
Property Registration and Land Record Management via Blockchains

*A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Technology*

by

Gunda Abhishek

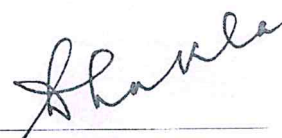


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

June 2019

CERTIFICATE

It is certified that the work contained in the thesis titled **Property Registration and Land Record Management via Blockchains**, by **Gunda Abhishek**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

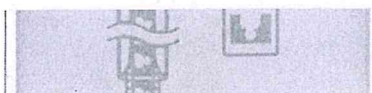


Dr. Sandeep Shukla

Department of Computer Science & Engineering

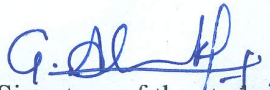
IIT Kanpur

June, 2019



Statement of Thesis Preparation

1. Thesis title: Property Registration and Land Record Management via Blockchain
2. Degree for which the thesis is submitted: ~~M.Tech~~ BT-MT (Dual Degree)
3. Thesis Guide was referred to for preparing the thesis.
4. Specifications regarding thesis format have been closely followed.
5. The contents of the thesis have been organized based on the guidelines.
6. The thesis has been prepared without resorting to plagiarism.
7. All sources used have been cited appropriately.
8. The thesis has not been submitted elsewhere for a degree.


(Signature of the student)

Name: Gunda Abhishek

Roll No.: 14807257

Department/IDP: CSE

Abstract

Name of student: **Gunda Abhishek** Roll no: **14807257**

Degree for which submitted: **M.Tech** Department: **CSE**

Thesis title: **Property Registration and Land Record Management via Blockchains**

Name of Thesis Supervisor: **Dr. Sandeep Shukla**

Month and year of thesis submission: **June 2019**

The birth of Blockchain technology in form of Bitcoin, has triggered a wide interest by demonstrating the possibility of eliminating the need of need of an intermediary and revolutionized the interactions between people and machines by increasing trust. Initially restricted to the domain of cryptocurrencies, people started realizing the potential of technology to go beyond just cryptocurrencies which led to the adoption of blockchain technology to solve real-world scenarios. One such scenario is the problems in E-governance systems among the public domain sectors. For the scope of this thesis we mainly focused on the problems in Land record and revenue sectors.

In this thesis, we analyze the integration of blockchain technology to the existing business processes and in doing so, we address problems such as data integrity, privacy, and more importantly the lack of common platform between the organizations involved. The evolution of blockchain technology led to the introduction of permissioned blockchain platforms, Hyperledger Fabric is a leading permissioned blockchain platform which is used in the development of our project. Finally the objective of this thesis is to evaluate the performance of blockchain based implementation of a land revenue & recording automation system.

Acknowledgements

First and foremost I would like to express my sincerest gratitude to my supervisor, Prof. Sandeep Shukla of the Department of Computer Science and Engineering, for introducing me to this new world of blockchain technologies and allowing me to work on this prestigious project. Without his guidance and invaluable suggestions throughout this thesis would not have been possible.

I feel extremely grateful and indebted to Shubham Sahai Srivastava for all those long discussions and brainstorming sessions during the implementation of this project and my brother for being a pillar of support and always lending me an ear in difficult situations.

I would like to thank my fellow lab mates Devendra Meena and Ajay Singh for their help and support. Finally, I would like to express my gratitude to my family for having faith in me, and for their advice in every stage of my life

Gunda Abhishek

Contents

Abstract	v
Acknowledgements	vi
Contents	vii
List of Figures	ix
List of Tables	x
1 Introduction	1
1.1 Land Records and Titles	1
1.2 Digitization of Land Records	2
1.3 Organization of thesis	3
2 Understanding the technology behind blockchain	4
2.1 Important properties	5
2.2 Evolution of Blockchain	6
2.2.1 Blockchain 1.0	7
2.2.2 Blockchain 2.0	8
2.2.3 Blockchain 3.0	8
3 Current Business Process of Land Registration	10
3.1 Department of Stamps & Registry	11
3.2 Board of Revenue	14
3.3 Need for a Blockchain based platform	15
4 Hyperledger Background	18
4.1 Introduction	18
4.2 Architecture	19
4.3 Projects	20

4.4	Hyperledger Fabric	21
4.5	Transaction Flow	25
4.5.1	Execution Phase	25
4.5.2	Ordering Phase	26
4.5.3	Validation Phase	26
5	Design	28
5.1	Design	28
5.1.1	SDK Layer	30
5.1.2	REST API Layer	31
5.1.3	Verity	31
6	Implementation	34
6.1	Working environment	34
6.2	Backend or Network Layer	35
6.2.1	Chaincodes	37
6.3	Middleware	41
6.3.1	SDK Layer	42
6.3.2	REST Server	43
6.4	Deployment	44
7	Evaluation & Results	49
7.1	Evaluation Metrics	49
7.2	Results	50
7.2.1	Impact of Arrival Rates	51
7.2.1.1	Insert Transactions	52
7.2.1.2	Query Transactions	55
7.2.2	Impact of Block Size	58
8	Conclusion & Future Work	61
8.1	Conclusions	61
8.2	Future Work	62
	Bibliography	63

List of Figures

2.1	<i>Structure of a blockchain</i>	5
2.2	<i>Historical evolution of blockchains</i>	9
3.1	<i>Hierarchical structure of DoSR</i>	12
4.1	<i>Sample Fabric network with two organizations</i>	25
4.2	<i>High Level transaction flow in fabric</i>	27
5.1	<i>Initial design architecture</i>	29
5.2	<i>Bridging Organizations with channel</i>	30
5.3	<i>Working of verity</i>	32
5.4	<i>Final design</i>	33
6.1	<i>Network configuration of the project</i>	36
6.2	<i>SQL schema of the sale-deed database</i>	39
6.3	<i>SQL schema of the land records Database</i>	40
6.4	<i>Sale deed asset</i>	40
6.5	<i>Land record asset</i>	40
6.6	<i>Assets used in the project</i>	40
6.7	<i>Deploying container to specified instance</i>	48
7.1	<i>Impact of arrival rates on performance</i>	53
7.2	<i>Impact of arrival rates on performance</i>	54
7.3	<i>Impact of arrival rates on ②</i>	55
7.4	<i>Impact of arrival rates on ④</i>	55
7.5	<i>Comparison of configurations ② and ④</i>	55
7.6	<i>Impact of arrival rates on performance</i>	56
7.7	<i>Impact of arrival rates on performance</i>	57
7.8	<i>Impact of Block on performance</i>	59

List of Tables

6.1	Summary of the keys involved in the assets	41
6.2	Details of the instances	45
6.3	Distribution of services within the instances	46
7.1	Various Network Configurations	51
7.2	Configuration for impact of Arrival Rate - ③	52
7.3	Configuration for impact of Arrival Rate - ①	54
7.4	Configuration for impact of Arrival Rate - ④	56
7.5	Configuration for impact of Arrival Rate - ①	57

List of Abbreviations

S.No	Term	Abbreviation
1	RoR	Record of Rights
2	PoW	Proof of Work
3	DoSR	Department of Stamps & Registry
4	BoR	Board of Revenue
5	SRO	Sub Registrar Office
6	DLT	Distributed Ledger Technology
7	SMR	State Machine Replication
8	MSP	Membership Service Provider
9	CA	Certificate Authority
10	SC	Smart Contract
11	CC	Chaincode
12	SDK	Software Development Kit

Dedicated to my Brother

Chapter 1

Introduction

Technology in its relationship to the global changes has always been at the forefront to disrupt the status quo and bring in a new fresh perspective to look upon the things. When it comes to the public domain related industries, the technological changes often end up entangled with the policy regulations and are presented with a great deal of difficulty while scaling up. One such example is E-governance and in particular *Property Registration and Land Records Management*. In this thesis, we go deeper into the understanding of the above example while conferring it to the Indian Context and the existing problems associated with it. The main purpose of this thesis is to assess whether integrating with the blockchain technology is a good solution or not. We will look at the properties of blockchain technology that makes it a desirable solution for the existing problems and finally we will present you the results obtained during performance evaluation of the system implemented.

1.1 Land Records and Titles

One of the biggest challenges that grip various states in India is the issue of Land ownership, which is partly due to the fact that there is no end-to-end effective Land

records management system till date. The issue of Land ownership is so critical that almost all financial institutions heavily depend on the ownership of the Land-based properties for collateral security. Due to the uncertainties on the ownership claims, it becomes more robust for the financial institutions to function properly with ease and thus hampering the growth of nation.

“Land Ownership in India is presumptive.”

The Land ownership in India is mainly established via the sale deeds(procured during the property registration) and some other documents. However, all these documents don't guarantee the claim of ownership but rather establish the fact that a transfer of property has been recorded and these recorded transactions are what we call a "Land Records".

The word "Land records" itself is a generic expression and is a culmination of many other documents providing various other information such as property maps, Record of Rights(ROR), sale deeds and others and these are maintained across different departments. Due to the poor maintenance and the lack of communication across the departments, we very often find discrepancies, outdated and data being out of sync in Land Records. These discrepancies are the root cause of ownership problem as one has to go back several years of records to verify the ownership claim.

1.2 Digitization of Land Records

The need for clear titles and conclusive land ownerships have been fairly detailed in the previous section. Thus over the last three decades, various schemes have been implemented in Modernizing the land records to move towards a clearer and conclusive land ownership from the presumptive property titles. We are not going deeper into the details of how modernization has been implemented and for the purpose of this thesis, we assumed that the land records are completely digitized.

1.3 Organization of thesis

The remaining part of this thesis is organized as follows. In Chapter 2 we look into the background of blockchain technology, the important properties associated with it and the evolution of this technology over the past few years. In Chapter 3 we did a literature study on the existing process of property registration and land record management, and in the process of doing so we will look at the departments involved and their works and the major problems associated with the existing process. Also in Chapter 3 we will argue about why blockchain technology is a desirable solution. In Chapter 4 we will look into the background of Hyperledger technologies, in particular, Hyperledger Fabric and their architectures. In Chapter 5 we will define the architecture of the blockchain network and the various layers of the design. Chapter 6 deals with the implementation of the defined architecture in Hyperledger fabric in a single-host environment first and later in a more distributed network. The implementation in chapter 6 is evaluated in Chapter 7 over different evaluation metrics and their comparisons. Finally, in Chapter 8 we provide some conclusions to the thesis work and some directions for the future work.

Chapter 2

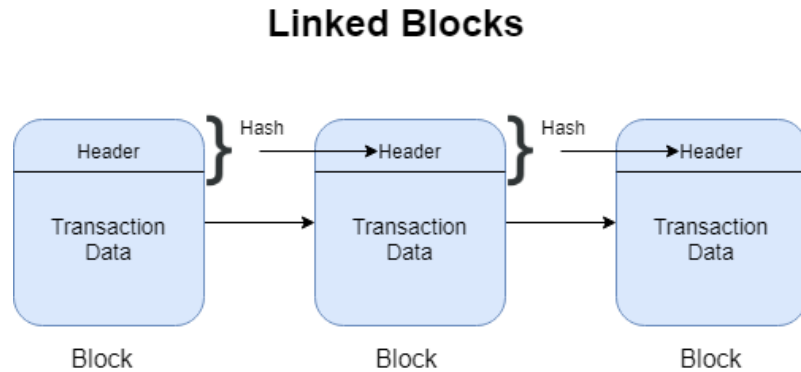
Understanding the technology behind blockchain

In the year of 2008, an anonymous developer or Group of developers going by the name of **Satoshi Nakamoto** published a white paper with the name of "**Bitcoin:a peer to peer electronic cash system**" [1]. In the paper while detailing about the project they coined a phrase *block chain* [2] which essentially is the backbone behind their project "bitcoin". The bitcoin came to prominence as it has successfully resolved the problem of **Double spending** [3] and eliminating the requirement of having a middle-man(Intermediary), which provides trust and security to the transaction by introducing a decentralized architecture and data storage to increase the trust and verifiability of a transaction.

Although the terms bitcoin and blockchain go side by side but, Bitcoin is just another cryptocurrency with **Blockchain** functioning at its heart and thus providing the following features.

Blockchain, as the name suggests is essentially a chain of blocks which are linked via cryptographic hashes. Each block is a data structure and comprises of

1)A cryptographic hash of the previous block 2)Transaction data 3)Timestamp

FIGURE 2.1: *Structure of a blockchain*

1. **Cryptographic Hash Function:** This can be thought of as a function which maps a given input of variable size to a string(hash) of fixed length and is implicitly a one-way function i.e easy to compute the output but reverse-engineering of input from the output is nearly impossible. One more ideal property of a hash function is the **Avalanche Effect** i.e a small change in the input should drastically change the output making it not correlatable to the original hash value.
2. **Transaction Data:** All the transactions in a given time frame are collected and are added into a block. The data being added records the transactions and the chain of blocks via hash linking creates a trail of transactions and thus making an analogy between a blockchain and a ledger.

2.1 Important properties

Decentralization: We have already seen that when bitcoin first came into the picture, the decentralization property is the one which introduced the trust and thus eliminating the intermediary from the picture. The decentralized architecture also brings the distribution of power among the nodes i.e. no single node has the authority over the network and each node will be sharing the same data as the other. This distributed replica of the same set of chained blocks over the network brings

the blockchain to be called a distributed ledger.

In a traditional centralized architecture, the central node holds the responsibility of adding the blocks but in our case, the onus is on the distributed nodes to somehow validate the blocks and link them to the chain, and in the context of blockchain this is done via consensus mechanisms. This process of having consensus is what makes sure that every single node on the system agrees on the state of the blockchain.

Immutability: In the context of a blockchain, the word immutability mainly refers to the transaction data. Once data is entered it can't be tampered with. In the context of a blockchain, the cryptographic hash linking of the blocks makes sure of the immutability. We already know that a small change in the data drastically changes the value of its hash. This newly computed hash has to be matched with the already linked chain of hashes. Also, the property of decentralization makes a very important role here because if a corrupted node tries to modify the data on its local state this has to be accepted during the consensus mechanism by all the other nodes which makes it difficult to modify the data.

So in a nutshell blockchain can be defined as an **immutable ledger, maintained within a decentralized network of nodes.**

2.2 Evolution of Blockchain

In the previous section, we have seen the origin of blockchain technology in the form of bitcoin, how a block chain works, important properties that make this technology tick. In this section, we are going to talk about the evolution of blockchain technology and how it has surpassed the barriers of being merely the technology supporting

a cryptocurrency to having the capability of creating an impact on the real world problems.

2.2.1 Blockchain 1.0

Most of the world still believes that Bitcoin and Blockchain are one and the same. Such was the hype around the bitcoin that led people to not see the potential of the Blockchain Technology which essentially is at the core of the Bitcoin. The Blockchain technology came to prominence after the release of a white paper **Bitcoin: a peer to peer cash electronic system** by an anonymous developer or group of developers going by the name of **Satoshi Nakamoto**. As we have earlier seen that the blockchain is an immutable, tamper resistant distributed ledger of transactions. The decentralized architecture of Blockchain has successfully eliminated the need of an intermediary and increased the trust between parties which paved the way for Bitcoin to be a successful cryptocurrency as an alternative to the traditional banking system.

The important properties of Bitcoin(Blockchain 1.0) are given as follows

- Successfully resolved the double spending problem
- Decentralization of currency and financial transactions
- Consensus Mechanism(Proof of Work) to verify all the transactions and maintaining the same state of blockchain across all the nodes in a distributed system.

In addition to the above properties, the Bitcoin blockchain also supports scripting language(Bitcoin Script) [4], this is often used to create the public and private keys to spend the transferred bitcoins. The very big limitation of Bitcoin Scripting language is that it is not a Turing complete which means that the automated system of the Bitcoin script cannot support real world human transactions.

Soon after the world started realizing the potential of blockchain technology which can go beyond just cryptocurrencies and can be used to solve some real-world problems. Thus the work on extending the Bitcoin scripting language began which led to the development of what we now call a **smart contract**.

2.2.2 Blockchain 2.0

Vilatek Buterin who was a member of the bitcoin community witnessed firsthand the above-discussed problems of bitcoin. In early 2014, he described his vision of Ethereum [5] to becoming a blockchain going beyond cryptocurrency. The launch of Ethereum blockchain not only gave the world a newer and better cryptocurrency in the form of "ether", it also gave the blockchain community a new perspective on using the blockchain technology by the introduction of developing "Decentralized Applications(DApps)" by utilizing the **Smart Contracts**.

2.2.3 Blockchain 3.0

The blockchain evolution has not stopped with the emergence of Ethereum. For sure ethereum has ushered in a new era of blockchain but it still has a lot of deficiencies and the new evolution of technology has sought to overcome these past deficiencies and has introduced some new features along the way.

When we look at the blockchain technologies in sections 2.2.1 and 2.2.2, the major concern was the privacy of the transactions/data on the blockchain. We know that both the Bitcoin and Ethereum blockchains are public blockchains i.e all the transactions entering the blockchain are publicly visible to the world. Also in case of the public blockchains, scalability is a very big issue as achieving consensus over a very large set of nodes is time-consuming and requires a very large computational power. So this level of privacy and scalability are concerning if we are to integrate blockchains into enterprise level solutions.

So in any typical real-world application, emphasis is mainly put on the data being shared in a cross-organizational workflow . Now if we are to integrate blockchains into these kind of scenarios, public blockchains are not a good fit because of the privacy concerns of the data being shared and not all users should have permission to join the network and have a stake in the consortium.

So the next evolution of blockchain has made a lot of improvements in order to overcome these deficiencies and the most important among them was the development of **private** and **permissioned** Blockchains. The following are some of the blockchain platforms that came into existence in the evolution of Blockchain 3.0, Hyperledger Fabric(Permissioned Blockchain) [6], Cardano [7], Zilliqa [8] , Zcash [9] e.t.c.

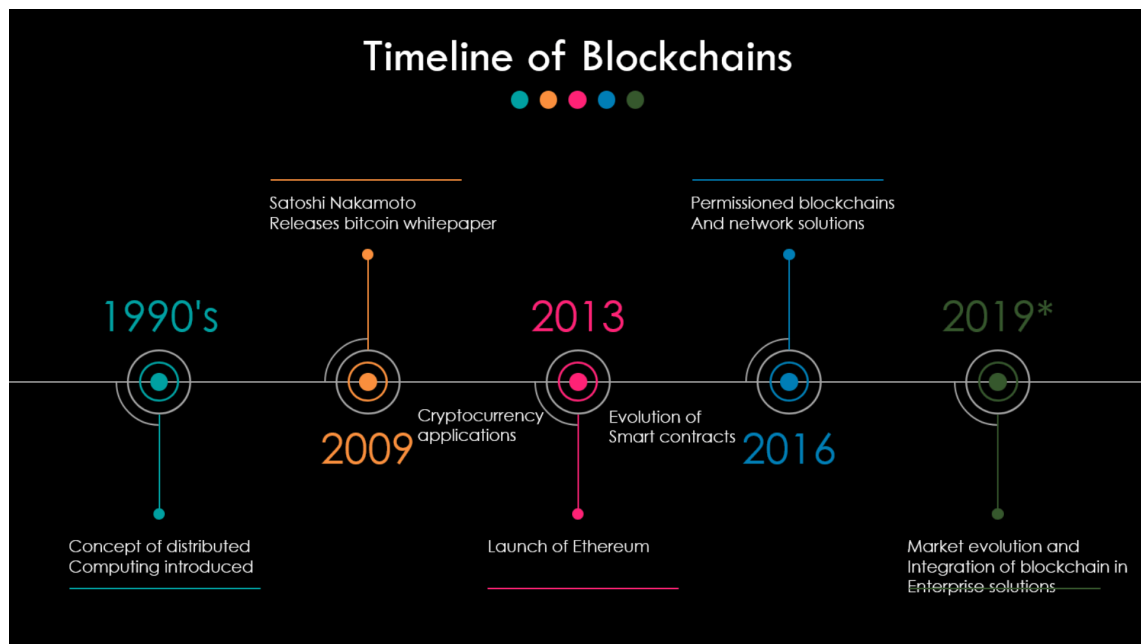


FIGURE 2.2: *Historical evolution of blockchains*

Chapter 3

Current Business Process of Land Registration

In this chapter, we describe the current scenario of the property Registration and the transfer of land in the Indian context and in particular for the state of Uttar Pradesh. We found that almost all the states in India have more or less adopted the same kind of the workflow, so we can assume that the findings we provided for the state of Uttar Pradesh can be extended to other states as well.

We found that in almost all the states of India usually two or three departments handles the Land related documents [10] and they are given as follows.

1. **Department of stamps & Registry(DoSR):** It handles the verification of documents during the registration of a transfer. It also has the added responsibility of collecting the stamp duty of the sale deed to be legally recorded.
2. **Revenue Department or Board of Revenue(BoR):** It handles the preparation and maintenance of the Textual Records or the transaction details
3. **Survey and Settlement department:** It handles the preparation and maintenance of the maps.

In this thesis, we have ignored the Department of Survey and settlement department as we found that the irregularities are too huge, and there is no record of having the spatial/geographic mapping of the records. We describe the functioning of the **DoSR** and **BoR** departments in further details over the next two sections.

3.1 Department of Stamps & Registry

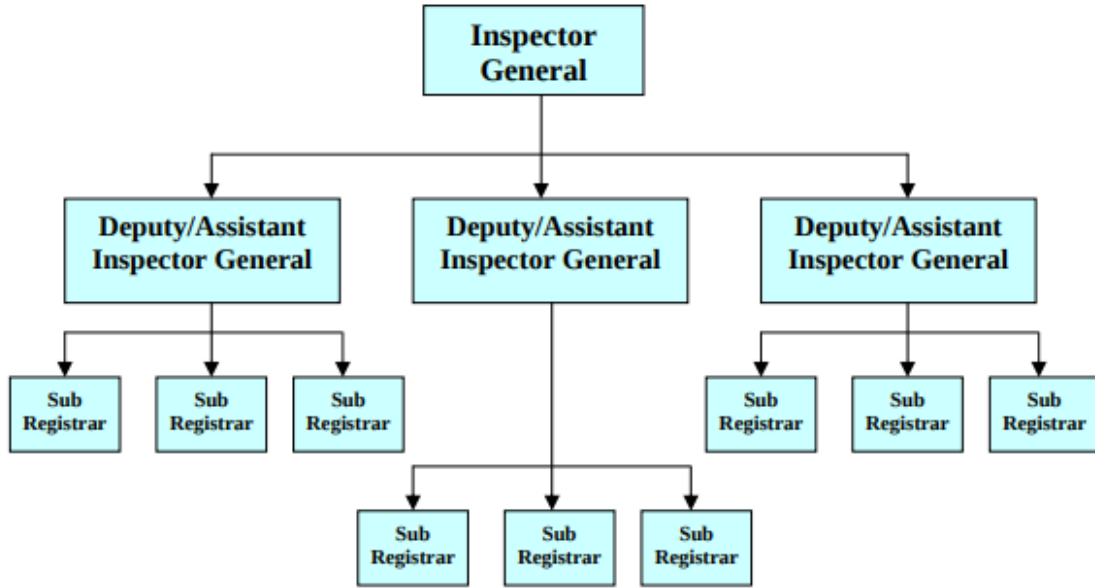
The DoSR currently handles around 62 different types of deeds over various domains such as transfer of immovable property, Marriage registration and bond papers etc. For the purpose of this thesis, we confine ourselves to the deeds related to the Land/Property Registration.

Given below are the different kind of possibilities in which you can acquire immovable property in India:

- Property inheritance
- Through Will
- Purchase of the property
- Property being granted by government or court

Each of these above transactions go under different types of deeds and thus differ in their work-flow. For this thesis, we will focus mainly on the registration of property acquired via purchase.

In the state of Uttar Pradesh the following three-tier hierarchical structure exists in the DoSR:

FIGURE 3.1: *Hierarchical structure of DoSR*

This hierarchical structure is later used in defining the role-based authorization of the information existing on the blockchain, thus restricting the access of sensitive information. The office of Sub Registrar(SRO) is the place where all the registration work takes place and has maximum interaction with the public. The various responsibilities of the office are listed below:

1. Estimation of property value
2. Collection of stamp duties
3. Registration of sale deed
4. Preservation of copies of sale deed
5. Issuing certificates of the registration

Under the registration act of 1908 [11], all the transactions that involve the sale of any immovable property have to be registered, thus ensuring all the information

on the transactions is valid and maintained. Owing to the digitization of all the documents, the property registration in India has become quite simple. Although the introduction of online portals have their advantages in the form increasing transparency in the evaluation of properties, it has not completely replaced the manual intervention where at stages, the user has to be manually present at the SRO for authentication purposes.

Given below are the steps during the registration of Immovable property in Uttar Pradesh:

1. **Verification of the Property:** This aspect of the registration generally lies with the buyer to have a thorough check on the title of the property. As earlier said in chapter 1 the land titles in India have a lot of inconsistencies and the transition to the online portals have not yet repaired the problems so the onus is on the buyer to verify the past ownerships of the property.
2. **Property Value Estimation:** Estimation of property value has to be done in order to pay for the stamp duty for the registration, which will be calculated as a percentage of the circle rate in the area.
3. **Preparation of stamp papers:** Non-judicial stamp papers equivalent to the value of the stamp duty are to be bought which are now made available online
4. **Stamp duty payment:** This part can now be done via online portal
5. **Registration at the SRO:** This part of the registration process requires manual intervention at the SRO for the authentication of the buyer, seller along with the witnesses.

3.2 Board of Revenue

As discussed earlier, in the state of Uttar Pradesh the BoR deals with the preparation and maintenance of the transaction details. The duties of the department differ greatly with the location of the immovable property i.e whether the property belongs to an urban category or rural. In the case of former the transfer of immovable property ends with the preparation of sale deed and the details of the transaction are recorded at the city municipal office, and the sale deed acts as the title document. Whereas in the case of latter the BoR has some added responsibilities of verifying the transactions and in case of any disputes a revenue court is held. We describe the latter part in a more detailed way over the following sections.

Similar to the hierarchal structure of DoSR in figure 3.1, the organizational structure of BoR is three-tier and the sub-divisional office sits at the lower level of the hierarchy and has the maximum interaction with the public.

There are around 22 documents maintained by the BoR which can be considered as the term "Land Records". The other important task undertaken by the BoR is the mutation proceedings of immovable property. So **mutation**, as the name suggests refers to the change in the ownership of the property in the BoR records and we look at the two main possibilities in which a mutation can happen.

1. **Succession:** As earlier discussed in the above section, we have looked at different ways in which you can obtain an immovable property. The cases of *property inheritance* and *through will* comes under the Succession category.
2. **Transfer:** The transfer of property via purchase is another category in which a mutation proceeding is initialized.

Although there are many other possibilities of initiating a mutation proceeding, in this thesis we confined ourselves to the above two discussed categories.

The mutation proceedings typically start with the submission of the sale deed procured at the DoSR followed by the verification of the property title as mentioned on the sale deed, if there are any disputes regarding the transaction the revenue courts takes the responsibility of the giving the final verdict. If the mutation goes through a new entry is added to the database updating the owner and the details of the transaction.

3.3 Need for a Blockchain based platform

Over the course of this chapter, we detailed the various responsibilities of the departments involved and the procedures of transferring an immovable property. In this section, we list out the problems in the existing solution and in the process of doing so, we argue why we think blockchain is a good solution to these problems. Despite the digitization of land records the existing process still faces many problems and security threats. We believe that the following are mainly responsible.

1. **Centralized Architecture:** We have already mentioned about the transition to the digital records and the online portals that have come into existence over time. But the important thing to consider is that all these transitions mentioned, work under traditional centralized databases. In any centralized database, the architecture is similar to a hierarchical pyramid where people up the chain are in a position to abuse power. These kinds of power abuse intentionally or unintentionally may be classified in general as an **Insider Attack**. In information security, insider attacks pose a tougher problem to be dealt with, as a malicious insider is too well concealed to be detected by traditional methods. There is a well-known argument that using proper logging mechanisms the insider attacks can be detected. We believe that with all the applications relying on a centralized architecture even the logging mechanisms any employee with admin privileges can attack the system and even corrupt

the logging mechanism making him/her untraceable. This over-dependence on the traditional centralized architecture is what blockchains aim to prevent. Earlier in the section 2.1, we mentioned the important properties of blockchains are decentralization and the immutability of data and we believe that these properties can certainly help to prevent the earlier mentioned attacks of a centralized architecture.

2. **Integrity of data:** This refers to the consistency, accuracy, and completeness of data, it is the assurance that the data we are accessing is attributable, legible, contemporaneously recorded, original and accurate often referred to as **ALCOA** [12]

- (a) **Attributable:** Data changes should be traceable i.e if data is changed then why and who should be accountable for the changes.
- (b) **Legible:** Data must be recorded permanently in a readable manner
- (c) **Contemporaneous:** Recorded data should be followed by the timestamps
- (d) **Original:** The recorded data should be original and not a certified true copy
- (e) **Accurate:** Modification of records should not be allowed without proper documentation and authorization

More often than not in traditional centralized database systems, the integrity of data is often not maintained i.e we don't have the absolute certainty that the data being shown on the application is actually the data that has been recorded originally. We have earlier seen that the blockchains are immutable, distributed ledgers and the transactions that are once recorded cannot be modified. Also the distributed architecture of the blockchains helps in preventing the unauthorized modification of data via consensus mechanisms.

3. **Trust in cross-organizational workflow:** We have seen that in a property transfer, both the departments are involved. They work on this shared data

for example in mutation proceedings of a property. At the BoR the verification of sale deed procured at the DoSR is done and the process is moved forward. More often than not, these departments work in silos and the data that is being shared across is not being properly updated and thus leads to lack of trust between the organizations. In the section [2.2.3](#), we have discussed about the public blockchains not being suitable in a cross-organizational workflow so we believe that a blockchain 3.0 platforms cater to the above needs perfectly.

4. **Public verifiability:** During the preparation of sale deed in the DoSR, we have earlier discussed that the onus is on the buyer to have a thorough verification of the property title. Even after the digitization of the records, they often seems to be out of sync or not up to date. This is partly due to facts that the current system is prone to insider attacks or due to the lack of trust between the parties involved. Due to these discrepancies, the buyer has to go back to several decades of documents to verify the property titles and such a process is highly inefficient. If the blockchain technology is successfully integrated to the current process, the public verifiability can be easily provided using the cryptographic hash properties of the transactions and the hash linkage of the blocks can be used to provide the traceability of a particular record.

Chapter 4

Hyperledger Background

4.1 Introduction

Hyperledger [13] first started in 2015 is an open source collaborative effort of blockchain and related technologies by Linux Foundation [14]. The earlier blockchain initiatives such as Bitcoin, Ethereum are designed as cryptocurrency alternatives running on public blockchain platforms. In the section 2.2.3, we have seen that the public blockchains are not suitable for the enterprise use cases. With these limitations in mind, hyperledger aims to provide support for integrating blockchain technology in real world business applications. The term **Hyperledger** does not refer to any blockchain technology as such but it refers to a dynamic community of developers, startups, and enterprises working together to advance the blockchain technology for business purposes.

4.2 Architecture

The requirements of a business blockchain network from application to application. In order to deal with these constantly changing requirements, hyperledger hosts and promotes the development of many blockchain technologies. A detailed list of projects hosted by hyperledger community are discussed in the following section. All the hyperledger projects follow a design philosophy which includes modular extensible frameworks, ability to inter-operate between frameworks and support for private and permissioned networks etc. The following listed components are what constitutes a Hyperledger architecture [15]

- Consensus Layer
- Smart Contract Layer
- Communication Layer
- Data store abstraction
- Crypto abstraction
- Identity services
- Policy services
- Interoperation

The detailed description of these components are mentioned during the explanation of Hyperledger Fabric in section 4.4.

4.3 Projects

All the projects hosted by the hyperledger working group can be classified under two categories and corresponding projects under the respective categories are listed below:

1. Distributed ledger technology(DLT) frameworks:

- **Hyperledger Burrow** [16]: Modular framework allowing the adaptation of ethereum smart contracts in enterprise networks.
- **Hyperledger Fabric** [17]: The most renowned framework of hyperledger, intended as a platform for building DLT solutions. It supports the usage of various components like consensus mechanisms, Smart contracts, Identity management etc.
- **Hyperledger Indy** [18]: Mainly built for decentralized identity management
- **Hyperledger Iroha** [19]: A simple easy-to-use modular framework with more emphasis on the end-user application development. It doesn't support the use of smart contracts or any other business logic.
- **Hyperledger Sawtooth** [20]: Most advanced framework yet developed under hyperledger. Uses a new form of chain-based consensus mechanism PoET¹. Designed to advance the scalability, privacy and security features of the earlier DLT frameworks.

For this thesis, we have implemented the project on the **Hyperledger Fabric**. A more detailed description of the components of fabric is discussed later in the chapter.

2. Tools:

¹Algorithm details can be found at <https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html>

- **Hyperledger Cello** [21]: Aims to simplify the management of networks and bring the deployment as a service model
- **Hyperledger Composer** [22]: A plug-and-play tool to create and deploy business networks over run-time DLT frameworks.
- **Hyperledger Explorer** [23]: A web-based application providing the visibility of functioning business network running on a DLT framework.
- **Hyperledger Caliper** [24]: A benchmarking tool for the performance analysis of a project made using any of the above defined DLT frameworks.

We have integrated some of the above-mentioned tools in the thesis project, the details and results of these tools are explained in the later chapters.

4.4 Hyperledger Fabric

Hyperledger Fabric [6] is one of the DLT frameworks developed within the hyperledger community. Fabric like other blockchain technologies is also an immutable ledger, maintained within a decentralized network of nodes. Fabric being 3.0 evolution of the blockchain technology has all the functionalities of the traditional 1.0 and 2.0 evolutions of blockchain technology.

We know that from the evolution of 2.0, blockchains have the capability of executing programmable logic in the form of smart contracts. This execution of smart contracts closely resembles the blueprint of State machine replication(SMR) [25] with some deviations. In fact, most of the existing blockchains follow the SMR with the implementation of *active replication* [26].

Active replication: This is a protocol used to achieve consensus in a distributed network. We first order the transactions and disseminate them to all the existing peers. In the second phase, each peer in the network executes all the received transactions sequentially. This architecture of ordering and executing the transactions is

called a *order-execute architecture*.

The order-execute architecture can be found in almost all the existing blockchains including the public blockchains like Bitcoin, ethereum to the permissioned blockchains such as Chain [27], quorum [28]. Although the order-execute architecture makes the blockchains to be more consistent and deterministic there are some limitations and issues as well.

In contrast to the existing architecture, Hyperledger fabric uses an *execute-order-validate* [29] architecture for the execution of transactions. This separates the flow of transactions in three different phases:

1. **Phase-1:** Execution of transaction with correctness check and thereby endorsing the transaction
2. **Phase-2:** Ordering the transactions through a consensus protocol
3. **phase-3:** Validation of transactions against the endorsement policies and consistency of updates (discussed later) are performed at each peer and thereby adding the transaction to the ledger

The above-listed phases are described in great detail during the explanation of transaction flow in section 4.5. Like all other DLT frameworks developed within hyperledger, Fabric is designed to be a modular and an extensible permissioned ledger aimed at advancing the scalability, confidentiality and privacy aspects of the blockchain technology.

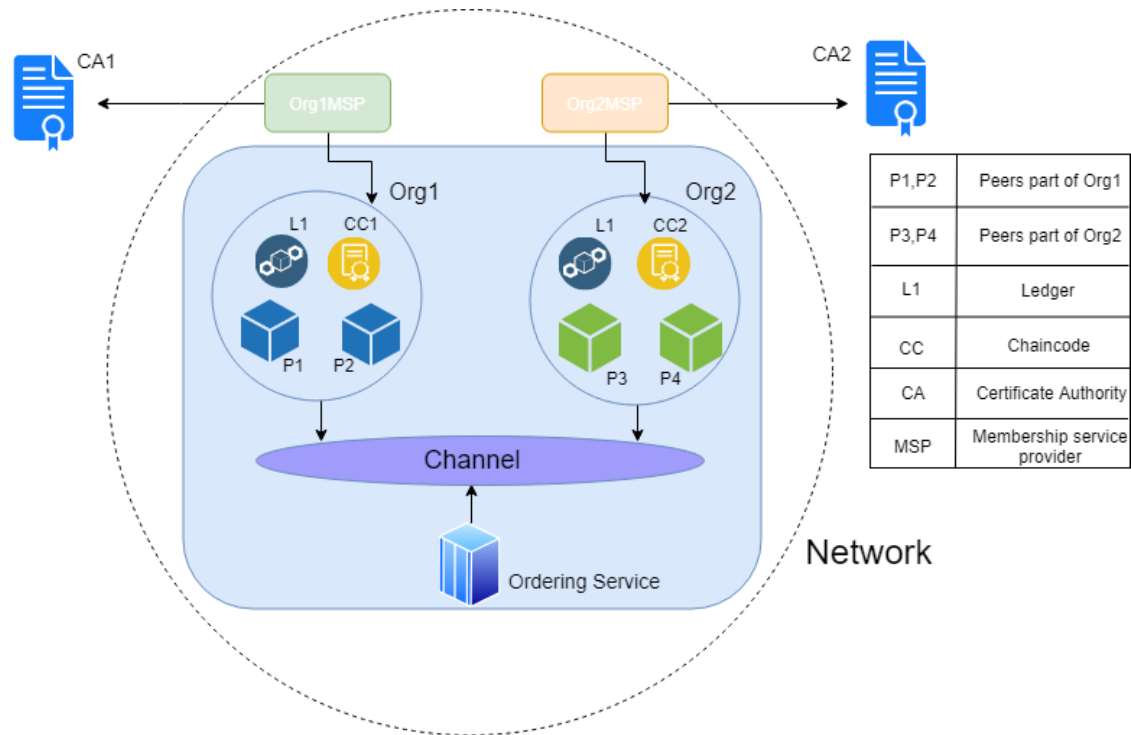
Permissioned network, privacy of transactions and identity Management are the most important characteristics of the Fabric which makes it break path from the traditional blockchain. The following listed are the major components in a fabric blockchain:

1. **Peers:** We know that blockchain is nothing but a distributed network of nodes (in this case peers) which makes the peers the building blocks of a blockchain

network. In fabric a peer by definition can host multiple instances of ledgers and smart contracts (defined later). Peers are classified into three categories which are given below:

- **Endorsing Peers:** Receive transaction proposals from the client application for endorsements. Smart contracts must be present to execute the proposal
 - **Committing Peers:** Commits the transactions and maintain the state of the ledger.
 - **Ordering Peers:** Special type of peers where the endorsed transactions are received, ordered into blocks and are distributed to validating peers for validation and the updation of their respective ledgers.
2. **Ledgers:** The term ledger in hyperledger fabric is associated with two distinct parts – a State Database and transaction log. The state DB records the current values of assets and makes it easy for the developer to access the values instead of traversing the entire chain. Fabric allows the support of two DB technologies – LevelDB [30] and CouchDB [31]. LevelDB is the default and deals with the simple objects and the latter deals with JSON documents making it suitable for the complex objects and allows rich queries. The transaction log is nothing but blockchain which records all the transactions that result in the current values of the assets.
 3. **Membership service provider(MSP):** In every permissioned blockchain, all the interactions between the nodes must be authenticated i.e each node in the network must be having an identity so that the authentication check can be performed. The MSP in fabric is the component which provides the credentials or certificates to all the participating clients and peers in the network and maintains their identities. For the provision of certificates the MSP uses a certificate authority (CA) to issue and verify the certificates. By default, Fabric-CA is used in the Fabric Blockchain.

4. **Organizations:** Also referred to as members, the organizations are a group of peers. Each organization is mapped to its own MSP which gives the identity to all the peers belonging to its own organization. A collection of organizations in a network form a Consortium.
5. **Channels:** Channels allow a specific set of peers of one or more organizations and applications to communicate with each other within a network. Each peer on the channel is associated with a unique ledger. In a fabric network, there can be multiple channels thus making the channels somewhat similar to private blockchains.
6. **Ordering Service:** Comprised of a group of orderer nodes with the task of broadcasting endorsed transactions to the validating peers and ensuring the ordering of all transactions in a delivery block. At present fabric supports three varieties of ordering service, namely SOLO [32], KAFKA [32] and RAFT [33].
7. **Smart Contracts:** These can be defined as a code of programmable logic which can be executed to modify the state values of assets on the ledger. Chaincode and smart contract(SC) are often used interchangeably in the context of fabric but SC's defines the transaction logic and are packaged into chaincode which is deployed on a peer. It is important to note that in a fabric network ledger can only be accessed through chaincodes. Fabric supports the use of high-level programming languages like Go [34], NodeJS, Javascript or Java in developing the chaincodes.

FIGURE 4.1: *Sample Fabric network with two organizations*

4.5 Transaction Flow

We have earlier talked about the execute-order-validate paradigm of the fabric blockchain. In this section we will give a general view of how a transaction executes in a fabric blockchain.

4.5.1 Execution Phase

The client application sends a transaction proposal to one or more endorsers for the execution based on the endorsement policy. We have earlier mentioned that the endorser peers must have chaincode installed in them. The endorsing peers execute the transaction on the local state DB of the peer and then endorses the transaction and sends it back to the client. The client collects the endorsements

from the peers until the endorsement policy specified in the chaincode is satisfied. The client application then packages the transaction along with endorsements and submit it to the ordering service

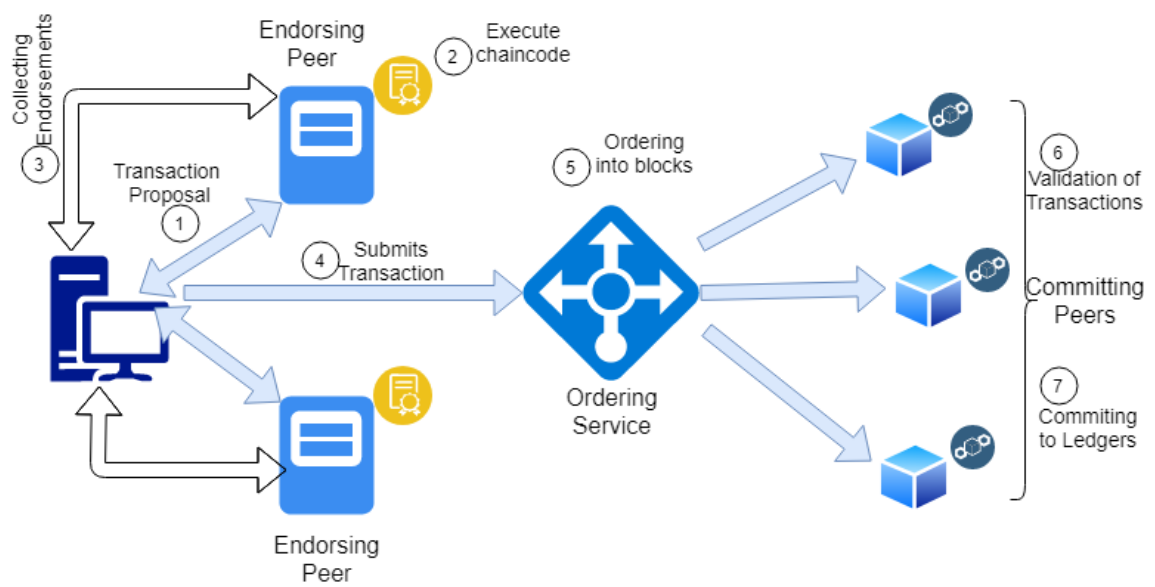
4.5.2 Ordering Phase

In the ordering phase, the orderer nodes validate the transactions against their respective endorsement policies. The next step is to agree on the ordering of the transactions according to the ordering mechanisms put in place which can be – namely SOLO, Kafka and RAFT. The orderers then batch multiple transactions into blocks and these blocks are broadcasted to the validating peers.

4.5.3 Validation Phase

Every block that has entered into the validation phase will execute the following steps:

- Validation of transactions against their respective endorsement policies is done
- All the transactions of a block undergo a read-write conflict check where the read-set is compared to the respective values of the write-set. If a transaction violates the conflict rules, it is marked invalid but it remains in the block with invalid tag.
- At the last stage, the local state DB of the peer is updated to the write set of the transaction and the block is appended to the transaction log of the peer.

FIGURE 4.2: *High Level transaction flow in fabric*

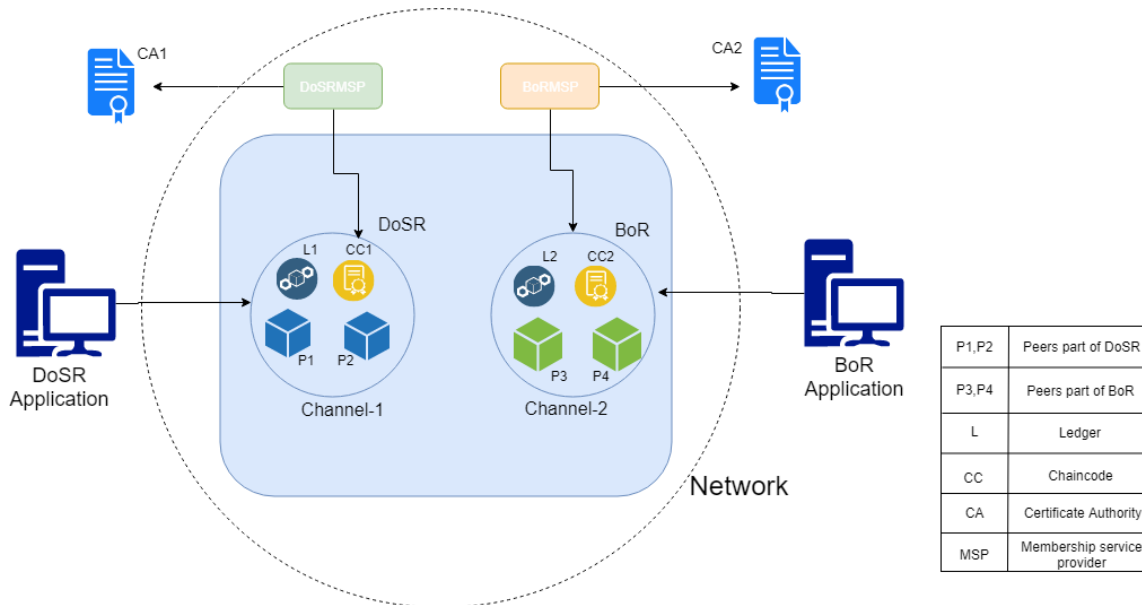
Chapter 5

Design

We have seen earlier in chapter 3 about the existing business process, the problems associated and the advantages of integrating blockchain technology. In this chapter, we are going to look at the design and implementation details of the Hyperledger Fabric integration to the existing process of land record maintenance & registration.

5.1 Design

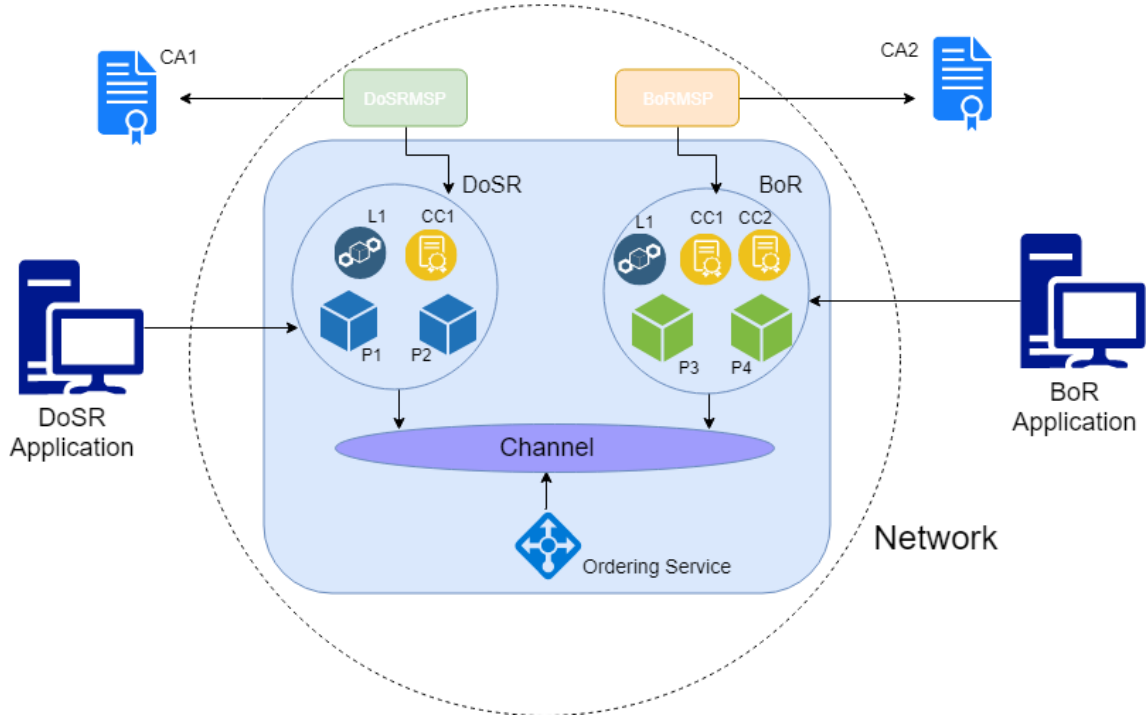
In the context of hyperledger fabric, we can see that an analogy can be drawn between the departments involved in the existing process and organizations in a fabric network. We know that in Fabric, the ledger is associated with a state DB modeled around assets and a transaction log. This can be used to serve the role of the database in the existing architecture. Chaincodes installed on the peers can be used to modify the ledger data, thus mimicking the functionalities of traditional database application.

FIGURE 5.1: *Initial design architecture*

When we talked about the problems in the existing architecture we mentioned that the data Integrity and the gaps in the cross-organizational workflow are the major ones.

By modeling the entire database on a ledger, we are ensuring the integrity of the data as we know from the properties of the distributed ledger technology, the data on the ledger is immutable and resistant to illegal modifications.

We now know that in a hyperledger fabric network, each channel is associated with a single ledger and the ledger can only be accessed by the peer on the same channel. In the current business process, the BoR do not have any way of communication with the DoSR as they are not connected and thus there is no way of digitally verifying the sale deed submitted during the mutation process. This gap can be bridged by bringing both the organizations on a single channel and by installing the chaincode of DoSR on the BoR. We can therefore verify the integrity of sale deed.

FIGURE 5.2: *Bridging Organizations with channel*

5.1.1 SDK Layer

By installing the chaincode CC1 on the BoR peer we are implicitly allowing the illegal modifications of the DoSR ledger data. To stop these modifications an authorization check needs to be performed before invoking the chaincodes.

In a fabric network, we know that the client application sends a transaction proposal to the endorsing peers which invokes the chaincode. This interaction between the client application and the fabric network usually happens through the Hyperledger Fabric SDK. Currently, Fabric has officially released support for two SDK's for Node.js and Java. The identity of any client application trying to invoke a chaincode can be accessed using the SDK layer, which can be used to restrict the access and thus preventing the illegal modifications.

More details on the SDK Layer and the identity check are discussed later describing our implementations in Chapter 6

5.1.2 REST API Layer

In the earlier section, we have seen that the invocation of a chaincode happens through the Fabric SDK. We have seen that in the context of Hyperledger fabric, the terms smart contracts and chaincodes are often used interchangeably but an SC refers to the transaction logic and multiple SC's can be involved in a chaincode.

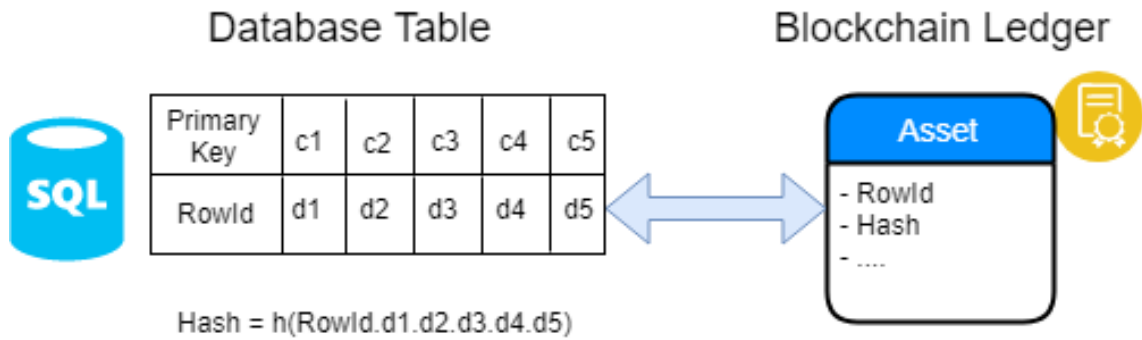
For every smart contract in the chaincode a corresponding function for the invocation is defined in the SDK layer. To hide the details of the smart contracts to the end user, a REST server is used to expose the endpoints of the SDK Layer.

5.1.3 Verity

We have earlier mentioned that modeling the database onto the ledger can solve the problem of data integrity but not without a cost. We know that blockchain technology is essentially a distributed network of nodes maintaining the same state of the ledger. During this maintenance, the data is replicated among all the nodes and with heavier data on the blockchain leads to slower transaction speed. So instead of migrating the entire database onto the blockchain, we need to figure out a way to protect the integrity of the data.

Verity [35] is a dataless framework residing between the database layer (SQL DBMS) and the blockchain network which claims to prevent the illegal tampering of data all the while preserving the data privacy.

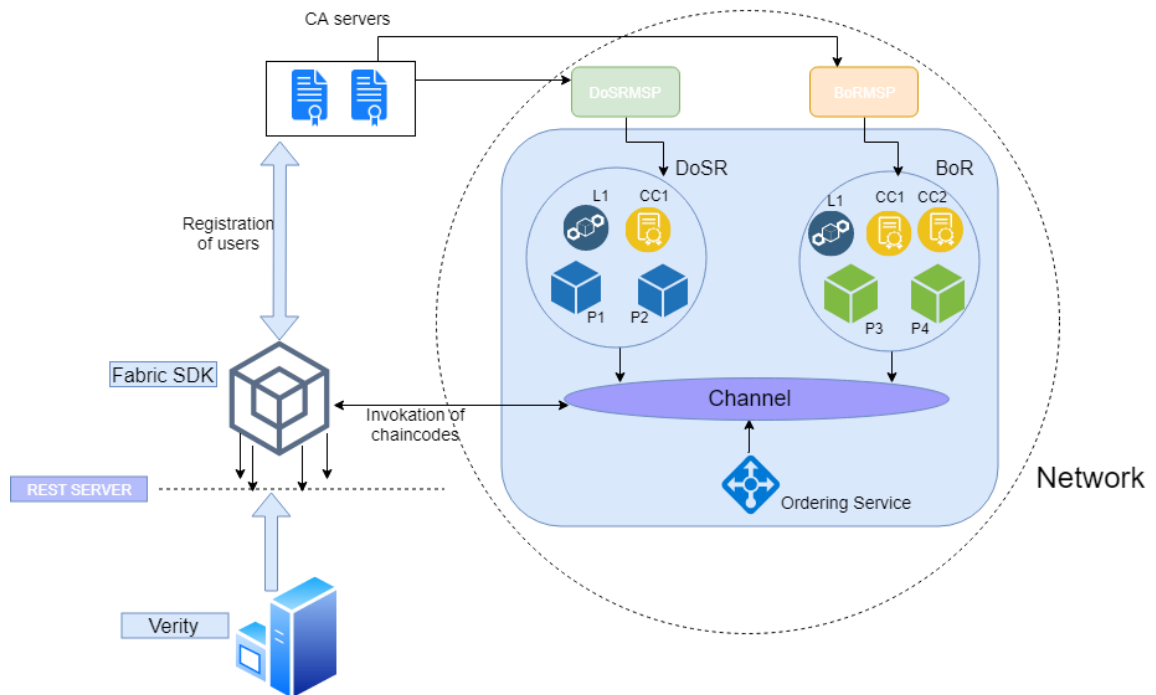
For the purpose of this thesis, we assume that the centralized databases used at the DoSR and the BoR can be categorized as SQL DBMS. To use the existing database with the blockchain network we need to create a link between the existing tuple on the database and the data stored on the ledger.

FIGURE 5.3: *Working of verity*

For every tuple in the DBMS, metadata about the data tuple is created. This created metadata (Cryptographic Hash) along with the value of the primary key of the table is to the ledger. So every time a data tuple is in contention, the following steps are followed:

- Metadata of the data tuple is created
- Ledger is queried for the metadata of the tuple in contention using the value of the primary key
- Verification of both the retrieved metadata is done. This implies that no illegal modification has been done to the DBMS by circumventing the verity protocol.

Similar protocols for SELECT, INSERT, UPDATE and DELETE are also provided by the Verity [35].

FIGURE 5.4: *Final design*

Chapter 6

Implementation

In this Chapter, we discuss the implementation details of the thesis project. The first part of this chapter focuses on setting up the project discussing each layer of a fabric network on a single host and then in the deployment section [6.4](#) we will discuss the details of deployment on a multi-host environment.

6.1 Working environment

Hyperledger Fabric technology is not yet quite matured and many releases of the technology are happening just in the span of few months. For the purpose of this thesis, Hyperledger Fabric version 1.4.0 has been used.

Technical Details:

- Operating System: Ubuntu 16.04 LTS

Hyperledger Fabric has a list of pre-requisite software's that need to be installed before starting a network. During the course of this project, we found that not all versions of the pre-requisites are compatible with each other so we are listing the versions of the softwares used in this project :

1. cURL *v7.47.0*
2. Docker *v18.09.5*
3. Docker-compose *v1.18.0*
4. Go *v1.11.4*
5. Npm *v6.4.1*
6. Node.js *v8.14.0*

All the important components of the Hyperledger Fabric like Peer, CouchDB, Ordering service are packaged into Docker images. Hyperledger Fabric Samples¹ provided by the community has the script to fetch all the related docker images from the Docker Hub and also the platform specific binaries later used in the network are also found in the Bin directory of the Fabric Samples. Installation details of the fabric can be found here in ².

6.2 Backend or Network Layer

From the image of final network design in figure 5.4 we know the architecture of the network and treading on the details carefully we decided to use the following configuration of the network in the thesis project. We have to remember that the configuration detailed below is for a single host deployment, we look at the multi-host configuration in the Deployment section 6.4

¹Available at <https://github.com/hyperledger/fabric-samples>

²<https://hyperledger-fabric.readthedocs.io/en/release-1.4/install.html>

Organizations	CA	MSP	Peers	Port	StateDB	Port
OrdererOrg	-	-	orderer.landrecords.com	7050	-	-
DoSR	CA0	DoSRMSP	peer0.dosr.landrecords.com	7051	CouchDB	5984
			peer1.dosr.landrecords.com	8051	CouchDB	6984
			peer2.dosr.landrecords.com	9051	CouchDB	7984
BoR	CA1	BoRMSP	peer0.bor.landrecords.com	10051	CouchDB	8984
			peer1.bor.landrecords.com	11051	CouchDB	9984
			peer2.bor.landrecords.com	12051	CouchDB	10984

FIGURE 6.1: *Network configuration of the project*

Ordering Mechanism: SOLO

Anchor Peers: peer0.dosr.landrecords.com, peer0.bor.landrecords.com

Now that the configuration of the network is decided we turn our attention to start the network. For any fabric network given below are the list of steps to be executed to form the base of a network :

1. **Generating crypto material:** As explained in section 4.4, Hyperledger fabric is a permissioned network and each component in the network has an identity. The cryptogen binary provided by the Fabric samples is used to generate all the certificates as provided by the configuration file which indicates the number of organizations, the number of peers associated with each organization and other details which are mentioned in figure 6.1 earlier. All the certificated to be used for various purposes are generated in a directory(crypto-config in our case). In a real deployment a Certification authority needs to be used.
2. **Generating Channel Artifacts:** In the case of our project, a consortium of both the organizations with a single channel connecting them is used. The configtxgen binary tool is used to generate the configuration files for the channels and the earlier defined anchor-peers in the network. A configuration

file(`configtx.yaml`) specifying the details of the organizations used in the network along with the consortium details is used as an input to the tool. Three different outputs are generated in this section –namely:

- `genesis.block`
- `channel.tx`
- Anchor peer transaction files for both the organizations involved

All the above files are placed under the directory `channel-artifacts`.

3. **Starting the docker containers:** We have mentioned that all the major components of Fabric are docker images. All the different components of the network should be deployed in isolated docker containers. A configuration file defining all the components discussed in the figure 6.1 along with their environment variables and other details are collected. Docker-compose can be used to deploy these listed services into containers.
4. **Channel Operations:** Now that all the containers are deployed, we need to join all the different components to the network. Below mentioned is the list of steps usually involved in starting the network:
 - Create the channel
 - Join all the peers involved in the consortium to the channel
 - Update the anchor peers
 - Install the respective chaincodes on to the peers
 - Instantiate the chaincodes with the endorsement policies

6.2.1 Chaincodes

Two chaincodes CC1 and CC2 are involved in this project. The chaincode CC1 is part of the DoSR organization and has the smart contracts to serve the functionalities of the SRO 3.1 which usually involves the following steps:

1. creation of sale deed applications
2. querying the sale-deeds
3. submission of application to the BoR for the mutation proceedings

Similarly the chaincode CC2 belongs to the BoR organization and serves the following functionalities

1. Integrity check of the submitted sale deed
2. Verification of the transaction details
3. Updating the Land record following the mutation

Usually in the fabric context, each chaincode involves one or more assets and the modification of these assets are handled by the smart contracts.

For the purpose of this thesis, we tried to mimic the existing process and in the process of doing so we toned down the functionalities to some degree.

The following abstractions of the databases involved in the network are made during the implementation of the project:

In the case of property registration, we know that the DoSR handles the digital records of the sale deeds and all the records are stored in a central database. A typical sale deed involving the transfer of immovable property deals mainly with the following details:

- 1) Details of property involved
- 2) Buyer Details
- 3) Owner Details
- 4) Other information on payment

We propose the following schema in figure 6.2 as an alternative to the original sale deed digital record

```
CREATE TABLE LAND (  
  APPNUMBER          INTEGER PRIMARY KEY NOT NULL,  
  L_LANDID           INTEGER NOT NULL,  
  L_TEHSIL           TEXT NOT NULL,  
  L_ADDRESS           TEXT NOT NULL,  
  L_CITY             TEXT NOT NULL,  
  L_STATE            TEXT NOT NULL,  
  O_ID              INTEGER NOT NULL,  
  O_NAME             TEXT NOT NULL,  
  O_ADDRESS          TEXT NOT NULL,  
  O_EMAIL            TEXT NOT NULL,  
  O_MOBILE           TEXT NOT NULL,  
  O_CITY             TEXT NOT NULL,  
  O_STATE            TEXT NOT NULL,  
  B_ID              INTEGER NOT NULL,  
  B_NAME             TEXT NOT NULL,  
  B_ADDRESS          TEXT NOT NULL,  
  B_EMAIL            TEXT NOT NULL,  
  B_MOBILE           TEXT NOT NULL,  
  B_CITY             TEXT NOT NULL,  
  B_STATE            TEXT NOT NULL  
);
```

FIGURE 6.2: *SQL schema of the sale-deed database*

Similarly during the mutation process, the BoR handles the modification of Land records and the following SQL schema is used :

```
CREATE TABLE OWNERS (
  L_LANDID INTEGER PRIMARY KEY NOT NULL,
  O_ID      INTEGER NOT NULL,
  O_NAME    TEXT NOT NULL,
  O_ADDRESS TEXT NOT NULL,
  O_EMAIL   TEXT NOT NULL,
  O_MOBILE  TEXT NOT NULL,
  O_CITY    TEXT NOT NULL,
  O_STATE   TEXT NOT NULL
);
```

FIGURE 6.3: *SQL schema of the land records Database*

For the integration of verity framework discussed in section 5.1.3, we have explained the importance of the primary Key of an SQL schema. In this project we have made an assumption of having a unique ID to represent an immovable property.

The key *L-LANDID* in figure 6.3 is used as the unique identifier for the land records. While discussing the workflow of verity we mentioned that the ledgers in the network won't be dealing with the entire database but rather they involve the metadata of the involved database, an example of the relation between DBMS and the asset in blockchain network can be in figure 5.3. Given below are the assets involved in the chaincodes CC1 and CC2 of the DoSR and BoR organizations

```
type landcontract struct {
  ObjectType string `json:"docType"`
  AppNumber  string `json:"appNumber"`
  LandID     string `json:"landId"`
  Tehsil     string `json:"tehsil"`
  Issuer     string `json:"issuer"`
  IssueTime  string `json:"issueTime"`
  Hashvalue  string `json:"hash"`
  Status     string `json:"status"`
}
```

FIGURE 6.4: Sale deed asset

```
type landownership struct {
  ObjectType string `json:"docType"`
  LandID     string `json:"landId"`
  Tehsil     string `json:"tehsil"`
  Issuer     string `json:"issuer"`
  IssueTime  string `json:"issueTime"`
  Hashvalue  string `json:"hash"`
  Status     string `json:"status"`
}
```

FIGURE 6.5: Land record asset

FIGURE 6.6: Assets used in the project

In the chapter 5 while designing the network we have mentioned about the necessity of having single channel in the network. This means that both the organizations work on the same ledger. In order to differentiate between the assets involved in the network in the stateDB an identifier is added to both the assets in this case *docType*. A brief summary of the data fields used in the assets are described in Table 6.1:

S.No	Field	Summary
1	docType	Identifier to distinguish objects in StateDB
2	appNumber	Primary key of the Sale-deed DB
3	landId	Primary Key of the land records DB
4	tehsil	location of the property involved
5	issuer	Identity of the client invoking the SC
6	issueTime	Timestamp of the transaction
7	hash	metadata given by the verity framework
8	status	Status of the process

TABLE 6.1: Summary of the keys involved in the assets

6.3 Middleware

From the design of the project, we can see that we have three different components involved in this section –namely:

1) SDK Layer 2) REST Server 3) Verity Framework.

The implementation of these three components are discussed below:

6.3.1 SDK Layer

We are using the Hyperledger Fabric Node SDK³ version in this project. We know that the SDK provides the API's to interact with the fabric network. A common connection profile (ccp) describing the configuration of the network is used by the SDK to connect to the network. We earlier mentioned that the installation of chaincode CC1 on both the organizations lead to the illegal modification of the DoSR ledger and the need for the authentication at the SDK. Everytime a client wants to interact with the fabric network, a registration of sorts is performed at the SDK layer which involves creating the identity and storing them in a *wallet*. These identities can be used to perform authentication and thus restricting the access to specific identities.

Also in a hierarchical architecture such as DoSR and BoR, not all the employees have the same clearances/authorizations over the operations. To overcome these problems, we have implemented a role-based authentication for interacting with the fabric network. While registering the clients to the fabric network, SDK provides the opportunity to enroll these clients under different affiliations or departments, and these affiliations can be later used to verify whether a certain client has the authorization to perform a certain action on the network.

For every smart contract in the chaincode a corresponding function is written in the SDK layer for invoking the smart contract. An example of the SDK function is as follows:

³Documentation can be found at <https://fabric-sdk-node.github.io/release-1.4/index.html>

Algorithm 1 Smart contract invocation in SDK Layer

```

1: FabricCAservices  $\leftarrow$  require('fabric-ca-client')
2: Gateway, Wallet  $\leftarrow$  require('fabric-network')
3: ccp  $\leftarrow$  Load ccp from the filepath
4: procedure SMARTCONTRACTINVOKATION
5:   // Check to verify whether the user is already registered in the wallet
6:   user  $\leftarrow$  await wallet.exists(username)
7:   If failed invoke the REGISTERUSER(USERNAME)
8:   // create new instance of gateway to connect to the network
9:   gateway  $\leftarrow$  await new Gateway()
10:  // Connect to the network using ccp
11:  await gateway.connect(ccp, wallet, user)
12:  // Role based Authentication
13:  VERIFYUSER(USER)
14:  network  $\leftarrow$  await gateway.getNetwork('channelname')
15:  contract  $\leftarrow$  network.getContract('Chaincodename')
16:  // Submit the transaction proposal to the contract
17:  await contract.submitTransaction('SCname', arguments)
18: procedure REGISTERUSER(username)
19:  CA  $\leftarrow$  gateway.getClient().getCertificateAuthority()
20:  // Create certificates and the register the user
21:  cert  $\leftarrow$  CA.register()
22:  // Enrolling the user into departments
23:  await CA.enroll(dept.name)
24:  // Import the identity to Wallet
25: procedure VERIFYUSER(user)
26:  CA  $\leftarrow$  gateway.getClient().getCertificateAuthority()
27:  // Create identity service instance from the gateway
28:  idservice  $\leftarrow$  CA.newIdentityService()
29:  //Get identity of the user
30:  identity  $\leftarrow$  await idservice.getOne(username)
31:  return the enrollment check from the identity

```

6.3.2 REST Server

In a typical software architecture, the programming logic on the back-end side should be completely hidden to the end-user. A REST server usually exposes all the functionalities of the back-end via API's, in this case, API's are interfaces to the SDK functions to interact with the fabric network.

In this project, the implementation of REST server is done using the Express.JS framework.

6.4 Deployment

Every distributed system involves multiple nodes connected together via some network. Up until now, we managed to deploy the Hyperledger fabric application in a single machine. In this section, we discuss the deployment of the project on a multi-host environment.

In a hyperledger fabric application major components of the network run in an isolated docker based containers. In case of runtime these containers need to communicate, for a single host environment channels can be used as the communication medium but for a multi-host architecture this won't work. So we found out that docker swarm⁴ with the help of overlay network can be used to communicate between containers deployed across multiple hosts.

The nodes in a docker swarm cluster can be classified into manager nodes and worker nodes. The manager nodes are responsible for creating the services and have the control over network whereas worker nodes receive and execute the tasks communicated by the manager nodes.

Network Configuration:

For the purpose of this thesis project, we are using 7 virtual instances. Usually in a docker swarm cluster the nodes are identified using their hostnames, so these along with the internal addresses of the instances are given below in table 6.2 and they are identified by these terms for the remainder of this paper.

⁴Details can be found at <https://docs.docker.com/engine/swarm/key-concepts/>

S.No	Hostname	IP address
1	instance-1 ⁵	10.160.0.7
2	instance-2	10.160.0.8
3	instance-3	10.160.0.9
4	instance-4	10.148.0.2
5	instance-5	10.148.0.3
6	instance-6	10.160.0.11
7	instance-7	10.160.0.12

TABLE 6.2: Details of the instances

All the software pre-requisites mentioned in the implementation section 6.1 along with the project code are to be setup on each instance.

All the instances used are having Ubuntu 16.04 LTS as OS and 6 GB RAM.

We have mentioned the different components involved in the network along with their details in figure 6.1. The following topology is used in this project to distribute the components across the instances.

⁵Manager Node in the configuration

S.No	Hostname	Service	Port
1	instance-1	orderer.landrecords.com	7050
2	instance-2	DoSR-CA	7054
3	instance-2	peer0.dosr.landrecords.com	7051
4	instance-2	couchdb	5984
5	instance-3	BoR-CA	8054
6	instance-3	peer0.bor.landrecords.com	7051
7	instance-3	couchdb	5984
8	instance-6	peer1.dosr.landrecords.com	7051
9	instance-6	couchdb	5984
10	instance-7	peer1.bor.landrecords.com	7051
11	instance-7	couchdb	5984
12	instance-4	peer2.dosr.landrecords.com	7051
13	instance-4	couchdb	5984
14	instance-5	peer2.bor.landrecords.com	7051
15	instance-5	couchdb	5984

TABLE 6.3: Distribution of services within the instances

Setup:

Initialize⁶ a docker swarm cluster with instance-1 as the manager node and make the other instances join the cluster as the worker nodes. An overlay network⁷ to help the instances communicate with each other is created.

In the section 6.2 we have described a list of steps in forming the base of a network and they are summarised as follows:

1. Generating Crypto Material

⁶Commands available at https://docs.docker.com/engine/reference/commandline/swarm_init/

⁷Command:docker network create --attachable --driver overlay NETWORK_NAME

2. Generating Channel Artifacts
3. Starting the docker containers
4. Channel operations

These same order of steps with some modifications are used to deploy the fabric network in a multi-host environment.

The first two steps involve the generation of identities and configurations of the channels involved in the network. So in a multi-host environment, all the nodes need to be having the same identity certificates of the components involved.

Make sure to copy all the crypto material and the channel artifacts generated in all the instances. Instead of having them at different paths across the instances we have used similar paths for the project configuration in all the nodes.

The next step is to deploy all the docker containers to the swarm cluster. Earlier we mentioned that a docker compose file detailing all the details of the components along with their environmental variables is used to deploy the services to containers. In order to deploy these containers to their specific instances, the details for the deployment of each container has to be specified. An example of the same has been given below

```
ca0:
image: hyperledger/fabric-ca:$IMAGE_TAG
environment:
  - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
  - FABRIC_CA_SERVER_CA_NAME=ca-dosr
  - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.dosr.landrecords.com-cert.pem
  - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/e715bdc32689aef69e828a84b8f26a5e949c3e5e2ffc9932f30560699754723c_sk
  - FABRIC_CA_SERVER_TLS_ENABLED=false
  - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.dosr.landrecords.com-cert.pem
  - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-config/e715bdc32689aef69e828a84b8f26a5e949c3e5e2ffc9932f30560699754723c_sk
command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
volumes:
  - ./crypto-config/peerOrganizations/dosr.landrecords.com/ca:/etc/hyperledger/fabric-ca-server-config
container_name: ca_peerDoSR
deploy:
  mode: replicated
  replicas: 1
  restart_policy:
    condition: on-failure
  placement:
    constraints:
      - node.hostname == $instance-4
ports:
  - published: 7054
    target: 7054
    mode: host
networks:
  - landnetwork
```

FIGURE 6.7: *Deploying container to specified instance*

Chapter 7

Evaluation & Results

In the following section, we will present the evaluation of the deployed project. The performance analysis of the project is done using the modified version of Hyperledger Caliper tool¹, which is a performance benchmarking tool of different hyperledger projects.

7.1 Evaluation Metrics

The performance of the implemented system is evaluated on the Execution Time, Latency, Throughput.

Average Latency:

In the context of blockchain, the confirmation of transaction happens only after the inclusion of transaction into the ledger. So latency of a transaction can be defined as the time taken by a transaction to complete from the submission of transaction to the network.

Transaction Latency = Confirmation time - submission time

$$\text{Average Latency} = \frac{\sum_{\#Transactions} TransactionLatency}{\#Transactions}$$

¹The documentation details of caliper can be found at https://hyperledger.github.io/caliper/docs/2_Architecture.html

Transaction Throughput:

This can be defined as the rate at which transactions are committed into the blockchain network. We have to remember that in a blockchain network, the transactions can be marked invalid in case of any error in the verification so while computing the throughput only the valid transactions are taken into consideration.

Transaction Throughput = Valid transactions committed / Total time taken

Execution Time:

In case of Hyperledger caliper, the transactions are submitted via batches to the blockchain network. So in the evaluation process, we considered execution time to be the total time taken to commit all the transactions in the batch to the ledger.

7.2 Results

All the results provided in this section are evaluated on the configuration provided in table 6.1. For benchmarking purpose we have evaluated the system under different working environments i.e single-host and the multi-host deployment of the project. To compare the performance analysis of the evaluation metrics, the results are evaluated under both the working environments while keeping the number of peers the same. Similarly, scalability of the system is evaluated by varying the number of peers over the same working environments. Cross-comparison of the working environments does not make sense in case of scalability comparison.

A total of 4 configurations over both the environments are used in the evaluation process is shown in table 7.1

S.No	Deployment	Peers
1	Single-host	4
2	Single-host	6
3	Multi-host	4
4	Multi-host	6

TABLE 7.1: Various Network Configurations

All the configurations listed in table 7.1 are referenced by ①, ②, ③ and ④ for the rest of this paper. Also for achieving better insights from the metrics, in the distributed architecture the nodes are selected from different server clusters to make this more similar to the physical world scenarios.

In table 7.1, the peers listed are the cumulative number of peers from both the organizations involved in the network, in our case two. So the four peer configuration essentially is two peers in both DoSR and BoR in the network and similar argument for the six peer configuration.

All the transactions in the blockchain network can be classified under two different categories.

- Transactions for creating new documents (INSERT)
- Querying transactions

These two categories are evaluated separately over the above-defined metrics.

7.2.1 Impact of Arrival Rates

We have earlier said that, in case of Hyperledger caliper the benchmarking of network is done via batches of transactions committing to the network. For every blockchain network there is an existing optimal throughput which can be handled, so if the

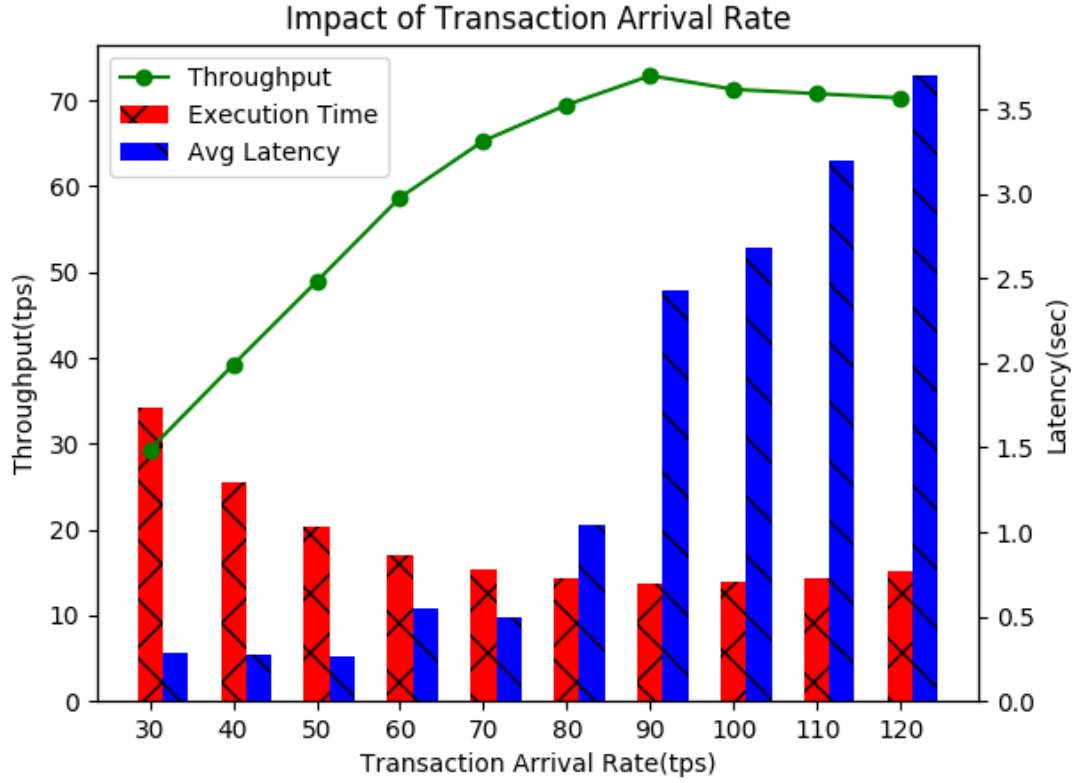
arrival rate of a batch is far less than the optimum throughput, each transaction stays less time in the network decreasing the average latency of the batch, and if the arrival rate is higher than the optimum throughput, the transactions remain longer in the queue increasing the waiting time of the transactions and thus increasing the average latency. So in-order to understand the effect of arrival rates and the optimum throughput of the network, this experiment executes batches of transactions at different arrival rates. In this section, we will first conduct the experiments on the insert transactions and later on the query transactions.

7.2.1.1 Insert Transactions

Figure 7.1 plots the values of evaluation metrics over different transaction rates on the configuration provided by table 7.2.

Parameters	Values
Configuration	③
Transaction Type	Insert
# Transactions	1000
Block Size	10
Arrival Rates	from 30 to 120 tps

TABLE 7.2: Configuration for impact of Arrival Rate - ③

FIGURE 7.1: *Impact of arrival rates on performance*

Observation 7.1. With increase in the arrival rate the throughput of the network increased linearly until it reached an optimal point, with a further increase in the arrival rate the throughput remained close to the optimal value.

Observation 7.2. With arrival rates lesser than the optimal value, the average latency of the batch is significantly lower (in order of 0.2-0.6 sec) because of the lesser waiting time for the transactions in the network. When the arrival rate just crosses the optimal throughput value there is a significant rise in the average latency (double in the above figure). With further increase in the arrival rates the rise in the latency gets bigger.

Observation 7.3. When the arrival rates are lesser than the optimal value, the network is at an under-performing stage and there would be no transactions in the waiting state, so the execution time of the batch would be nearer to the arrival time

of the transactions.

When the arrival rate just crosses the optimal value, from observation 7.1 we found that the throughput remain nearly the same and in a similar way the execution time remain flattened.

We found the following relation explains the nature of execution time in a fabric network:

$$ExecutionTime \approx \frac{\#Transactions}{ArrivalRate}$$

The optimal throughput for configuration ③ was found to be ≈ 73 tps.

Figure 7.2 plots the throughput and average latency of the transaction batches over different transaction arrival rates according to the configuration in table 7.5.

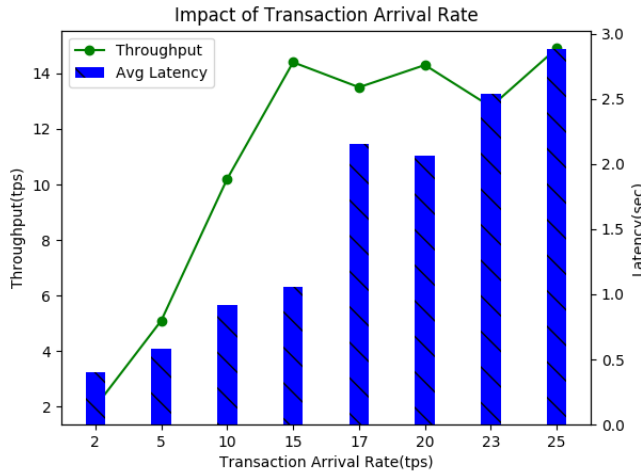


FIGURE 7.2: *Impact of arrival rates on performance*

Parameters	Values
Configuration	①
Transaction Type	Insert
# Transactions	200
Block Size	10
Arrival Rates	from 2 to 25 tps

TABLE 7.3: Configuration for impact of Arrival Rate - ①

From the figure 7.2, we can verify the validity of all the observations 7.1, 7.2, 7.3 claimed earlier.

In case of configuration ① (Single-Host), we found the optimal throughput of the system to be around **14.5 tps** which is significantly lesser when compared to the **73**

tps observed in the configuration ③ (Multi-host).

The reason for this can be attributed to the distribution of peers in the multi-host architecture, where the load of the network is similar to all the other nodes. In a single-host architecture the CPU intense operations during the validation stage of transactions over-heats the machine which increases the execution time of the transaction, thus lowering the throughput. Although there is more network latency during the gossip communication in a multi-host architecture, we found that the effect is much less compared to the CPU overuse in the single-host architecture.

Observation 7.4. *The system performs significantly better in a multi-host architecture than a single-host architecture.*

To further validate the observation 7.4, Figures 7.3 and 7.4 provides the results on configurations ② and ④ under similar conditions as in tables 7.5 and 7.2.

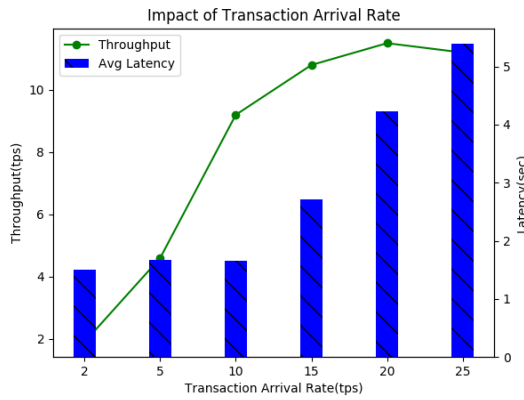


FIGURE 7.3: Impact of arrival rates on ②

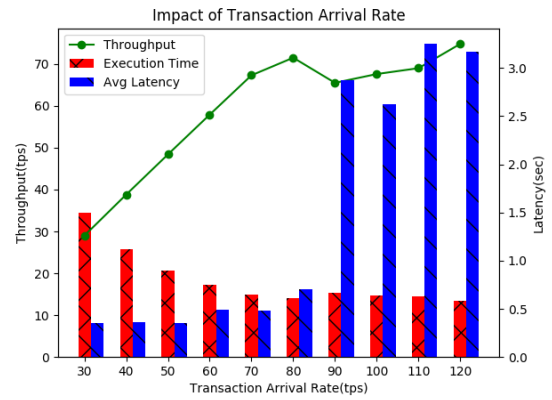


FIGURE 7.4: Impact of arrival rates on ④

FIGURE 7.5: Comparison of configurations ② and ④

7.2.1.2 Query Transactions

In case of query transactions, every transactions leads to some operations on the stateDB. These transactions in comparison to the insert transactions have higher

operations at the database level. So the choice of stateDB has a significant effect on the performance comparison between insert and query transactions. Hyperledger fabric currently offers support to two state DB's GoLevelDB and CouchDB. It is observed that GoLevelDB offers significantly higher performance than the CouchDB and the reason for these differences is that the GoLevelDB is an embedded database on the fabric peer whereas the CouchDB is an external database that is accessed via REST API's. The network latencies involved in the REST API calls increases the execution time of a transaction with CouchDB and thus decreasing the throughput. In this section, we will observe the performance of the network under different arrival rates in case of query transactions.

Figure 7.6 plots the average latency and throughput of the system. Table 7.4 presents the various parameters involved in the experiment.

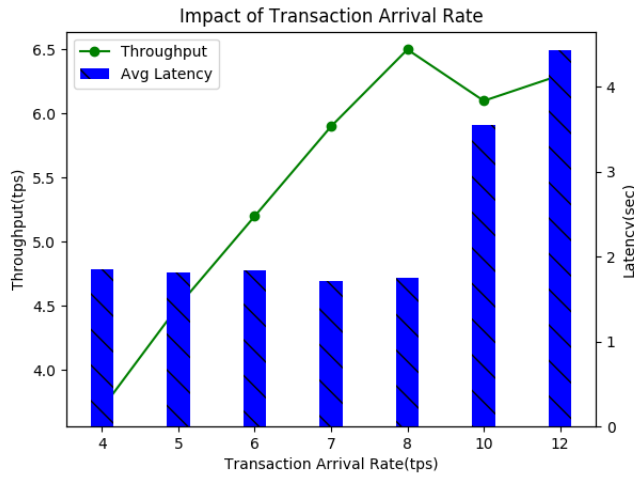


FIGURE 7.6: *Impact of arrival rates on performance*

Parameters	Values
Configuration	④
Transaction Type	Query
# Batch Size	64
Block Size	10
Arrival Rates	from 4 to 12 tps
# Ledger Transactions	1000

TABLE 7.4: Configuration for impact of Arrival Rate - ④

Observation 7.5. *The throughput observed in the query transactions (≈ 6.5 tps) is significantly lesser when compared to the insert transactions (≈ 74 tps) in multi-host configuration. This is because, in insert transactions the operations performed (usually PUT) on the ledger are less time consuming when compared to the query operations which have to query the entire ledger in case of search or update requests. So with*

a less execution time insert transactions have a higher throughput.

Similar to the insert transactions, the query transactions show similar behaviours as observations 7.1 and 7.2. For arrival rates less than optimal throughput, the average latency observed under query transactions is ≈ 1.5 sec which is significantly higher when compared to the 0.4 sec observed under the insert transactions. Also, in case of query transactions, the jump in the latency during the optimal throughput is lesser than what was observed in the insert transactions.

Optimal throughput observed in configuration ④ for query transactions is ≈ 6.5 tps

Figure 7.7 plots the throughput and average latency of the transaction batches over different transaction arrival rates according to the configuration in table 7.5.

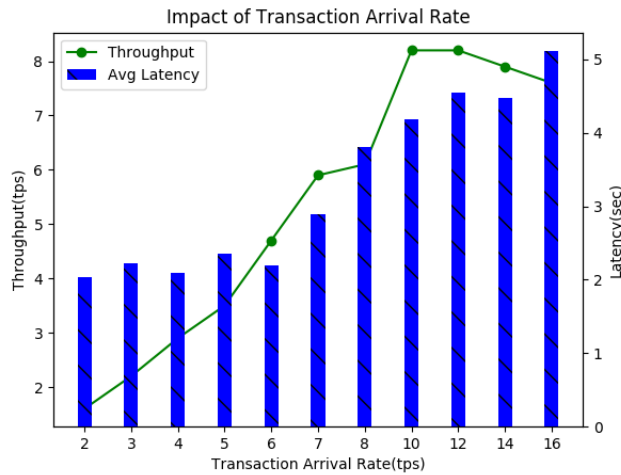


FIGURE 7.7: *Impact of arrival rates on performance*

Parameters	Values
Configuration	②
Transaction Type	Query
# Batch Size	64
Block Size	10
Arrival Rates	from 2 to 16 tps

TABLE 7.5: Configuration for impact of Arrival Rate - ①

The above figure corresponds to the single-host architecture ②, unlike the behaviour observed in observation 7.5, the single host architecture don't show a significant drop in the throughput between insert and query transactions.

Observation 7.6. *Under moderate loads in query transactions, the single-host architecture performs slightly better compared to the multi-host architecture.* For example, optimal throughput in configuration ② (Single-host) is ≈ 8 tps whereas the Multi-host configuration ④ shows a throughput of ≈ 6.5 tps. This is because in a multi-host architecture, the couchDB databases are distributed across the nodes and the network latencies involved in the REST API calls is not present in case of single-host architecture. We already know that in case of single-host architecture, the execution time of transaction is usually higher due to the CPU over-usage and in this case we found that the network latencies caused by the external REST API calls in multi-host architecture has greater effect on the performance.

From observation 7.4, we found that in insert transactions the system under multi-host architecture performs far more significantly better than the single-host architecture. In case of query transactions, under moderate loads we found that system single-host architecture performs slightly better than the multi-host architecture. To arrive at a conclusion, we further experimented the system in case of query transactions under significantly higher loads (Ledger size), we found that the single-host architecture is unable to handle higher loads due to the starvation of transactions in the validation phase.

From the above given arguments, we can conclude that the system under multi-host architecture performs better than the single-host architecture.

7.2.2 Impact of Block Size

In the earlier section 7.2.1, we have concluded that system under multi-host architecture performs better, so in this section all the experiments and the observations inferred are made under the distributed architecture.

We know in fabric transaction flow happens majorly in three phases which were explained in section 4.5, ordering phase is where the transactions submitted are ordered into blocks with the defined blocksize and these are further distributed to peers for

validation phase. Unlike the validation phase where the validation of blocks happens in transaction wise, in case of ordering phase the cryptographic verification happens in blocks. In earlier section we studied the behaviours of throughput and latency across different arrival rates. Varying blocksize in conjunction with the arrival rates have an effect on the throughput of the system and their behaviours are studied in this section.

Figure 7.8 show the behaviour of throughput and latency of the system under environment (4) for insert transactions while varying the block-size and arrival rates.

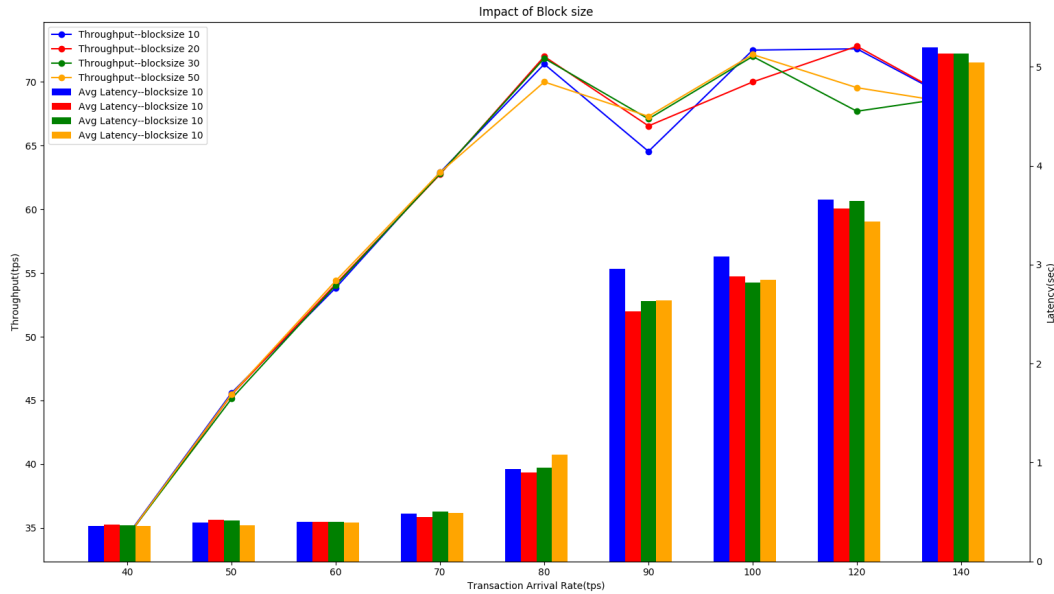


FIGURE 7.8: *Impact of Block on performance*

Theoretically the effect of block size on throughput can be explained during the ordering and validation phases of the transaction flow. With increase in the block size, the time taken to package the transactions into block increase at the ordering phase increasing the waiting time of a transaction in this phase. Similarly in the validation phase where the validation of blocks happens in per-transaction wise, the increase in the block-size implies more transactions packed in a single block. It is observed that the time taken in accessing the block is much higher due to the read-write latencies,

so with higher blocksize the time taken during the validation phase is lesser.

Observation 7.7. *With arrival rates more than the optimal throughput, the latency of the system shows a slight decrease with increasing block-sizes.* For example, consider arrival rate 100 tps with block sizes 10 and 50 . The throughput of the ordering phase is very high so with the transactions readily available in this phase , the block creation time has a very little effect. In the validation phase, however, the system with block-size 10 has higher no: of blocks to be processed compared to the system with block-size 50. We have earlier explained that in validation phase, p transactions in a single block takes less time when compared to $\frac{p}{q}$ transactions in q blocks

Usually in the context of land registrations in India, the arrival rate of transactions can be assumed at 10-15 tps. Also with a higher block-size the waiting time of transactions in block creation during the ordering phase increases. So a lower block size (10) makes more sense in this case.

In this chapter, we evaluated the performance of our system under different working environment across various parameters. In section 7.2.1, we evaluated the system separately for both insert and query transactions and in each case the evaluation was done on different working environments (Single-host vs Multi-host) and, arguments for the observations made are presented. At the end of this section 7.2.1, we concluded that the system under distributed environment performs better when compared to single-host environment.

In section 7.2.2, we evaluated the performance of the multi-host system by varying the block size of the network.

Chapter 8

Conclusion & Future Work

8.1 Conclusions

The main purpose of this thesis is to find whether integrating blockchain technology to the current business process of Land record management is a viable solution or not? To answer this question a literature study is done on the existing business process and during this study we discovered the problems associated with the existing process. We then made an argument about why we think blockchain technology is the best possible solution in section 3.3 in particular Hyperledger Fabric. Also in this thesis, we have established the fact that Hyperledger Fabric can be used to integrate blockchain technology into existing enterprise solutions. Although we have made an application to mimic the existing process of land records, the main emphasis of the thesis is put on the designing part and performance evaluation of the blockchain network.

8.2 Future Work

We have talked about the many hyperledger tools integrated with this project and since the start of the project, many of these tools have released newer and upgraded versions. This lack of maturity in hyperledger technologies has hampered a lot of the work. We have listed below some major improvements that can be done as part of the future work.

1. **Ordering Service:** Currently we are running the SOLO ordering service with a single node in the project. Since the project is running on a single ordering node, this can always be a single point of failure in the system. With the support to new ordering services like KAFKA and RAFT beginning, we can shift to a more safer and scalable ordering service
2. **Penetration Testing:** In this thesis, we have not made any significant steps to protect the network against exploitable vulnerabilities. Since this is more of an application, pen testing should be used to uncover some those vulnerabilities.
3. **Database sharding:** Although we are not storing the entire data on the blockchain as explained in section 5.1.3 we have seen the dip in the performance while observing the evaluation metrics on the Query operations. We believe that these can be optimized by using sharding techniques on the existing couchDB database.
4. **Application Layer:** In the course of this thesis we have made some abstractions and assumptions along the way to simplify the existing process such as migration of the databases to SQL, assumption of a unique identifier for property records to name a few. A clear understanding of the existing technical implementation can clear up some of these abstractions.

Bibliography

- [1] Satoshi Nakamoto. "bitcoin: A peer-to-peer electronic cash system" <http://bitcoin.org/bitcoin.pdf>, 2008.
- [2] Wikipedia authors. *Blockchain*. <https://en.wikipedia.org/wiki/Blockchain>.
- [3] Jake Frankenfield. *Double Spending*, 2018. <https://www.investopedia.com/terms/d/doublespending.asp>.
- [4] *Script* <https://en.bitcoin.it/wiki/Script>.
- [5] Ethereum white paper. <https://github.com/ethereum/wiki/wiki/white-paper>.
- [6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Genady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 30:1–30:15, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5584-1. doi: 10.1145/3190508.3190538. URL <http://doi.acm.org/10.1145/3190508.3190538>.
- [7] Cardano <https://www.cardano.org/en/home/>.

- [8] The Zilliqa team. *Zilliqa: A Secure, Scalable Blockchain Platform*, 2018. <https://docs.zilliqa.com/positionpaper.pdf>.
- [9] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, May 2014. doi: 10.1109/SP.2014.36.
- [10] Rita Sinha. *Moving towards clear land titles in India: Potential benefits, a road-map and remaining challenges*, 2009. https://www.fig.net/resources/proceedings/2009/fig_wb_2009/papers/country/country_sinha.pdf.
- [11] Registration act, 1908 <https://indiacode.nic.in/bitstream/123456789/2190/3/A1908-16.pdf>.
- [12] N. Vishal Gupta P. Kiruba Rachel. *Data Integrity – Regulations and Current Scenario*, 2017. <http://globalresearchonline.net/journalcontents/v43-1/05.pdf>.
- [13] Hyperledger <http://www.hyperledger.org>.
- [14] Linux foundation <http://www.linuxfoundation.org>.
- [15] Hyperledger architecture, volume 1. https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.
- [16] Hyperledger burrow. <https://github.com/hyperledger/burrow>.
- [17] Hyperledger fabric. <https://github.com/hyperledger/fabric>.
- [18] Hyperledger indy. <https://github.com/hyperledger/indy-sdk>.
- [19] Hyperledger iroha. <https://github.com/hyperledger/iroha>.
- [20] Hyperledger sawtooth. <https://github.com/hyperledger/sawtooth-core>.

- [21] Hyperledger cello. <https://github.com/hyperledger/cello>.
- [22] Hyperledger composer. <https://github.com/hyperledger/composer>.
- [23] Explorer. <https://github.com/hyperledger/blockchain-explorer>.
- [24] Hyperledger caliper. <https://github.com/hyperledger/caliper>.
- [25] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990. ISSN 0360-0300. doi: 10.1145/98163.98167. URL <http://doi.acm.org/10.1145/98163.98167>.
- [26] Bernadette Charron-Bost, Fernando Pedone, and André Schiper, editors. *Replication: Theory and Practice*. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 3-642-11293-5, 978-3-642-11293-5.
- [27] Chain <http://www.chain.com>.
- [28] Quorum <http://www.jpmorgan.com/global/Quorum>.
- [29] Behind the architecture of hyperledger fabric. <https://www.ibm.com/blogs/research/2018/02/architecture-hyperledger-fabric/>, Feb 2018.
- [30] Leveldb <https://github.com/google/leveldb>.
- [31] Couchdb <http://couchdb.apache.org/>.
- [32] J. Sousa, A. Bessani, and M. Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 51–58, June 2018. doi: 10.1109/DSN.2018.00018.
- [33] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC’14, pages 305–320, Berkeley, CA, USA,

-
2014. USENIX Association. ISBN 978-1-931971-10-2. URL <http://dl.acm.org/citation.cfm?id=2643634.2643666>.
- [34] Go programming language <https://golang.org/>.
- [35] Shubham S. Srivastava, Medha Atre, Shubham Sharma, Rahul Gupta, and Sandeep K. Shukla. Verity: Blockchains to detect insider attacks in DBMS. *CoRR*, abs/1901.00228, 2019. URL <http://arxiv.org/abs/1901.00228>.