

FRAMEWORK FOR DESIGN AND DEVELOPMENT OF BLOCKCHAIN APPLICATION USING SMART CONTRACTS

By

Sumati Kulkarni

May 2020

Director of Thesis: Dr. Nasseh Tabrizi, PhD

Major Department: Computer Science

There is a lot of excitement around Blockchain technology and its ability to disrupt many traditional industries and business practices. First invented as a part of Bitcoin's underlying infrastructure, Blockchain technology offers a platform for decentralized and transparent transaction management between untrusting parties. Many believe this aspect of blockchain can revolutionize traditional supply chain practices typically involving many untrusting entities from the time raw material extraction to the final consumption of a finished product by the end consumer. While there have been many claims regarding its obvious benefits in Supply chain management, there are only few technical applications developed so far that are useful in real world scenarios. In this thesis, we review different real-world implementations of block chain technology in the supply chain domain, especially those that leverage smart contracts. Smart contract is a computer protocol that facilitates, verifies, enforces performance of a contract digitally using Blockchain technology. Since smart contracts are trackable, irreversible and allow performance of credible transactions without third parties, it can be deployed effectively to replace existing supply chain mechanisms that require working with an intermediate entity such as a bank that often comes with a price tag for their services. In this thesis, we present a framework to enable

sale of goods between untrusting entities typically in different geographies leveraging smart contract technology that can effectively replace the "letter of credit" payment mechanism. An novel algorithm for dispute resolution is developed and a decentralized app (Dapp) is built and deployed on Ethereum block chain using smart contracts developed in Solidity. Last, we discuss the effectiveness of such a system, potential drawbacks or known security threats that may hinder the adoption of such an app in the real world.

FRAMEWORK FOR BLOCKCHAIN BASED DECENTRALIZED ECOMMERCE
APPLICATION USING SMART CONTRACTS

A Thesis

Presented to The Faculty of the Department of Computer Science
East Carolina University

In Partial Fulfillment of the Requirements for the Degree
Master of Science in Computer Science

by
Sumati Kulkarni
May 2020

© Sumati Kulkarni, 2020

FRAMEWORK FOR DESIGN AND DEVELOPMENT OF BLOCKCHAIN
APPLICATION USING SMART CONTRACTS

by
Sumati Kulkarni

APPROVED BY:

DIRECTOR OF THESIS

Nasseh Tabrizi, PhD

COMMITTEE MEMBER

Mark Hills, PhD

COMMITTEE MEMBER

Venkat Gudivada, PhD

CHAIR OF THE DEPARTMENT
OF COMPUTER SCIENCE

Venkat Gudivada, PhD

DEAN OF THE
GRADUATE SCHOOL

Paul J. Gemperline, PhD

TABLE OF CONTENTS

LIST OF FIGURES	vi
CHAPTER 1: INTRODUCTION	1
Motivation	1
Thesis Objectives	1
Thesis outline	2
CHAPTER 2: RELATED WORK.....	3
Blockchain Technology	3
History of Blockchain.....	3
Working of Blockchain.....	4
Types of Blockchain	5
Blockchain vs. traditional database application.....	6
Known issues with Blockchain technology	7
Existing Blockchain Platforms	8
Hyperledger.....	8
Ethereum.....	9
Stellar	10
Comparison of Blockchain Platforms.....	10
Smart contract technology	10
Working of Smart Contracts	11
Advantages and disadvantages	13
Known issues with Smart contracts	14
Potential countermeasures	15
Supply chain management	16
Supply chain management objectives.....	17
Flows in Supply Chain.....	17
Mapping Study of existing blockchain applications.....	18
Results of mapping study.....	21

CHAPTER 3: DEVELOPMENT OF BLOCKCHAIN APPLICATION	22
Developing Proof of concept	22
Letter of Credit.....	22
Proof of Concept: Decentralized Letter of Credit.....	23
Design & development of decentralized application using solidity smart contracts .	25
Requirements	25
Proposed framework	25
Metamask, Truffle Suite and Ganache.....	25
Ethereum environment setup	28
Solidity Smart contracts	29
Buyer and Seller registration	31
Implementation of sale transaction	32
ADMIN selection and dispute resolution algorithm.....	33
CHAPTER 4: EVALUATION	36
Implementation using AngularJS.....	36
Results	43
CHAPTER 5: CONCLUSIONS AND FUTURE WORK	44
Conclusion	44
Future Work	44
BIBLIOGRAPHY	46

List of Figures

Figure 1: Comparison of Permissionless and Permissioned Blockchain platforms

Figure 2: Decision flowchart for selecting Blockchain vs. traditional database technology

Figure 3: Working of Smart contracts

Figure 4: Stages in a typical Supply Chain

Figure 5: Flows in Supply Chain

Figure 6: Included research work for mapping study

Figure 7: Analysis of Blockchain research in Supply chain by year and publishers

Figure 8: Analysis of Blockchain research in Supply chain by Industry

Figure 9: Analysis of implementation maturity of Blockchain applications in Supply chain

Figure 10: Comparison of features in popular blockchain platforms

Figure 11: Flows in supply chain with Letter-of-credit

Figure 12: Potential outcomes with decentralized purchasing

Figure 13: Proof-of-concept of decentralized letter of credit application

Figure 14: Working with Metamask

Figure 15: Coding smart contract in Solidity

Figure 16: Constructors and functions in Solidity

Figure 17: User registration function in Solidity

Figure 18: Implementing sale transaction in Solidity

Figure 19: Dispute resolution in Solidity

Figure 20: Dispute resolution at optimal time using algorithm

Figure 21: Letter of Credit DApp Homescreen

Figure 22: Letter of Credit DApp – Selecting Seller account

Figure 23: Letter of Credit DApp – Seller Products Page

Figure 24: Letter of Credit DApp – Seller Add Product

Figure 25: Letter of Credit DApp – 0.1 ETH GAS FEE & 1.0 ETH DISPUTE CLEARING FEE from seller account when product is added to the list

Figure 26: Letter of Credit DApp – Logout from Seller account and login to Buyer account to start transaction

Figure 27: Letter of Credit DApp – Buyer Purchase the product

Figure 28: Letter of Credit DApp – 0.1 ETH GAS FEE, 100.0 ETH PRODUCT COST, 1.0 ETH DISPUTE CLEARING FEE from buyer

Figure 29: Letter of Credit DApp – Buyer bought the product

Figure 30: Letter of Credit DApp – Seller ships the Product

Figure 31: Letter of Credit DApp – Buyer does not confirm delivery

Figure 32: Letter of Credit DApp – Seller initiates dispute

Figure 33: Letter of Credit DApp – Disputed Items

Figure 34: Letter of Credit DApp – Admin1 votes seller

Figure 35: Letter of Credit DApp – Admin2 votes seller

Figure 36: Account balances in Ganache showing accurate working of algorithm

CHAPTER 1: INTRODUCTION

1.1 Motivation

Blockchain technology offers an innovative platform for decentralized and transparent transaction management. Blockchain was first invented as part of Bitcoin's underlying infrastructure in 2008 (Nakamoto, 2008). Blockchain uses a distributed, peer-to-peer network to make a continuous, growing list of ordered records called blocks to form a digital ledger. Each transaction, represented in a cryptographically signed block, is then automatically validated by the network itself. Despite initial doubts about this technology, recently governments and large corporations have investigated to adapt and improve this technology in various domains of applications, from finance, social and legal industries to design, manufacturing, and supply chain networks. At the same time, there is an on-going debate among researchers regarding the applicability of Blockchain technology to solve real-world problems.

To provide some background on supply chain management and why it is being considered as a potential application area for blockchain technology, we review some of its definitions and fundamental concepts. Supply chain management (SCM) is defined as the active management of supply chain activities to maximize customer value and achieve sustainable competitive advantage. In commerce, the management of the flow of goods and services, involves the movement and storage of raw materials, of work-in-process inventory, and finished goods from point of origin to point of consumption (Chopra, 2013). Organizations increasingly find that they must rely on effective supply chains, or networks, to compete in the global market and networked economy. In new management paradigms (Drucker, 1998), this concept of business relationships extends beyond traditional enterprise boundaries and seeks to organize entire business processes throughout a value chain of multiple companies. However, with complicated interactions among players, mistrust between players becomes a self-fulfilling prophecy leading to a lack of visibility/transparency and ultimately reducing the value delivered to customers. Improving trust and visibility across a Supply chain has proven to have a positive impact on all entities involved.

1.2 Thesis Objectives

Blockchain technology can address a critical aspect of the supply chain by enabling reliable, accurate information sharing across multiple parties without trust. This can also eliminate multiple "middlemen" currently used to conduct transactions between parties without trust. A typical example of this would be during the import of high-value items from a supplier located in a distant country. Due to a lack of trust between the two parties, it is common practice to take the assistance of a financial institution such as a bank to mediate the purchase of goods and transfer of funds. The bank for a small fee will ensure the transfer of funds is only conducted after ensuring the physical transfer of goods. This process can effectively be managed by a blockchain application that can ensure the release of funds only after obtaining proof of shipment from the supplier. There are many other such applications which we will further explore in this thesis and identify good use cases that can be developed into real-world blockchain applications. The objectives of this thesis are:

- Perform a mapping study of blockchain as applied to supply chain management
- Create a framework for a blockchain-based letter of credit that addresses challenges with using current techniques.
- Systematic review of technology to identify best set of tools
- Develop a novel algorithm to resolve disputes while executing smart contracts.
- Evaluate the effectiveness of this framework by creating a web application.

1.3 Thesis outline

This thesis is organized as follows: We start off understanding fundamental concepts of Blockchain technology, existing blockchain platforms and Smart contracts in chapter 2. We also introduce basic concepts of supply chain management to understand applications that would be most beneficial in the real world. We then conduct a literature survey to understand the current research landscape related to blockchain applications in the Supply chain. In chapter 3 we bring all our learning together and propose a proof of concept application. We provide details of development and implementation of decentralized applications using solidity smart contracts. We also develop an algorithm for selecting ADMINS and dispute resolution which is unique to the problem selected here and addresses several issues with existing Online dispute resolution (ODR) systems. Finally, we conclude with details of complete implementation, results, and future work in chapter 4.

CHAPTER 2: RELATED WORK

2.1 Blockchain Technology

Blockchain technology is an immutable ledger of transactions over a peer-to-peer network (Nakamoto, 2008). It can be a decentralized distributed database that stores a time-stamped immutable public ledger of all transactions. Through a consensus protocol, the peers can view and validate transactions and are grouped into blocks that are linked using cryptography. Every new entry is stored in a new block and tampering with any block impacts the whole chain. This is by design and ensures chronological sequence and integrity of data. Validating new blocks is done through a set of protocols and consensus obtained from every participant of the network before the block is stored permanently on the chain. Pointers and linked list data structures are used to create this chain with each block pointing to the previous block. Each block, in turn, contains multiple transaction data along with timestamps and links to the previous block which is generated by a secure hash algorithm (Krishnan, 2020).

In traditional client/server architecture, user access is typically controlled by administrators through roles and responsibilities. In contrast, a public blockchain is a decentralized peer to peer network where all participants have equal access to and can control the network (Herlihy, 2017). While other types of blockchains exist that can be more restrictive, the main idea behind blockchain technology is to carry out transactions in a secure auditable manner even in presence of unknown untrusted parties without the need for an intermediary. As blockchain is a nascent technology, it is evolving continuously, and many different variations of this technology can be found which we will discuss later.

2.1.1 History of Blockchain

Early work on blockchain was pioneered by Stuart Haber and W. Scott Stornetta developed a framework for a cryptographically secured chain of blocks in 1991 (Haber, 1991). Later they went on to incorporate Merkle trees into their blockchain design to improve performance and scalability by grouping several transactions on a single block (Bayer, 1993).

After many more attempts of creating a viable cryptocurrency and several years later, Blockchain was invented by one or more people using the pseudonym Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin (Nakamoto, 2008). Bitcoin was the first invention built on blockchain technology and was the first digital cryptocurrency to solve the double-spending problem without the need for a trusted intermediary or central server. The invention of the blockchain for bitcoin paved way for the development of several other cryptocurrencies and decentralized applications leveraging blockchains that are widely available for public use and review. One such application worth mentioning would be Ethereum including their smart contract feature (Buterin, 2013). Smart Contracts are a set of executable code that can directly run on top of the blockchain systems. Agreement between untrusted parties without the requirement of a third party is enforced by this technology. Smart contracts can be used effectively

in a variety of applications and is not limited to financial transactions like Bitcoin (Bartoletti, 2017).

2.1.2 Working of Blockchain

Blockchain, as the name suggests, involves chaining a series of blocks consisting of data transaction data along with some metadata to make the chain work. Each block has a pointer to the immediately previous block which is essentially a value of the previous block. When users initiate a transaction in the network, a block is created and broadcasted to all participants of the network. In a blockchain, blocks are connected by referencing the hash value of the previous block. Since every node of the network has a complete chain, the index of any new block being inserted must be greater than the latest block. A hash value is computed using a combination of the previous block hash and the new block's transaction data. Participants then analyze and validate this block using a consensus algorithm such as proof-of-work or proof-of-stake. Once a consensus is formed in the network validating the block, it is added to the chain and becomes a permanent immutable record accessible by all participants.

Since transactions are permanent and immutable on blockchains, it becomes clear that there needs to be ample memory and computing power to be sustainable. This is where blockchain leverages Merkle Tree, which is a way of structuring data so that large volumes can be verified quickly and accurately. Furthermore, Merkle Trees help validate that later versions of a block include everything from an earlier version. They ensure that data is recorded in chronological order and can help verify that no prior records have been altered or tampered with. Merkle Tree works by repeatedly using hashing functions such as MD5, SHA-3 and SHA-256 that take inputs and generate unique output. First the entire volume of data is split into pairs. Every pair is repeatedly hashed and stored until only one remains which is called the Merkle root that is used to verify blocks on a blockchain (Asharaf, 2017).

Also, every time a new node is added to the block chain, a synchronization is needed to update the blockchain to include the new node. This is done by exchanging messages with peers in the network and updating local copies.

Lastly, there are different ways to validate and add new blocks to the blockchain. One of the very first algorithms was Proof of Work which was used in Bitcoin cryptocurrency. This widely used algorithm adds a block after miners can solve a difficult puzzle such as finding a hash that begins with a certain number of zeroes. First miner to successfully find such a hash will be rewarded by adding his block to the network and the miner collects the transaction fee. Other consensus mechanisms are also being used such as Proof of Stake, Proof of activity, Proof of Burn time and so on (Nguyen, 2018).

2.1.3 Types of Blockchain

While Blockchain maintains its integrity across a distributed network through a consensus mechanism across all the participants of the network, who can participate in the network creates different dynamics and widely different Blockchains.

Blockchains such as Bitcoin which is open to the public and can be accessed by anyone and does not prohibit anyone from participating are as Public blockchains. In such blockchains anyone can access the current state of the blockchain and add new blocks. There are also private blockchains typically developed by large companies, access to which is granted through a centralized entity. While this essentially strips the decentralized, transparent nature of a blockchain it still provides immense value to business as the transaction records on a blockchain act as a single source of truth across the organizations and entities using such a blockchain. Therefore, such blockchains systems provide more privacy for transactions being recorded and in turn loses some transparency that is seen in a public block chain.

Blockchains can be broadly classified into the following three categories (Buterin V. a., 2014):

Public Permission less Blockchain:

Public blockchains are designed to be fully decentralized, with no single entity controlling how transactions are recorded in the blockchain or how they are processed. They offer full open access to everyone and all transactions on such a blockchain can be verified by any participant. Public blockchains are highly transparent and resistant to tampering.

Private Permissioned Blockchain:

Private blockchains are designed and developed primarily for enterprises who want to collaborate and share data but do not trust each other completely enough to share more sensitive data. These chains are more centralized and need not be transparent. The data collected on such blockchains can also be tampered with using the power of central authority.

Properties	Permissionless Blockchains	Permissioned Blockchains
Public Verifiability	✓	!
Transparency	✓	!
Privacy	!	✓
Integrity	✓	!
Redundancy	✓	✓

Figure 1: Comparison of Permissionless and Permissioned Blockchain platforms

Consortium or Hybrid Blockchain:

It is worth noting here that there is essentially a tradeoff between transparency and privacy that results in the choice of public vs. private blockchain. Consortium or hybrid model of Blockchain covers the remaining spectrum between Transparent to Private blockchain. Such blockchains are controlled by a group of entities rather than a single entity. This is very similar to a private blockchain but can be more decentralized based on the mix of entities controlling the blockchain and how they are selected. This model is considered ideal for organizations with competition.

2.1.4 Blockchain vs. traditional database application

As we move from public Blockchain to private blockchain the line between traditional database architecture and blockchain grows increasingly thin. Considering that we achieve immutability and other desirable properties of blockchain even in a traditional database setting provided the administrator is trusted. Furthermore, traditional databases are much faster compared to blockchain transactions which require the consensus of a large group of people on the network. In the paper 'Do you need a blockchain' the author has conducted detailed study and provides a useful flowchart to guide users on when it makes sense to build a blockchain application over a traditional database (Wust, 2018).

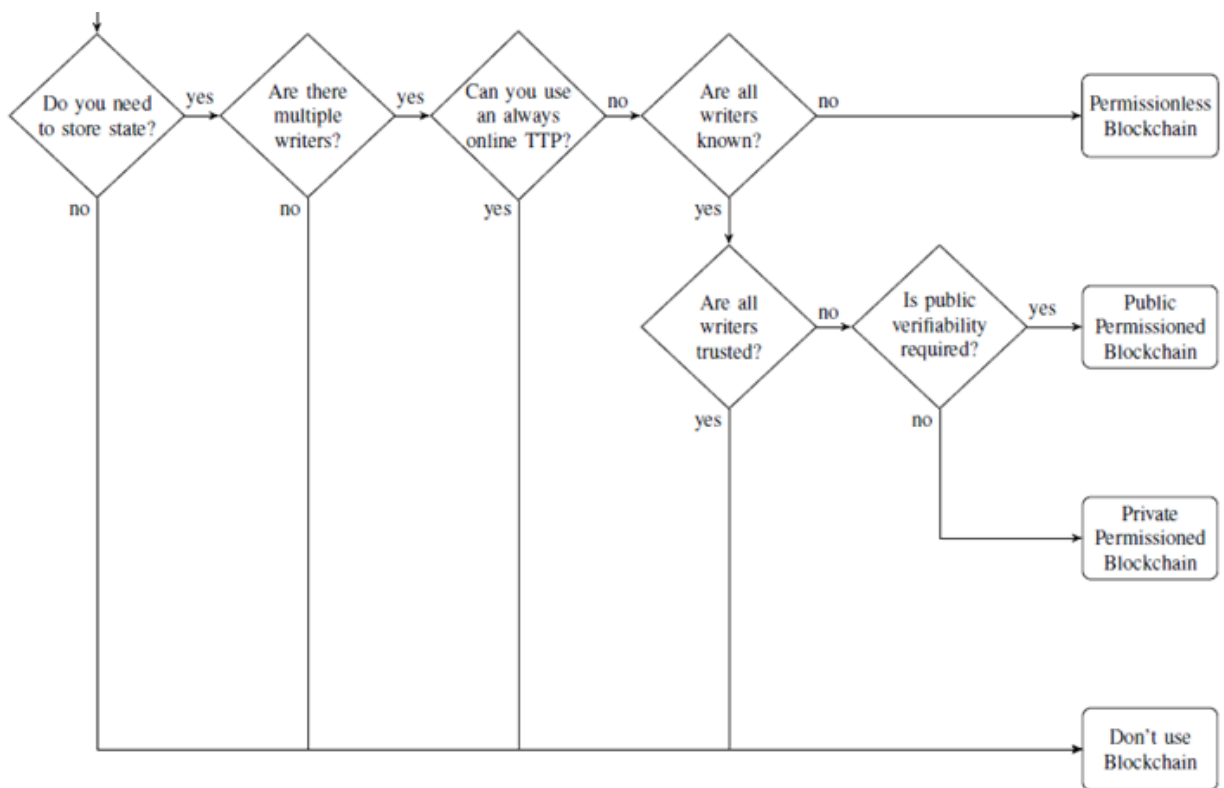


Figure 2: Decision flowchart for selecting Blockchain vs. traditional database technology

As detailed in the flow chart, a permission less public blockchain makes sense when all participants of the network are not known as is the case with well-known cryptocurrencies. When all the

participants are known but not trusted a Permissioned blockchain is recommended. Further, if public verifiability is required for certain organizations then a public permissioned blockchain is a viable option. In all other cases where all participants are known and trusted a simple database can easily outperform blockchain.

Hence, we can conclude that choosing blockchain over traditional database for development of any application only makes sense when the following hold true:

- multiple entities want to interact and change the state of a system
- mutually mistrusting
- not willing to agree on an online trusted third party

2.1.5 Known issues with Blockchain technology

Block chain technology has come a long way since its birth a decade ago. There have been many issues and major failures, but the technology has evolved to come out stronger after each major setback. Some of the major security issues in the past are as follows:

1. Fork Problems

Whenever a block chain software is upgraded, it affects all the blocks in the chain including old ones that were created based on a different set of rules. If there is a new consensus algorithm being implemented, there is always a chance that old nodes created until this version behave differently and can lead to potential security issues. Furthermore, there are two kinds of forking that could occur. A fork is termed ‘hard fork’ when the new version is not compatible with old nodes. In such cases, all the old nodes must be upgraded to the new version and there will essentially be a new chain for the new version and the old version can continue on the old chain with the old version of software if desired. A ‘Soft fork’ occurs when the new software is not incompatible with the old nodes, but blocks mined by old nodes will never be approved. As such, both old and new nodes can continue to work on the same chain while old nodes are upgraded gradually. However, this does create some disparity between different nodes and can result in a potential security flaw that could be exploited.

2. 51% Majority Attack

Since in a Proof of Work based consensus mechanism, the probability of mining a block depends on the work done by the miner and incentivizes miners to band together to form “mining pools”. If such Mining pools become large enough to hold 51% computing power in a blockchain network, it can then take control of the entire blockchain. This is a serious security issue because if someone manages to accumulate 51% computing power then they can almost always find Nonce value quicker than others and in turn become the sole authority to decide which block is added to the chain. This could result in a range of problems and breaks the very foundations of a blockchain that is fair, immutable, and permanent.

3. Other Attacks

The DAO attack was a contract implementing a crowd-funding platform which raised \$150M before being attacked. The attacker managed to steal approximately \$60M after which a hard fork had to be made to nullify the effects of transactions involved in the attack. There have also been many cases where bugs in the codes become public knowledge and are exploited. For example, incorrect scoping of the constructor function in Rubixi let anyone invoke the function and withdraw funds.

4. Scale of Blockchain

Since every node has a copy of the entire blockchain, as the blockchain grows, it will take an enormous amount of resources to sustain the blockchain and keep it performing at an acceptable level. There have been workarounds to this problem where in all data not necessarily required to be stored in the blockchain is handled off-chain. There are also algorithms or technologies that make verification faster and less resource intensive such as 'Simplified Payment Verification'. Furthermore, the continuous improvement in storage and computational technology can help us sustain a growing blockchain if the amount of data stored and computational effort required to complete transactions are well thought through.

5. Transaction Time

Blockchain transactions are known to be much slower than transactions on any application working on a traditional database due to the decentralized consensus mechanism for validating and confirming any transaction. Some consensus mechanisms like proof-of-stake are better than others such as proof-of-work. Other potential solutions include Lightning Network where in implementation of Hashed Time lock Contracts (HTLCs) with bi-directional payment channels allows payments to be routed across multiple peer-to-peer payment channels.

2.2 Existing Blockchain Platforms

To choose the right blockchain to develop POC of the decentralized application we review some of the major options available as listed below:

2.2.1 Hyperledger

Hyperledger project was published by Linux Foundation in 2015 with the goal of providing a blockchain-based open source technology through which companies will be enabled to build robust and industry-specific systems for secure transaction processing (Burgwinkel, 2016). Many different companies collaborate and contribute to the Hyperledger project and build frameworks. The new version of the Hyperledger blockchain is called Fabric. Hyperledger is a permissioned blockchain and does not provide a cryptocurrency. However, since the consensus mechanism is based on a plug-in, it is possible to run the system with a cryptocurrency. Lastly, there is a cryptographic plug-in, as Hyperledger does not define a specific cryptographic algorithm (like for

example the secure hashing algorithm SHA 256 for Bitcoin) and therefore depending on the plug-in, different algorithms are used. Through the open plug-in architecture, the system can be adapted to changes in the future (Burgwinkel, Blockchain technology: Einfuhrung fur business-und IT manager, 2016). Fabric is primarily designed for integration projects where a Distributed Ledger Technology is used, offering no user facing services other than an SDK for Node.js, Java and Go. Fabric supports chaincode in Go, JavaScript and other languages such as Java that can be accessed by installing corresponding modules. As such it is more flexible than other blockchain platforms that only support a closed Smart Contract language.

Hyperledger has a consensus mechanism based on current implementation of practical byzantine fault tolerance (PBFT) (Castro, 1999). It includes trust anchors to root certificate authorities as an enhancement to the asymmetric cryptography and digital signature features with SHA3 and ECDSA (Cachin, 2016). The permissioned nature of Hyperledger enhances security of the network by preventing attacks involving unauthorized generation of malicious peers that can potentially take over the network. Also, smart contract implementation in Hyperledger is based on chaincode, which can self-execute conditions or resource transfers among peers in fraction of a second (Samaniego, 2016), (Smith, 2016). Thus, by applying smart contracts based on chaincode and a unique PBFT implementation which offsets computational overhead for increased networking among peers, Hyperledger offers a well-rounded platform for applications for Internet-of-Things.

2.2.2 Ethereum

Ethereum is an open source public blockchain-based distributed computing platform and operating system on which smart contracts can be deployed (Buterin V. , 2013). Ethereum was first developed by Vitalik Buterin in 2013 and launched in 2015 after an online crowd sale that took place in 2014 (Buterin V. a., 2014). Ethereum currently works on the proof-of-work mechanism like bitcoin and Ether is the cryptocurrency generated. Ethereum began as an alternative cryptocurrency solution to compete Bitcoin but further on things have changed. It has some special characteristics, as it is an adaptable blockchain implementation with an implementation of smart contracts and a derivative of proof-of-work consensus known as Ethash. This also applies to directed acyclic graphs to manage probabilistic hash generation in matters that will prevent potential abuse from specialized hardware where other proof-of-work algorithms are vulnerable to (Natoli, 2016).

Ethereum contracts are executed on the Ethereum Virtual Machine (EVM), a decentralized virtual machine. EVM can execute contracts using an external network of public nodes. The virtual machine's instruction set is Turing-complete which makes it much more usable in the real world without elaborate restrictions. A small fee called "Gas" is charged per transaction to mitigate spam and allocate resources on the network (Wood, 2014).

In addition to implementing smart contracts, Ethereum transactions can also store custom data. This allows use of Ethereum for several applications beyond cryptocurrency transactions. Due to Ethash being based upon proof-of-work, Ethereum is very fast compared to Bitcoin's Proof-of-Work and may require between 10 to 20 seconds to produce a block. Still high frequency and time

sensitive IoT device operations may not support such delays. While Ethash prevents abuses from potential specialized hardware, it does not necessarily enhance fault tolerance. At scale, IoT devices would need to rely on trusted and computationally powerful peers to ensure fault handling. Storage also presents another problem, as Ethereum requires all peers to store a blockchain that is tens of gigabytes larger. IoT devices that normally do not have such storage capacity, will either need to intercommunicate with a proxy server that will act as a peer in the Ethereum network or accommodate large storage. Ethereum, as it is used longer than most distributed ledger implementations, has IoT prototypes, such as handling tokens and contracts for electronic lock sharing and supply chain assurance prototypes (Christidis, 2016).

To keep the increasing number of users and applications sustainable on Ethereum and to improve transaction speed Ethereum 2.0 is being developed. It aims to introduce a proof-of-stake consensus mechanism, which will eliminate the need for expensive proof-of-work mining. Also, Ethereum 2.0 plans to introduce sharding, which will improve the speed and throughput of ETH transactions (Crypto, 2020).

2.2.3 Stellar

Stellar features a public blockchain with its own consensus algorithm which is like Practical Byzantine Fault Tolerance (PBFT) (Castro, 1999) but uses elements from Social network modeling. The difference is that a node agrees on a transaction if the nodes in its neighborhood agree. Nodes in the neighborhood are more trustworthy than the others. When the transaction has been accepted by a threshold number of nodes in the network, a cascading effect ensues due to homophily and the transaction will be confirmed by the entire network with a high degree of certainty. As such, this protocol requires much less computing power, as it does not require solving of cryptographic puzzles. Unlike Ethereum, there is no specific language for smart contracts; it is still possible to assemble some transactions and write them atomically within the block chain. Stellar also features special accounts called multi-signature which essentially lets several owners handle a single account. To perform operations from these accounts, a minimum level of consensus must be reached among the owners. Transaction chaining and multi-signature accounts can be combined to make more complex contracts (Mazieres, 2015).

2.2.4 Comparison of Blockchain Platforms

After considering and reviewing some of the popular blockchain platforms available we can list (Wu, 2019) and compare their main difference in Figure 10.

2.3 Smart Contract Technology

As conceptualized by Nick Szabo, a smart contract is a computerized transaction protocol that executes the terms of a contract (Szabo, 1994). The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and

Features/Characteristics	Hyperledger Fabric	Ethereum	Stellar
Blockchain Type	Public	Public	Public
Privacy	Private	Public/Private	Public
Permission Restrictions	Permissioned	Permissionless	Permissionless
Average Transaction fee	N/A	\$2.50	0.00001 XLM
Node Scalability	Low	High	N/A
Performance Scalability	High	Low	N/A
Consensus Mechanism	PBFT	PoW	Stellar
Turing Complete	Yes	Yes	No
Contract Language	Go	Solidity	JavaScript, Go
Encryption of Transaction data	Yes	No	N/A

Figure 10: Comparison of features in popular blockchain platforms

enforcement costs, and other transaction costs (Szabo, Formalizing and securing relationships on public networks, 1997). While the idea was conceptualized in 1994, it was not until the invention of blockchain that such contracts could execute in a truly decentralized and autonomous manner. Unlike legal contracts which are meant to be referred to and act as a guideline for execution, Smart contracts perform the execution of contract terms themselves while ensuring it meets all the terms agreed upon to begin with (Tjong Tjin Tai, 2017). Vending machines can be used as an analogy to understand Smart Contracts. They are like smart contracts in that the machine operates based on pre-written software code. When the required amount of coins is deposited in the machine and a selection of items made, the vending machine dispenses this item along with any change that needs to be returned. Smart contracts work in a similar fashion and perform tasks of contract only after ensuring conditions of contract are met. This ensures trust and reliability as once the contract terms are agreed upon and coded, neither party will have the ability to change the way it functions. Furthermore, smart contracts on a blockchain provide even more security and trust as terms of the contract code are always available to the public for scrutiny and all transactions are recorded permanently and cannot be changed (Omohundro, 2014).

2.3.1 Working of Smart Contract

Smart contracts are executable programs (Buterin V. a., 2014). They are usually written in high-level computer programming languages in order to represent business logic or predefined criteria to trigger transfer of values. For a smart contracts engine to be effective in supporting a wide range of use cases, the language needs to be Turing complete, that it can solve any computation problem. Therefore, even though Bitcoin has its own scripting language, it is not considered to have smart contracts. On the other hand, Ethereum smart contracts are Turing complete and have been used to solve some of the most challenging problems in real-world (Le, 2018).

When a user submits transactions, smart contracts gets executed by the blockchain nodes to process transactions. A blockchain transaction has a designated target smart contract function, a payload

that contains input values to the function call and is always signed by the submitter. A transaction can be submitted to any node in the blockchain network, which broadcasts it to the entire network so all the nodes will see the transaction. At some point, the transaction gets processed by each individual node using the executable program in the target smart contract. If the transaction execution is successful, the internal state of the blockchain will be updated. A smart contract may also consider the input to be invalid and reject the transaction as failed, in which case the state is not affected.

Smart contracts must be executed by a set of blockchain nodes independently. Unlike traditional databases, blockchains are decentralized. As such, every node assumes others are potentially malicious and never trusts states maintained by other nodes within the network. Instead each node executes the transactions themselves using the smart contract code and maintains its own state. Since all nodes have the identical beginning state, same input values and therefore the same execution logic. If all three parts are identical, the top state is sure to be identical. The chain of blocks with the linked hashes each representing the total list of transactions input and therefore the starting state, play a critical role in forming consensus among the blockchain nodes. To ensure the correct smart contract code is executed to process the transaction, Ethereum smart contract code stores a copy of itself on the blockchain directly as state. In Hyperledger Fabric and Corda, contract code is stored off-chain, and an on-chain hash is used to identify the correct version of the contract. The main purpose of smart contracts is to take care of program states. State is an arbitrary piece of knowledge that gets updated by executing a transaction. So, a blockchain can be conceptualized as a database, although it is designed for data consistency and immutability and not for speed of performance of queries. Most of the blockchain protocols are designed to follow a state transfer conceptual model where each smart contract maintains its own set of states. Most transactions submitted to a blockchain involve a contract, except for pure value transfers that do not involve smart contracts. Whenever a transaction is executed, the state of the target smart contract is updated. Good contracts can call another smart contract and question the downstream contract's state or update it. Smart contracts may be thought of as program functions: there are inputs, logic to process the inputs, and output. Execution of smart contracts often ends up in updated states (Zhang, 2019).

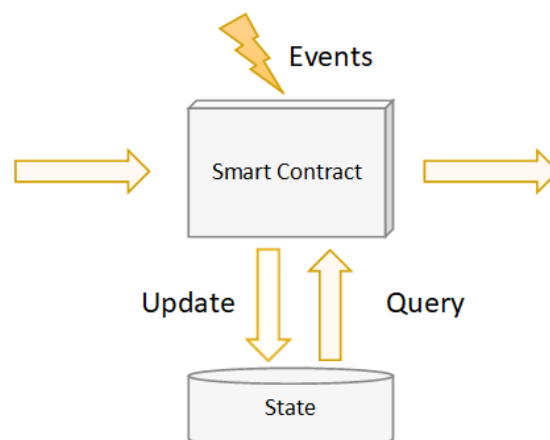


Figure 3: Working of Smart contracts

When smart contracts are executed only valid transactions will result in updated states. Invalid transactions resulting from exceptions thrown by the smart contract are rejected by the network or included as failed depending on the blockchain platform.

Smart contracts can also publish events that send notifications to the outside world when the block containing the transaction gets committed to the blockchain on the node.

A smart contract may have multiple public functions that can be called by any transaction. These functions could either result in a state change or simply return the latest state after performing internal calculations. Some of common functions that update states are:

`transfer(to, amount)`

`approve(delegate, amount)`

`transferFrom(from, to, amount)`

`mint(to, amount)`

`burn(from, amount)`

Functions that only query the latest states for information and do not result in a state change:

`balanceOf(account)`

If a transaction calls a function that requires a state change, then it must be handled by a consensus mechanism, so that the system ensures all the copies maintained by the blockchain network's participating nodes have identical records.

On the other hand, querying the latest state and retrieving information without updating state can be done with the help of just one node. Since the consensus mechanism ensures all nodes have the same information it does not matter which single node we query. Hence, we can conclude that write operations on a blockchain are far more expensive than read only operations.

2.3.2 Advantages and disadvantages

Smart contracts offer several advantages over traditional contracts, some of which are listed below:

1. **Lower cost:** Even though each transaction requires a small fee, applications using smart contracts require much less manual handling or verification and as such will reduce overall costs/financial charges by a significant amount.
2. **High Accuracy:** Since these transactions are processed automatically without manual intervention at any point, there are fewer errors resulting from human error.
3. **Increased Speed:** as smart contracts are essentially software codes that automate complex tasks involving decision making, they can increase the speed of transactions as well as the entire business process.
4. **Lower Risk:** Since smart contracts are stored and executed on the block chain, an immutable and permanent record of transactions are stored, it would be virtually impossible

to manipulate or cheat the system which reduces the overall risk associated with doing business.

5. **No Middlemen:** Since smart contracts can function autonomously in a reliable manner, it is often used to remove third party intermediaries whose sole purpose is to be a bridge between two untrusting parties.

While Smart contracts provide many advantages, they are not without their fair share of disadvantages as well (O'hara, 2017), (Mulligan, 2018) , (Raskin, 2016), (Giancaspro, 2017):

1. **Privacy:** Since smart contracts are executed on a blockchain, every transaction that needs to be added to the blockchain needs to be broadcasted to the network of nodes to reach consensus. So, any node participating in the blockchain can essentially deduce all the information regarding any transaction. As such privacy is never guaranteed even when some amount of information can be obfuscated.
2. **Limited Scope:** While many contracts existing today, particularly those relating to business transactions are well suited to be converted to a smart contract, there are many others that may include ethical and social issues that may be hard to do so.
3. **Performance issue:** Most blockchains have a high latency or low transaction speed depending on the consensus mechanism used. This makes it inapplicable to many applications that require instantaneous confirmation of transactions.
4. **Governance:** If blockchains are to be sustainable in the long run, serious consideration should be given to ethics and framework for governance models. Nascency of the technology coupled with pseudonymity of account holders and complexity of underlying concepts make it prone to deception and fraud.

2.3.3 Known issues with Smart contracts

There are many known vulnerabilities in Smart contracts that can be exploited to perform certain attacks that are provided below.

1. **Out-of-gas:** When a function is trying to transfer ether to another account, it is possible to encounter an out-of-gas exception if the sender does not have sufficient gas to cover the transaction. This may result in contract execution if not handled appropriately.
2. **Invalid transfer:** When sending ether, recipient address must be specified accurately. If some ether is sent to an incorrect address, it could be lost forever. Even if the incorrect address is valid and someone else did receive the ether, it may be hard to get it back. As such it is important that the correct recipient addresses are used especially when retrieving these from an array or other complex data types.
3. **Exception handling:** Solidity raises an exception when one the execution runs out of gas; the call stack reaches its limit or when the command throw is executed. However, the way Solidity handles different exceptions is not uniform and developers should be careful on how these will be handled.
4. **Reentrancy:** Unlike some other programming languages, it is not guaranteed that when a non-recursive function is invoked, it cannot be reentered before its termination. Due to the fallback mechanism an attacker may be able to re-enter the caller function. This could cause

unexpected behaviors and loops which might end up draining all the gas before coming to a stop throwing an out-of-gas exception.

5. Private fields: Again, unlike some common programming languages, privacy of private fields is not guaranteed. Since every transaction is sent to miners and broadcasted on the blockchain, elements of the transactions are available for anyone to inspect.
6. Call stack depth limit: Whenever a contract invokes another contract, the call stack associated with the transaction increases by one frame. Since the call stack is limited to 1024 frames, an exception is thrown when an invocation is made beyond this limit. As such it is highly recommended to avoid using recursive functions.
7. State: The state of a contract is determined by the value of its fields and balance. In general, when a user sends a transaction to the network in order to invoke some contract, he cannot be sure that the transaction will run at the same state as the contract was at the time of sending that transaction. This may happen because, in the meanwhile, other transactions have changed the contract state. Even if the user was fast enough to be the first one to send a transaction, it is not guaranteed that such a transaction will be the first to run. Indeed, when miners group transactions into blocks, they are not required to preserve any order; they could also choose not to include some transactions.
8. Timestamp dependency: Timestamps should be avoided in critical parts of the code as the miners can manipulate the timestamps.

Solidity also provides a list of known bugs with their corresponding severity level (List of Known Bugs, 2016-2020).

2.3.4 Potential countermeasures

It is suggested that known vulnerabilities in smart contracts can be prevented and risk mitigated by using tools (Mense, 2018) (Dika, 2017) such as listed below:

1. ZeppelinOS: is an operating system for smart contract applications developed by Zeppelin Solutions that enables development of smart contracts by using already developed and secure smart contracts.
2. HackThisContract: is a crowdsourcing experimental website where smart contracts uploaded will be attacked and tried to be exploited for potential vulnerabilities by other developers. This helps eliminate a lot of common issues and makes the smart contract more secure before deployment.
3. Hard Fork: It is always recommended to upgrade the Ethereum platform adding functionalities that can improve operational semantics and face security issues such as: guarded transactions to deal with transaction ordering dependence (TOD), deterministic timestamp and exception handling.
4. Oyente: This tool extracts the control flow graph from EVM bytecode of a smart contract and executes it symbolically to detect vulnerability patterns. This tool identifies vulnerabilities arising due to non-handling of possible exceptions such as not checking the return code of call or issues with reentrancy.

5. Remix: is a web-based IDE that allows users to write, deploy and run Solidity smart contracts. Remix includes an integrated debugger and a test-blockchain network. It can be used to analyze the Solidity code and reduce coding mistakes by performing a security analysis using deductive program verification and theorem provers.
6. Town Crier: TC acts as a high-trust bridge between existing HTTPS websites and the Ethereum blockchain. It scrapes website data and delivers it to contracts on the blockchain as concise pieces of data called datagrams. TC uses a combination of Software Guard Extensions, Intel's recently released trusted hardware capability, and a smart-contract front end. It executes its core functionality as a trusted piece of code in an SGX enclave that can prove to remote clients that it is interacting with a legitimate, SGXbacked instance of the TC code (Zhang F. a., 2016).

2.4 Supply Chain Management

A supply chain consists of all parties involved, directly or indirectly, in fulfilling a customer request. The supply chain includes not only the manufacturer and suppliers, but also transporters, warehouses, retailers, and even customers themselves. Within each organization, such as a manufacturer, the supply chain includes all functions involved in receiving and filling a customer request (Chopra, 2013). These functions include, but are not limited to, new product development, marketing, operations, distribution, finance, and customer service. Each organization can be thought of as a company buying from its suppliers and selling to its customers after adding some value. Value addition in supply chain context can be any activity that increases value to the end customer. As such, moving an item from supplier location A to manufacturer location B is considered a value adding activity as it enables availability of goods at the point of use.

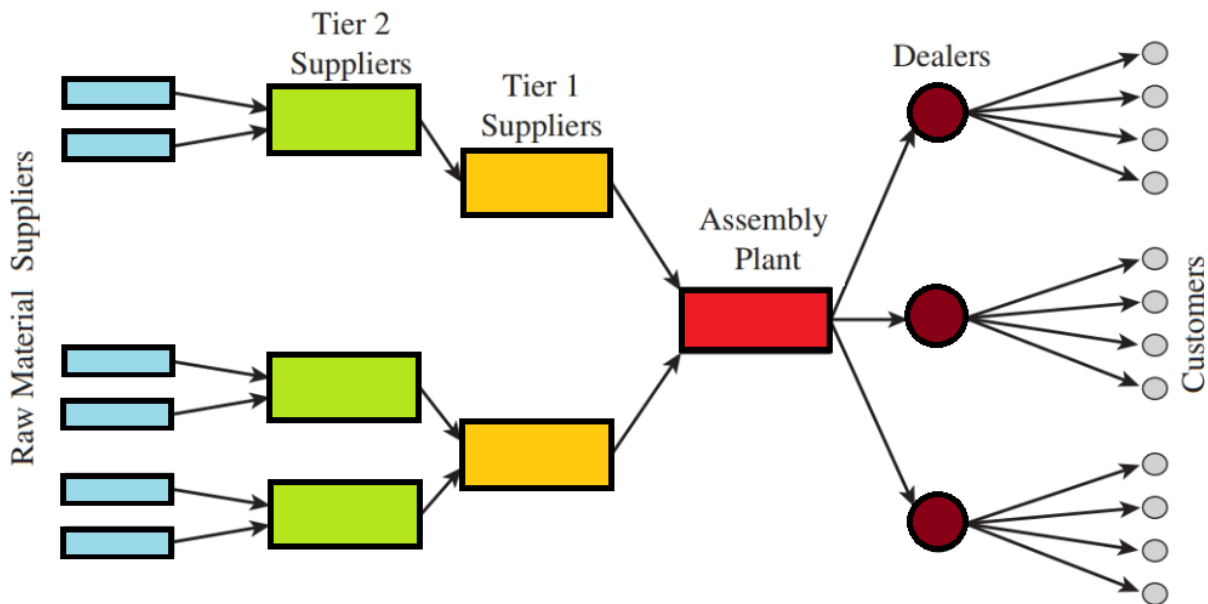


Figure 4: Stages in a typical Supply Chain

2.4.1 Supply chain management objectives

Objective of any supply chain would be to maximize the net value generated. Effective supply chain management involves the management of supply chain assets and product, information, and fund flows to grow the total supply chain surplus. A growth in supply chain surplus increases the size of the total reward, allowing all contributing members of the supply chain to benefit. The net value a supply chain generates is the difference between what the value of the final product is to the customer and the costs the entire supply chain incurs in filling the customer's request. This difference is referred to as the supply chain surplus.

$$\text{Supply Chain Surplus} = \text{Customer Value} - \text{Supply Chain Cost}$$

Supply chain success should be measured in terms of supply chain surplus and not in terms of the profits at an individual stage. For most profit-making supply chains, the supply chain surplus will be strongly correlated with profits. Based on this formula, a sure way to increase Supply chain surplus would be decrease supply chain cost while maintaining the same customer value.

2.4.2 Flows in Supply Chain

For the sake of understanding, let us consider the supply chain of a typical online retailer. When a customer goes online and buys an item of interest, the retailer ships the item to the customer. In the event the customer does not like the item they can typically return it for a refund.

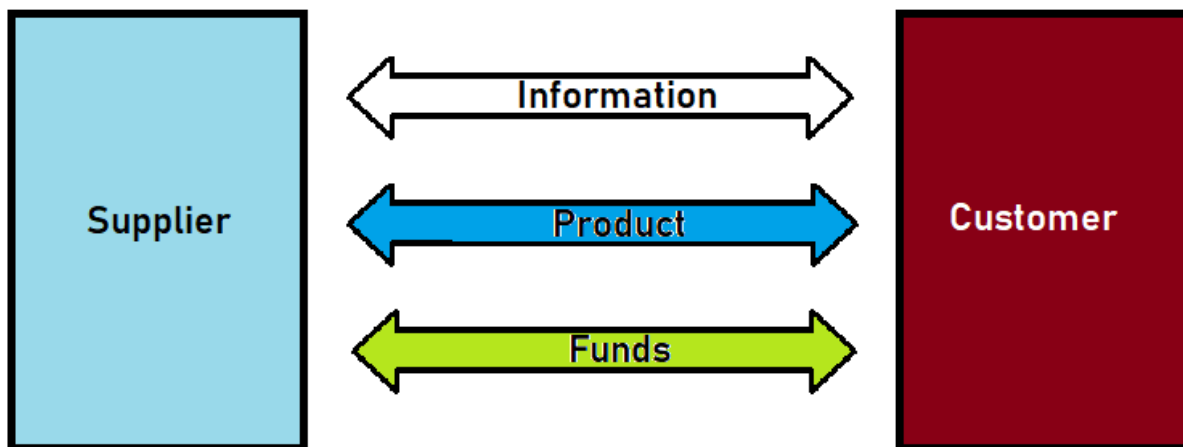


Figure 5: Flows in Supply Chain

We can note here that there is exchange of information when the customer accesses the retailer's website and places an order. Also, there is exchange of product and funds between the two parties. The exact timing of when funds are transferred and when the product is shipped and delivered to the customer varies for each supply chain depending on the trustworthiness of supplier vs. customer, cost of product, type of product and so on.

2.4.3 Mapping Study of existing blockchain applications

A mapping study involves searching literature to determine what sorts of studies addressing the systematic review question have been carried out, where they are published in what databases they have been indexed, what sorts of outcomes they have assessed, and in which populations (Jorgensen, 2007). While this is like a literature survey, the data extracted is much broader and involves the following steps:

- i. Identification of research (searching)
- ii. Selection of primary studies (inclusion/exclusion)
- iii. Study quality assessment (bias/validity)

i. Identification of research

Considering that we are interested in looking at all research related to blockchain applications in supply chain domain, we have searched for technical publications on scholar.google.com using the following search terms:

“Supply chain” “blockchain”

While this resulted in 9240 results, we are unable to access records after the 1000th item. We have considered the first 100 papers within this search result to represent the entire population. While this is convenience sampling, we believe the “sort by relevance” feature within scholar.google.com ensures that the first 100 papers result in a representative sample indicative of most important research in this area.

We also repeated this using academic.microsoft.com terms and with search query provided below:

“logistics” “blockchain”

“Supply chain management” “blockchain”

These queries returned fewer results, although we did find a significant number of overlaps within the first 100 papers further indicating the relevance of our sample.

ii. Inclusions/exclusions

Out of the 100 papers considered, we further exclude papers that are only related to either Supply chain management or blockchain technology alone as these are not relevant to our study.

22 of the original 100 papers selected were related to Supply chain management alone and did not mention applicability of blockchain technology. Another 2 were excluded as they were technical papers related to block chain and did not mention any specific applications/relevance to Supply chain management domain.

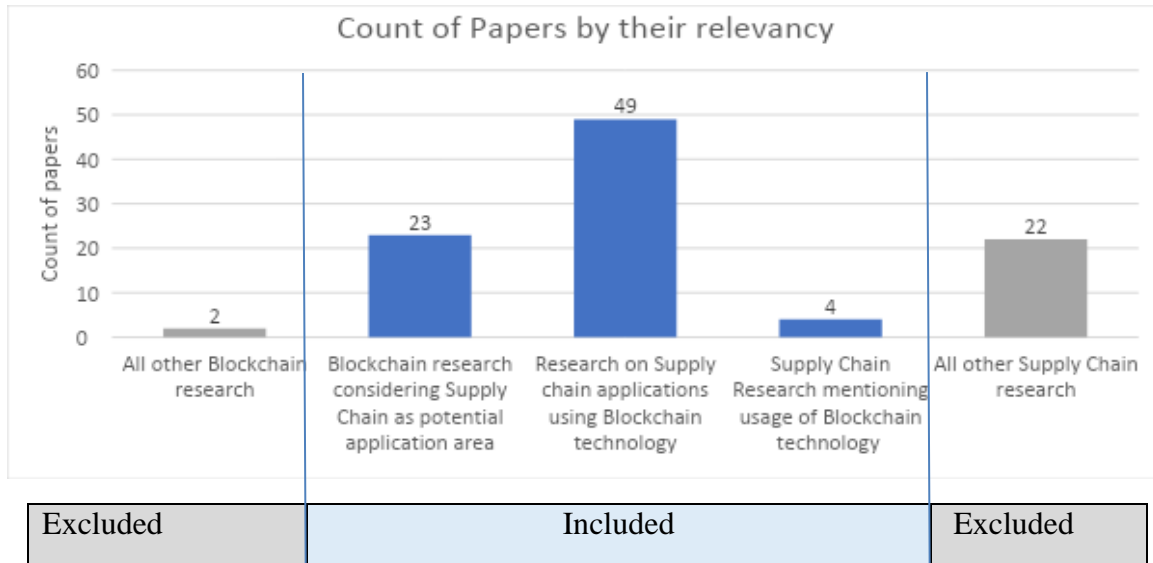


Figure 6: Included research work for mapping study

iii. Bias/Validity

All the remaining 76 papers have been included and categorized by the publisher, published year, overarching interest and a generic term for the aim of the paper.

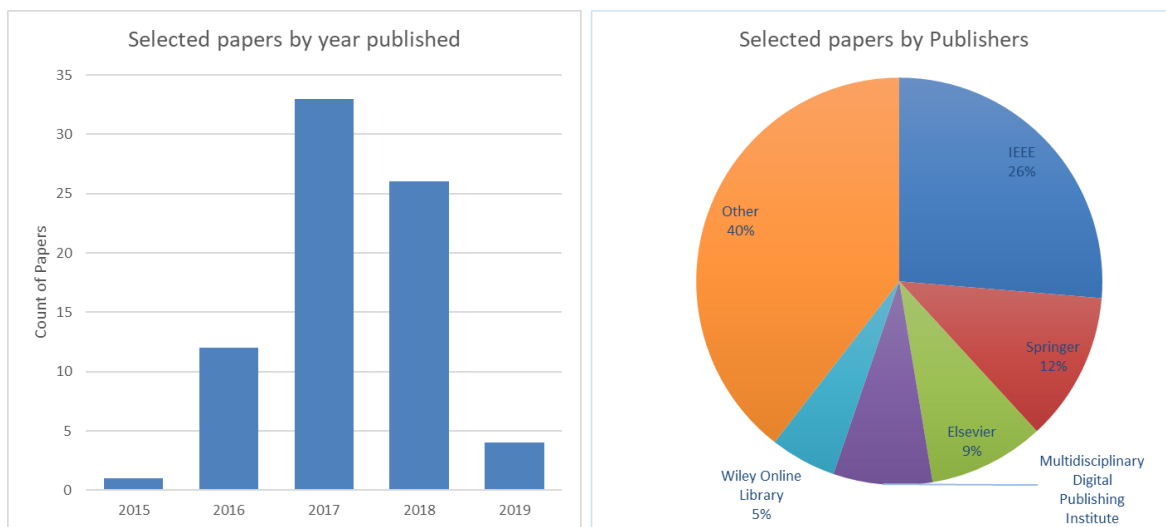


Figure 7: Analysis of Blockchain research in Supply chain by year and publishers

Looking at the above charts, there seems to be significant interest in applying blockchain technology to supply chain management areas. While most of these papers published in technical journals were optimistic in the ability of blockchain to revolutionize supply chain there seems to be a lack of similar interest from Supply chain management journals in leveraging Blockchain

technology. Also, there is disproportionate research that is optimistic about blockchain’s potential to revolutionize supply chain management which could be a result of publication bias.

We then looked at the distribution of papers by industry to answer our last research question as shown below. While much of the research is generic in nature and not limited to any industry, we do see some industries such as Agriculture-Food and healthcare that seem to be keener on leveraging blockchain technologies. These industries are particularly interested in tracking provenance to ensure drugs/food consumed by end users are unquestionably safe and whose quality has not been compromised in any way.

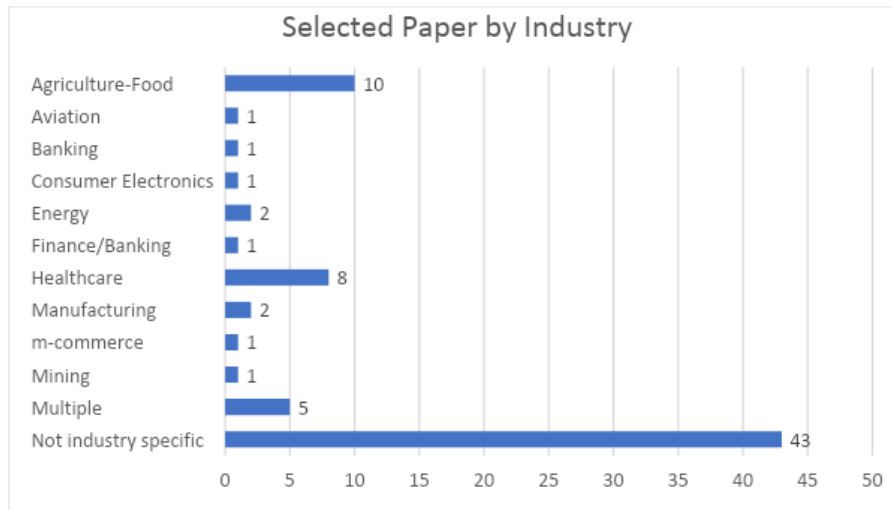


Figure 8: Analysis of Blockchain research in Supply chain by Industry

Lastly, we categorized the selected papers into the following four “implementation maturity” categories ranging from discussions on whether blockchain technology can even be used to solve supply chain problems to actual real-world implementations along with lessons learned for future.

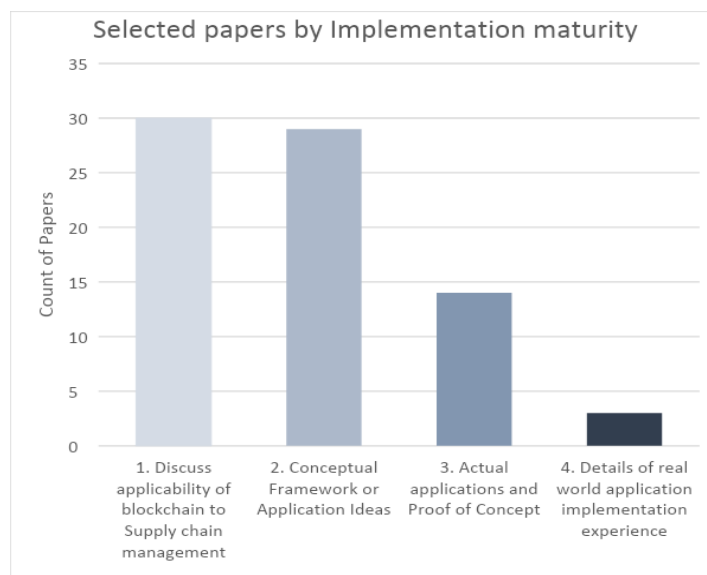


Figure 9: Analysis of implementation maturity of Blockchain applications in Supply chain

2.4.4 Results of mapping study

This is a rather green field with a lot of unstructured approaches to applying blockchain technology to solve supply chain management problems with a handful of application areas emerging.

While most of these papers fell into only one classification some papers spread across more than one classification. In such cases, we have classified it under the heading that is most relevant or prominent within the paper.

Based on our findings shared above, we now proceed to answer our initial research questions with the assumption that the selected set of papers represent the overall body of research related to blockchain applications in supply chain management domain.

1. What is the current state of research related to blockchain applications in the supply chain management domain?

There seems to be considerable research in evaluating usage of blockchain applications to solve supply chain management problems. While there has been considerable debate owing to blockchain technologies' inability to offer any security in the physical world, there are also many applications being developed along with stories of real-world implementation.

2. Is there significant evidence to suggest applicability/non-applicability of blockchain technologies to the supply chain management domain?

While there is no clear answer to this question as blockchain technology has its strength and weakness that may be more suitable to some supply chain than others, the fact that many applications have been developed and successfully implemented increases the likelihood that blockchain technology can be used to solve supply chain management problems. Also, the developing nature of blockchain technology itself provides more reasons to believe some of the roadblocks to successful development of block chain applications may be overcome soon.

3. Which markets or focus areas within the supply chain domain are conducting more research on using blockchain technologies?

Although much of the research work seems to be not industry specific, we certainly see some clusters such as Agriculture-Food/Healthcare that have more industry specific research related to blockchain applications than others. Provenance/Origin tracking seems to be the primary motivator in both these cases.

CHAPTER 3: DEVELOPMENT OF BLOCKCHAIN APPLICATION

3.1 Developing Proof of concept

Let us consider the case of large retailers like Amazon or Walmart, customers typically transfer funds first and the supplier ships the item afterwards. It is also not uncommon for the transfer of funds to occur after delivery of a product especially in B2B (Business-to-Business) transactions. In other words, either the buyer or the seller needs to have trust in the other party and transfer goods or funds in good faith that the other party will hold their end of the deal. While there is always legal recourse in the event one of the parties does not act as per the initial agreement, it might be difficult to enforce legalities over international borders. When some product is only available from a single source and outside the country limits, buyers would be at risk of losing their funds with no legal recourse. Conversely, this situation could very well happen at the sellers' end when they are selling to unknown buyers outside their country and expecting payment after delivery of goods. To aid both parties in this situation, financial institutions such as banks have an instrument to exchange goods and funds while absolving both parties from all risk associated with the transfer. This instrument is called a 'Letter of Credit'.

3.1.1 Letter of Credit

A letter of credit, or "credit letter" is a letter from a bank guaranteeing that a buyer's payment to a seller will be received on time and for the correct amount. In the event the buyer is unable to make a payment on the purchase, the bank will cover the full or remaining amount of the purchase (KAGAN, 2020).

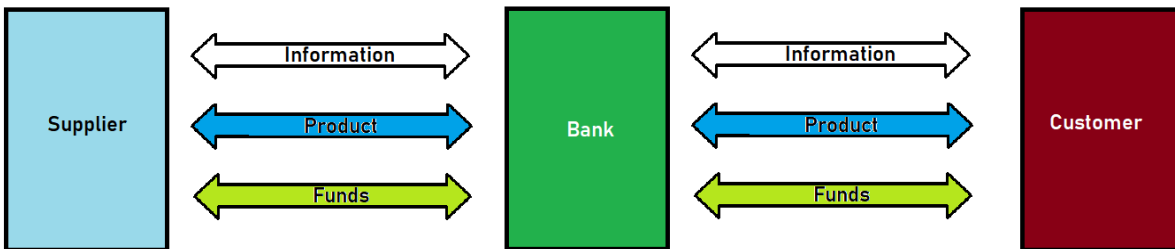


Figure 11: Flows in supply chain with Letter-of-credit

Now, the bank acts as an intermediary and ensures funds are transferred from Customer to Supplier only after goods are delivered as agreed. While this process enables transactions between two mutually untrusting parties, the bank charges a fee starting at 0.75% of the item being bought. In case of expensive items these fees quickly add up to increase supply chain cost and reduce overall supply chain surplus. Furthermore, adding an intermediary increases the time taken to complete a transaction by introducing more touchpoints and handshakes in the process.

Such transactions would be a good candidate for a permissionless blockchain application as all the actors are not known from start and transparency/public verifiability are essential to prevent fraudulent activities.

3.1.2 Proof of Concept: Decentralized Letter of Credit

In this section we propose a decentralized marketplace application that can effectively replace a letter of credit. Let us assume that a buyer wishes to purchase a specific item from an unknown seller. For the sake of simplicity, we are assuming that there are no returns of products and no partial shipment/refund being made. That is, the buyer is refunded the full amount* or pays the full amount after successfully obtaining the item. While there are a myriad of supply chain issues that can be inspected from the point an item is bought by the buyer to seller receiving payment, we consider only the following four cases with either buyer or seller being fraudulent and provide provisions within the decentralized app to resolve these efficiently.

Potential outcomes after purchase		Seller	
		Good	Bad
Buyer	Good	Seller ships item, buyer confirms receipt and seller receives payments.	Seller doesn't ship/ships wrong/defective item and still receives payment from buyer.
	Bad	Seller ships item but buyer does not confirm receipt and does not release payments	When both parties are fraudulent there's no transaction. Seller doesn't ship and buyer doesn't pay

Figure 12: Potential outcomes with decentralized purchasing

As can be seen here, the DApp needs to address cases either one of the actors are fraudulent as when both are fraudulent, or both are not there is no issue to be fixed. Hence, the smart contract based DApp should be able to issue funds to good sellers even when the buyer does not confirm receipt of the item. Also, the DApp should be able to refund a legitimate buyer in the event they do not receive an item as promised from a fraudulent seller. We are assuming all the details of expected delivery would be agreed upon by the buyer and seller prior to the purchase.

Here we propose a decentralized marketplace application that would enable sellers to post their items for sale along with terms and conditions. The buyers can negotiate these terms and once they reach an agreement, they can purchase the item. At this stage, Ether is transferred from the buyer's account to an escrow account by the smart contract. The money from the buyer will be held here until successful completion of transition. This also triggers a notification to the Seller prompting them to ship the item. Once the customer receives the goods as expected, they can 'confirm receipt' of the item causing the smart contract to release funds and record this transaction in a new block.

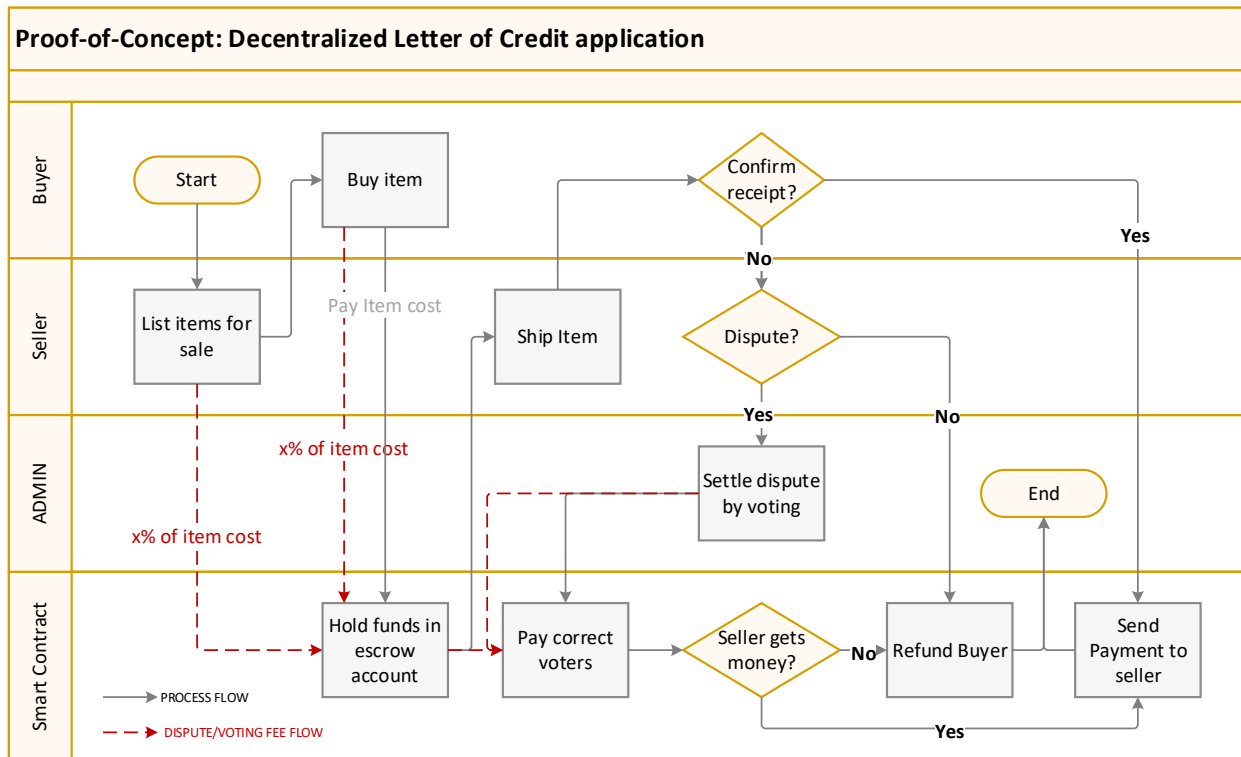


Figure 13: Proof-of-concept of decentralized letter of credit application

Now let us consider the cases where the buyer fails to confirm receipt after receiving an item as detailed in the agreement before purchase. If the buyer does not confirm within the agreed number of days, the marketplace website would provide the seller with an option to ‘dispute’ the transaction. Once a seller requests for a dispute settlement, the transaction is shared with ADMINS who can act as Jury. These ADMINS can then look at all the documents shared by the seller within the stipulated amount of time along with historical transactions of buyers and sellers to decide if buyer or seller are acting fraudulently. At the end of this stipulated time votes of all the ADMINS are counted to determine if the buyer needs to be refunded or if the seller needs to be paid. Upon receiving the judgement from ADMINS, the smart contract will issue a refund to the buyer or process payment to the seller and the transaction is recorded on the blockchain. In order to ensure ADMINS act as good jury and are not arbitrarily casting their vote, only those that voted for the majority group will receive a fee for acting as jury.

Furthermore, the fee for settling a dispute is deducted directly from the buyer or seller’s account. This discourages fraudulent behavior and encourages the parties to settle any minor misunderstanding offline without necessarily starting disputes.

As the usage of this smart contract becomes widespread and a large number of ADMINS are available to settle disputes, ADMINS can be provided increasingly better access to vote on transactions with larger sums of money based on their history of correctly identifying the party

with majority votes. This will serve the dual purpose of incentivizing ADMINS to cast their votes carefully while also penalizing ADMINS that vote arbitrarily just to receive a dispute settlement fee. The number of transactions ADMINS can vote on can also be time phased so that any single ADMIN cannot place many votes in a small period. Lastly, each transaction can have a max number of votes that decays over time upon reaching which the decision will be made and a new transaction written to the blockchain.

3.2 Design and Development of Decentralized Application Using Solidity Smart Contracts

3.2.1 Requirements

To develop a decentralized application that can effectively substitute a traditional letter of credit, we would primarily need a website that acts as a marketplace for buyers and sellers. Here sellers must be able to post their items for sale and buyers must be able to buy using Ether which should be held by the smart contract until transaction completion. Once the item is delivered/service rendered, the buyer must have the ability to confirm receipt of goods/services that should trigger the smart contract to disburse funds to the seller. In the event the buyer does not confirm receipt within a pre-agreed number of days, the seller should have an option to start a dispute and attach necessary evidence. This dispute must then be broadcasted to admins accounts who act as jury and settle the dispute in favor of either the buyer/seller by placing their votes. The contract must then be able to transfer money to the winning party and pay the admins who voted for the majority group as dispute settlement fee marking completion of contract. In the event no action is taken by both parties after purchase for a long period of time then the buyer should receive a refund for the amount paid for purchase of the goods/service.

3.2.2 Proposed framework

To meet these requirements, we can make use of Truffle suite that provides a framework with all the necessary resources packaged and ready for the development and implementation of a decentralized application. Metmask can be used to transact with digital currency. The smart contract can be developed in Solidity and tested in remix IDE. Front end can be built using Angular JS and Web3 which is also supported by Truffle Suite. Backend can be developed for offline transaction handling using any supported database such as mongodb or sqlite. Lastly, the smart contract can be developed and deployed on Ganache which emulates blockchain networks on local machines. For testing and quality assurance, we could deploy the smart contract on the Ropsten Ethereum test network which is a real time blockchain for testing new applications. Once the smart contracts are tested and ready, they can be deployed on the main Ethereum network.

A decentralized letter of credit application as conceptualized in this section would likely be used by unknown parties. Since all of them cannot be trusted it would be hard to accomplish this task in a permissioned blockchain such as Hyperledger. Such a blockchain platform is more suitable for enterprise level implementation where all participants are known and trusted. The application conceptualized here would be deployed directly on the blockchain and needs to be accessible across the globe. While Hyperledger provides high scalability in terms of performance, this application should by its very nature not witness high volumes of transactions since letter of credit

is usually sought for high value goods/services that cannot be sourced quickly. Furthermore, typical delivery lead times for such goods/times can range from a couple of weeks to several months. In such cases, a few minutes or even hours of transaction time would not make much difference. Transaction fee is not a concern either considering that the number of transactions for any participants would be low and individual transactions values would be high. A higher transaction fee may even help as it deters low value commodity sellers from choking the network bandwidth. Security, reliability, and audit trail of transactions are crucial for such an application. Lastly, since smart contract logic can get complicated quickly with changing business scenarios the platform on which it is deployed needs to be turing complete.

Considering these features and implications as mentioned in section 2., we proceed with developing our decentralized letter of credit application on Ethereum using Solidity language for writing smart contracts. Even with a dispute settlement fee the overall cost incurred for a transaction would be much less than going through a central agency such as a bank. Furthermore, when there is no dispute there is no cost incurred by the buyer or sellers other than the nominal transaction fee which would be insignificant compared to cost of the goods/service and the utility it offers to both parties.

3.2.3 Metamask, Truffle Suite and Ganache

MetaMask is a plugin for browsers that allows users to manage accounts and their keys in a variety of ways, while isolating them from the site context (Lee, 2019). This greatly improves security as storing user keys on a single central server, or even in local storage, can lead to mass account thefts. Also, this plugin lets developers interact with the globally available Ethereum API that identifies the users of web3-compatible browsers and whenever a transaction signature is requested, MetaMask will inform the users of the transactions. MetaMask helps retrieve data from blockchain and lets users securely sign and manage transactions on blockchain. MetaMask supports different networks including Ethereum main network, Ethereum test networks provided by infura and Ganache local blockchain network.

Truffle Suite is a framework for building, testing, and deploying applications on the Ethereum network that was founded by Tim Coulter. The Truffle Framework consists of three primary development frameworks for Ethereum smart contract and decentralized application (dApp) development called Truffle, Ganache, and Drizzle (Mohanty, 2018).

Truffle is a development environment, testing framework and deployment pipeline for Ethereum dApps primarily. Truffle takes care of managing contract artifacts and includes support for custom deployments, library linking and complex Ethereum applications. It also provides automated tests for contracts in both JavaScript and Solidity. Last, Truffle provides an interactive console, which includes access to all Truffle commands and contracts built.

Truffle can be used to bootstrap contracts and run a network-aware script. Truffle is operated in the Terminal and has a range of handy commands that can be used at different stages of developing a dApp. Using Truffle's `unbox` command, we can download a pre-built boilerplate project to

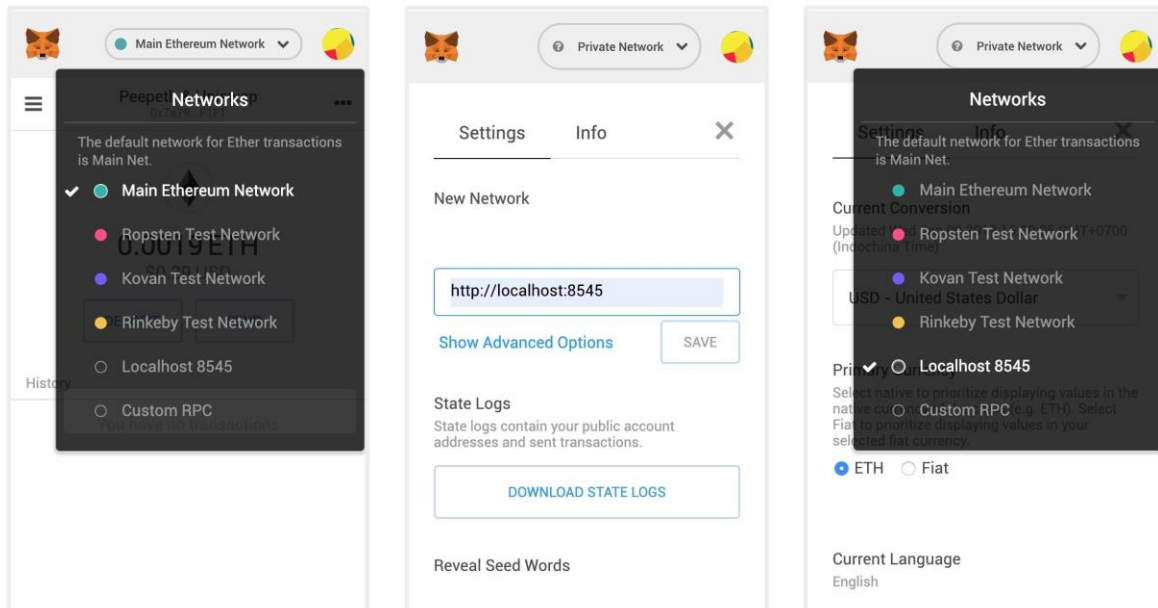


Figure 14: Working with Metamask

bootstrap a dApp. These pre-built projects range from React with Truffle and Webpack boilerplate to ERC20 smart contract examples and tutorials.

A project can be started from scratch by running `truffle init` in the root project directory. This will create a bare bone structure for developing the dApp. After running this command, we can find a folder structure within the root project directory including `contracts/`, `migrations/` and `tests/` along with a `truffle-config.js` that acts as a configuration file for Truffle. These folders serve the following purpose:

`contracts/` — hosts all the Solidity (.sol) smart contract files. Contracts are developed within the `contracts` folder, before being migrated onto a blockchain, and then tested using Truffle’s automated testing capabilities. Contracts can be written, and IDEs such as Sublime Text provide syntax styling for Solidity.

`migrations/` — hosts all migration files that help deploy smart contracts onto an Ethereum blockchain.

A migration is essentially just a set of instructions on how the smart contracts need to be deployed and could look like the following:

```
var MyContract = artifacts.require("MyContract");
module.exports = function(deployer) {
  // deployment steps
  deployer.deploy(MyContract_1);
  deployer.deploy(MyContract_2);
  ...
};
```

Next, a blockchain must be selected where these contracts will be deployed. This could be the mainnet, a testnet or a local private blockchain, which is also provided by Truffle Suite and is called Ganache. Together Truffle and Ganache generate a blockchain on a local machine to test and deploy smart contracts.

Also, for Truffle Migrations to work, a **Migrations** contract is required which provides an interface for managing Truffle deployments. There is also an `1_initial_migration.js` file available in the `migrations/` folder ready to deploy the contract. Truffle remembers which migrations have already been run, and only re-runs every migration when the special `--reset` flag is used with `truffle migrate`.

Lastly, there is another folder generated as part of `truffle init` command which is `tests/`:

`tests/` — is the directory for hosting unit tests for smart contracts. Tests can be written in Javascript, Typescript and Solidity. In Javascript, Truffle uses the Mocha testing framework and Chai for assertions, providing tried and tested tools for the job.

3.2.4 Ethereum environment setup

To deploy smart contracts on an Ethereum blockchain we can connect to it by using `truffle-config.js` in the root directory. By default, this file includes a development network that has already been configured for localhost. We can expand this config file to include more networks, such as a live mainnet.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*" // Match any network id
    }
  }
};
```

Ganache provides a GUI for displaying blockchain state. Upon blockchain instantiation 10 accounts are created to aid in development and testing. This blockchain is assigned to a random network ID and has no relation to real-world public Ethereum blockchains. A mnemonic is also generated to uniquely identify this blockchain which is automatically instantiated and is accessible via localhost:8545. The network itself may be configured within the settings section. By default, Ganache speeds up transactions on blockchain, that is, blocks are auto mined, and transactions are processed instantly. By default, Truffle is ready to communicate with Ganache out of the box.

Now, to deploy contracts on the live network, we can call migrate with the `--network` flag:
`truffle migrate --network live`

This allows us to define a range of networks and migrate to them as required, which is helpful while moving from testing on local blockchains to deployment on live mainnet.

```
module.exports = {
  networks: {
    development: {
      ...
    },
    live: {
      host: "<host_ip_address>",
      port: 80,
      network_id: 1
    }
  }
};
```

Along with the folders, the front-end app folder will also be present in the root directory of the Truffle project. For example, if a Create React App is employed with Truffle, the React app is present within the root directory and resides in a separate web-app/ folder. Next, front must communicate with our smart contracts, and this may be done via web3. Web3.js is a collection of libraries that enables us to send Ether from one account to a different, read and write data from smart contracts and build smart contracts. Web3.js talks to Ethereum Blockchain through JSON RPC, which is a "Remote Procedure Call" protocol. Since ethereum is a peer-to-peer network of nodes, it stores duplicates of all data and code on the blockchain. Web3.js allows us to request data from a private Ethereum node using JSON RPC and write data to the network. It's quite like using jQuery with a JSON API to read and write data with an internet server. By default, web3.js looks for a web3 provider — a blockchain client or light node running that can handle contract communication and transact on an Ethereum blockchain.

3.2.5 Solidity Smart contracts

Smart Contracts are written in a Solidity programming language. Solidity syntax is similar to JavaScript language which supports inheritance, libraries and complex user-defined types. Solidity is a high-level language used for creation of smart contracts. The Solidity integration of C++, Python and JavaScript languages and it targets the Ethereum Virtual Machine (EVM) (Dannen, 2017).

A contract in the sense of Solidity is a collection of code (functions) and data (state) that resides at a specific address on the Ethereum blockchain. All the code of the smart contract is visible to the public, and we can allow anyone connected to the network to call functions on the smart contract.

The `pragma` keyword is used to enable certain compiler features or checks. A `pragma` directive is always local to a source file, so `pragma` needs to be added to all files if it needs to be available across the whole project. If another file is imported, the `pragma` from that file does not automatically apply to the importing file. As we are using Solidity 0.6.4 in this project, this contract file would not compile with a compiler earlier than 0.6.4, and it also won't work on a compiler starting from version 0.7.0. As there will be no breaking changes until version 0.7.0, the code will compile as intended.

Solidity is a statically typed language, so we must first specify the data type of variables as shown below:

```
pragma solidity ^0.6.4;

contract LOCdAPP {
    using SafeMath for uint256;
    enum UserType{noUser,buyer,seller}
    address[] admins;
    struct ProductStruct{
        string name;
        string description;
        uint256 price;
        uint256 amount;
        uint256 disputePrice;
        address payable buyer;
        address payable seller;
        bool isOrdered;
        bool isShipped;
        bool confirmDelivery;
    }

    mapping (uint256 => ProductStruct) public product;
    mapping (address => uint256[]) productList;
}
```

Figure 15: Coding smart contract in Solidity

Variables declared within the contract block are called a “state variable” as the value of this variable is stored on the blockchain and is accessible to all functions of the contract.

Structs are custom defined types that can group several variables.

Enums can be used to create custom types with a finite set of ‘constant values’.

Mapping types use the syntax `mapping(_KeyType => _ValueType)` and variables of mapping type are declared using the syntax `mapping(_KeyType => _ValueType) _VariableName`. The `_KeyType` can be any built-in value type, bytes, string, or any contract or enum type. Other user-defined or complex types, such as mappings, structs or array types are not allowed. `_ValueType` can be any type, including mappings, arrays and structs.

Mappings can be thought of as hash tables, which are virtually initialized such that every possible key exists and is mapped to a value whose byte-representation is all zeros, a type's default value. However, the key data is not stored in a mapping, only its `keccak256` hash is used to look up the value. Because of this, mappings do not have a length, or a concept of a key or value being set, and therefore cannot be erased without extra information regarding the assigned keys.

Functions are the executable units of code within a contract. Function Calls can happen internally or externally and have different levels of visibility towards other contracts. Functions accept parameters and return variables to pass parameters and values between them (Solidity documentation, 2016-2020).

```
mapping (uint256 => ProductStruct) public product;
mapping (address => uint256[]) productList;

constructor (address[] memory _admins) public {
    admins = _admins;
}

function SellProduct(
    string calldata _name,
    string calldata _description,
    uint256 _price) external payable {
    require(uint(user[msg.sender].userType) == 2, 'Invalid User');
    require(msg.value == _price.div(100), 'Insuffent Dispute value');
    P_ID++;
    product[P_ID].name= _name;
    product[P_ID].description= _description;
    product[P_ID].disputePrice= msg.value;
    product[P_ID].price= _price;
    product[P_ID].seller= msg.sender;
    productList[msg.sender].push(P_ID);
}

function BuyProduct(uint256 _P_ID) external payable {
```

Figure 16: Constructors and functions in Solidity

3.2.6 Buyer and Seller registration

Here we have shown how a function can be used to validate and register new users who wish to use this application.

Solidity uses state-reverting exceptions to handle errors. Such an exception undoes all changes made to the state in the current call, all its sub-calls and flags an error to the caller. The convenience functions `assert` and `require` can be used to check for conditions and throw an exception if the condition is not met. We use the `require` function here to ensure that the user account is not already registered as either a buyer or seller. User registration is bound to the user address as can be seen from metamask and a user cannot have more than one role. We use `require` function here as `assert` function is meant for testing internal errors and to check invariants only.

```

function UserRegistraion(
  string calldata _name,
  string calldata _location,
  string calldata _email,
  uint256 _mobile, UserType _userType ) external {
  require(uint(user[msg.sender].userType) == 0, 'Already Registered');
  user[msg.sender]= UserStruct(_name, _location, _email, _mobile, _userType);
}

```

Figure 17: User registration function in Solidity

We can retrieve the address of the account that's calling the function with `msg.sender`. Solidity provides this value inside the `msg` global variable that also lets us retrieve other account related values such as current ether and other user defined variables.

3.2.7 Implementation of Sale transaction

To enable a sale transaction, we first let the supplier call a 'ListProduct' function and collect a dispute settlement fee upfront as a percentage of the item cost. Each product is uniquely identified by product ID 'P_ID' and retains the address of the seller and buyer once a buyer purchases the item by calling the 'BuyProduct' function. We ensure there is enough ether available to complete transactions using the `require` statements again.

```

function ListProduct(
  string calldata _name,
  string calldata _description,
  uint256 _price) external payable {
  require(uint(user[msg.sender].userType) == 2, 'Invalid User');
  require(msg.value == _price.div(100), 'Insuffent Dispute value');
  P_ID++;
  product[P_ID].name= _name;
  product[P_ID].description= _description;
  product[P_ID].disputePrice= msg.value;
  product[P_ID].price= _price;
  product[P_ID].seller= msg.sender;
  productList[msg.sender].push(P_ID);
}

function BuyProduct(uint256 _P_ID) external payable {
  require(uint(user[msg.sender].userType) == 1, 'Invalid User');
  require(_P_ID > 0, 'Invalid Product Id');
  require(product[_P_ID].isOrdered == false, 'Product Already Ordered');
  require(msg.value == product[_P_ID].disputePrice.add(product[_P_ID].price), 'Insuffient Ethereum');
  product[_P_ID].buyer = msg.sender;
  product[_P_ID].amount = msg.value.sub(product[_P_ID].disputePrice);
  product[_P_ID].disputePrice += msg.value.sub(product[_P_ID].amount);
  product[_P_ID].isOrdered = true;
}

```

Figure 18: Implementing sale transaction in Solidity

Other functions are added similar to list and buy products to enable smart contract to keep track of product and payment when it's shipped, and delivery confirmed. When a product is shipped and delivery is confirmed by the buyer, payment needs to be processed to the suppliers and dispute

settlement is refunded to both parties. In the event the buyer does not confirm receipt of goods after receiving it, the seller can start a dispute.

3.3 ADMIN selection and dispute resolution algorithm

When a dispute is started, ADMINS can see this transaction along with comments and proof of delivery from the seller. The ADMINS can then act as jury and place their votes indicating if they think the seller should receive payment or if the buyer should receive a refund. ADMINS will be charged a small fee to ensure votes are not being placed arbitrarily. After dispute settlement, the winning party gets full payment/refund including the dispute settlement fee paid upfront as a deposit with the contract. The dispute settlement fee collected from the losing party along with the voting fee collected from each ADMIN is then shared between the ADMINS voting for the winning party equally. If all ADMINS voted for the same party the entire dispute settlement fee and voting fee collected is split and distributed equally between the ADMINS.

```
if (dispute[_D_ID].count == admins.length) {
  if (dispute[_D_ID].bVote > dispute[_D_ID].sVote) {
    // buyer wins
    product[dispute[_D_ID].productId].buyer.transfer(product[dispute[_D_ID].productId].amount);
    product[dispute[_D_ID].productId].buyer.transfer(product[dispute[_D_ID].productId].disputePrice.div(2));
    product[dispute[_D_ID].productId].disputePrice = product[dispute[_D_ID].productId].disputePrice.div(2);
    product[dispute[_D_ID].productId].buyer = address(0);
    product[dispute[_D_ID].productId].isOrdered = false;
    product[dispute[_D_ID].productId].isShipped = false;
    PayAdmin(_D_ID,true);
  } else {
    // seller wins
    product[dispute[_D_ID].productId].seller.transfer(product[dispute[_D_ID].productId].amount);
    product[dispute[_D_ID].productId].seller.transfer(product[dispute[_D_ID].productId].disputePrice.div(2));
    product[dispute[_D_ID].productId].disputePrice = product[dispute[_D_ID].productId].disputePrice.div(2);
    product[dispute[_D_ID].productId].confirmDelivery = true;
    PayAdmin(_D_ID,false);
  }
}
```

Figure 19: Dispute resolution in Solidity

Success of this application hinges on dispute settlement and is already a major focus area in smart contracts. Since disputes with any contract is inevitable there are several companies providing online dispute resolution services some of which are listed below (Schmitz, 2019):

Kleros

An online crowdsourced arbitrator for smart contract dispute resolution in Ethereum. Kleros system is built on game theory and discovering a “Schelling point” for resolving disputes. Schelling point (or focal point) is a solution that people choose by default in the absence of communication (Schelling, 1980). Kleros works by enlisting random admins as jury from around the world based on the number of cryptocurrencies, they deposit to show their availability and interest.

Aragon

Aragon is another crowdsourced ODR that creates flexible human-readable agreements that parties can enforce via Ethereum by depositing collateral in the form of cryptocurrency. A party can appeal by posting an even larger bond as the complaint moves up the process, and finally may

reach the Aragon "supreme court" judges-these judges are those with the highest success rates on the network.

Jur.io

Jur is similar to Kleros in that disputing parties offer resolutions along with a number of tokens to "stake" their proposals. Voters decide which proposal to uphold and a decision is rendered at the end of time period chosen by the parties. Voters who vote against the majority are penalized by losing tokens and is expected to encourage fair voting.

One of the key shortcomings in these approaches are that the number of voters are preselected and fixed or the time period within which the voters can place their votes is fixed. While the payment of a fee by the voters discourages votes placed without due consideration of the evidence, it would also decrease the number of voters available in the system to settle dispute. There is again a trade off between quality and quantity of votes that can be obtained by tuning the cost of voting fee.

Following are some issues that need to be addressed for proper functioning of this process:

1. Incorrect decision due to too few voters
2. Not sufficient votes collected in time
3. Trigger for dispute settlement. Time bound or based on total number of votes?

Considering the above we develop an algorithm that automatically balances between quality and quantity of votes with a decaying function that determines the number of votes required to settle dispute. Pseudocode for this algorithm is as shown below:

```
after every new vote and at predefined time intervals calculate:
maxvotes = f(p,t)           {where p are transaction parameters, t is
time since dispute}
if total_votes > maxvotes then:
    if lastvote = buyer then:
        buyer_votes = buyer_votes - 1
    else :
        seller_votes = seller_votes - 1
    settle_dispute
elseif total_votes == maxvotes then:
    settle_dispute
else:
    if (vote = buyer) then
        buyer_votes = buyer_votes + 1
    else :
        seller_votes = seller_votes + 1
    total_votes = total_votes + 1
    if total_votes == maxvotes:
        settle_dispute
```

$f(p,t)$ is a decaying function over time that always results in an odd number

In this algorithm, number of votes required to settle dispute is calculated based on transaction parameters and is broadcasted to that many number of ADMINS selected randomly divided by a factor α that represents historical percentage of votes received from the number of ADMINS selected for voting. If all the ADMINS selected places their votes by paying their voting fee, then the dispute is settled immediately. However, if the ADMINS are taking time to place their votes, considering that time is of the essence in many of these contracts, the total number of votes required to settle dispute reduces over time.

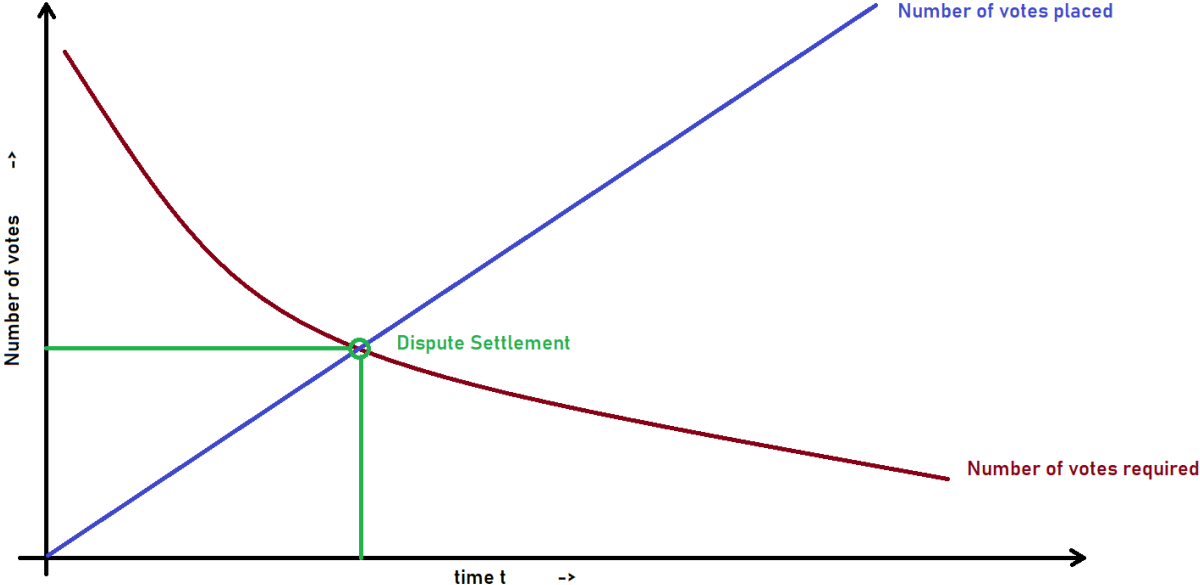


Figure 20: Dispute resolution at optimal time using algorithm

Considering that votes placed cannot be taken back or modified, it is safe to assume that the number of votes placed increase continuously over time. Since the algorithm developed here reduces the number of votes required to settle dispute over time, we can be assured that the algorithm will terminate. Furthermore, as long as some votes are being place, the time take to resolve dispute will be much less than the maximum dispute resolution time agreed by both parties at the beginning of transaction. The algorithm detailed here will essentially result in an automatic dispute settlement at an optimal time based on availability of ADMINS and time since dispute was started.

CHAPTER 4: EVALUATION

4.1 Implementation using AngularJS

Finally, we use AngularJS which is a JavaScript-based open-source front-end web framework that helps in developing single-page applications for our front end (Jadhav, 2015). This website includes pages for new user registration. Roles specific pages include adding new products and creating a dispute for the seller, confirm delivery or claim refund for the buyer and dispute settlement page for the ADMINS where all the disputed transactions are listed. Front end is started after deploying the contract on a local blockchain network provided by Ganache.



Figure 21: Letter of Credit DApp Homescreen

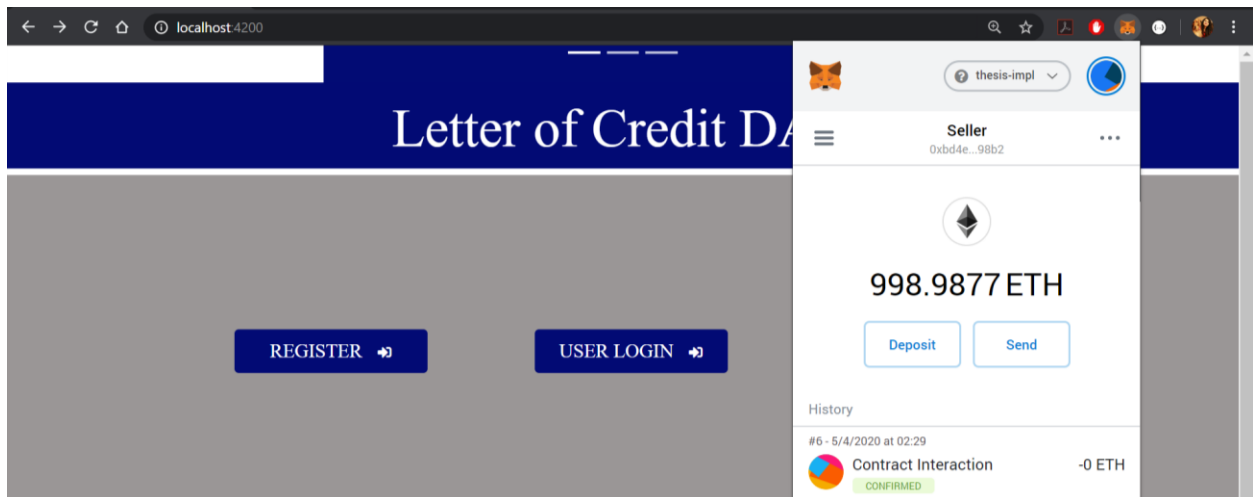


Figure 22: Letter of Credit DApp – Selecting Seller account

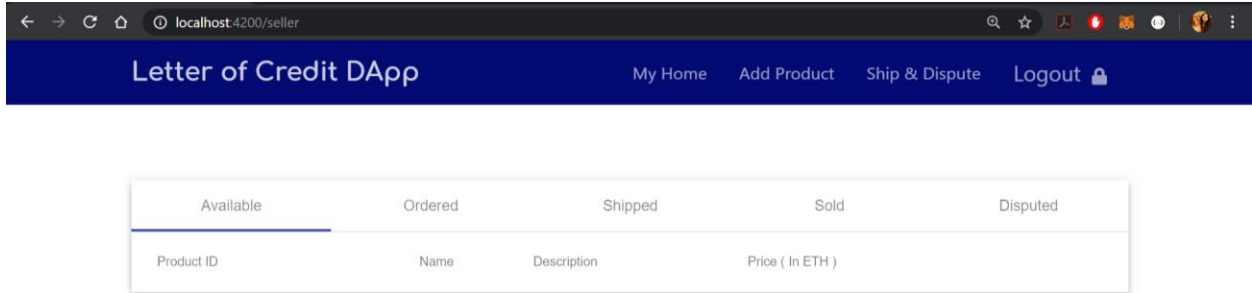


Figure 23: Letter of Credit DApp – Seller Products Page

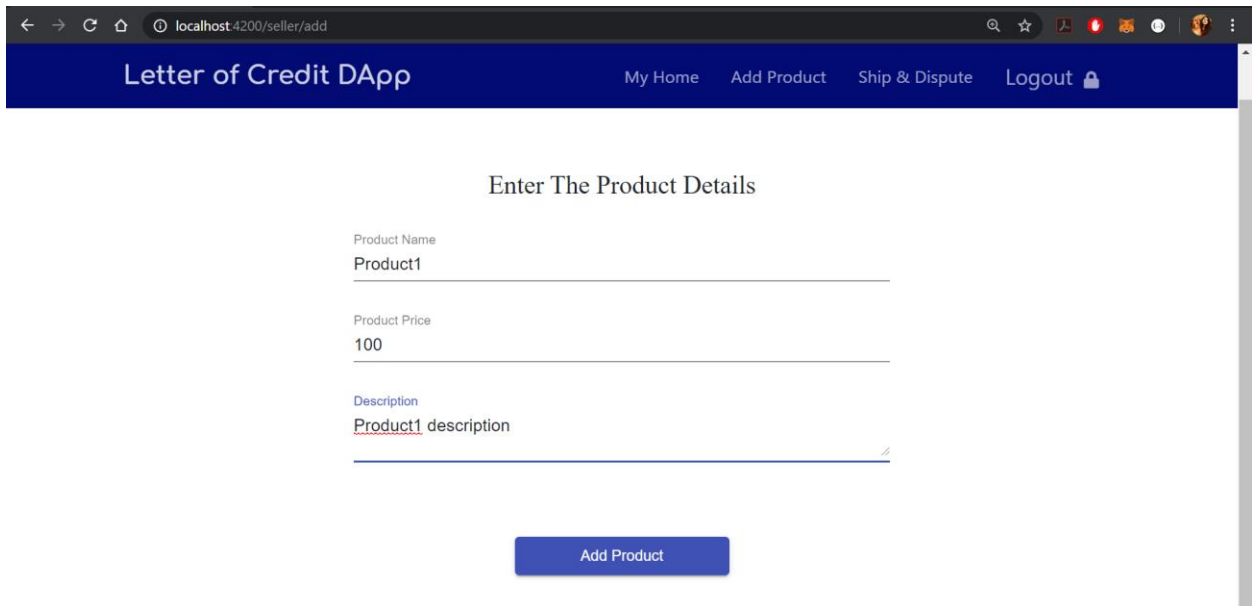


Figure 24: Letter of Credit DApp – Seller Add Product

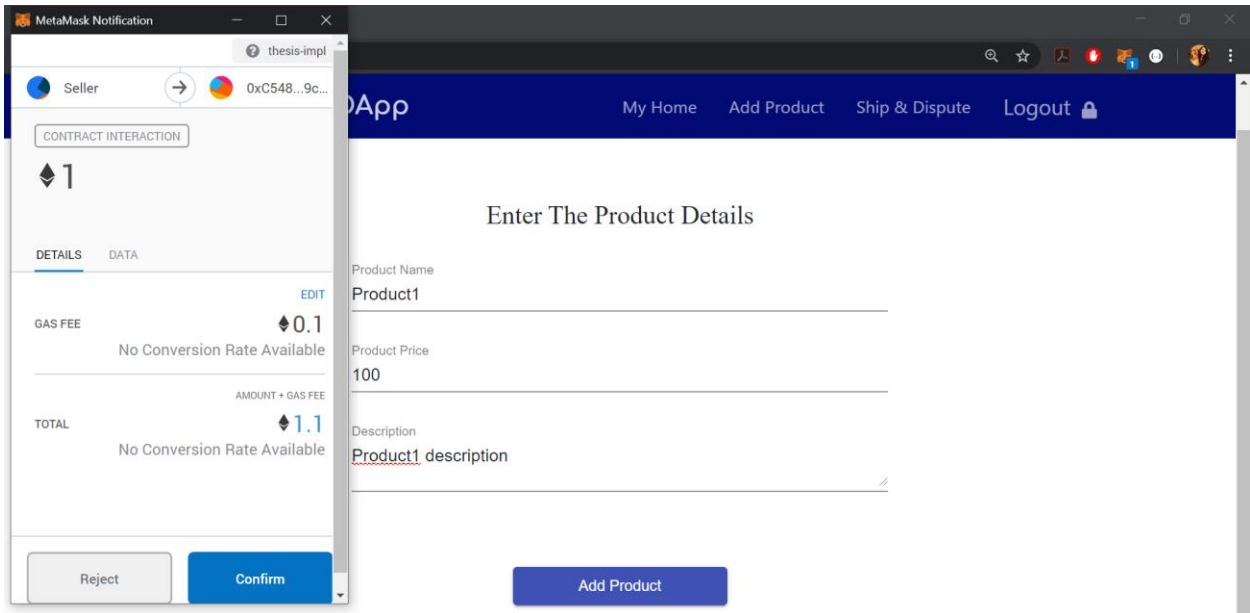


Figure 25: Letter of Credit DApp – 0.1 ETH GAS FEE & 1.0 ETH DISPUTE CLEARING FEE from seller account when product is added to the list

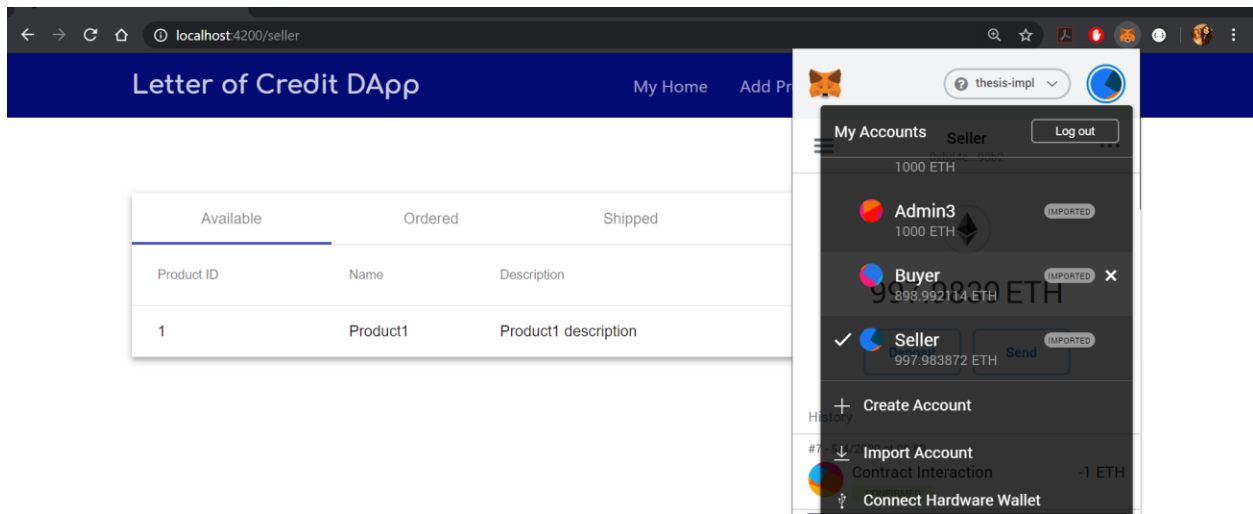


Figure 26: Letter of Credit DApp – Logout from Seller account and login to Buyer account to start transaction

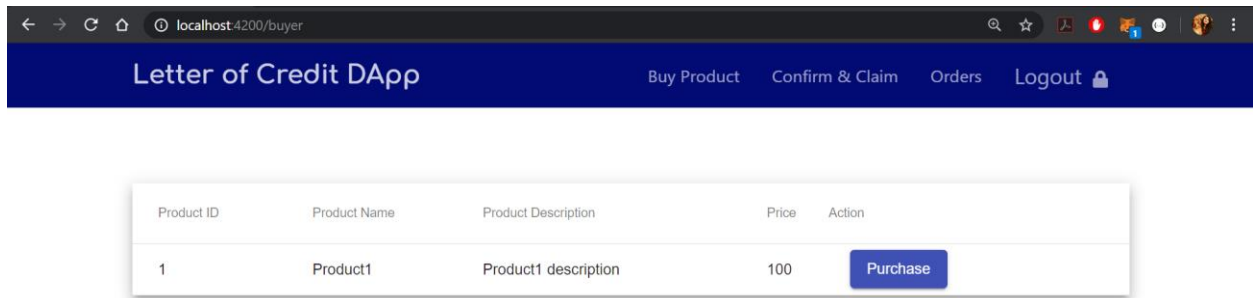


Figure 27: Letter of Credit DApp – Buyer Purchase the product

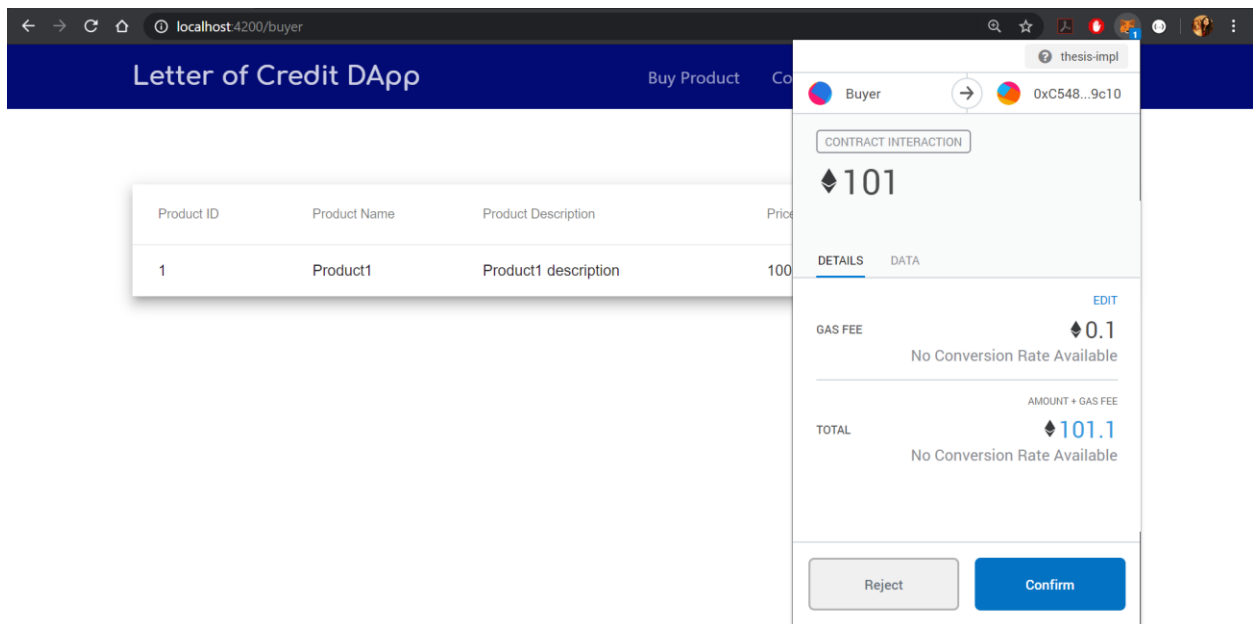


Figure 28: Letter of Credit DApp – 0.1 ETH GAS FEE, 100.0 ETH PRODUCT COST, 1.0 ETH DISPUTE CLEARING FEE from buyer

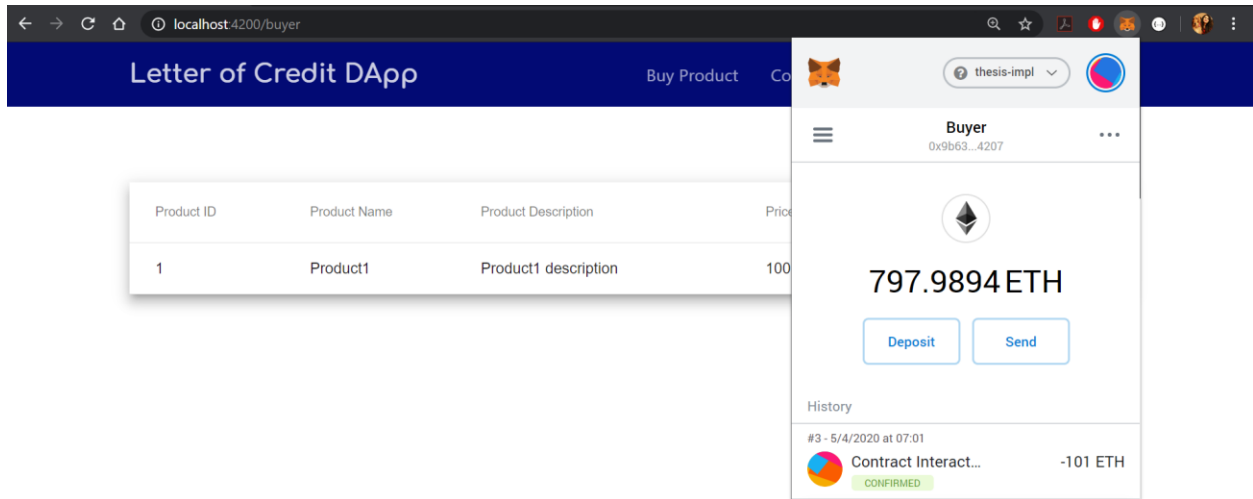


Figure 29: Letter of Credit DApp – Buyer bought the product

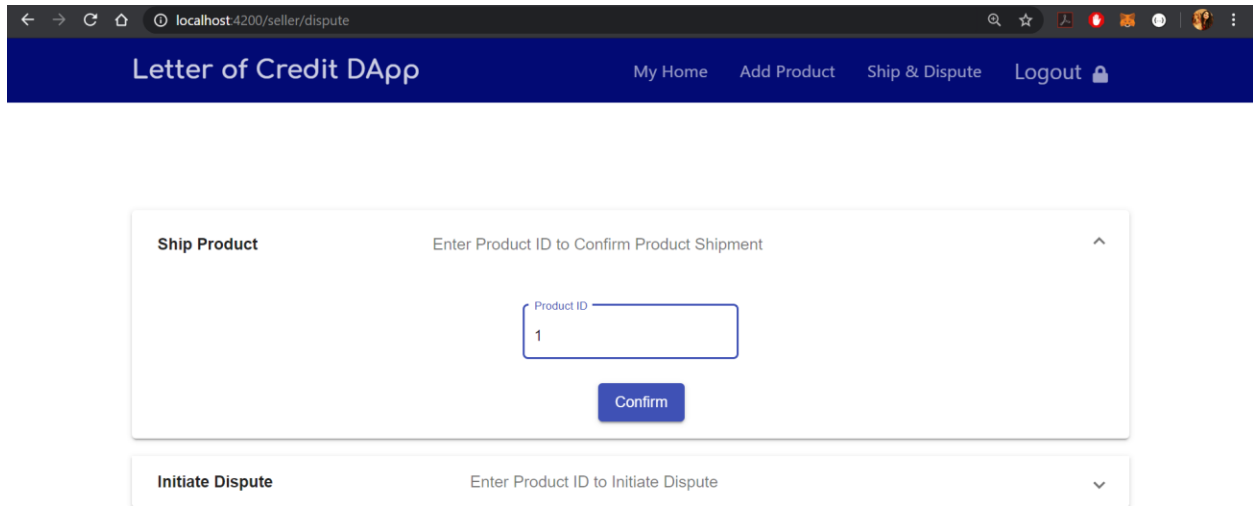


Figure 30: Letter of Credit DApp – Seller ships the Product

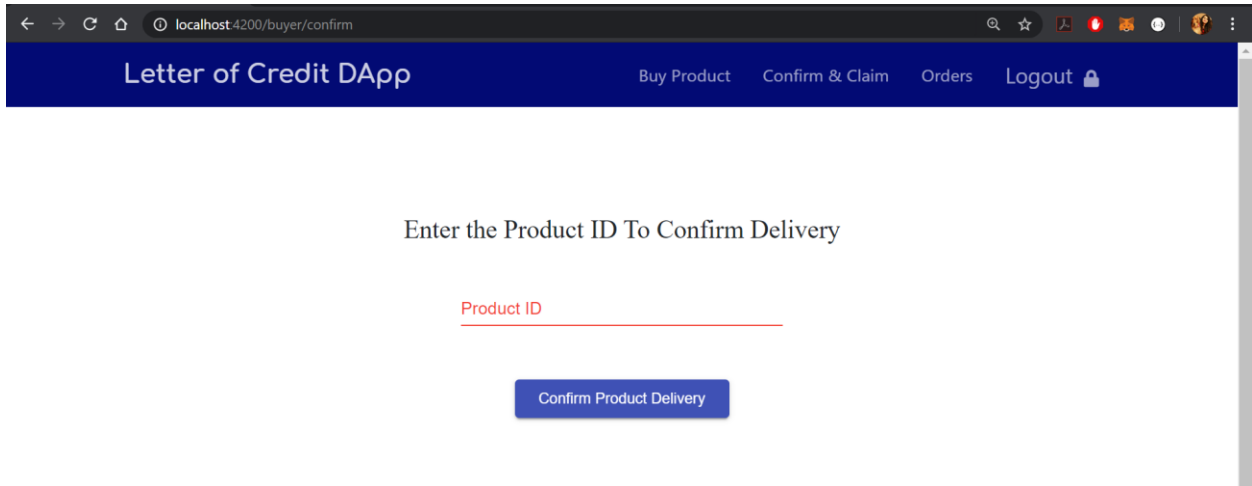


Figure 31: Letter of Credit DApp – Buyer does not confirm delivery

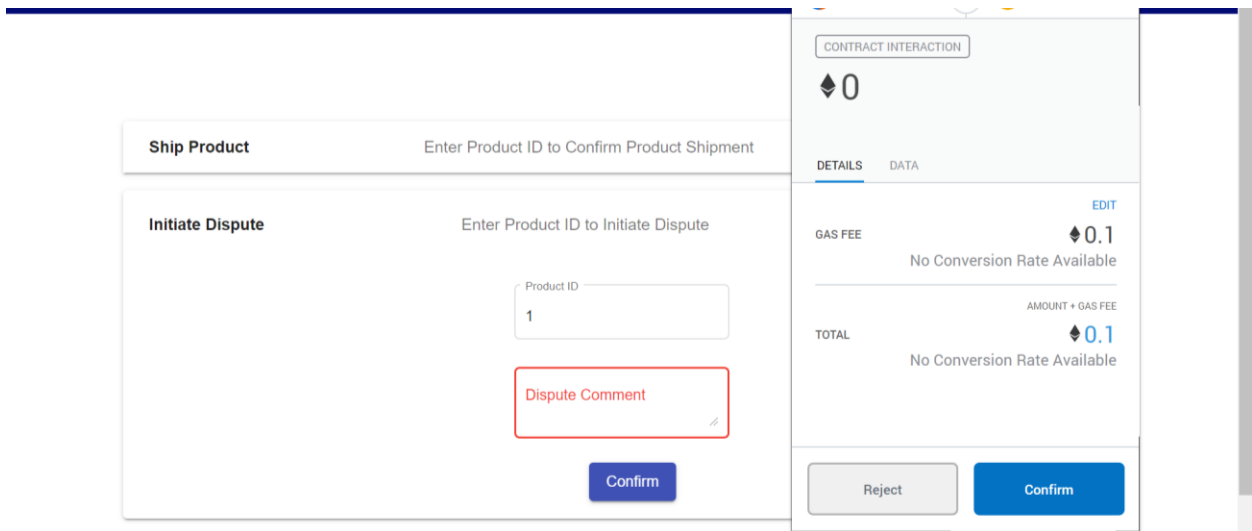


Figure 32: Letter of Credit DApp – Seller initiates dispute

The screenshot shows a web browser at localhost:4200/seller. The page title is "Letter of Credit DApp" and the navigation bar includes "My Home", "Add Product", "Ship & Dispute", and "Logout". A table with five tabs (Available, Ordered, Shipped, Sold, Disputed) is displayed, with the "Disputed" tab selected. The table contains one row of data.

Product ID	Name	Description	Price (In ETH)
1	Product1	Product1 description	100

Figure 33: Letter of Credit DApp – Disputed items

The screenshot shows a web browser at localhost:4200/admin. The page title is "Letter of Credit DApp" and the navigation bar includes "Logout". A table with columns "No.", "Dispute ID", "Product ID", "Comment", and "Vote For" is displayed. The first row contains the values 1, 1, 1, and two buttons labeled "Buyer" and "Seller".

No.	Dispute ID	Product ID	Comment	Vote For
1	1	1		<input type="button" value="Buyer"/> <input type="button" value="Seller"/>

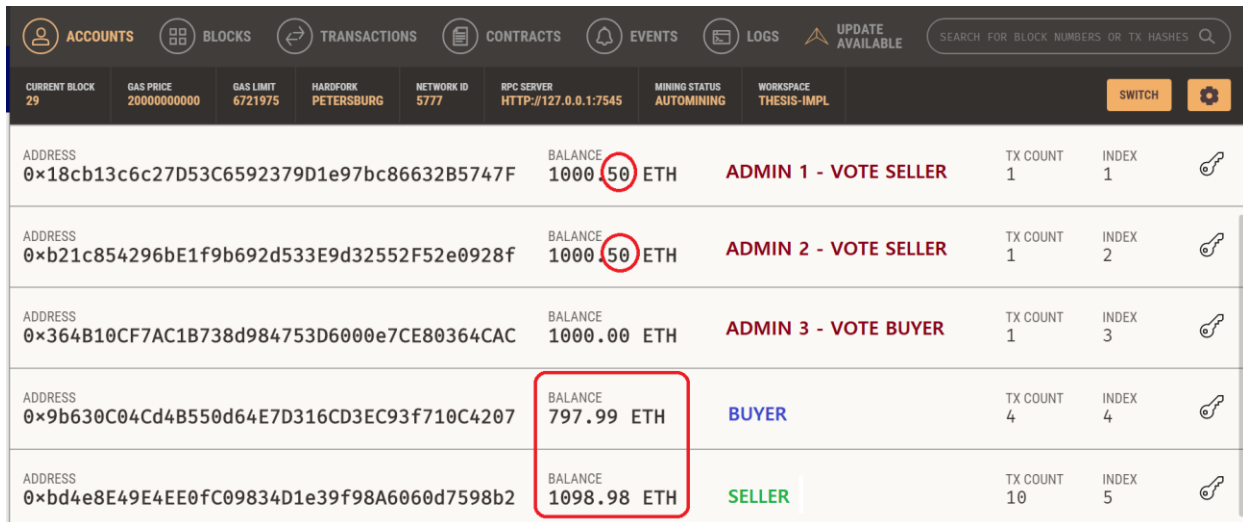
Figure 34: Letter of Credit DApp – Admin1 votes seller

The screenshot shows a web browser at localhost:4200/admin. A dark modal dialog box is displayed in the center with the text "localhost:4200 says Dispute Vote Successful" and an "OK" button. Below the dialog, the same table from Figure 34 is visible.

No.	Dispute ID	Product ID	Comment	Vote For
1	1	1		<input type="button" value="Buyer"/> <input type="button" value="Seller"/>

Figure 35: Letter of Credit DApp – Admin2 votes seller

4.2 Results



ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS	UPDATE AVAILABLE	SEARCH FOR BLOCK NUMBERS OR TX HASHES		
CURRENT BLOCK 29	GAS PRICE 2000000000	GAS LIMIT 6721975	HARDFORK PETERSBURG	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING	WORKSPACE THEISIS-IMPL	SWITCH	⚙️
ADDRESS 0x18cb13c6c27D53C6592379D1e97bc86632B5747F	BALANCE 1000.50 ETH	ADMIN 1 - VOTE SELLER	TX COUNT 1	INDEX 1	🔗				
ADDRESS 0xb21c854296bE1f9b692d533E9d32552F52e0928f	BALANCE 1000.50 ETH	ADMIN 2 - VOTE SELLER	TX COUNT 1	INDEX 2	🔗				
ADDRESS 0x364B10CF7AC1B738d984753D6000e7CE80364CAC	BALANCE 1000.00 ETH	ADMIN 3 - VOTE BUYER	TX COUNT 1	INDEX 3	🔗				
ADDRESS 0x9b630C04Cd4B550d64E7D316CD3EC93f710C4207	BALANCE 797.99 ETH	BUYER	TX COUNT 4	INDEX 4	🔗				
ADDRESS 0xbd4e8E49E4EE0fC09834D1e39f98A6060d7598b2	BALANCE 1098.98 ETH	SELLER	TX COUNT 10	INDEX 5	🔗				

Figure 36: Account balances in Ganache showing accurate working of algorithm

Here we have shown how a disruptive public decentralized application can be developed using completely open source tools that can substitute established traditional instruments such as letters of credit. We also develop an algorithm for dispute resolution that in theory will perform better than other online dispute resolution solutions available. While users of this application may end up spending some money on transaction fee this would still be much less than 0.75% of an expensive item charged by the bank for a letter of credit. Since both parties benefit most when they perform as per their agreement and do not dispute, this encourages ethical and professional behavior. This should result in dispute settlements being exceptions rather than the norm. Furthermore, users or traders holding their end of the bargain are always protected for the sum of money they have invested and need not have to bear any risk. Over time as the usage in the network grows and everyone is transacting fairly, the cost associated with obtaining a letter of credit can be brought down to a few cents. Today, Letters of credit cover 12.5% (1.7%) of world trade, or \$2.3 trillion (\$310 billion). This huge market can be captured by effective development and implementation of this decentralized application. Lastly, from an ethics point of view, unlike several other cryptocurrency transactions or decentralized applications that provide complete anonymity allowing illegal transactions to take place, this application would always be subject to review by the admins who would most certainly report concerned agencies if illegal trade activities are found (Friederike Niepmann, 2016).

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

To conclude, I have been able to achieve the objectives of this thesis set forth in section 1.2 as listed below,

- Performed a mapping study of blockchain as applied to supply chain management
- Created a framework for a blockchain-based letter of credit that addresses challenges with using current techniques.
- Systematically reviewed technologies to identify best set of tools
- Developed a novel algorithm to resolve disputes while executing smart contracts.
- Evaluated the effectiveness of this framework by creating a web application.

5.2 Future Work

While we have considered many issues related to purchase of high value to goods/services in absence of trust, there are many more issues that could arise including but not limited to customs, change of ownership, hazardous goods, force majeure events, insurance and fraud. The concepts and framework provided in this thesis are meant to be a step in the right direction and in an effort keep the idea simple, we leave a lot of the complexities to be handled by the admins. Fortunately, we can always build upon this smart contract and carefully add new issues as they are found during dispute settlement. Furthermore, in this proposal, admins can be any independent agents.

Success of this idea hinges on admins doing a good job of settling disputes. While implementation shown here does incentivize only admins who have correctly voted for the majority player, this system can be further strengthened by broadcasting higher valued transactions preferentially to admins who have voted for the majority party consistently. If during any point in time an admin who has always voted correctly starts voting incorrectly, then they should be penalized by only having voting rights on lower value transactions. They can improve their credibility by voting correctly and work their way up to higher valued transactions. This will also deter any fraudulent activities or admins teaming up with each other to turn the case in favor of buyer or seller. The number of votes required could initially be set based on the value of transaction, but it could deteriorate slowly to allow for dispute settlement with fewer votes as time passes. These gamifying mechanisms would go a long way in ensuring dispute settlement is done accurately. Furthermore, voters do not have to work against each other as admins stand to benefit even if all voters pick the same party. Lastly, to reduce chance of malpractice, the admins chosen to vote can be selected randomly.

Considering that disputes can occur for various reasons and one of the parties need not necessarily be at fault for any transaction. We can provide options for admins to decide on a partial payment in case of delivery of goods/services was done but not up to the expectation of the buyer.

We could improve the structure of this application by letting the voting rights remain with all blockchain participants through proof-of-stake. This way all blockchain participants are benefitted even when there are few disputes and it also encourages the jury to review the disputes and vote carefully. As incorrect voting over time will lead to deterioration in trust over the application and could eventually lead to fewer transactions and usage. This is not desirable for stakeholders in the blockchain. This framework would ultimately encourage all parties involved to act ethically and professionally in a truly decentralized and democratic manner. An Ethereum token can also be developed specifically for transactions of this nature and for use with this application that could make this framework more robust.

Lastly, I believe that the framework developed here, is a novel approach to developing framework for decentralized blockchain application. However, since I have not been able to conduct an exhaustive search of all existing frameworks developed for this purpose, it would be worthwhile to consider this task for future work.

Bibliography

- Asharaf, S. a. (2017). *Decentralized Computing Using Blockchain Technologies and Smart Contracts: Emerging Research and Opportunities: Emerging Research and Opportunities*. IGI Global.
- Bartoletti, M. a. (2017). An empirical analysis of smart contracts: platforms, applications, and design patterns. *International conference on financial cryptography and data security* (pp. 494--509). Springer.
- Bayer, D. a. (1993). Improving the efficiency and reliability of digital time-stamping. In *Sequences II* (pp. 329--334). Springer.
- Burgwinkel, D. (2016). Blockchain technology: Einführung für business-und IT manager. In D. Burgwinkel. Walter de Gruyter GmbH & Co KG.
- Burgwinkel, D. (2016). Blockchain technology: Einführung für business-und IT manager. In D. Burgwinkel. Walter de Gruyter GmbH & Co KG.
- Buterin, V. (2013). Ethereum: The ultimate smart contract and decentralized application platform. *Libro blanco de Ethereum*. Consultado en <http://web.archive.org/web/20131228111141/http://vbuterin.com/ethereum.html> el.
- Buterin, V. a. (2014). A next-generation smart contract and decentralized application platform.
- Cachin, C. a. (2016). Architecture of the hyperledger blockchain fabric. *Workshop on distributed cryptocurrencies and consensus ledgers*, (p. 4).
- Castro, M. a.--. (1999). Practical Byzantine fault tolerance. *OSDI*, 173--186.
- Chopra, S. a. (2013). *Supply chain management: strategy, planning, and operation*. Boston, MA: Pearson.
- Christidis, K. a. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 2292-2303.
- Crypto, W. (2020). Ethereum 2.0: The One Tech Upgrade With The Biggest Potential Impact On The Crypto Space. *Benzinga Newswires*.
- Dannen, C. (2017). Introducing Ethereum and Solidity.
- Dika, A. (2017). *Ethereum smart contracts: Security vulnerabilities and security tools*. NTNU. *DOCUMENTATION*. (n.d.). Retrieved from Truffle Suite: <https://www.trufflesuite.com/docs>
- Drucker, P. F. (1998). Management's new paradigms. *Forbes magazine*, 98--99.
- Friederike Niepmann, T. S.-E. (2016). *VOX CEPR Policy Portal*. Retrieved from Trade finance around the world: <https://voxeu.org/article/trade-finance-around-world>
- Giancaspro, M. (2017). Is a 'smart contract' really a smart idea? Insights from a legal perspective. *Computer law & security review*, 825--835.

- Haber, S. a. (1991). How to Time-Stamp a Digital Document. *Crypto '90, LNCS 537*, Springer.
- Herlihy, M. (2017). Blockchains and the future of distributed computing. *Proceedings of the ACM Symposium on Principles of Distributed Computing*, (pp. 155--155).
- Jadhav, M. A. (2015). Single page application using angularjs. *International Journal of Computer Science and Information Technologies*, 2876--2879.
- Jorgensen, M. (2007). Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting, Elsevier*, 449--462.
- KAGAN, J. (2020, April 14). *Letter of Credit*. Retrieved from Investopedia.com: <https://www.investopedia.com/terms/l/letterofcredit.asp>
- Krishnan, S. a. (2020). *Handbook of Research on Blockchain Technology*. Academic Press.
- Le, T. C. (2018). Proving conditional termination for smart contracts. *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*, (pp. 57--59).
- Lee, W.-M. (2019). Using the MetaMask Chrome Extension. *Beginning Ethereum Smart Contracts Programming*, 93--126.
- List of Known Bugs*. (2016-2020). Retrieved from Solidity: <https://solidity.readthedocs.io/en/v0.6.6/bugs.html>
- Mazieres, D. (2015). The Stellar Consensus Protocol. *A Federated Model for Internet-level Consensus. Version July*.
- Mense, A. a. (2018). Security vulnerabilities in ethereum smart contracts. *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, (pp. 375--380).
- Mohanty, D. (2018). Deploying smart contracts. *Ethereum for Architects and Developers*, 105--138.
- Mulligan, C. a. (2018). Blockchain beyond the hype: A practical framework for business leaders. *World Economic Forum, White Paper*.
- Nakamoto, S. (2008). A peer-to-peer electronic cash system. *Bitcoin*.--URL: <https://bitcoin.org/bitcoin.pdf>.
- Natoli, C. a. (2016). The blockchain anomaly. *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)* (pp. 310--317). IEEE.
- Nguyen, G.-T. a. (2018). A Survey about Consensus Algorithms Used in Blockchain. *Journal of Information processing systems*.
- O'hara, K. (2017). Smart contracts-dumb idea. *IEEE Internet Computing*, 97--101.
- Omohundro, S. (2014). Cryptocurrencies, smart contracts, and artificial intelligence. *AI matters*, 19--21.

- Raskin, M. (2016). The law and legality of smart contracts.
- Samaniego, M. a. (2016). Hosting virtual iot resources on edge-hosts with blockchain. *2016 IEEE International Conference on Computer and Information Technology (CIT)* (pp. 116--119). IEEE.
- Smith, B. a. (2016). IBM Blockchain: An enterprise deployment of a distributed consensus-based transaction log. *Proc. Fourth International IBM Cloud Academy Conference*, (pp. 140--143).
- Solidity documentation*. (2016-2020). Retrieved from Functions:
<https://solidity.readthedocs.io/en/v0.6.6/structure-of-a-contract.html#functions>
- Szabo, N. (1994). Smart contracts. *Unpublished manuscript*.
- Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*.
- Tjong Tjin Tai, E. (2017). Formalizing contract law for smart contracts. *Tilburg Private Law Working Paper Series*.
- Wood, G. a. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 1--32.
- Wu, M. a. (2019). A comprehensive survey of blockchain: From theory to IoT applications and beyond. *IEEE Internet of Things Journal*, 8114--8154.
- Wust, K. a. (2018). Do you need a blockchain? *Crypto Valley Conference on Blockchain Technology (CVCBT)* (pp. 45--54). IEEE.
- Zhang, F. a. (2016). Town crier: An authenticated data feed for smart contracts. *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, (pp. 270--282).
- Zhang, J. (2019, October 1). *What are Smart Contracts and How Do they Work?* Retrieved from kaleido.io: <https://kaleido.io/how-do-smart-contracts-work/>