A Blockchain Based Freight Management Solution

By

Ramsha Rizwan

00000205747

Supervisor

Dr. Syed Taha Ali

Department of Electrical Engineering

A thesis submitted in partial fulfilment of the requirements for the degree of Masters of Science in Information Security (MS IS)

In

School of Electrical Engineering and Computer Science

National University of Science and Technology (NUST)

Islamabad, Pakistan

(July-2019)

Approval

It is certified that the contents and form of the thesis entitled “A Blockchain Based Freight Management Solution” submitted by Ramsha Rizwan have been found satisfactory for the requirement of the degree.

Advisor: Dr. Syed Taha Ali

Signature:

Date:

Committee Member 1: Dr. Saad Qaisar

Signature:

Date:

Committee Member 2: Dr. Fahd Ahmed Khan

Signature:

Date:

Committee Member 3: Dr. Arsalan Ahmed

Signature:

Date:

Thesis Acceptance Certificate

Certified that final copy of MS thesis written by Ramsha Rizwan, Registration No. 00000205747, of Department of Computing, SEECS has been vetted by undersigned, found complete in all respects as per NUST Statutes and Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Advisor: Dr. Syed Taha Ali

Signature:

Date:

Dedication

Dedicated to my beloved parents and siblings.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Ramsha Rizwan

Signature:

Acknowledgement

I would first like to thank my thesis advisor Dr. Syed Taha Ali for his supervision and scheduling weekly meetings to ensure I am consistently working during my research phase. His guidance and working strategy propelled me to achieve my goal. Without his passionate participation and input, I might have not finished my thesis on time.

I would also like to thank my Committee Member Dr. Saad Qaisar for motivating to meet all deadlines for tasks discussed in meetings and giving his valuable suggestions on my work.

Finally, I must express very deep gratitude to my parents and friends for supporting and encouraging me throughout my research phase. This accomplishment would never have been possible without them.

Table of Contents

List of Figures

List of Tables

Abstract

Freight Management System streamlines shipping procedure. It is a sub-process of supply-chain realm which focuses on transportation and logistics solution. Freight Management System allows its participants to use technology to automate shipping process. Thus, saving time and cost for shipments.

Since all shipments are not similar in nature. Some are time sensitive, whereas others are temperature sensitive. Thus, they require different freight services which can handle them with extreme care. For this reason, businesses do not use same carrier for different shipments. It is very common to observe different logistics services dealing with different shipments. As a result, freight tracking process becomes extremely scattered and challenging. Depending on goods and routes they cover, a heap of paperwork is required which include certification of compliance, invoices, clearance permits, dispatch notes etc. Which means integrity needs to be preserved not only for goods, but also for the paperwork. A wrong or missing paper from its pile create many troubles in logistics, such as inconveniently long delays, misplaced consignments etc. It is very clear paperwork is of utmost importance, but the challenge is how to ensure providing correct details about freight, its sender and receiver without needing to process paper-trail.

One way to address problems in freight management system is to expose system to the latest technologies. One of those technologies is blockchain which allows to secure data in an immutable fashion. This might prove to be a good architecture to address traceability issues in freight management system, making it fully automated and digital.

Thesis focuses on researching over issues involved in Freight Management Systems and finding out whether blockchain can address them. Hence, we begin with baselining our hypothesis blockchain architectures seem to be a good match for Freight Management Systems.

We gleaned major requirements that are necessary for Freight Management System. Later, we made an extensive literature review to pick one out of many existing blockchain frameworks which suffice to fulfill if not all, majority of those listed requirements.

In the end, we validated our proposed solution to find out extent to which it can fulfill listed requirements. From its validation, we were able to reach out to some conclusions. The fact that our design architecture proved to meet majority of requirements for Freight Management System, but a few others remain unaddressed. Our aim was to explicitly define achievable and unachievable criteria through our proposed solution and to show potential of blockchain in freight management realm. We hope our contribution can pave way for future research to integrate blockchain in freight management systems.

Keywords: Freight Management Systems, Traceability, Blockchain, Immutability, Interoperability.

Chapter 1

Introduction

Pakistan's economy leans on export to facilitate trade with other countries. Unfortunately, our country's deteriorating road and rail infrastructure creates a huge bottleneck in economic growth. Although constructive measures under CPEC claims to restore Pakistan's trade network with other countries, country needs to revise its freight management service for commodity flow to enjoy full benefits of CPEC. Logistic companies are making efforts towards this end to search for right technologies to make underlying processes efficient.

1.1 Traditional Freight Management

From Finance to Customs and logistics, global trade involves commodity transfer between these actors sitting in different countries. Consequently, global trade becomes extremely scattered. Moreover, operations related to commodity transfer are still based on paperwork due to lack of secure online data exchange system. Some organizations have deployed their information systems, but those serve as information silos. Data transfer between such systems is not possible because they lack integration compatibility. Transfer of each document from one organization to another is incumbent on shipping agent. The documents are mostly printed documents. The agent submits required documents to custom office before ship arrives at port to obtain permission to berth. Shipping agent submits similar documents to harbor master including custom permissions to berth. Shipping agent then submits all permissions from custom office and harbor master to port authority so that ship can actually berth. After the ship berths, custom officers board on ship to inspect cargo and its documentations. After inspection, custom officers allow the cargo loading on ship. Ship captain transfers loading details to port personnel. Port personnel records berth number and yard operations and enter this information on computers. Port officers submits ship information, such as name, ship characteristics, location and status to harbor master [1].

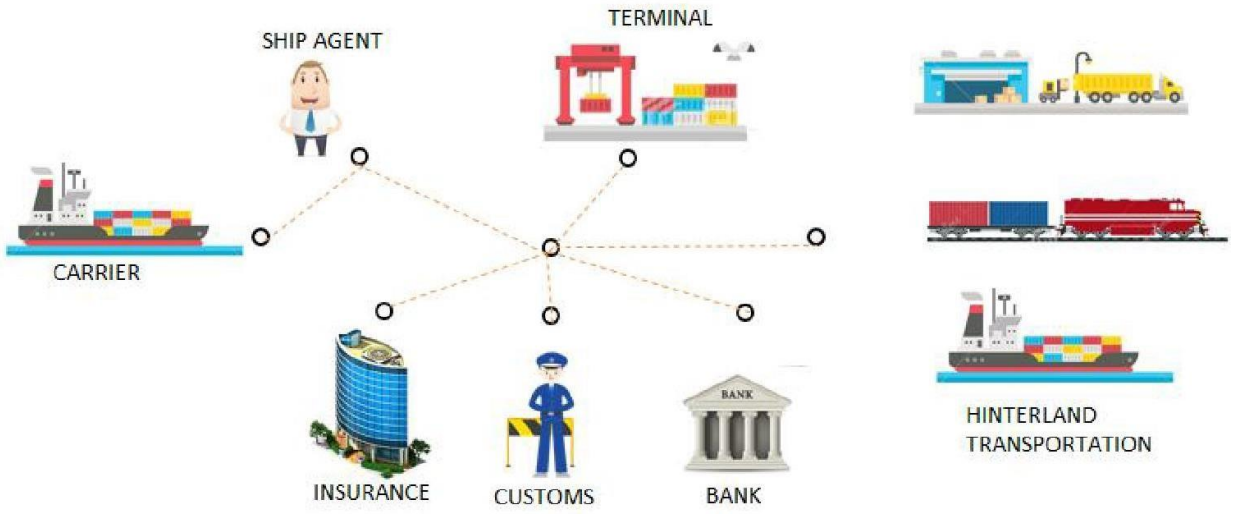

Figure 1.1: Traditional Freight Management System

1.2 Challenges

As previously mentioned, until today, a paper-trail keeps a track of a cargo's change of ownerships. Bill of Landing, a legal document enlists all details about cargo. It functions as a receipts too. Supplier and transporter signs Bill of Landing after ship loads cargo. Whereas, receiver and transporter signs Bill of landing after it reaches destined location. Thus, Bill of landing is used to verify transfer of ownerships. It satisfies buying party about selling party's intent to sell this cargo to buyer only. The reason why Bill of Landing still remains a paper based contract is because one wants to be completely certain about cargo's title transfers and location. Heavy paper based trail for title transfers goes around commodity trading [2]. Every participant in the business network needs to negotiate with each another for verification. But, since the individual parties maintains different information tracking systems, shipment process is still challenging.

1.3 From Blockchain Technologies to Freight Management System

Blockchain, a shared and immutable ledger actuates recording transactions and tracking assets' trajectory through its entire journey during entire freight management life cycle. An asset could be tangible or intangible. Hence, an asset or colloquially anything of value can be tracked or traded on a blockchain network.

Following a traditional freight management, every participant on a network maintains its own ledger of records. This approach involves paying exorbitant charges to intermediaries for their service provision to each participant. It is clearly evident that delays involved in synchronizing information silos from participants make it inefficient. Additional house-keeping for maintaining heaps of ledgers further deteriorates its efficiency. The approach is susceptible to cyber-attacks due to its centralized nature. Being a single point of failure, getting caught in cyber-attack bait could shut-down entire business network.

Blockchain allows participants over its network to share one ledger, whose entries are updated through mutual consensus of all participants. Every participant or a node can transmit transactions to other nodes in a network and receive transactions coming from those within network. Data is synchronized all over blockchain network because one ledger is transferred to all parties [3].

Blockchain network is **economical** because it eliminates all house-keeping involved in maintaining different ledgers. It is efficient because it makes the network independent of intermediaries. It has a **consensus mechanism** which fortifies its consensus model, allowing each node to verify authenticity of information. Thus, a transaction can make it to a blockchain block only if all network's participants vote it as valid. It is **transparent** as participants can pin-point asset's provenance and see it changing ownerships. It is **immutable** because no participant can modify transaction once the blockchain records it. It is a **single source of truth** as a single and shared ledger refers to all the participants a single place to determine asset's ownership. [4]

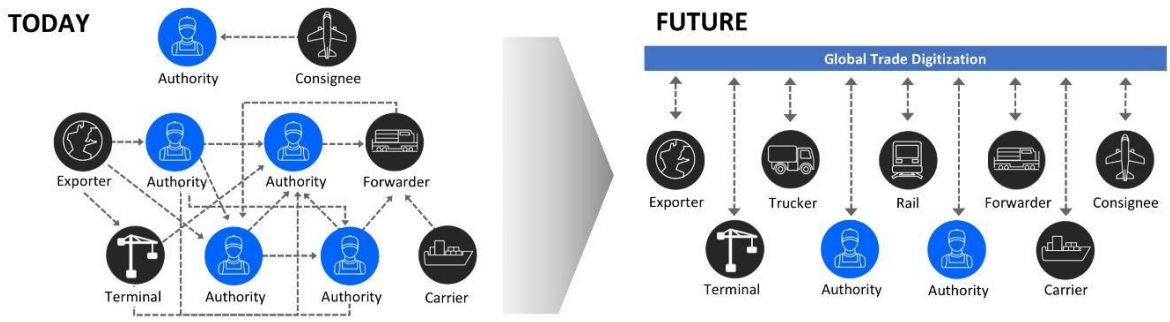

Figure 1.2: Digitizing Freight Management System

1.4 Motivation

Foreign trade depends on paper based process. Most crucial of all transport documents is the bill of landing. Technological advancement claims electronic documents to replace paperwork in business processes. Companies are looking forward to digitize bill of landing and fit in all functionalities of paper counterparts. However, there is still no certitude of them successfully digitizing bills. There are obstacles that hinder their shift to paperless trade. Using centralized system to transfer the paper process to electronic systems requires integration between parties. This situation is difficult to achieve because full integration requires high degree of mutual trust among trading parties. Lack of mutual trust brings intermediaries onboard to integrate trading parties. Greatest practical barrier in electronic trade is standardization of electronic documents. International standard for paperless trade is not defined yet [5]. Developing international coordination among entities for a project to set up a common system is extremely difficult. Nevertheless, removing papers from trade entails a common platform to unite all parties involved in business. Validity of electronic document depends on electronic signature. Every country's law defines the validity of electronic signature differently. Electronic signature is a legal problem at international level trade. Trading parties might need to sign multitude of agreements to clarify legal status. Therefore, Law insists on inclusion of paper documentations, written signatures etc. Every electronic document has a code attached to it which shows it is authentic. This code represents electronic signature. Electronic signature saves a lot of costs, but considering electronic documents in a legal sense requires electronic signatures in a manner equally acceptable by different laws.

Blockchain can meet this requirement for electronic signature by its simple mathematical algorithms and provide high authentication using cryptography. Thus, making cryptographically secured signature acceptable world-wide [6]. Blockchain can potentially overcome other shortcomings too. It can extricate system from intermediaries and ensure full integration among trading parties. Multiple trade documents like Bill of Landing, Certificate of Origin, and Packing List etc. can be replaced by a single smart contract on Blockchain. This cuts down business costs to a huge margin and saves time.

1.5 Objectives

The main goal of thesis is to research whether blockchain is a good substitute for a traditional freight management system to solve its common problems. Thesis also aims to find out technological requirements of modern freight management system, so it can fit to the needs of modern projects, such as CPEC. There are a myriad of small tasks in freight management which blockchain can automate. This thesis will also figure out which of those tasks are best applied by blockchain.

Primarily, the thesis focuses to determine whether blockchain frameworks are feasible enough to improve traditional freight management system to cater to the needs of CPEC project. For this purpose, it is necessary to conduct a research over issues in traditional freight management system and validate them, in order to propose a blockchain based freight management solution to address these issues.

1.6 Thesis Structure

Other than Introduction, thesis covers 5 more chapters, which are divided into three parts.

1.6.1 Background and State of Art

It forms a basic foundation to understand the technology, as well as explains different frameworks for blockchain and its application to freight management system. Chapter 2 and Chapter 3 of this thesis covers Part 1 to set the required background.

- Chapter 2, Background Information, discusses very basic and important concepts of blockchain architecture, along with analyzing different blockchain frameworks and comparing them.
- Chapter 3, “Literature Review”, expatiates over applying blockchain to freight management system, possible advantages, challenges and analyzing existing blockchain frameworks suitable for freight management use case.

1.6.2 Problem, Research and Solution

It goes in depth to provide a detailed explanation for problem, objectives and proposed methodology.

- Chapter 4, “Problem Statement”, specifies thesis statement and questions that needs to be answered to form a conclusion.

- Chapter 5, “Solution Design and Implementation” presents a blockchain framework and proposes a solution, in the form of building a proof of concept (PoC) project to meet freight management system requirements.

1.6.3 Conclusion

It collects all information from results to make a solid statement, as well as discusses difficulties and future work.

- Chapter 6, “Conclusions”, presents an overview of work thesis covers, provides an answer for the previously discussed problem, as well as discusses possible future work in this area.

Chapter 2

Background Information

Chapter covers building blocks of blockchain technologies which are crucial for learning its applications. It uncovers latent potential of blockchain to serve freight management system.

2.1 Introduction

Distributed systems face a classical problem known as “The Byzantine Generals’ Problem”. It states that there’s no guarantee for consistency in any distributed system due to lack of general consensus on state of system at any given time [7].

Satoshi Nakomoto’s practical blockchain implementation seems the most suitable fix for this problem. Being able to handle consensus issue, it earned the term “Practical Byzantine Fault Tolerance (PBFT) algorithm” [8]. Since majority of distributed systems faced Byzantine Generals’ problem, blockchain tolerance to those make it more appealing.

Some advanced features are incorporated into traditional blockchain, such as smart contracts, which makes it even more suitable for a variety of applications, such as Internet-of-Things, Identity Management, Health Care, Insurance etc. In each of these areas, blockchain can store and process information and provide services.

2.2 Core Concepts and Features

The very first blockchain was develop for bitcoin, with an aim to create a distributed online payment system having no involvement of trusted third parties for transaction verification. A cryptocurrency or a digital token was associated with blockchain which served as digital representation of real money.

With time, blockchain evolved into something better, with new uses, other than being used for payment system. The algorithm bitcoin uses doesn’t define blockchain in a broader term, but it is only one of many other applications.

Blockchain peer-2-peer architecture continually stores and updates blocks chained together in chronological order. Each block stores information relevant to its application and a previous block hash value. A hash pointer is a name for previous hash block value, since it can be treated as an address through which each block can point to its previous block. This concept is depicted in figure 2.1.

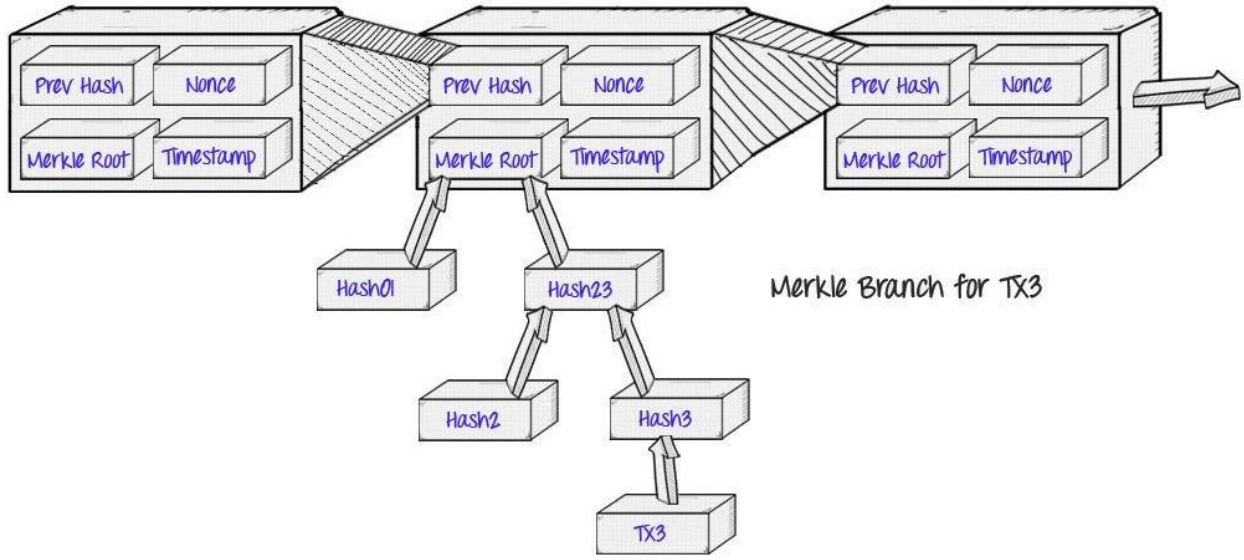

Figure 2.1: Blockchain Architecture

2.2.1 Immutability

These hashes not only make sure blocks are linked to one another, but also ensures blockchain integrity. If someone tries to alter one block, its hash stored in block that follows it will find a mismatch. So, he would have to alter the hash on second block. As a consequence, third block will find a mismatch, and so on. These linked chain of hashes make blockchain immutable. It is impossible to tamper a block in the middle without tampering many others. As a result, tampering never goes unnoticed on blockchain [9].

2.2.2 Consensus

Nodes make up blockchain peer-to-peer network. Nodes are machines running core code of blockchain system. Nodes receive incoming information, shares among each other in the form of blocks and validate blocks according to the established rules. All nodes (if they are not corrupt and makes no attempt to alter information) contain same blockchain information and structure, since they all show their consent through consensus mechanism.

Some nodes are public, thus they can receive information from an outside peer-to-peer network and spreads it to rest of nodes in the network. A small set of nodes, known as miners, will collect information from peer-to-peer network, bunch it into blocks and adds it to blockchain. Not all the nodes can make their block a part of blockchain simultaneously, because they will have different versions of blockchain. Therefore, nodes mutually decide which block shall enter blockchain next.

In public blockchain, all hard work in incumbent on miners, usually incentivised by earning cryptocurrencies. Cryptocurrencies are digital tokens which only associates themselves to the blockchain they belong to. These digital tokens play a crucial role in blockchain. They are rewarded to miners for their veracity and hardworking. Without incentivising cryptocurrencies, public blockchain will probably collapse.

Nevertheless, this is one of many other ways to actuate blockchain correct working. Other types of consensus mechanisms are also designed, but those for public blockchain will always involve cryptocurrencies as a reward.

2.2.3 Public, Private and Hybrid Blockchain

Public Blockchain

Traditional blockchain was public by nature. But, as many organizations learnt about possible gains this technology can bring, they started investments over it. As a result, private blockchain and consortium blockchain are tailored to meet business needs. Figure 2.2 shows node topologies for private, consortium and public blockchain.

Private Blockchain

Organizations own private blockchain, with fine-grained access control policies defined. It restricts write access to specific peers within its organization. It can also restrict read access, but leaves it on organization's discretion. Unlike public blockchain, it is not driven by cryptocurrencies, since now organizations are in charge of maintaining it. All consensus-making nodes are a part of organization in a private blockchain. The greatest benefit over public blockchain is it can fit in huge number of transactions to process them per second.

Consortium Blockchain

This is a hybrid approach, but many of its aspects are skewed to private blockchain. Different organizations control consortium blockchain. Some pre-selected nodes fromorganizations handle consensus. This consensus might involve certain conditions, say if there 20 nodes, it required at least 15 to sign block). Permissions for read could be public or permissioned. We can say consortium blockchain as partially decentralized blockchain.

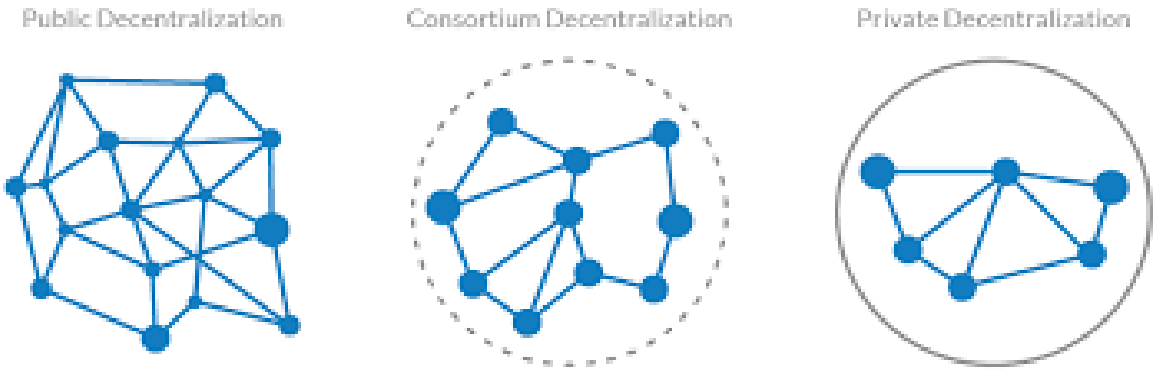

Figure 2.2: Node Topologies for Public, Consortium and Private Blockchain

A short comparison between all three types of blockchain is shown in table 2.1

Property	Public blockchain	Consortium blockchain	Private blockchain
Consensus determination	All miners	Selected set of nodes	One organization
Read permission	Public	Could be public or restricted	Could be public or restricted
Immutability	High (Nearly impossible to tamper)	Could be tampered	Could be tampered
Efficiency	Low	High	High
Centralised	No	Partial	Yes
Consensus process	Permission-less	Permissioned	Permissioned

Table 2.1: Comparison between types of blockchain

2.2.4 Types of Consensus Mechanisms and Algorithms

Some famous consensus mechanisms, such as Proof-of-Work (PoW) , Proof-of-Stake (PoS) and Proof-of-Activity (PoA) are showcased in Table 2.1.

Type	**Working**	**Use case**
Proof of Work (PoW)	In PoW, all nodes dedicate their computational power in a hunt for a specific hash that fulfils certain conditions. When a node finds that hash, it can append his block to the blockchain. The node broadcasts his block to all other nodes. To avoid forks, PoW makes an assumption that the longest chain, having valid mining hashes and correct content is the correct chain. Moreover, the miner of block receives a reward for mining it.	It is suitable for Public blockchain, such as Bitcoin and altcoins etc.
Proof of Stake (PoS)	In PoS, miners put their cryptocurrencies on stake, as if to bet for a block they want to include in blockchain. Whoever bets the largest number of cryptocurrencies has the greatest probability to mint that block. Therefore, it is in interest of all miners to act honestly in network. PoS consumes less energy resources than PoW, which makes it more efficient.	It is feasible for Public and Consortium blockchain
Proof of Authority (PoA)	Some authorized nodes, acting like "admins" does all work related to validating transactions and accommodating them in blocks. This consensus mechanism is more like a centralized one.	It is feasible for Private blockchain

Proof of Elapsed Time (PoET)	Every network participant waits for an undefined slot of time, and the first participant to finish its wait state mints the block.	It is feasible for Private and Consortium blockchain
Byzantine Fault Tolerance (BFT)	In BFT, preselected nodes verifies and orders transactions	It is feasible for Private and Consortium blockchain

Table 2.2: Comparison between different consensus mechanisms, Available in [10]

2.2.5 Transparency

Blockchain availability is not only restricted to mining nodes in the network, even though they actively participate to let the chain grow. In contrast, blockchain records are publically available. Whosoever wants to see it, can see it and validate authenticity of data. This property makes blockchain transparent. In a private or permissioned blockchain, only few nodes might see those records, depending on access control rules organization dictate. An organization do it through asymmetric key pairs.

2.3 Applications

In general terms, there are a different approaches to implement blockchain (private, public and consortium). Business specific goals drive blockchain architectural choice. Ongoing research on blockchain is opening horizons for areas where blockchain can make its place [11]. Some of them are discussed below:

1) **Identity Management**: For notary services where documents are verified are recorded
2) **Insurance**: Insurance can be claimed, given the conditions specified in smart contracts are met.
3) **Health Care**: Blockchain coupled with IoT sensors can help monitor patient's health status and protect integrity of IoT sensor readings.
4) **Distributed Cloud Storage**: Blockchain has a potential to shift traditional centralized cloud to a distributed cloud.

5) **Voting**: Known hurdles in e-voting systems are related to security concerns. Blockchain can increase feasibility and security of e-voting system.
6) **Internet-of-Things (IoT)**: IoT sensors, connected to Internet can send their recorded values to Blockchain where they are stored and processed.

2.4 Blockchain Frameworks

Several frameworks exist to build blockchain, along with other platforms to participate in those already implemented. We do not need to build a blockchain from scratch, since a lot of work is done already. This chapter delves into detail to explain the most important frameworks and the ways they are applicable to different cases.

2.4.1 Ethereum

Ethereum was launched in 2014, with an aim to build a blockchain that is more than a mere backbone for online payment systems. Vitalik Buterin wrote a white paper to explain technicalities of this platform. People begin exploring its inner workings, and its popularity inflated ever since then. Ethereum allows creation of consensus based applications which are standard, scalable and interoperable [12].

Smart Contracts

Ethereum utilizes Turing-complete programming language to store code in the form of contracts. Nick Szabo presented rudimentary idea about smart contracts in 1996 where he suggested smart contracts as a set of promises for each party and some protocols with dictate how each part will fulfill the promise [13]. Although the idea of smart contracts was old, but Ethereum allows for the first realistic implementation of smart contracts. At a broader level, smart contract is nothing but a source code, having a structure following some pre-defined set of rules to transfer assets their titles. A smart contract responds based on its interactions with other elements of blockchain. Those elements can either be specified as people or some other contracts. In simple terms, Ethereum stretches beyond the domain of currency and opens up new horizon for other decentralized applications to run on top of blockchain. Solidity is a custom language which is used to write smart contracts in Ethereum, and later they are compiled to byte codes. Byte code is deployed over chain where Ethereum Virtual Machine (EVM) interprets it.

In contrast to Bitcoin, Ethereum not only records transactions, but also saves states of smart contract. Since smart contract run on transaction validating nodes, therefore, it is also subject to reach consensus.

Currency

Smart contracts being executed by peer-to-peer network nodes takes Ethereum availability to a global level. Nevertheless, computational cost becomes a price to be paid. Every time a function in a contract is called, some mining node does all computations on each line of code. For that computational work, mining node charges a fee. Ethereum's cryptocurrency is called Ether which essentially fuels the network.

Consensus

Similar to Bitcoin, Ethereum uses Proof-of-Work consensus protocol. Few projects, such as Casper [14] are trying to shift Ethereum to Proof-of-Stake. In Proof-of-Stake, mining is based on trust. The fact that a miner having its cryptocurrencies on stake in network will want to act as an honest node in the network. For that reason, a Proof-of-Stake does not dissipate huge computational power. In proof-of-Work, miners do an exhaustive search for a target hash. The one who finds the target hash first wins the chance to add his block to blockchain. All energy that a miner puts in finding the target hash goes wasted [15]. Also, some other concerns like performance and scalability issues indicates Casper can do a better job than Ethereum.

Performance and Scalability issues

Ever since Ethereum was launched, questions were arising whether Ethereum's Latency and Throughput suffice to cater to a myriad of applications running simultaneously. The launch of CryptoKitties degraded Ethereum's performance. It created a bottleneck in the main network and a sluggish processing of transactions started to raise questions over Ethereum's scalability.

These concerns are highly valid for freight management application. If Ethereum were to be used for freight management, glancing over Ethereum's performance and scalability first is of extreme importance. An important thing to note here is Public and Private blockchain will have different performance and scalability. Moreover, certain factors (such as, block time) that influence

performance can also have an impact on scalability. These attributes depend on one another and require a balance where both attributes reach an optimal level. A network comprising only a set of nodes can be set up for Ethereum to separate it from main network. This network will also be public and might face same issues main network faces.

Security becomes a bigger concerns in a public network. Blockchain parameters like block size is kept in a way to prevent different attacks. Adjusting these parameters in a private blockchain is much easier. Whereas, in public chains, tweaking these values causes a bifurcation in chain (known as a Fork). Deploying a private blockchain concentrates power in a fewer nodes. Such networks are more trustworthy and less prone to different attacks. This factor makes private blockchain more scalable.

Average throughput, telling the number of transactions per second measures an overall performance of Ethereum. Two major factors influence throughput:

1) **Block size**: In Ethereum, a "gas" limit defines a block size. Every transaction making into block spends some "gas" value, and when "gas" reaches its threshold, block accepts no more transactions. A "gas" defines computational power required to process transaction or contract. Greater the computational power required, greater the gas value becomes.
2) **Block Interval**: It is average time which a block requires to publish. It is 10 minutes for Bitcoin. However, for Ethereum, it can be extremely short, such as 12 seconds [16]. Block interval is interdependent on Latency. Latency dictates the time a transaction takes to enter the block. Transactions making into block at their earliest shortens latency and block interval.

Variable parameters, such as transaction and block size make it harder to describe Ethereum's performance at a specific point in time. Besides, there's a great need to stem Ethereum's block size, since nodes could not store huge blockchain to validate it.

This is an important security concern, as only a few nodes who can store blockchain will validate them. This situation is unfavorable in Ethereum, where every nodes has to store and process transaction, along with storing entire snapshot of blockchain.

Proposed Solutions

Some of the projects aimed to resolve above mentioned issues in Ethereum are Casper, Raiden, Sharding and Plasma.

Casper

Casper aims to implement Proof-of-Stake consensus algorithm in Ethereum. It functions by letting anyone who holds a stake (in this case, Ethers) in the system to pick up next block. Casper introduces accountability in system. Nodes have to deposit some of their currency to join mining process. Since, they have their own currency at stake, they are required to act as honest nodes. If any malicious act is discovered, they will end up losing their entire stake. Proof-of-Stake mechanism makes blockchain much more efficient. Miners do not require heavy hardware. This in turn reduces electricity bills.

Sharding

Sharding targets to improve scalability of Ethereum. It makes a subset of nodes and transactions. Every subset of nodes verifies a subset of transaction [17]. This allows parallel processing of transaction. More transactions processed per second augments network's throughput. Security is maintained in a better way now, since there are enough nodes validating subset of transactions.

Raiden

Raiden allows a secure token transfer on a side channel outside a blockchain, without a global consensus using hash-locked transfers known as balanced proofs [18]. A side channel is opened for a certain number of transactions, which closes after those transactions being performed. Transactions are then submitted back to chain. Raiden's protocol implements routing between the nodes, making side channel transactions possible. [19]

Plasma

Plasma uses smart contracts to form hierarchical side chains which can be assumed as children of blockchain. These hierarchical side chains will process information independently, having their own set of rules. Transactions are revertible in side chains, since they are built separately from main blockchain. There can be as many side chains as possible, thus increasing scalability [20].

2.4.2 Hyperledger Fabric

The Linux Foundation hosts open-source Hyperledger with an aim to improvise idea of blockchain to enable its use in different ways. Hyperledger Fabric framework is used to create private and consortium blockchain to increase its usability in industry and address issues in existing frameworks, such as Ethereum. Although it shares many similarities with Ethereum, it does have a lot of differences as well.

Chaincode

Hyperledger names its smart contract as chaincode. They function very similar to state machines. Like in Ethereum, events drive a chaincode. Every time they run on ledger, they update its existing state. Hence, they are able to move assets and their ownerships around different participants.

What makes a chaincode different from Ethereum's smart contract is the language it uses. Source code is scribbled in Go language, but a compatibility for general purpose languages like Java and Node.js is embedded in them. Although using general purpose language to write a smart contract is a benefit, a non-deterministic code written in general purpose language can give rise to forks. For this reason, Ethereum sticks to language that it compiles and runs on Ethereum Virtual Machine.

Architectural Revision

Fabric v.1.1 is a later version for Fabric. It has many architectural changes involved which are different than that of its previous versions (such as v.0.6). The architectural changes were made to resolve underlying issues in technology.

Hyperledger Fabric's previous version shares many architectural similarities with other blockchain frameworks. Hyperledger Fabric v.0.6 also uses the same order-execute architecture. In order-execute architecture, after all transactions fit in the block and block is mined, they are sequentially executed by all other nodes. This architecture included many drawbacks for permissioned blockchain. Few issues are discussed below:

1) **Non-deterministic code**: A non-deterministic code might give rise to forks in an order-execute architecture, since a state for each node might change differently.

2) **Sequential execution of smart contract**: A slow smart contract can make others wait for long time.
3) **Hardcoded consensus**: There's a need for alternate protocols, since a different protocol may fulfil needs of a particular use case.
4) **Confidentiality of execution**: In permissioned blockchain, it is sometimes required that a smart contract logic to be run on specific nodes only. Order-Execute architecture does not allow for this type of confidential execution, since states are broadcasted to all nodes in the end.

The revised version, V.1.1 implements "Execute-Order-Validate" architecture to overcome the issues mentioned right above [21]. Revised version promises scalability, chaincode trust flexibility, pluggable consensus availability and confidentiality [22].

System Architecture and Consensus

The two types of transactions in Fabric are either deployment transactions or invoke transactions. Deployment transactions initiate a new chaincode, whereas invoke transactions perform required operations on chaincode.

In a permissioned blockchain, it is required for a node to authenticate itself and bear some identity prior to having any interaction with transactions. Only certain participants of network can see transactions and data, and the way data is partitioned to be visible to certain participants only is done through a channel. Users belonging to a specific channel can only see transactions. Similarly, only channel-specific users can participate in consensus. There is only one ledger per channel. Each channel has a specific set of rules, defining what actions every user can perform.

Besides occurrence of permissioned channels, another big difference in Fabric's revised version is consensus mechanism's functioning. The steps corresponding to consensus mechanism are described below:

Endorsement

The act of endorsing a transaction is simply an endorser signing it to show confirmation. Some endorsers validates transactions first and show their consent to accept or reject it afterwards.

Endorsement policies dictates minimum number of endorsers required for a transaction to be accepted.

Ordering

Transactions accepted during a specific time slot are bunched into a block and committed in the same order.

Validation

Here, transaction's endorsement policy is made into consideration to see if transaction satisfies it. Three different types of nodes are involved in consensus process.

1) **Client**: A transaction is submitted to endorser nodes through client.

Peer Node: peers construct peer-to-peer network, maintains ledger state and chaincode. They can be endorsers and committers

2) **Endorser**: They simulate process of transaction execution and determines whether every condition is met for endorsing it. Endorsers also act as committers.

Committer: They validates endorsements and transactions.

3) **Orderer Node**: Orderer nodes combine to provide ordering service. A pluggable protocol for ordering service orders transactions received by peers, bunch them in a block and broadcast it to committers.

Steps written below describes workflow of consensus, which are also shown in Figure 2.2

1. Client submits transaction to endorsing nodes
2. Endorsing nodes simulate transaction to decide whether to endorse it or not. They simply sign transaction to show they have endorsed it and send it back to client.
3. Endorsed transaction goes to ordering service from client
4. An Orderer groups it with other endorsed transaction into a block and broadcasts the block to all peers. Peers validate endorsements to see if they fulfil endorsement policies.

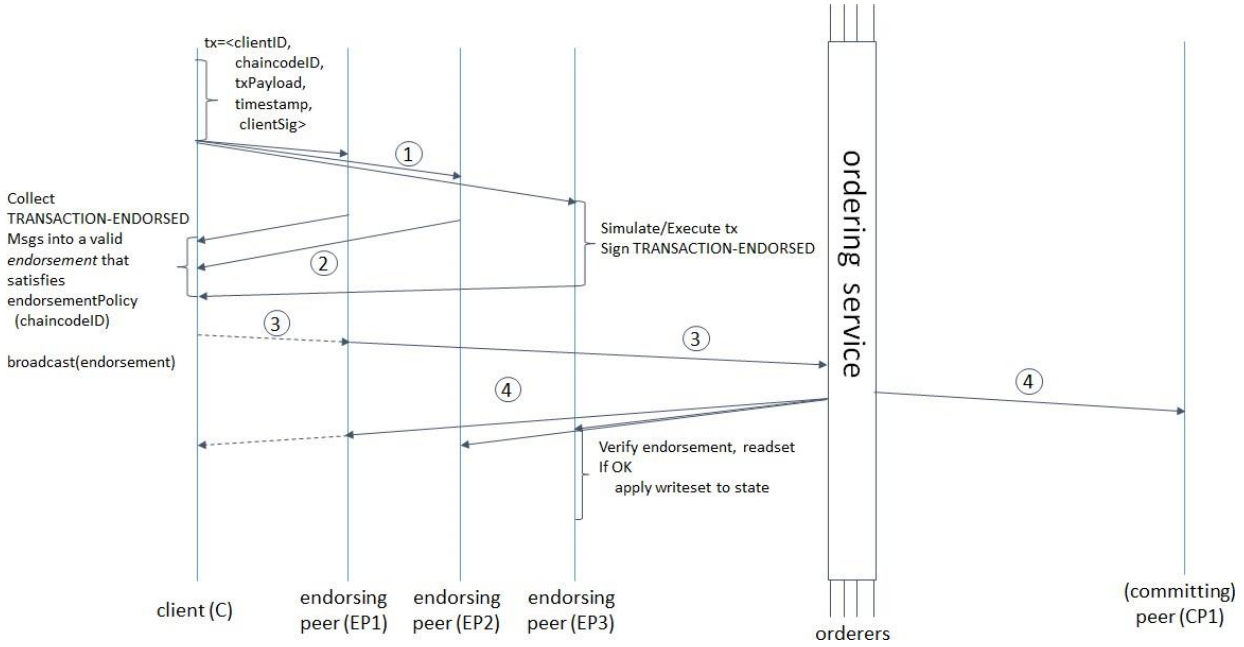

Figure 2.3: Hyperledger Fabric Consensus Workflow

In comparison with other blockchain platforms, this consensus bears some advantages in some use cases. Table 2.2 shows some key differences between a blockchain using Proof-of-Work and others, such as Hyperledger Fabric using a Byzantine Fault Tolerance state machine consensus.

Consensus Type:	Proof of Work (Ethereum, Bitcoin)	Byzantine Fault Tolerance (Hyperledger)
Membership Type:	Permission-less	Permissioned
Node Identity Management	Decentralized, Anonymous	Centralized, All nodes recognize each other
Scalability (no. of clients)	Greater than 100 nodes, without decreasing performance	It can scale to 100 nodes, with performance metrics at stake
Throughput	Limited (15 transactions/second)	Excellent (Greater than 10,000 transactions/second)
Power efficiency	Very poor (Requires special hardware)	Good (Requires simple hardware)
Temporary forks	Possible	Not possible
Consensus Finality	No	Yes

Latency	High	Low

Table 2.3: Comparison between consensus mechanisms

In short, Hyperledger gives up on some decentralization to achieve better scalability and performance. Hyperledger consensus mechanism overcomes issues in public blockchain. Moreover, it allows for strictly authorized access of data to protect sensitive data. This is one of the chief requirements of Freight Management System which is quite impossible to achieve in a public blockchain.

2.4.3 Hyperledger Composer

Hyperledger composer is another framework which functions on the top of Hyperledger Fabric to ease the process of creating blockchain network. It's a layer of abstraction over Hyperledger Fabric to reduce line of codes. It simplifies business network creation, its deployment as well as does a better job in managing network participant identities. It gives a user-friendly graphical interface where a user can interact with all of its components. A REST server is also available to make blockchain available to an application from outside world. [23]

Business Network

A business network defines all objects, functions, transactions and participants that can interact with one another, with all their interactions getting saved in a ledger. It's a chaincode's layer of abstraction which gets installed over Fabric. Since a business network is a network model, it can be run on a specific nodes. A business network is written in composer's modelling language. There are four main components of a business network as shown in figure 2.3.

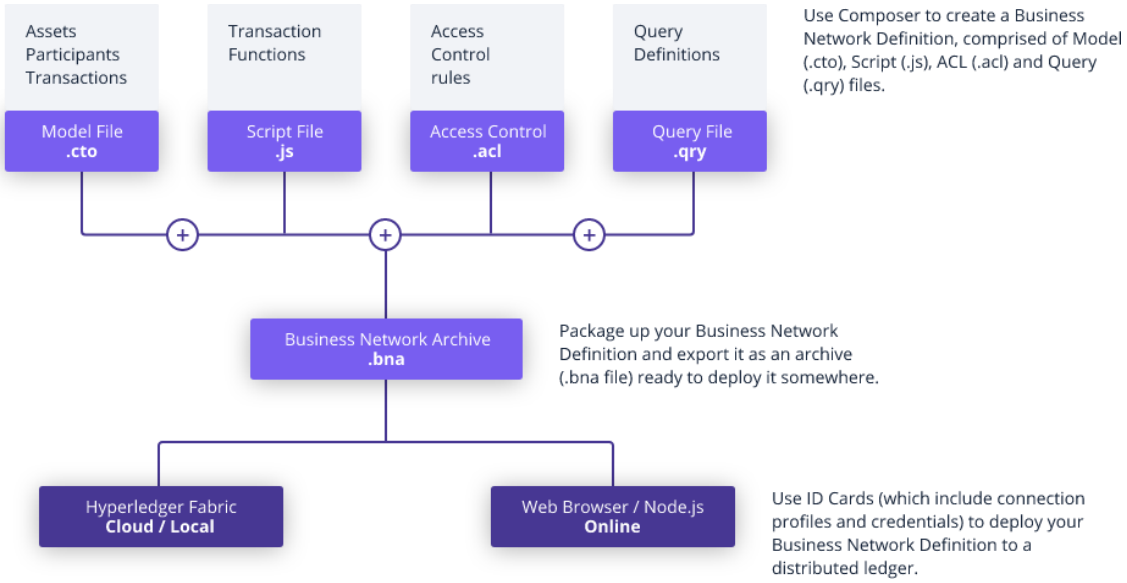

Figure 2.4: Hyperledger Composer over Hyperledger Fabric, Available in [24]

Model File

A model file contains business network definitions. Assets are main objects of model file. An asset can be a variety of things, be it a car, house etc. “Participants” make up individuals participating in business activities on blockchain. Then there are “Transactions” which are the most crucial type of objects defined in a model file. They define what possible actions are to be taken. There can be a “BuyHouse” transaction which has parameters related to House. An authorized participant (such as owner of house) can call this transaction. All three objects of a model file: Asset, Participant and Transaction have their separate registry, where Asset and Participant registries are mutable, but Transaction registry is not. An Event object can be defined in a Model File. A transaction emits an event and an application subscribe it to generate notifications.

Script File

A file containing functions that dictates behavior of a transaction, type of data it can process and output.

Access Control File

All rules related to authorization are dictated in access control file. This file can describe some rules to restrict the rights for invoking transactions, reading/writing of data to specific individuals only.

Query File

This files creates some queries to get some information related to assets, participants and transactions stored in blockchain.

All these four components bunched together forms a Business Network Archive (.bna) file, ready to be deployed over fabric.

Authentication and Identities

Hyperledger Composer also manages identities. By an identity, we mean a private key and a digital certificate. This is analogous to providing an account on a permissioned business network to interact with it. Also, within a business network, an identity is provided with a specific Participant type. The identity will only be able to interact with blockchain according to access control rights provided to that participant type.

To ease the process of managing identities, Hyperledger Composer has Business Network Cards. These cards are simply files holding information about specific identity, network and private key. If users hold a card, they can show it to confirm their identity to interact with blockchain.

2.5 Ethereum vs Fabric

Comparing key differences in two significant blockchain platforms (Ethereum vs Fabric) based on their general characteristics and consensus mechanism shows that Ethereum is a general purpose platform, whereas Fabric is more oriented towards addressing domain specific problems. Also, Fabric has greater malleability than other blockchain platforms. Therefore, a permissioned blockchain, such as Hyperledger suits more to the needs of Freight Management System [25] [26].

Chapter 3

Literature Review

This chapter portrays how blockchain can be integrated into industry in general and Freight Management System in particular. It describes the gains blockchain can bring to industry, with a strong focus on its positive impact on Freight Management System. We will also discuss few challenges blockchain might encounter while making its place in industry. Some applications that have integrated Blockchain into Freight Management System are also discussed. The chapter closes with an elevator speech for some design alternatives, which will become the foundation for design analysis later on.

Chapter covers topics of blockchain and freight management to justify if they can make a good fit together.

3.1 Benefits of Blockchain in Freight Management System

Some potential gains due to features inherent in blockchain are discussed below:

1) **Immune to errors**: It will reduce errors involved in manual entries and those in IoT devices due to certain attacks they are prone to.
2) **Secure transactions**: The ledger is not only immutable, but is capable of detecting any attempts of tampering.
3) **Improved tracking**: The ledger is easy to analyze and transactions happen in near real time, making it feasible to find out status of asset at a specific point of time. Any accident or unfavorable incident happening into system is easily traceable.
4) **Improved consumer trust**: Customers can trace an entire trajectory of their product all the way from its origin to final delivery.
5) **Reduced governance costs**: It can cut down heavy costs due to physical scrutiny at customs and clearance.
6) **Establishing trust between business participants**: It is very significant that participant involved in a business trust the information flow among them and blockchain lets this trust integrated.

Last point is one the most important functionality to achieve in Freight Management System's use case. Trust means being able to rely on data others provide and believe the data is not tampered.

Trust is the driving factor in Freight Management System. Without its presence, system's performance will degrade. In a scenario where you expect to have information quickly, maintaining trust is necessary.

If Blockchain can provide a symmetric information flow in Freight Management System, while maintaining a significant level of trust between business participants, then system will find an improvement in its performance, since business participants will be more satisfied. Therefore**, Trust seems to be a vital factor improving efficiency of existing Freight Management Systems** [27].

3.2 Challenges in Blockchain Application to Freight Management

Blockchain is not fully evolved yet, it is still something people are researching over. Although, some of the advantages it promises are quite obvious, but many other disadvantages are ignored. Considering those disadvantages is important while deciding to implement a blockchain based Freight Management System.

3.2.1 Technical Limitations and Scalability Concerns

Throughput, latency, size, bandwidth and security limitations are involved when using blockchain.

1. **Throughput**: Private blockchain, such as Hyperledger, provides a high throughput, but it is still below what centralized systems offer. This is a big concern in Freight Management System. Blockchain cannot process information faster than current centralized freight management systems. Thus, a blockchain might incur more delays. Companies might have to sacrifice speed for decentralization in freight management. Although, it is quite unfair to compare speed of centralized and a decentralized system, since latter one offers much more functionalities including scattering information between different participants. In a freight management system, a slow flow of information at a global level, among various parties is preferred over a fast flow of information locally. As a consequence, we might look to achieve a trade-off between functionality and throughput.

2. **Latency**: Similar to throughput, latency is also an important factor to bring into consideration. With blockchain applications, like Bitcoin, each transaction entails a large window of time for validation and might require a fee as well. Permissioned blockchain, such as Hyperledger reduces latency, even with a large number of transactions. It also does not have any fee involved.
3. **Size**: Blockchain inflates as the number of transactions increases. In the context of Freight Management System where we want to deploy blockchain at a global level, the chain will grow too large in a short span of time. Nevertheless, this is not a worst case scenario, as there is a lot of research taking place over optimizing size of blockchain.
4. **Security**: Another concern is how a blockchain incorporates security. It is a bigger concern in public blockchain with a Proof-of-Work consensus mechanism. In our Freight Management scenario, we will not be using Proof-of-Work because it is not feasible in our case. We need to take hash functions into account, since a lot of hast functions are rendered useless because of being broken. This concern poses a threat to immutability of blockchain. Any aspect of proving provenance and asset tracking will lose its groundwork.

3.2.2 Lack of Interoperable Standards

Companies often have their own system for inserting their data. Also, companies choose data entry format on their own discretion. Now, if any other company wants to access their data, it will give them a hard time looking for the data, since they will not know where the desired data resides. This is the first problem where due to absence of interoperable standards, companies do not find a common ground to understand each other's data.

Another problem, which is even bigger is lack of interoperability is systems themselves. Many a times, information is inserted manually into systems and there are no APIs with which external systems can get themselves connected.

3.3 Similar Existing Applications

Some projects have implemented blockchain based solution for improving freight management. This section explores some of these applications.

3.3.1 CargoX

CargoX project tried to digitize Bill of Landing document. In a freight management system, a cargo ship inside container delivers product. Bill of landing contains exact values declared on goods. It serves following purposes:

1) A carrier issues bill of landing to acknowledge receipt of cargo for shipment.
2) A bill of landing dictates terms of contract for carriage.
3) It declares the ownership of cargo.

These features make Bill of Landing of extreme importance. It must be transferred from carrier to company requesting products in a tamper-free way. If Bill of Landing is lost, all rights related to goods in shipment will be lost.

CargoX uses Ethereum's Smart Contract to shift paper based documents into blockchain. It has a built-in token system to transfer document's ownership right after a payment is made [31].

Reliance and Applicability

CargoX uses a public blockchain, such as Ethereum for exchanging extremely valuable documents. It does not seem like a good fit for this use case, since it can raise security concerns. If Ethereum becomes a prey to 51% attack, integrity of Bills of Landing will become questionable.

CargoX uses a token system that seems more skewed to monetizing the concept and moving currency around rather than the Bill of Landing.

The idea to transfer documents is a significant factor in integrating information flow, since these documents prove the transfer of ownerships. Undoubtedly, Blockchain can be good way to digitize transfer of titles in a traceable way.

3.3.2 Eximchain

Eximchain acts as a ledger and keeps a trail of all transactions. It uses smart contract and serves as an inventory management tool. The platform was designed using a fork of Ethereum, known as Quorum, which is a permissioned version of Ethereum. Consequently, it can run on its own network as well [32].

Smart contracts of Eximchain allows making payments and validates orders placed. The ledger records all transactions storing data regarding flow ofcommodity. This enables suppliers to claim their delivery is reliable [33].

All goods are traceable, with an easy accessibility of information from all business participants.

Relevance and Applicability

This applications is more focused on important aspects of freight management: process of transmitting commodities, integration of information related to goods and easy tracking of goods by trading parties. This is extremely practical and the thesis tries to obtain similar objectives, which are to alleviate challenges in current freight management systems by integrating information and making it available in near real time.

3.3.3 OriginTrail

Similar to CargoX, OriginTrail runs on Ethereum network and possess tokens specific to project, which are called ERC_{20}. It integrates all freight management data and uses these tokens as rewards. The network uses zero knowledge technique to verify data without making data available. A custom network topology protocol severs as a privacy layer. Data Holder and Data Creator nodes in custom network topology protocol coordinates to receive, transmit, store and process information. Later, blockchain receives same information and people holding ERC_{20} tokens can interact with blockchain.

Project's objective is to make sure products are traceable along their entire journey, while ensuring that nobody tampered data, which means project aims on attaining high privacy. ERC20 tokens serves the purpose to change ownerships of data.

Relevance and Applicability

OriginTrail aims to make traceability ofgoods easy for public, so they can know where goods are coming from and cross-reference it with what a company claims. This allows a better insight into products to ensure their integrity and authenticity. A good use case for this project could be a situation where a spoiled product is detected and a batch recall is made. OriginTrail makes it easy to pinpoint the batch and remove it from supplying any further. [28]

Although, it is a more specialized solution. It can also be simply achieved by integrating blockchain in freight management systems.

3.3.4 Ambrorus

Ambrorus focuses on making sure goods on their way meets quality assurance standards. This is designed for food and pharmaceutical companies. Not only that it traces products' entire trajectory, but also integrates IoT sensors to send quality metrics of products in real time over blockchain.

Similar to OriginTrail, Ambrorus also uses its own protocol AMB-NET to host smart contracts and make its communication with Ethereum feasible. [29] [30]

Relevance and Applicability

This is another project focused on quality assurance of goods through IoT sensors. Not all industries are applying it, but to some, it is the most important of all features. Nevertheless, the project can handle quality tracking much efficiently.

3.3.5 IBM and Maersk's Demo

IBM and Maersk launched a joint project, known as TradeLens which aims to create an open platform designed on Hyperledger Fabric to share information on a broader scale. [31]

It features a pipeline of shipping information, fully integrated by hosting it over blockchain. It also aims to shift trade to a paperless paradigm. It deals with transporting goods and automating all processes involved in it.

Relevance and Application

Although it serves the purpose this thesis is trying to achieve, but the project aims to achieve IBM and Maersk's expectations. Not many companies are content with this platform, since it does not satisfy industrial requirements and neither ensures common standards. Other than this, there's very less information publically available over its progress [32].

3.4 Designing a Blockchain-based Freight Management System

Blockchain is not always a solution that fits all applications. Instead, application requirements should be thoroughly considered to tailor blockchain in accordance with its needs. This section describes some important agendas to have in mind when looking forward to making decisions for design of blockchain based freight management.

3.4.1 Integration Models

Point to point: In this type of integration, each two specific points must be interlinked. Each new connection needs to be modeled separately. This model doesn't work fine when implemented at a broader scale.

One-to-many entities: A company can establish a connection endpoint with which other companies can also connect to using hub communication standards or its API. This way, a company can build connections with many intermediaries.

Many-to-many entities: This model gives full integration where information freely floats between companies. A public blockchain tries to achieve this ultimate goal, but it requires deploying interoperability standards, which are not developed until now. Otherwise, this type of integration is the most cost effective.

3.4.2 Key Implementation Components and Features

All the projects mentioned in this chapter, like CargoX, Eximchain, OriginTrail and TradeLens, each project serves its own purpose and address some specific goals. Similarly, these are some key components that blockchain inherits, and some features that must be kept in consideration:

Information Storage

The most important feature of blockchain is its ability to record data in an immutable fashion, along with notifying about important events. This information storage is very important in Freight Management System. Blockchain also provides ease in Inventory Management, traceability and provenance of products.

Ledger and Transactions

Allowing transactions to get recorded over blockchain is also important in Freight Management System in context to payments made between businesses.

Smart Contracts

Smart Contracts can be an essential part of Freight Management System. They can be used to transfer ownerships of either data or products using digital tokens. This is one of many ways smart contract can work. Other types of smart contracts include tracking commodities by their location or specified condition, automatically updating commodities status over blockchain, as well as notifying about important events occurring. Another feature that smart contracts possess is they automate payments upon delivery.

Smart contracts are the most innovative component of blockchain. They are code of programs run over blockchain, so they can be tailored to serve different purposes based on what an application wants to fulfill.

Chapter 4

Research Methodology

This chapter expatiates on aims of this thesis, clearly defines thesis statement, as well as the approach it will follow to find out how valid statements and underlying assumptions are.

After having mentioned some background information about blockchain, its frameworks, as well as about freight management system and its issues. Now, it's time to explore what these issues actually mean. Hence, we will formulate and test out a possible way to getover these issues using blockchain.

4.1 Objective

This section introduces main points of research content and objectives of research work.

4.1.1 Point of Focus

In a freight management system, most products travel from one place to another, undergoing iterations of processing and shipments, along with changing owners. These phases are involved in almost every industry, even for the simplest of products (which don't require any processing) will be shipped from one place to another, where they will be sold finally.

The improvement of freight management life cycle is one the greatest aim of this thesis. This improvement, however, involves many points of focus. Some of them are summarized below:

Speed of delivery

Products from one part of globe get shipped to another part in weeks or a few days. The world is moving forward so quickly, so a matter of days or weeks might seem long. The faster the products get in hands of buyer, the faster can his needs be satisfied.

Synchronization

Most of the times, data residing in a company is in sync with its own servers and software, and in protocols and formats that a specific software can decipher only. Hence, if many companies share the same specific software, they can integrate their information easily. The real problem arises

when companies cannot find a common ground and there's no automatic way to transfer data. This leads to a tedious manual work to export data from one system and import it to another. Although, there might be many other causes for this problem, but the logical assumption is the problem stems from the following two main reasons:

i) Lack of data integration standards in freight management industry.
ii) Lack of common technology storing all data, so that each company can extract information using its own software.

Tracking

There are many possibilities of alterations in a product's record during entire freight management life cycle. Many times, records pertaining to origin of product are lost during a product's journey. They are often falsified or skip the process of making an entry in registry. All these lead to a minimized trust and reliability in goods a consumer uses. This may also happen that products not being properly tracked does not comply with the quality standards regulatory parties define. This can lead to a product not safe to be consumed.

Security

This is one of the most important aspect to be discussed, as security itself encompasses many other aspects, such as: Who is authorized to access information? How to restrict information access? What authentication methods shall be used? How to detect fraudulent activities efficiently?

Information related to freight management is very sensitive. It should be regulated such that only trusted entities can gain access to it. Most companies compete to make the most deliveries and be the one with the fastest product cycles. Thus, the information generated in the process of managing freight might be too sensitive to share, to keep the company on the edge of competition. Moreover, the information generated and entered into system should be verified to thwart human errors and malicious activities.

4.1.2 Possible Solution

The above mentioned focus points are not actually problems, but improving these areas can benefits industry and its consumers. So, the actual problem narrows down to **finding out a way to cater to needs for improvement through technology.**

Traditional solutions for this problem call for a distributed system, and many solutions are readily available as well. However, distributed systems like cloud or distributed databases do not cater to needs of freight management systems. Thus, researching for an alternate design is a good approach to explore a new horizon that may revolutionize the system entirely.

So, the question remains same, whether there exist any technology can be implemented to improve freight management system, in a scalable and secure way.

Blockchain has many of its aspects similar to distributed databases. There's a need to find out if might or might not be a good approach to address freight management system issues.

4.1.3 Thesis Statement

Main goals of thesis are already been defined: to improve security, tracing goods and process events, synchronizing information and increasing speed of delivery of goods. Blockchain seems to be a nice approach to address these goals.

The statement of this thesis is defined as:

"Blockchain possesses a good architectural design for freight management system"

What remains unanswered yet is, whether the attributes mentioned right above are the main points to focus on, and if so, can blockchain be a good substitute for other distributed architectures to achieve them. The statement seems inclined on some assumptions which arises further questions. Since, we are assuming points of focus mentioned in section 4.1.1 are of extreme significance.

So, the questions that call for answers first, to solidify conclusion for the main thesis statement are mentioned below:

1) Which Blockchain framework is the most suitable for developing an architecture to support these requirements?
2) Is it possible to create a feasible architectural design, using such tool or framework for implementing all these requirements?

4.1.4 Approach

The answer for the questions raised in section 4.1.3, next chapter is written over solution design and implementation, which focuses on reaching a conclusion while answering questions that thesis statements puts forward.

Solution Design and Implementation

This section focuses to apply knowledge over which are the most important aspects of freight management system to be improved, to answer the two final questions: “Which Blockchain framework is the most suitable for developing an architecture to support these requirements?” and “Is it possible to create a feasible architectural design, using such tool or framework for implementing all these requirements?”

Finally, to answer the final question about a feasible blockchain based design for freight management system, we implement a proof of concept, using the platform we chose.

The approach goes with the following phases:

1) Design: Building a test model which can implement the requirements
2) Implementation: Program system according to design
3) Validation: Verify if built system fulfills requirements fully, and if not, give a solid explanation.

After finishing this approach, we will analyze our results to find out which use cases can be implemented and which cannot. We will also analyze the limitations found. Taking this kind of analysis will allow making some conclusions related to some remaining questions.

After answering all two questions related to thesis statement, we can analyze the validity of thesis statement much better. Although, future work might bring some new answers, since blockchain is still evolving due to different researches being carried on it.

Chapter 5

Solution Design and Implementation

This chapter aims to answer the two fundamental questions. The first question, “What Blockchain framework is the most suitable to develop an architecture to support requirements?” entails a thorough analysis of available frameworks. Second question, “Is it possible to build a feasible architecture and design by using framework to implement all requirements?” requires forming a list of requirements, building an architectural design and implementing it over the chosen framework. At the end, we need to verify the extent to which Proof of Concept meets these requirements

This chapter follows a sequential approach to deal with these tasks. The first section focuses on analyzing which framework best satisfies the requirements of freight management use case, and later chooses that framework. The latter ones specify requirements, follows up with a design, implements it using the chosen framework, and finally, verifies the requirements.

5.1 Framework Comparison and Choice

It’s time to answer our first question: “What Blockchain framework is the most suitable to develop an architecture to support requirements?”

In order to choose a good framework, our analysis should focus on the most important functionalities an information system could offer, along with what functionalities are the most important to achieve for freight management use case. We will also bring into our consideration the performance analysis made in section 2.4.5 to choose a good framework.

5.1.1 Framework Requirements

The requirements for making choice for a framework are derived from the attributes that are main focus point of this thesis: **synchronization**, **security** and **traceability**.

Security

Improving privacy is not a concern in freight management systems. Current systems are already providing much improved privacy. There’s a need to consider other security aspects, such as access

control and detecting fraudulent activities are the most important requirements to achieve for our use case.

Thus, the chosen framework should support an environment where good authentication mechanisms are placed, which require proper authentication for a user who wants to interact with the system. The actions against all users should be properly authorized through fine-grained access control rules. There should be an additional support for fraud activity checks.

Traceability

Proposed system should be able to keep a track of all information and any changes made in information, which includes maintaining registry for assets, network participants and transactions. Moreover, fraud detection schemes should be in place through smart contracts.

Therefore, the chosen framework must do all house-keeping related to data management for assets, participants and transactions, which should be made available only to specific set of entities.

Synchronization

Any external system that needs to query or enter blockchain data shall be made blockchain readily accessible.

Thus, the chosen framework should make the blockchain accessible to outside world through REST APIs.

This information is abridged in table 5.1.

Focus Area	**Framework Requirements**
Security	• Authentication and Authorization mechanisms for access control. • Fraud detection through smart contracts.
Traceability	• Managing registries for assets, participants and transactions. • Fine-grained access control checks for data, but it should be visible to auditors.
Synchronization	• Expose data to outside world through REST APIs.
Financial Sector	• Using native cryptocurrencies or an alternate way to simulate cryptocurrency functionality.

Table 5.1: General Framework Requirements

5.1.2 Framework Choice

Now, we can deduce some conclusion for framework choice based on framework requirements. First of all, a private blockchain seems more suitable, because it meets all requirements related to security. Secondly, framework should not only perform asset management, but also manage network identities, where network related participants can be permitted to interact with blockchain. Last, framework should have APIs.

Cross-referencing these requirements with the abilities of frameworks discussed in the previous chapters, we can analyze their applicability for our use case.

Ethereum

A public framework that uses a native cryptocurrency, and makes a strong use case in financial domain. However, it charges fee to run the code. Hosting required functionalities on Ethereum will become much more expensive than in a private blockchain. Moreover, it does not have identity management feature, authentication and authorization checks. Although, it has strong financial capabilities, but it fails to serve security aspects and handling costs.

Hyperledger Fabric

It features authentication and authorization mechanisms, an essential requirement for our use case. Similar to other blockchain platforms, it also features smart contracts. It can manage data and asset much efficiently, as it is highly customizable network. And for synchronization, it features REST servers. It has one drawback, although it is customizable, but it does not support cryptocurrencies functionality for financial transactions, so they need to be designed from scratch to support financial transactions.

Thus, Hyperledger Fabric seems to fulfill MANY of these requirements.

So, our first question, "**What Blockchain framework is the most suitable to develop an architecture to support requirements?**" can be answered. Hyperledger Fabric, together with

Hyperledger Composer is the framework that meets most of requirements for freight management system, along with cutting huge costs that other blockchain platforms will incur.

5.2 Requirements Specification

The layout of this specification will be similar to organization of IEEE 830-1998 requirements specification standard [42]. Specification is presented in the following way:

1. Introduction - Product purpose, scope, overview and users.
2. Requirements - Specific requirements including functional requirements, non-functional requirements, permission list and design constraints.

5.2.1 Introduction – Project Drivers

Scope of Work

The project presents a proof of concept to validate feasibility of implementing a blockchain-based freight management system using Hyperledger Fabric and Composer. It does not showcase a fully developed project. Nevertheless, the goal is to present a concept of product which can improve freight management, by making data accessible in near real time, reducing time required to synchronize information from various participants, providing improved integration and a tool that guarantees end-to-end traceability.

Scope of Product

Proof of concept develops Hyperledger Composer business network over Hyperledger Fabric's node topology. Business network also comprises of blockchain ledger model and integration endpoints. We will design a ledger to fit transactions for freight management. We will also execute smart contracts, in the form of transactions, for asset and identity management of participants. Building integration modules for external systems is out of scope of this project. However, we will create API's required for integration.

Client, Customers and Stakeholders

The project will benefit any industry using freight management system, but it is more oriented to serve any company who hopes to integrate their own systems with this ledger, to maintain a common information transmission platform, as well as maintain an indelible trail of records.

Possible stakeholders for this system and their respective gains are listed below:

1. Freight Management Executives, Managers and Employees.
2. Shippers, Shipping Lines, Freight Forwarders, Customs and Clearance Brokers, Port and Terminals: Availability of information related to partner entities can speed up the process and increase trust.
3. The consumer: consumers might be able to track their goods more precisely.
4. Auditors and Certificate Authorities: It becomes easy for auditors, since they only have all necessary information available at a single place.

Examples of users of the products are:

1. Freight management members: Employees from all companies will be categorized as common users. They can record incoming and outgoing product details on blockchain
2. Regulatory entities: Auditors can view all information related to all companies, to check if system is performing fine without any fraud.
3. System administrator: They are in charge of network and see if network needs maintenance. They make sure issues are resolved on time.
4. Integration developers: They are responsible for integrating companies with the blockchain network.

The system follows a Role Based Access Control approach. Each user's identity is tied to a specific role in the system, according to the tasks they are given to perform. The roles, together with the system itself are actors of system:

1. System: Hyperledger Fabric and Hyperledger Composer have certain adjustable behaviors, which makes system itself to act like an actor with some specified requirements
2. Regulator Entity
3. Admin
4. Freight Management Members
    i. Shipper
    ii. Shipping Line
    iii. Freight Forwarder
    iv. Customs
    v. Port and Terminal Authorities

5.2.2 Functional Requirement Drivers

Functional requirement list begins with a brief explanation, followed by each requirement, with ID and description. Since we are using Hyperledger Fabric and Hyperledger Composer, their requirements will contain software specific terms, such as asset registry, transaction registry and participants.

System

The system should be able to keep a track of every data and accommodate them into blocks. System records data in transactional format, which dictates all actions performed in the system. Similar to transaction registries, there should be registries for network participants and assets as well. It is very important that only participants having permissions should be able to invoke and view these transactions.

Identity management, access control, consensus mechanism and synchronizing information are system related concerns. Some of them are provided by Hyperledger and some are required to be implemented. Functional requirements are listed below:

1. FMS1- System shall execute chaincode transactions.
2. FMS2- System shall record all user-related actions into registry, along with identifying user and activity it performed.
3. FMS3- System shall keep an indelible record of all past transactions in the form of blocks.
4. FMS4- System shall permit submission of transaction batch, which means sending many transactions simultaneously.
5. FMS5- System shall associate a timestamp with each transaction.
6. FMS6- System shall permit creation of multiple ledgers through different channels to support a subset of permissions and separating information of organizations.
7. FMS7- System shall notify important events.
8. FMS8- System shall keep track of total batch count under each participant's possession.
9. FMS9- System shall define separate permissions for each role, such as permissions for invoking data, reading ledger data etc.
10. FMS10- System shall expose all user performed actions to outside world through a REST API

11. FMS11- System shall authenticate user through a REST API

Freight Management Members

For freight management members, all these requirements are only performable in case the actor has permissions to do so, as we mentioned in S9. In this system, freight management members shall be able to manage their assets and shipments, with their actions (performed by invoking transactions) recorded on immutable ledger. Users can invoke a transaction for actions, such as: create, edit and delete assets, interact with shipments, read shipment and asset status. All these actions help data management, traceability aspects.

1. FMM1- Freight management members shall be able to invoke transactions.
2. FMM2- Members shall read information related to assets, transactions and other participants, in accordance with assigned permissions.
3. FMM3- Members shall write and deploy contract based agreements on blockchain.
4. FMM4- Members shall be able to query and acquire all states an asset has gone through, making tracing effective enough to pinpoint its origin and trace it all the way down to the point the product currently is in the journey.
5. FMM5- Members shall query assets a user has in his possession.
6. FMM6- Members shall create assets.
7. FMM7- Members shall delete or edit assets they possess.
8. FMM8- Members shall create shipments with assets they hold.
9. FMM9- Members shall add contract based agreements to their shipments
10. FMM10- Members shall query shipment specific information, including owner and holder of shipment, together with assets involved in shipment.
11. FMM11- Members shall query shipments owned by specific user, if they have permission for it.
12. FMM11- Members shall query user who has specific asset in possession, if they have permission for it.
13. FMM13- Members shall check a shipment status, status of all assets in their shipment, if they are either buyer, seller or current holder of shipment.

14. FMM14- Members shall submit a report for item damage for an asset during their shipment.
15. FMM15 - The members shall update the shipments they hold, as well as their status.
16. FMM16 - The members shall be able to modify their identity.
17. FMM17 - The members shall submit input to an XML file, which has a standard format to automate data submission.
18. FMM18 - The members shall hold a cryptocurrency token or a balance, in order to allow payments, penalties and contractual agreements.

Freight Management Regulatory Entity

An auditor is given a role for manual fraud detection and ensuring proper functioning of system. For that purpose, an auditor needs to have a full read access to system.

1. FMRE1- Regulatory entity shall query and get information about steps particular item has gone through, as if to effectively trace entire trajectory from its origin to the current point in its journey.
2. FMRE2- Regulatory entity shall query all assets, participants and transactions in the system.

Freight Management Admin

An admin severs a major purpose in a blockchain ecosystem. Any problem related to node maintenance has to be solved by admin. Admin registers all other users who want to interact with the blockchain network. It is an authoritative entity who can thwart fraud attempts and make system stable and secure. Admin deals with identity management, authentication and authorization protocols. Thus, it is Admin who enforces different requirements.

1. FMA1- Admin shall revert a transaction by submitting a second transaction that has an opposite effect of first transaction.
2. FMA2- Admin shall create and delete new channels.
3. FMA3- Admin shall create network identity cards for users.
4. FMA4- Admin shall issue network identity card to a network participant, such as Sara's card can be associated to a network participant Auditor#21.

5. FMA5- Admin shall create, delete and edit assets.
6. FMA6- Admin shall submit any transaction.
7. FMA7- Admin shall give others a permission to change roles.

These requirements are jotted down with Hyperledger architecture and Freight Management System needs under consideration. They are loosely written because they will be mentioned again during implementation of proof of concept.

Data Requirements

Another important requirement apart from user-related actions is the format data is written in. One main objective of a blockchain based system encompassing entire freight management, is to standardize data, to make it easy for organizations to import and export data from ledger to their systems, and data retains in a format an organization can understand.

In a real scenario, these data requirements need to be met perfectly. However, this project serves as a proof of concept only, so it will serve minimum number of fields required to cater to basic system functionalities.

5.2.3 Non-Functional Requirements Drivers

At an abstract level, the product should follow these parameters

Usability

Product, APIs and documentation should be transparent enough to allow developers to implement oracle, which is a software that links blockchain with external systems. It will serve as a medium to push and pull data to and from blockchain and external systems.

Performance

We have already analyzed that throughput and latency of Hyperledger Fabric in previous chapters. The throughput of the system is not required to be as high as centralized systems, but time required to synchronize information between companies might augment (decreased latency). The objective is to expedite product delivery to meet the ultimate business goal, even if it does not provide other performance metrics better than available frameworks.

Accuracy

It shall record exact copy of data entered. Any mismatch in data entry shall not go undetected.

Reliability and availability

The product shall be unavailable only if all nodes fail simultaneously, which is quite impossible, unless a coordinated attack happens on system. If a few nodes fail, it might lower the response time.

Scalability

The product requires number of nodes, commensurate with number of companies involved in freight management.

Maintenance and portability

The product will run on Linux based system, having compatibility for nodejs, docker, golang versions that Hyperledger Fabric uses. Creating new nodes and moving former nodes should involve easy procedure, with least complications.

Security

Concerns related to security which are specific to our use case are listed below:

Privacy

Appropriate visibility of transactions and product must be considered by system. Sharing some private data might be extremely risky for a company.

Immutability

No one shall be able to alter contents of ledger.

Authorization

People possessing data should approve any chances to be made in it. For instance, a shipment transaction for delivery shall be approved by person delivering it and its recipient.

Constraints

1. There should be defined APIs for the product to allow flow of incoming and outgoing data between ledger and external system.
2. The product shall be built over open source framework. We will use Hyperledger Composer and Hyperledger Fabric for this project.

Project Issues

Open issues

The architectural complexity involved in Hyperledger Fabric and Composer makes it very difficult to study them completely. The open source nature of these software renders them constantly evolving. Therefore, it is not a complete and reliable product for fulfilling all requirements. In our case, since it is a proof of concept, so we are taking the risk.

User problems

A company who wants to use product will have to understand APIs and build their own oracle for data synchronization.

Limitations in Implementation Environment

The number of companies the product can accommodate is still unknown, since it is interdependent on the number of nodes each company will run and the amount of data passing through the network.

Costs

Creating a blockchain network heaps no cost at all, as it only requires a few nodes and each of them might be controlled by different companies.

5.3 Design and Implementation

Although few projects design entire blockchain network software from scratch, while others build partially custom software, whereas using existing frameworks, like Hyperledger Composer can expedite pace of designing a fully functioning prototype.

List of requirements, specifically functional requirements will serve as a base for prototype design. We can divide design in 4 phases:

1. Model Design for Composer Business Network
2. Access Control Design and Identity Management
3. Network Topology and Deployment
4. Integrating Existing Systems and Building External Applications

It is not necessary to execute these requirements in order, but following this order during development can bring the best results. The thesis does not explore all these listed aspects in depth, since scope of development in this thesis is not very broad. Hence, the project focuses on functional side of applying blockchain to freight management, not very much on quantitative side which would include running tests on network to check network's performance (throughput and latency). It means that our project mainly focuses on model design, access control design, identity management and access control management. Our thesis also discusses essentials of integrating existing systems and building external application, but only on a shallow level.

This section is divided into different sections to give reasonable explanation for achieving the listed aspects. At the end, we will reach a final conclusion to answer our question: "Is it possible to build a build a feasible architectural design using such framework to implement all our requirements"

5.3.1 Composer Business Network – Model Design

The business network model for Hyperledger Composer was designed and built, keeping in mind a list of functional requirements (discussed in section 5.2.2). A .cto file specified definitions of all class types, such as participants, assets, transactions and events. A script file implemented transactional code, whereas a query file contained custom queries to fetch specific data from blockchain.

Participants

The very first step in designing system was to define users to model participants of business network. This is not very difficult, since we have already discussed actors of freight management system previously. The participant types are listed below:

1. Auditor:
2. Freight Management Members: Freight Management Members are the actual users who will interact with the blockchain network, so it makes sense to bring them into

consideration while modeling design. There are subtypes defined in the model, since different participants may behave differently, so they should have different access control rights defined based on their subtype.

i. Shipper
ii. Freight Forwarder
iii. Inland Transporter
iv. Customs
v. Port and Terminal Authority

Both auditor and freight management members would have attributes, such as company name, personal identification etc. Admin does not show up here, because an admin does not require a user type to invoke transaction. They only require their admin card, which is why theydon't need to be associated to network participant class.

Asset

Assets are one of the main components of system, because asset management is an important part of freight management. If we want to trace products, we need to model these products as assets, so that network can maintain a registry for which assets exist, their status and any changes that happen to them.

Proposed assets for freight management system were: Commodity, ShipmentBatch and Contract. Each of them serves a different purpose. A diagram depicting asset-participant relationship is shown below in Figure 5.1.

Commodity- A commodity is a single product being exchanged in freight management, having attributes, such as Product ID, name, description, status of item, ID of person who is the owner of product.

ShipmentBatch- It represents a physical shipment which a buyer orders from a seller. It includes all shipment related information, such as tracking number, shipment status and location. Shipment also has a list of Commodity that it carries, it also possesses information related to the current holder of shipment and other participant owners. Since a shipment has an origin and a destination as well, it is important to have a contract associated with each shipment.

Commercial Invoice- This digital contract makes contractual shipment agreements possible. It contains shipment conditions, expected arrival time and location, buyer and seller.

Transactions and Transactional Script

Participants can interact with network and assets through transactions. A network participant invokes certain parameters of transaction's chaincode function. Assets and participants model users and storing data, whereas transactions model network's behavior by accessing ledger registries for participants and assets and making changes in it. Any transaction being invoked is always recorded over immutable ledger.

Hyperledger Composer has a built-in support for create, delete and update transactions, but we need to model all these behaviors. In many use cases, we want to customize default behaviors, as if to ensure system's integrity remains intact.

Following list shows transactions modeled for freight management network.

1. **CreateShipment**: It creates a shipment and associates a contract with it for certain parameters, such as buyer, seller and commodities in a shipment.
2. **UpdateShipment**: It updates tracking status for a shipment, its location and transfer it to a new owner**. It emits event notification "check for fraud" if shipment does not arrive at expected location**
3. **UpdateCommodity**: It is a customized transaction to update a commodity rather than using default transaction. The aim is to apply access control over this behavior.
4. **ReportDamagedGood**: Many times, commodities are spoiled during shipment. This transaction changes status of transaction and reports a description of a good's damage.
5. **TransferCommodityPossession**: It transfers the title of commodity to another business network participant. **It emits event notification about change of ownership.**

Queries

We have transactions to model user actions and behavior, but to fetch the same information, we need something else. Composer retrieves information about participants, assets and transactions through queries. Hyperledger Fabric stores asset and participant registries in a relational database and Composer uses queries to fetch data from database.

In our proof of concept, we programmed some of the queries to retrieve important chunks of information to make sure traceability is achieved. The designed queries are shown in Table 5.2

Queries are easy and simple to program. We have designed a basic query of type SELECT, followed by what object we want to pick, with a WHERE statement in the end specifying conditions like equality, inequality etc.

Query No.	Query Name	Retrieve	Parameters
1.	selectCommodityItemsinBatchByID	Batch Count	Freight Member ID
2.	selectBatchTransferbyTimestamp	Batch Count	timestamp
3.	selectShipmentByID	Shipment	Shipment ID
4.	selectShipmentByOwner	Shipment	Shipment Owner
5.	selectShipmentByHolder	Shipment	Shipment Holder
6.	selectShipmentByCountry	Shipment	Location Country
7.	selectShipmentByTrackingcNumber	Shipment	Tracking Number
8.	getHistorianRecords	Transaction	None
9.	getHistorianByPerson	Transaction	Participant
10.	getHistorianByType	Transaction	Transaction Type
11.	getDamagedFreightTransactions	Transaction	None
12.	getCreatedShipmentsTransactions	Transaction	None
13.	getCommodityOwner	Commodity	GTIN
14.	getShipmentWhereCommodityExists	Shipment Batch	Commodity

Table 5.2: List of Queries for Freight Management Data Retrieval

5.3.2 Composer Business Network – Identity Management and Access Control

Creating a model design for the network acquires all basic functionalities expect one, which is the most important aspect required in freight management: security and access control mechanism. As previously explained in section 2.4.3, composer features network participants, which represents users, but it also features their identity cards which serves purpose of private keys. An identity card is tied to a participant, serving as a virtual ID card for a person.

Access control rules dictates actions specific to certain participant class types, specific participant instances and specific identity cards. A set of access control rules are designed to fulfill requirements, such as only specific people are permitted to access certain information, so that it renders people's data unavailable to unspecified people.

These access control rules are written in access control composer language in .acl file. The designed permissions are divided into following types

1. Transactional data and invocations.
2. Asset data and CRUD actions.
3. Participant data and CRUD actions.

Permissions and Access Control Rules

Transactional data and invocation

1. **CreateShipmentAndContract**: All participant types, except **customer** participants can invoke this transaction.
2. **ReportDamagedGoods**: Only the owner of shipment holding specific commodity can invoke this transaction.
3. **UpdateShipment**: Only the owner of shipment can invoke this transaction.
4. **TransferShipment**: Only the owner of shipment can invoke this transaction.
5. **UpdateCommodity**: Only the owner of commodity can invoke this transaction.

Commodity

1. Create: Freight Management Member participant type can perform this action.
2. Read: Freight Management Member participant type who owns the commodity being read and auditor participant type can perform this action.
3. Update: Freight Management Member participant who owns the commodity being updated can perform this action.
4. Delete: Only Admin can perform this action.

Commercial Invoice

1. Create, Update and Delete: Only Admin can perform these actions.
2. Read: Owner, Current holder of shipment, Buyer and Auditor can perform this action.

Participants

Freight Management Members

1. Create, Update and Delete: Only Admin can perform these actions.
2. Read: Auditors can perform this action, as well as allowed for a participant to read its own details.

Auditors

1. Create, Update and Delete: Only admin can perform these actions.
2. Read: Auditors can read their own details.

These access control rules gives insight to some basic facts. Auditors are allowed to read anything in the network, whereas Admin can read, update, delete and invoke any transaction. Other participants can read details specific to themselves. Same rule goes for assets and transactions, means the participants holding assets can read its details and can invoke transactions on assets that are associated to them (they need to be either buyer, holder or owner of asset to invoke transactions).

The CRUD actions that Freight Management Members can execute on assets and participants are also limited, as if to maintain the integrity. This is important because composer, by default, has no mechanism to implement checks on CRUD actions. This is why, some custom transactions are designed to implement those checks.

Conclusions and Security Concerns

Identity Management is a difficult thing to achieve in Hyperledger Composer Framework. Human errors and social engineering attempts can easily bypass its Identity Management mechanisms. Hyperledger Composer only gives an immutable ledger feature.

Moreover, some transactions can be made more secure if “multisig” feature is incorporated, where multiple parties can sign a transaction. It can make UpdateShipment transaction work better, since it will require both parties, such as a buyer and shipment holder to acknowledge a transaction. **Hyperledger Composer does not feature multisig yet, therefore it does not completely satisfy security requirements**.

5.3.3 Network Topology and Deployment

Hyperledger Composer features different topology designs for network, such as choosing number of organizations featuring the network, number of nodes each organization deploys and endorsement policy. This type of information can be extremely useful to carry out a quantitative analysis to test what topological setup is the most suitable to maintain a trade-off between performance metrics and scalability of system. Since the scope of our proof of concept is limited to only testing whether the requirements stemming from the problem statement questions can be successfully implemented or not, it does not tests different topological behaviors to see which one meets industrial needs the best. We worked with the simplest network topology, such as one organization topology, with two peer nodes to see if we can achieve our requirements.

5.3.4 Rest Server API and Authentication

Composer features a fast functioning API server, which lists all CRUD operations, queries and invoked transactions. Rest Server API also provides authentication using identity card files. We can also write a customized Rest Server API using Angular applications, but the default one already has features required for implementing basic functionalities.

5.4 Results and Validation

5.4.1 Requirements Validation

In order to find out answer to the second problem statement question, *"Is it possible to build a feasible architectural design, by using such a tool, to implement all these requirements?"* we need to evaluate design and implementation objectively.

Proposed methodology entails validating functional and non-functional requirements from specification.

For functional requirements, we need to determine which of the requirements were feasible for implementation, which ones were partially feasible, which ones were found infeasible and which ones were not determined due to lack of time for development to reach a conclusion.

For non-functional requirements, we will make a few comments in regard to whether each of them was fully satisfied, partially satisfied or unsatisfied.

5.4.2 Functional Requirements Validation

FMS1	Permit chaincode transactions	possible	**FMM10**	Query a specific shipment	possible
FMS2	Keep a track of all user acts	possible	**FMM11**	Query a shipment a user holds	possible
FMS3	Keep an immutable track of all transactions in blocks	possible	**FMM12**	Query a user asset	possible
FMS4	Submit transaction batches	not possible	**FMM13**	Check status of shipment, location and status of item	possible
FMS5	Timestamping transactions	possible	**FMM14**	Submit damage item reports	possible
FMS6	Create multiple channels in composer	not possible	**FMM15**	Update shipment location and status	possible
FMS7	Notify important events	possible	**FMM16**	Edit own identity	possible
FMS8	Track batch quantity	possible	**FMM17**	Give an XML file as input to add data to	possible
FMS9	Set different permissions	possible	**FMM18**	Hold cryptocurrency or allow its functionality using account balance	partially
FMS10	REST API creation	possible	**FMRE1**	Query all steps of product's trajectory	partially
FMS11	REST Authentication	possible	**FMRE2**	Query system registry	possible
FMM1	Invoke transactions	possible	**FMA1**	Revert transaction	not possible
FMM2	Read assets, participants, transactions according to specified access control	possible	**FMA2**	Create and delete new channels	partially
FMM3	Write and deploy contract based agreements	partially	**FMA3**	Create network identity cards	possible
FMM4	Query all steps of product life cycle	partially	**FMA4**	Assign identity cards to business network participants	possible
FMM5	Query owner of asset	possible	**FMA5**	Update participant details	possible
FMM6	Create new assets	possible	**FMA6**	Create, delete and edit asset	possible
FMM7	Edit and delete assets in possession	possible	**FMA7**	Submit any transaction	possible
FMM8	Create shipment of assets in possession	possible	**FMA8**	Change a user role	possible
FMM9	Add a contract based agreement to shipment	possible	**FMA9**	Assign others a permission to change roles	possible

Table 5.3: List of Validated Functional Requirements

S4: **Transaction Batches**: Hyperledger Composer submits transaction one by one, and not all transactions might get into same block, so transaction batches concept is not yet incorporated into Composer.

S6: **Multiple Channels**: Composer has no support for multiple channels, so it is not possible to give organizations a feature of private information sharing through Hyperledger Composer.
A1: **Reverting Transaction**: Composer does not feature revert transaction, since the run-time API has no methods available to access transactions, which means accessing transactions occurred in past requires interaction with script file.

Requirements that were partially met are listed below:
FMM3: **Deploying contractual agreements**: It is only possible to make contractual agreements in a specified format, which means there's no space for submitting any customized agreements in Hyperledger Composer.
FMM4 and RE1: **Querying all steps of product's trajectory:** The product might have gone through some transformations, during journey. Current version of composer creates deletes the entry for commodity that has undergone transformation, and create a new entry for it. Therefore, it does not support to keep a complete track of transformed commodities.
FMM18: **Holding a cryptocurrency**: The concept of cryptocurrency is incorporated in Composer through "balance", which is why it is partially implemented. It retains same usability criteria as that of cryptocurrency, in a sense that a balance cannot be double-spent because of consensus mechanism. It has a few limitations, since it cannot be transformed into other currencies.
A2: **Create and delete new ledger channels**: Fabric supports this feature, whereas Composer does not. Theoretically, an admin can create a channel in Hyperledger Fabric, but cannot make Composer to use new channel.

5.4.3 Non-functional Requirements Validation

Usability	API design is simple. It has APIs for assets, participants, transactions and query.	Satisfied
Reliability	Greater the number of nodes, greater will be reliability, but endorsement policy decides the subset of nodes for some specific transactions. So if those nodes are down, transactions might get stuck	Partially Satisfied
Maintainability	Easy to add new nodes	Satisfied

Privacy	Access control rules ensure privacy	Satisfied
Immutability	Blockchain, by its design is immutable	Satisfied
Authorization	Some users might require permissions from greater number of users, which is not possible yet.	Partially satisfied

Table 5.4: List of Validated Non Functional Requirements

5.5 Conclusion

This chapter is all about listing requirements, designing a system which suits these requirements, implementing it and verifying to what extent requirements are met.

All these steps lead to answering the question: *"Is it possible to create an architectural design using the chosen framework to implement all these requirements*

By scrutinizing results, we reach to a conclusion that it is NOT possible to implement ALL requirements through this design. Nevertheless, the design achieves MAJORITY of the requirements. And since it satisfies most of the requirements and the design is possible to implement, it would be harsh to say the design is not feasible, just because it fails to achieve a few requirements. So, our final answer to the question remains in favor of design*: "Yes, it is possible to create an architectural design using the chosen framework to implement majority of these requirements"*

Chapter 6

Conclusion

This chapter reiterates all the statements we considered to draw our conclusions. The chapter also mentions the difficulties we face while formulating this thesis.

6.1 Overview

Generally, the whole thesis revolves around finding enough facts to validate the statement made in Chapter 4: “Blockchain will make a good architectural design for freight management system”. With regard to this, we formulated two questions in the problem statement:

1. What blockchain framework will be most suitable to develop an architecture to support these requirements?
2. Is it possible to build a feasible architectural design using such framework to implement these requirements?

Chapter 5 reveals answer for these two questions, which leads to reaching a final conclusion about whether or not Blockchain is favorable for freight management.

6.1.1 Thesis Statement

At this point of writing, we have almost sorted out everything to reach a final conclusion for our thesis statement. However, we need to expatiate a little bit over the statement “good architectural design for freight management”. It seems ambiguous, in a sense what makes an architectural design “good” for a specific our use case.

If we make a comparison between a new architectural design and other previous designs, then the new design will be considered good if it inherits all the functionalities of previous designs, along with its new efficiencies. But, we made it clear in the beginning, the approach of thesis is qualitative in nature, so it is more skewed towards validating framework according to the mentioned requirements rather than comparing it with different architectural designs (a quantitative approach addresses it).

Therefore, a design which can fulfill most of the requirements is considered as good architectural design.

6.1.2 Difficulties

Although, many recent projects used different blockchain architectural designs to show its potential in freight management, there's still less literature review available which can provide a quantitative base to evaluate efficiency metrics of our design by comparing it with others. Another difficulty was that many blockchain frameworks are not stable yet. Some of the most popular ones are getting extinct, as there's no further development. Whereas, few of them are under developed and few others are constantly evolving.

6.2 Future Work

This thesis contributes to the work at its basic level, as it tries to find out requirements for freight management system and their fulfilment through designed architecture. Therefore, this research can serve as a base for other greater contributions such as:

1. Taking other available blockchain frameworks and trying to fulfil the list of requirements to cross-reference which one serves requirements better.
2. Test different topologies for this proof of concept to find out which topology is the most optimal, means which one achieves the best trade-off between scalability and performance metrics.

Appendix A

Business Network Archive- Model Code Snippets

```
namespace org.freightmanagement

enum ProductKind {
  o FOOD
    o SHOES
    o MEDICINES
}

enum ItemStatus {
  o GOOD_CONDITION
  o DAMAGED
  o LOST
}
```

```
enum ShipmentStatus {
  o LABEL_PENDING
  //A label is pending, this means that the label has not been generated yet.
  o LABEL_GENERATED
  //A label has been generated, the next step is to schedule a drop-off or pickup
  o DROP_OFF_SCHEDULED
  //A Parcel is scheduled to be dropped off at the designated location
  o PICKUP_IN_PROGRESS
  // A Courier is in progress of picking up your parcel from the pickup location
  o PENDING_TRACKING_EVENT
  //This means that the parcel is in transit and waiting to be scanned at the first tracking point.
  o TRACKING_INFO_RECEIVED
  //This status is that we have received information that the parcel is in transit to the next destination
  o IN_TRANSIT
  //Your parcel is in transit and we are waiting for it to arrive at its destination
  o OUT_OF_DELIVARY
  //The parcel has arrived at destination and is out for delivery
  o FAILED_TO_DELIVER
  //The courier was unable to deliver the parcel

}
```

```
transaction CreateShipment{
  /*
  * Shipment parameters
  */
  o String shipmentId
  o String trackingNumber
   o Location location
  o String message
  o ShipmentStatus status default = 'LABEL_PENDING'

  --> freightMember owner
  //--> FreightManagmentnMember holder, the one who initially owns shipment
  --> Commodity[] assetExchanged

}
```

```
 */
transaction UpdateShipment{
  --> Shipment shipmentToupdate
  o ShipmentStatus status
  o Location location
  o String message
  --> freightMember newHolder optional

}
```

```
 transaction ReportDamagedFreight {
 --> Commodity damagedFreight
 o DateTime dateOccurred
 o DateTime dateReported
 o String occurrenceDescription
 o String itemConditionDescription
 o ItemStatus itemStatus

}
```

```
  transaction UpdateCommodity {
--> Commodity commodityToUpdate
o ProductKind type
o String name
o String description
o ItemCondition itemCondition
}
```

```
    transaction BatchTransfer {
 o Double batch_transfer
 -->batch_count fromaccnt
 -->batch_count toaccnt
}
```

```
asset Shipment identified by shipmentId {
  o String shipmentId
  o String trackingNumber
  o String message optional
  o ShipmentStatus status
  o Location location
  --> freightMember owner
  --> freightMember holder //currently in possession of shipment
    --> Commodity[] assetExchanged
}
```

```
asset Commercial_Invoice identified by InvoiceId {
o String InvoiceId
--> freightMember buyer
--> freightMember seller
--> Shipment shipmentid
o Location expectedArrivalLocation
o DateTime arrivalDateTime
o Double paymentPrice
o String PackageQuantity
o String TermsofPayment
}
```

```
  asset Commodity identified by GTIN {
  o ProductKind type
  o String GTIN
  o String name
  o String description
  o ItemCondition itemCondition
  --> freightMember owner
}
```

```
  asset batch_count identified by FreightMemberAccntId {
  o String FreightMemberAccntId
   o Double CommodityItemsinFreightMemberAccountId
   o DateTime timestamp
 }
```

```
  concept ItemCondition {
  o String conditionDescription
  o ItemStatus status
}

  concept Address {
  o String city optional
  o String country
  o String street optional
  o String zip optional
}

concept Location {
  o String globalLN //Global Location Number
  o Address address
}
```

```
  abstract participant freightMember identified by FreightMemberId {
  o String FreightMemberId
  o String email
  o Address address
  -->batch_count acnt
}
participant Shipper extends freightMember {
}
participant FreightForwarder extends freightMember {
}
participant InLandTransport extends freightMember {
}
participant Customs extends freightMember {
}
participant PortandTerminal extends freightMember {
}
```

Appendix B

Business Network Archive- Logic Code Snippets

```
async function freightmemberExists(freightMember){

    if(freightMember === undefined){
        return false;
    }

    var memberID = freightMember.getIdentifier();
    var memberType = freightMember.getType();

    if(memberID === undefined || memberID == '' || memberID === null ){
        return false;
    }else{
        return getParticipantRegistry('org.freightmanagement.' + memberType)
        .then(function (participantRegistry) {
        // Determine if the specific driver exists in the driver participant registry.
        return participantRegistry.exists(memberID);
        })
        .then(function (exists) {
        // Process the the boolean result.
        return exists;
        })
        .catch(function (error) {
        // Add optional error handling here.
        });
    }
```

```
/**
/**
 * Sample transaction function.
 * @param {org.freightmanagement.BatchTransfer} sampletransfer
 * @transaction
 */
function sampletransfer(sampletransfer){
 sampletransfer.fromaccnt.CommodityItemsinFreightMemberAccountId -=sampletransfer.batch_transfer;
 sampletransfer.toaccnt.CommodityItemsinFreightMemberAccountId +=sampletransfer.batch_transfer;
 return getAssetRegistry('org.freightmanagement.batch_count')
 .then(function(batch_accountRegistery){
 return batch_accountRegistery.updateAll([sampletransfer.fromaccnt,sampletransfer.toaccnt]);
 });


}
```

```
/**
 *
 * @param {org.freightmanagement.UpdateShipment} updatedItems - the UpdateShipment transaction
 * @transaction
 */
async function UpdateShipment(upgrade) {

  shipment = upgrade.shipmentToupdate
shipment.status = upgrade.status
  shipment.location = upgrade.location


    //UPDATE SHIPMENT
    const shipmentAssetRegistry = await getAssetRegistry('org.freightmanagement.Shipment');
    await shipmentAssetRegistry.update(shipment);
}
```

```
/**
 *
 * @param {org.freightmanagement.CreateShipment} createshipment - the CreateShipment transaction
 * @transaction
 */
async function CreateShipment(createshipment){
    //TODO: CHECK PERMISSIONS

    return freightmemberExists(createshipment.owner).then(function(exists){
        console.log("Exists? " + exists);
        if (!exists)
            throw 'Shipment owner does not exist.'
        else
            return freightmemberExists(createshipment.buyer);
    }).then(function(exists){
        console.log("Exists? " + exists);
        if (!exists)
            throw 'Shipment buyer does not exist.'
    }).then(function(exists){
        // Check if shipment owner is also the owner of all the assets
        var assetExchanged = createshipment.assetExchanged;
        for (var i = 0; i < assetExchanged.length; i++) {
            if (assetExchanged[i].owner.FreightMemberId!= createshipment.owner.FreightMemberId) {
                throw 'The shipment owner is not the owner of all the commodities in the shipment (check if all the commodities exist).';
            }
        }
```

```
      //Checking that the actual arrival date is AFTER the current date
        var now = new Date();
        if (createshipment.arrivalDateTime <= now) {
            throw 'Arrival Date is set to before the current date.';
        }
        console.log("before creating");
        // after all the checks
        CreateShipmentAuxiliar(createshipment);

        return getAssetRegistry('org.freightmanagement.Shipment')

    }).then(function(shipmentAssetRegistry){
        //console.log("Finally... " + shipmentAssetRegistry);
        return getAssetRegistry('org.freightmanagment.Commercial_Invoice');
    });

}
```

Appendix C

Business Network Archive- Access Control Code Snippets

```
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

```
rule AuditorAllowReadAll {
  description: "Auditor can read everything"
  participant: "org.freightmanagement.Auditor"
  operation: READ
  resource: "org.freightmanagement.**"
  action: ALLOW
}

rule AuditorDenyActions {
  description: "Auditor can not submit any transaction."
  participant: "org.freightmanagement.Auditor"
  operation: CREATE, DELETE, UPDATE
  resource: "org.hyperledger.composer.system.**"
  action: DENY
}

rule ParticipantAllowReadSystem {
  description: "Participants can access the system"
  participant: "org.hyperledger.composer.system.Participant"
  operation: READ
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}
```

```
rule ParticipantReadSelf {
  description: "Participants can their own info"
  participant(p): "org.hyperledger.composer.system.Participant"
  operation: READ
  resource(r): "org.hyperledger.composer.system.Participant"
  condition: (r.getIdentifier() == p.getIdentifier()  && r.getFullyQualifiedType() == p.getFullyQualifiedType())
  action: ALLOW
}

rule MembersReadOwnShipment {
  description: "Any supply chain member can read the shipment batches they own, hold or are buying."
  participant(p): "org.hyperledger.composer.system.Participant"
  operation: READ
  resource(r): "org.freightmanagement.Shipment"
  condition: (r.owner.getIdentifier() == p.getIdentifier() || r.holder.getIdentifier() == p.getIdentifier() || r.contract.buyer.getIdentifier() ==
p.getIdentifier())
  action: ALLOW
}
```

Appendix D Business Network Archive- Query Code Snippets

```
/**
 * New query file
 */

query getCommodityItemsByFreightMemberAccountID
{
  description: "Select all Batch count based on Freight Member Account ID"
  statement:
  SELECT org.freightmanagement.batch_count
  WHERE (FreightMemberAccntId == _$FreightMemberAccntId)
}

query getBatchTransferbyTimestamp
{
  description: "Select all Batch count based on timestamps"
  statement:
  SELECT org.freightmanagement.batch_count
  WHERE (timestamp == _$timestamp)
}
```

```
query getShipmentByID {
  description: "Select all shipments based on their ID"
  statement:
      SELECT org.freightmanagement.Shipment
          WHERE (shipmentId == _$shipmentId)
}
```

```
query getShipmentByOwner {
  description: "Select all shipments based on their Owner"
  statement:
      SELECT org.freightmanagement.Shipment
          WHERE (owner == _$owner)
}
```

```
query getShipmentByCountry {
  description: "Select all shipments based on their country"
  statement:
      SELECT org.freightmanagement.Shipment
          WHERE (location.address.country == _$country)
}
```

```
query getShipmentByHolder {
  description: "Select all shipments based on their Holder"
  statement:
      SELECT org.freightmanagement.Shipment
          WHERE (holder == _$holder)
}
```

```
query getHistorianByType {
  description: "Get all Historian records by type"
  statement: SELECT org.hyperledger.composer.system.HistorianRecord
  WHERE (transactionType == _$type)
}
```

```
query getHistorianByPerson {
  description: "Get all historian records by participant"
  statement: SELECT org.hyperledger.composer.system.HistorianRecord
  WHERE (participantInvoking == _$participantInvoking)
}
```

```
query getShipmentByTrackingcode {
  description: "Select shipment based on tracking code"
  statement:
      SELECT org.freightmanagement.Shipment
          WHERE (trackingNumber == _$trackingNumber)
}
```

```
query getHistorianRecords {
  description: "All transactions recorded"
  statement: SELECT org.hyperledger.composer.system.HistorianRecord

}
```

References

[1] M. Oude Weernink, W. Van Den Engh, M. Francisconi, and F. Thorborg, "The Blockchain Potential for Port Logistics," *Erasmus Univ. Delft Univ. Technol.*, no. 2 January 2018, p. 16, 2017.

[2] J. Laurens and W. E. Forum, "The digitisation of trade's paper trail may be at hand," pp. 1–8, 2018.

[3] Y. Okazaki, "Unveiling the Potential of Blockchain for Customs," no. 45, pp. 1–24, 2018.

[4] S. Godbole, "How Blockchain can transform Global Trade Supply Chains."

[5] M. E. Civelek and A. Özalp, "Blockchain Technology and Final Challenge for Paperless Foreign Trade," vol. 15, no. 1, pp. 1–8, 2018.

[6] A. Tricoli, "Legal Considerations On Blockchain- Based Bill Of Lading," vol. 971, no. 0, pp. 1–8, 2019.

[7] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," vol. 4, no. 3, pp. 382–401, 1982.

[8] Yuhefizar, "Membangun Toko Online Itu Mudah," pp. 1–9, 2013.

[9] D. Guegan, "The Digital World : II – Alternatives to the Bitcoin Blockchain ? HAL Id : halshs-01832002 Centre d ' Economie de la Sorbonne Documents de Travail du," 2018.

[10] M. Vukoli, "The Quest for Scalable Blockchain Fabric : Proof-of-Work vs . BFT Replication."

[11] G. Foroglou and A. L. Tsilidou, "Further applications of the blockchain," *Conf. 12th Student Conf. Manag. Sci. Technol. Athens*, no. MAY, pp. 0–8, 2015.

[12] V. Buterin, "Ethereum White Paper Made Simple."

[13] SZABO, Nick. Formalizing and Securing Relationships on Public Networks. **First Monday**, [S.l.], sep. 1997. ISSN 13960466.

[14] V. Buterin and V. Griffith, “Casper the Friendly Finality Gadget,” pp. 1–15, 2017.

[15] P. Fairley, “Ethereum Plans to Cut Its Absurd Energy Consumption by 99 Percent - IEEE Spectrum,” pp. 1–6, 2019.

[16] L. Fekkes, “Comparing Bitcoin and Ethereum,” 2018.

[17] Y. Gao, H. N.-P. of the A.-P. Advanced, and undefined 2017, “A Proof of Stake Sharding Protocol for Scalable Blockchains,” *Journals.Sfu.Ca*, 2017.

[18] Hutchins, P. (2018). Council Post: Creating Scalability On Ethereum. URL: https://www.forbes.com/sites/forbestechcouncil/2018/10/02/creating-scalability-on-ethereum/

[19] Hees, H. Raiden Network. URL: https://raiden.network/101.html

[20] J. Poon and V. Buterin, “Plasma: Scalable autonomous smart contracts,” *White Pap.*, pp. 1–47, 2017.

[21] E. Androulaki *et al.*, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” 2018.

[22] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, “Performance Analysis of Hyperledger Fabric Platforms,” *Secur. Commun. Networks*, vol. 2018, pp. 1–14, 2018.

[23] The Hyperledger White Paper Working Group, “Hyperledger Overview,” 2018.

[24] Hyperledger Composer Development Team. Introduction @ hyperledger.github.io, 2018. URL https://hyperledger.github.io/composer/latest/introduction/introduction.html.

[25] C. Ma, X. Kong, Q. Lan, and Z. Zhou, “The privacy protection mechanism of Hyperledger Fabric and its application in supply chain finance,” *Cybersecurity*, vol. 2, no. 1, 2019.

[26] G.Emmanuelle, *Can Blockchain Revolutionize International Trade?* 2018.

[27] Y. Chang, E. Iakovou, and W. Shi4, “Blockchain in Global Supply Chains and Cross Border Trade: A Critical Synthesis of the State-of-the-Art, Challenges and Opportunities,” pp. 1–51, 2019.

[28] B. Rakic, Msc, T. Levak, Z. Drev, S. Savic, and A. Veljkovic, “First purpose built protocol for supply chains based on blockchain,” 2017.

[29] Alex Cavanagh, “Ambrosus will Reform Supply Chains with Blockchain,” pp. 1–5, 2018.

[30] B. K. Wiederhold, G. Riva, and G. Graffigna, “White paper,” *Annu. Rev. CyberTherapy Telemed.*, vol. 11, p. 7, 2013.

[31] R. Aitken, “IBM Forges Global Joint Venture With Maersk Applying Blockchain To ‘Digitize’ Global Trade,” *Forbes*, pp. 1–13, 2016. URL: https://www.forbes.com/sites/tomgroenfeldt/2017/03/05/ibm-and-maersk-apply-blockchain-to-container-shipping/#1965cbc33f05

[32] O. Andersen and L. Vogdrup-Schmidt, “Rivals reject blockchain solution from Maersk and IBM,” pp. 1–4, 2018. URL: https://shippingwatch.com/carriers/Container/article10602520.ece