results based on random number value generated
	System Timestamp	Difficult to ensure timestamp functions are executed at same time for different peers
Non-determinism Caused From Accessing Outside	Web Service	Need to ensure that the results of calling the web service is not different among peers
	System Command Execution	Need to ensure systems hosting the peer return same output for each peer

of Blockchain	External File Accessing	Need to ensure files are synced for all peers
	External Library Calling	Need to consider external library behavior
State Database Specification	Range Query Risk	Two functions to call range query cause phantom read of the ledger. Phantom read reads data that other transactions add or delete and changes the result of the process.
Fabric Specification	Cross Channel Chaincode Invocation	Addresses of memory depend on execution environment, endorsing peer needs to be hosted on the machine hosting the peer for second channel
	Read Your Write	A value written for a key cannot be read within the same transaction call
Practices	Unchecked Input Arguments	Should verify input arguments for every method
	Unhandled Errors	Should handle every error, even on unsuccessful returns

Approach to avoid non-determinism:

To avoid the above categories of non-deterministic practices for our platform, we confirm having made sure all our chaincodes comply with the following conventions:

1. Non-determinism Arising from Language Instructions

- No use of global variables
- No key-value iteration inside any transaction call
- No use of memory dependent objects
- Zero concurrency, concurrency at API level if any
- Send timestamps from API while invoking a chaincode method rather than reading timestamps within chaincode

2. Non-determinism Caused from Accessing Outside of Blockchain

- No external services called or returned from
- No system command execution
- Files packed within the chaincode package for any reference
- Use verified libraries from fabric sample projects [35] for any external operation

3. Fabric Specification

- Made sure chaincodes for the two channels are hosted on the same peer for transactions where cross chaincode invocation is required
- Not reading the value from ledger before the transaction on that key is committed

4. Common Practices

- Hyperledger Fabric makes sure a method is invoked with the correct number and types of arguments
- Handled all errors, or throwing an empty error when an invoked method is exiting without any execution

4.3.4 Common Security Attacks and its Prevention

Below mentioned are some of the common attacks possible on the application and their corresponding preventive measures.

1. **51% attack:** 51% attack refers to such an attack where a single entity is capable of taking the entire decision of what should happen next in the network. [36] Since hyperledger is a private blockchain framework, it is not completely decentralized which introduces the possibility of such type of attack. Charity acts like an operator in our application and is available on all the channels. Charity can be vulnerable to

this type of attack and can be hacked. To prevent this type of attacks in our application, each chaincode has endorsement policies that involve all the actors for that channel. This ensures that the task of proposing a new transaction is shared amongst the actors involved in the chaincode's policy.

Malicious charity nodes can try to obtain the complete access of project and pool asset. The pool asset is owned by charity and can be exploited. To avoid this, we have implemented a voting-based mechanism to determine the usage of pool funds. Further to ensure that the voting process is legitimate, we only allow donors who have a good donation record to participate.

2. **Invalid id attack**: In this attack a user with invalid id or who is unauthorized tries to gain access to the transactions and data in the system. [37] When multiple actors/users are operating on the same channel such type of attack is easily possible. Consider the scenario where a borrower is not able to get sufficient funds for its project. The project goes to ext_fund state and bank has to decide whether or not fund the project. Since all the actors are operating on same channel, the borrower will have the complete view of the project asset and the pool asset. Two scenarios are possible here:

    - Borrower will have the view of the pool asset and can try contacting the charity directly to provide the funds.
    - Borrower will have the access to banks information and can try reaching out to the bank to negotiate the terms and conditions for the loan.

Both these scenarios are not acceptable as, as the security and confidentiality of the system is compromised.

To prevent from such a scenario to arise in our application, we have 4 different channels. Any updates happening to the project or the pool asset during the lending process are not visible to the borrower as depicted in the transparency section above. If the charity decided to fund the project, the borrower will only receive the information that the project is funded. They cannot know how or by whom. If the project is funded by bank, then also the bank details are not shared with the borrower. Only the information that the project is funded, and loan amount is shared with the borrower.

3. **Spoofing:** Spoofing is a method whereby a user can pretend to be an authenticated user and act on behalf of them. This is the most classic case for blockchain based cryptocurrencies. To alleviate this situation, our approach follows a two layered user authentication process. Chainrider provides a way to enable user authentication on the rest APIlevel using JSON web tokens or simply JWT. JWTs are comprised of three parts that are encoded and clubbed to form the token. Header which consists of the signing algorithm, payload which consists of information such as application level identification attributes held by a user and signature part is a signature constructed using the encoded forms of header and payload, a secret and the signing algorithm. Once the user logs in with the proper payload, a JWT is returned by the application which needs to be passed with subsequent API calls. Further, it expires overtime making sure the JWTs are not stolen or reused by any other individual. We are using the following parts for creating payload for registering users for JWTs:

Table13. Attributes for payload of JWT

User Name	user1@charity.com
Org Name	charity
Role	client
Attributes	""
MSP Secret	xyz

The organization name and the *Membership Service Provider* secret are used to make sure that the user is registered and enrolled with the correct MSP on the network. Now the network level security comes into picture. The *Certificate Authority* issues *X.509* certificates for every component as well as members on the system. X.509 certificates are the basis for protocols like TLS and SSL. Each entity and module on the network interact using their public key certificates. Any member of an organization is supposed to sign a transaction with their private key before they can propose or commit a transaction. This is verified by the orderer node and rest of the participating peers with the public key of the member before it can be committed to the chain.

As mentioned in the beginning of this section, token asset is basically the money on our platform. Spoofing attacks can affect the token asset management and compromise the motive behind this platform of attracting donors by ensuring trust and transparency. A user who is exercising this attack can steal a member's owned token assets. To prevent this, we have associated token assets to a member's X.509 certificate-based fingerprint. Any user who tries to access their token should have a valid certificate issued by the CA and enrolled by their MSP. The two-layered authentication protocol described in above paragraph therefore plays a key role in establishing user and token asset’s security.

CHAPTER 5

CONCLUSION

5.1 Privacy, Transparency and Security Symmary

In this thesis, I investigated on why and how to use blockchain to make a secure application for peer to peer microlending. I have used private blockchain to build my application as it provides with features that help in ensuring the objectives stated in the hypothesis discussed in section 1.9.

The use of charity as an actor to the application helped in ensuring that only valid projects are introduced in the system, the different project and pool attributes are updated continuously as the assets state change and it also ensure that cause of microlending is supported by providing funds to the borrowers from pool fund if no donors are available. The different views of the assets discussed in the section 4.2 helps us in visualizing how the information from different actors is being shared and how the transparency is maintained while the project states change.

The project into multiple milestones and progress of the milestones is shown to the donors which will help them in tracking their donations and will also motivate them to provide funds for other milestones. If donors have good donation record, we entitle them as premium donors in the system which helps in enforcing trust between the application and the donors and in turn helps in attracting more donors. These premium donors govern the voting process as well not giving the complete monopoly to the charity.

Summarizing the discussion from section 4.3 it mentions the various security features provided by chaincode and steps taken to ensure the chaincode's security. The use of multiple channels, different endorsement policies, use of tokens in combination with the certificate authority helps in minimizing some of the common security threats to the system.

5.2 Future Work

Currently our design supports multiple charity institutions enrolled onto the network, but each charity can manage only their individual pools of funds and donors. This is a limitation in the sense that each charity might have limited funds and a small group of donors enrolled in their pool, not turning out to be trustable and eventually inefficient for the borrowers. We can rather allow multiple charity organizations to club their pool of funds and have a common pool of donors. This will result in a large sum of pool funds and a better reach for the borrowers.

Assets are recorded on the blockchain as key-value pairs. Currently our pool asset's key is constructed by combining a pool name and the owner id that is the charity. We can remove the singularity of these pools by having a common key for the pool, so that it can be accessed by any charity organization on the donor channel. The formula below shows this where f is a concatenation type function:

Current *POOL_KEY = f(pool_name, CharityID)*
Future *POOL_KEY = f(pool_name)*

The figures below show an overview of current and future architecture for pool management where C denotes a charity actor, P denotes the pool and D denotes a donor.

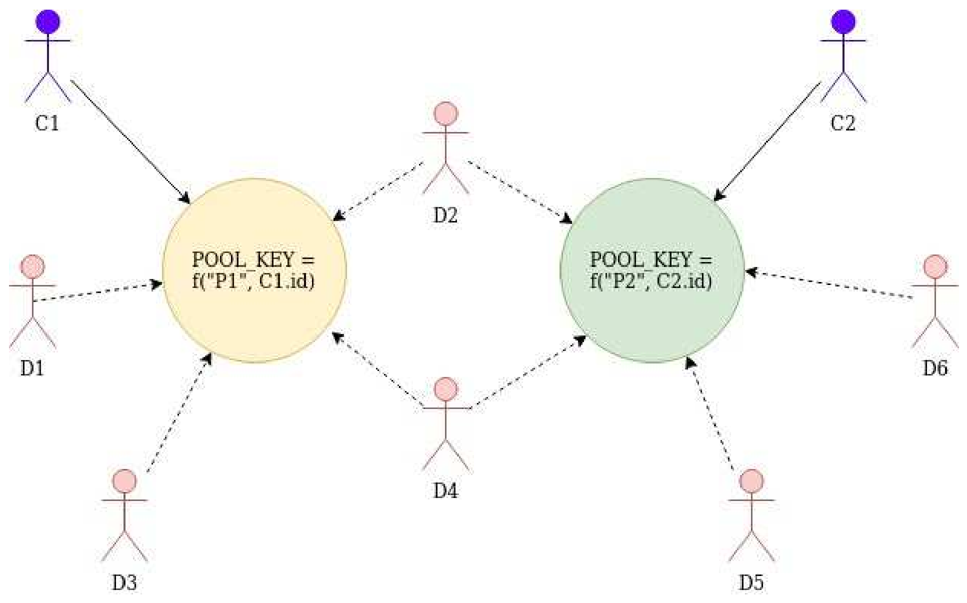


Figure 26. Current Architecture of Pool per Charity Basis

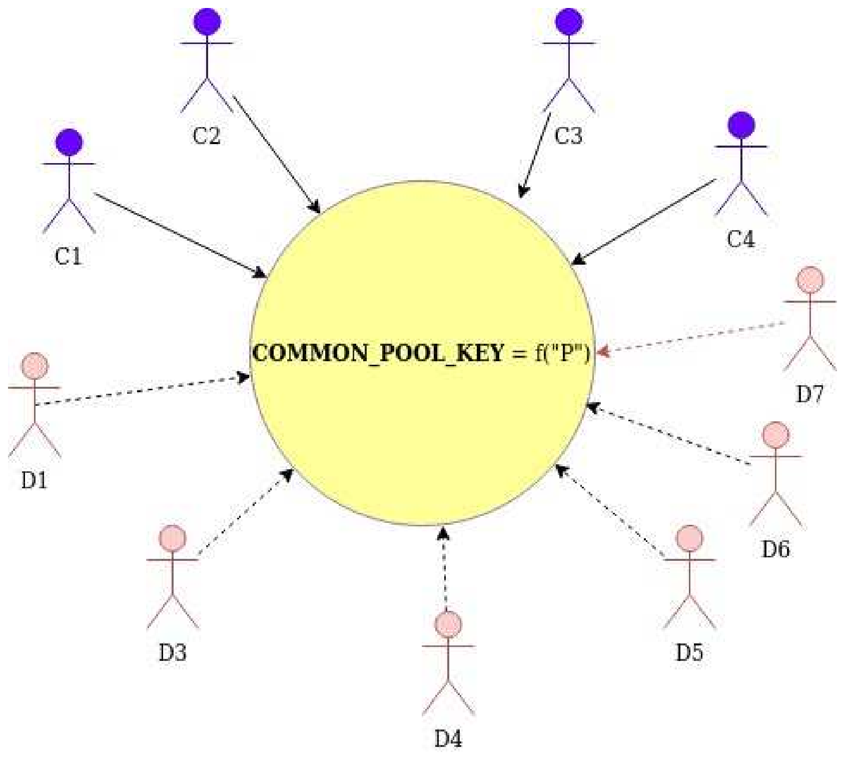


Figure 27 Architecture for a Common Fund Pool for All Charities

This can come along with security threats like some charity institution exploiting pool resources and probable disagreements amongst them regarding their decisions on their extents to support a project. We will have to establish new rules for voting mechanisms for this common shared pool. This can be resolved by revising the terms encoded in the smart contracts and an external monitoring system for our pool asset. Hyperledger's private data collections can also be used to manage private data if any for charity organizations within the same channel.

REFERENCES

1. Patel, Raj, Akhil Sethia, and Shyam Patil. “Blockchain - Future of Decentralized Systems.” In *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, 369–374. IEEE, 2018.

2. “Public Vs. Private Blockchain: A Comprehensive Comparison.” 2019. Blockchain-Council.Org. August 7, 2019. https://www.blockchain-council.org/blockchain/public-vs-private-blockchain-a-comprehensive-comparison/

3. "What Is Microlending? Definition And Examples". 2020. The Balance. https://www.thebalance.com/microlending-315625.

4. “Bitcoin - Open Source P2P Money.” 2009. Bitcoin.Org. 2009. https://bitcoin.org/en/.

5. “Home.” n.d. Ethereum.Org. https://ethereum.org/en/.

6. Androulaki, Elli, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, et al. 2018. “Hyperledger Fabric.” *Proceedings of the Thirteenth EuroSys Conference on - EuroSys ’18*. https://doi.org/10.1145/3190508.3190538.

7. “Introduction — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html#modularity.

8. “Introduction — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html#privacy-and-confidentiality.

9. “Introduction — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html#performance-and-scalability.

10. “Glossary — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.2/glossary.html#organization.

11. “Membership Service Provider (MSP) — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/membership/membership.html.

12. “The Ordering Service — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/orderer/ordering_service.html.

13. Paul, Moses Sam. 2018. “Hyperledger — Chapter 6 | Hyperledger Fabric Components — Technical Context.” Medium. December 19, 2018. https://medium.com/swlh/hyperledger-chapter-6-hyperledger-fabric-components-technical-context-767985f605dd.

14. “Peers — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/peers/peers.html.

15. “Channel Capabilities — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/capabilities_concept.html.

16. “Smart Contracts and Chaincode — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html.

17. “Ledger — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed June 25, 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html.

18. Rijmenam, Dr Mark van. 2019. “5 Blockchain Benefits for the Financial Services Industry.” Medium. August 28, 2019. https://medium.com/@markvanrijmenam/5-blockchain-benefits-for-the-financial-services-industry-20b760d4bb3a.

19. “We.Trade.” n.d. Www.Ibm.Com. Accessed June 25, 2020. https://www.ibm.com/case-studies/wetrade-blockchain-fintech-trade-finance.

20. Norta, Alex, et al. “Lowering Financial Inclusion Barriers with a Blockchain-Based Capital Transfer System.” IEEE INFOCOM 2019 - IEEE Conference on

Computer Communications Workshops (INFOCOM WKSHPS), 2019, doi:10.1109/infcomw.2019.8845177.

21. Choo, Jaegul, Changhyun Lee, Daniel Lee, Hongyuan Zha, and Haesun Park. 2014. “Understanding and Promoting Micro-Finance Activities in Kiva.Org.” *Proceedings of the 7th ACM International Conference on Web Search and Data Mining - WSDM ’14*. https://doi.org/10.1145/2556195.2556253.

22. “ChainRider Private Blockchain Documentation.” n.d. Chainrider.Io. Accessed June 25, 2020. https://chainrider.io/docs/private-blockchain/#blockchain-as-a-service.

23. “Fabric-Client.” n.d. Npm. Accessed July 16, 2020. https://www.npmjs.com/package/fabric-client.

24. “Fabric-Shim.” n.d. Npm. Accessed July 16, 2020. https://www.npmjs.com/package/fabric-shim.

25. “Apache CouchDB.” n.d. Couchdb.Apache.Org. Accessed July 16, 2020. https://couchdb.apache.org/.

26. "Access Control Lists (ACL)" Hyperledger. Accessed July 16, 2020. https://hyperledger-fabric.readthedocs.io/en/release-2.0/access_control.html.

27. “Peer Channel-Based Event Services — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed July 16, 2020. https://hyperledger-fabric.readthedocs.io/en/release-2.0/peer_event_services.html.

28. “Fabric Chaincode Lifecycle — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed July 16, 2020. https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode_lifecycle.html#deployment-scenarios.

29. “Fabric Chaincode Lifecycle — Hyperledger-Fabricdocs Master Documentation.” n.d. Hyperledger-Fabric.Readthedocs.Io. Accessed July 16, 2020. https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode_lifecycle.html#install-and-define-a-chaincode.

30. Yamashita, Kazuhiro, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun. "Potential Risks of Hyperledger Fabric Smart Contracts." *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2019. https://doi.org/10.1109/iwbose.2019.8666486.

31. "ChainSecurity Chaincode Scanner Analyse Hyperledger Fabric Chaincode for Vulnerabilities." n.d. Chaincode.Chainsecurity.Com. Accessed July 16, 2020. https://chaincode.chainsecurity.com/.

32. "ESLint." 2020. Wikipedia. June 27, 2020. https://en.wikipedia.org/wiki/ESLint.

33. "Standard ECMA-262." n.d. Www.Ecma-International.Org. Accessed July 16, 2020. https://www.ecma-international.org/publications/standards/Ecma-262.htm.

34. Durieux, Thomas, João F. Ferreira, Rui Abreu, and Pedro Cruz. 2020. "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts." *ArXiv:1910.10601 [Cs]*, February. https://doi.org/10.1145/3377811.3380364.

35. Hyperledger. "Hyperledger/Fabric-Samples." GitHub. Accessed July 16, 2020. https://github.com/hyperledger/fabric-samples.

36. Frankenfield, Jake. n.d. "What Is a 51% Attack?" Investopedia. https://www.investopedia.com/terms/1/51-attack.asp#:~:text=A%2051%25%20attack%20refers%20to.

37. Dabholkar, Ahaan & Saraswat, Vishal. (2019). Ripping the Fabric: Attacks and Mitigations on Hyperledger Fabric. 10.1007/978-981-15-0871-4_24.