DIGITAL ASSETS TRANSMISSION BETWEEN ORGANIZATIONS: MUSIC INDUSTRY USE CASE

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
In Partial Fulfillment of the Requirements
For the Degree of Master of Science
In the Department of Computer Science
University of Saskatchewan
Saskatoon

By

MARCO ANTONIO MAIGUA TERÁN

PERMISSION TO USE

In presenting this thesis/dissertation in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis/dissertation in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis/dissertation work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis/dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis/dissertation.

Requests for permission to copy or to make other uses of materials in this thesis/dissertation in whole or part should be addressed to:

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

ABSTRACT

This research addresses the following experiences as a contribution to the topic of Blockchain applications. First, the modeling of a Music Industry revenue distribution problem. Second, the Integration of Blockchain platforms and Legacy software. Third, the design of an algorithm that solves the distribution of Digital Assets across organizations within the Music Industry. Ultimately, the analysis of the Performance of Blockchain platforms (Ethereum and Hyperledger) in terms of Latency and Throughput. Additionally, the purpose of the research is to show that the modeling of a Music Industry payment system is possible using Blockchain Technology. Therefore, the old business model of the Music Industry, which possessed flaws and inefficiencies, could potentially change into a trustless environment benefiting all the participants y paying their contributions instantaneously. Moreover, the necessity of a solution is reinforced by an internship experienced in a MITACS project in conjunction with a company called Membran to design and implement a Blockchain solution that shortens the gap between Spotify and the payment to the Labels and Artists.

The system distributes value by automatically calculating payments whenever the Digital Assets (Music Tracks revenue) are imported. The application defines specific roles and variables to simulate the Music Industry. For example, Distributors as an entry point and Artists at the end of the chain. Although, any participant within the network can create agreements and benefit from the distribution.

The implementation of this research took the Hyperledger Composer framework to use the Hyperledger Fabric Blockchain as the Private Distributed Ledger, and the public Blockchain Ethereum with the Ganache Client for development purposes. Extensive research of the strengths and weaknesses of these technologies included the descriptions of features like the consensus algorithms, modular architectures, and smart contracts.

Ultimately, the performance of these technologies compared Hyperledger Composer and Ethereum in terms of Latency and Throughput. The conclusion of this research pointed that Hyperledger Composer with features like the role-based architecture for applications, Programmable ChainCode (Smart Contracts), and Business Network Definitions, is better suitable for modeling customized solutions and outperforms Ethereum in terms of performance when testing the same number of transactions, the same logic of the chain code and the same machine environment.

ACKNOWLEDGEMENTS

I want to express my most profound appreciation to Dr. Ralph for all his support and his interest in my career towards a future of great collaboration and giving to the academia and the human knowledge in general. Thanks to my family's support with which I could have never been the person I am today and thanks to the creative expression of music to give me the ideas that I hope, could help to push forward the human spirituality and achievement for the following generations.

CONTENTS

PERMISSION TO USE……**i**
ABSTRACT……..**ii**
ACKNOWLEDGEMENTS……**iii**
TABLE OF CONTENTS ……...**iv**
LIST OF TABLES ……**vi**
LIST OF FIGURES ……..**vii**
LIST OF ABBREVIATIONS ……**viii**
CHAPTER 1: INTRODUCTION ..……1
CHAPTER 2: PROBLEM DEFINITION ……4
CHAPTER 3: LITERATURE REVIEW……10
3.1 Definition of Blockchain ……...10
3.1.1 Transaction Tracking and Digital Asset Transmission……...11
3.1.2 Smart Contracts……...11
3.2 Generations and History of Blockchain……12
3.3 Bitcoin……13
3.4 Use Cases and Applications of Blockchain Technologies……15
3.4.1 New Decentralized Economies……...……16
3.4.2 Social Change……18
3.4.3 Land Registration……...…….....19
3.4.4 Blockchains in the Entertainment Industry……20
3.5 Limitations of Blockchain Technology……...……..……21
3.6 Benefits of Blockchains Technology……22
3.6.1 Audit Systems and Data Transmission Automation……23
3.6.1.1 Data Preservation and Integrity of Data……...24
3.6.2 Privacy……25
3.6.3 Security……26
3.6.4 Business Model Abstractions and Decentralized Autonomous Organizations (DAO)……...26
3.6.5 Efficiency……27
3.6.6 DBMS vs. Blockchains……...…27
3.7 Table of the Scope of the Research……27
CHAPTER 4: ARCHITECTURE ……..29
4.1 Layers of Abstraction……29
4.1.1 Machine Consensus Layer……...29
4.1.2 Application Layer……29
4.2 Hyperledger Project……31
4.2.1 Consensus Algorithms……32
4.2.1.1 Types of Consensuses……..32
4.2.1.2 Permission Voting Algorithms……33
4.2.1.3 General Process Flow of Hyperledger Project Consensus…...34
4.2.1.4 Which Consensus to Apply?.....35

4.2.2 Hyperledger Fabric……..35
4.2.2.1 Hyperledger Fabric Architecture ……36
4.2.2.2 Hyperledger Fabric Consensus Mechanism……36
4.2.2.3 Docker Containers……38
4.2.2.4 Hyperledger Composer ……..……..38
4.3 Ethereum Blockchain……38
4.3.1 Ethereum Philosophy and Concepts……....39
4.3.2 Process of Ethereum Virtual Machine Mining……42
4.3.3 Ether Mining……43
CHAPTER 5: IMPLEMENTATION……...45
5.1 Definition of Environment Variables……....45
5.2 Workflow of Data……...46
5.2.1 Balances of Users……47
5.3 Distribution Algorithm……...48
5.3.1 First Distribution……49
5.3.2 Recursive Distribution……52
5.4 Limitations of the Algorithm……..54
5.4.1 Scenario 1: Two sources of revenue to one node……54
5.4.2 Scenario 2: Two sources of different revenue lengths……55
5.4.3 Scenario 3: Infinite loop failing to evaluate branches……...…..56
5.5 Hyperledger Fabric……57
5.5.1 Architecture of HF Software Components……..57
5.5.2 Hyperledger Fabric Components……58
5.5.3 HF Data Modeling……...60
5.6 Ethereum……62
5.6.1 Architecture of Software Components……62
5.6.2 Ethereum Data Modeling – Solidity……65
5.7 Web-Application-Front-End…….…….....68
5.9 Development Tools……71
CHAPTER 6: EXPERIMENTS AND EVALUATIONS……73
6.1 Settings of the Reference Experiment……73
6.2 The approach of this research ……76
6.3 Qualitative Analysis……...77
6.4 Data Analysis Implementation Architecture……..78
6.5 Quantitative Analysis……81
6.5.1 Assumptions of the experiment in Hyperledger Fabric……..83
6.5.2 Hyperledger Fabric Results……84
6.5.3 Assumptions for Ethereum Environment……86
6.5.4 Ethereum Ganache Client Results……...87
6.6 Limitations of the Performance Analysis……...89
6.7 Discussions and Conclusions of the Data Analysis……...90
CHAPTER 7: CONCLUSIONS, CONTRIBUTION, AND FUTURE WORK…….…..93
7.1 Summary of the Implementation and the Data Analysis……93
7.2 Where is the current research on blockchain technology?.....95

REFERENCES……...99

LIST OF TABLES

Table 3.1 Scope of the Literature Review and Important Papers……..…28
Table 4.1 Comparison of Permission Consensus Approaches and Standard PoW……..33
Table 4.2 Comparison of Consensus Algorithms used in Hyperledger Frameworks……..34
Table 6.1 Table of the number of techniques that combine with Ethereum Ganache Client...81
Table 6.2 Table of the number of methods that connect with Ethereum Ganache Client……82
Table 6.3 Results of JMeter in Hyperledger Fabric…….84
Table 6.4 Ganache Client Ethereum JMeter Results……87

LIST OF FIGURES

Figure 1.1 The old business model of the Music Industry based on the sales contract of albums and records……2
Figure 1.2 Comparison between the current business model(Streaming Services Hierarchy) of track revenue distribution and the proposed new workflow using Blockchain…….2
Figure 1.3 Music Streaming Industry Process……3
Figure 2.1 Blockchain Approaches for the Idle Music Industry……7
Figure 2.2 Proposed distribution workflow……8
Figure 3.1 Block of Transactions Mechanism……..10
Figure 3.2 Digital Asset Unspent Transaction Output (UTOX)……..11
Figure 3.4 Emerging Technologies……..13
Figure 4.1 Proposed distribution workflow……..31
Figure 4.2 Generalized Hyperledger Consensus Process Flow……37
Figure 4.3 Average Time for each transaction on a client with different amount of RAM….40
Figure 4.4 Ether token system……..40
Figure 4.9 Ethereum Blockchain transaction list and state ……..……...42
Figure 4.10 Ethereum GPU Miner…….……..44
Figure 5.1 Example of Distribution with agreements …….46
Figure 5.2 General Diagram of Data Flow……..47
Figure 5.3 Logic of the Balance Enabled and Balance Disabled in Track Revenues……..48
Figure 5.4 Distribution Algorithm Workflow……..50
Figure 5.5 Evaluate Receivers Process…….51
Figure 5.6 Last Node Distribution ……..……53
Figure 5.7 Receipt Evaluation Branch-Evaluation Process……..54
Figure 5.8 Scenario 1……55
Figure 5.9 Scenario 2……56
Figure 5.10 Scenario 3……..57
Figure 5.11 Architecture of Hyperledger Solution……...58
Figure 5.12 SOLO Configuration for Hyperledger Composer ……59
Figure 5.13 Software Implementation Architecture of the Back End…….....60
Figure 5.14 BNA file for the Business Definition……61
Figure 5.15 Bash script for installing the BNA file and setup the network…..……...61
Figure 5.16 Architecture of Ethereum Solution……...62

Figure 5.17 Process of deploying contracts in EVM……63
Figure 5.18 Process of deployment the application with Web3……....64
Figure 5.19 Process of deployment of our app in NodeJS using Web3 ……..64
Figure 5.20 Code Snippet of Web3 Events ……64
Figure 5.21 Software Implementation Architecture of the Back End……..65
Figure 5.22 Java Bean Like Solidity Contract Structure……..66
Figure 5.23 Wrong attempt for importing and calling another contract……..67
Figure 5.24 Proper way to import and interact with other contracts……68
Figure 5.25 Login Page Front End……...69
Figure 5.26 List of Tracks Page ……...70
Figure 5.27 Creation of Agreement Page ……70
Figure 5.28 List of Transactions (Receipts Page)……71
Figure 5.29 MVC Relationship……72
Figure 6.1 Performance Experiment Settings of the Research Reference ……..74
Figure 6.2 Comparison of average latency between Ethereum and Hyperledger……75
Figure 6.3 Comparison of average throughput between Ethereum and Hyperledger……..75
Figure 6.4 Average throughput of Ethereum and Hyperledger with varying number of transactions of TransferMoney function ……75
Figure 6.5 Average throughput of Ethereum and Hyperledger with varying number of transactions of TransferMoney function transactions of TransferMoney function…….76
Figure 6.6 Data Analysis of Scenario 1 Source Code……..78
Figure 6.7 Architecture of the testing environment…….79
Figure 6.8 Data Flow of the Data Analysis in Hyperledger Fabric ……79
Figure 6.9 Data Flow of the Data Analysis in Ethereum-Ganache …….80
Figure 6.10 Total Execution Time for both Scenarios using Hyperledger Fabric……...85
Figure 6.11 Average Throughput for both Scenarios using Hyperledger Fabric……..……..85
Figure 6.12 Throughput in terms of Distribution per second for both Scenarios using HF.....86
Figure 6.13 Total Execution Time for both Scenarios using Ethereum Ganache……..…….88
Figure 6.14 Throughput in terms of number of Tx for both Scenarios using Ethereum Ganache……89
Figure 6.15 Throughput in terms of Distribution per second for both Scenarios using Ethereum Ganache……...89
Figure 6.16 Total Execution Time of HF vs. Ethereum in both Scenarios…….....90
Figure 6.17 Throughput in terms of distributions of HF vs. Ethereum in both Scenarios…...91

Figure 6.18 Throughput in terms of transactions per second of HF vs. Ethereum in both Scenarios……..91
Figure 7.1 Publication year of the selected primary papers…….....96
Figure 7.2 Source of the selected primary papers ……...96
Figure 7.3 Geographic distribution of the selected primary papers ……96
Figure 7.4 Publication type …….....96
Figure 7.5 Paper types per year ……...97
Figure 7.6: Summary of the identified challenges and solutions of blockchain …….....97

LIST OF ABBREVIATIONS

AWS	Amazon Web Services
API	Application Program Interface
BNA	Business Network Definition
BFT	Byzantine Fault Tolerance
DBMS	Database Management System
DAO	Decentralized Autonomous Organizations
DoS	Denial of Service
DApps	Distributed Applications
DLT	Distributed Ledger Technology
EVM	Ethereum Virtual Machine
EOA	Externally Owned Accounts
GCP	Google Cloud Platform
GFE	Google Front End
HC	Hyperledger Composer
HCA	Hyperledger Composer Application
HF	Hyperledger Fabric
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
ISO	International Standard Organization
JSON	JavaScript Object Notation
MVC	Model View Controller
OASIS	Organization for the Advancement of Structured Information Standards
SDK	Software Development Kit
PoW	Proof of Work
PoET	Proof of Elapsed Time
PKI	Public Key Infrastructure
RBFT	Redundant Byzantine Fault Tolerance
REST	Representational State Transfer
RPC	Remote Procedure Call
SHA	Secure Hash Algorithm
SBFT	Simplified Byzantine Fault Tolerance
URL	Uniform Resource Locator
UTXO	Unspent Transaction Output

TERMINOLOGY OF THE USE CASE: MUSIC INDUSTRY

The following terminology applies to the Music to guide some of the variables that the application of the research is using.

Digital Asset. - piece of content that is digitally stored in a system and that has value to a community of individuals.
Track Revenue. - Royalties of Artistic Content such as music tracks, music collaborations, etc.
Distributors. -Music Streaming Services such as Spotify, Youtube, and other channels.
Label Record Companies. - Companies dedicated to the professional production of music.
Content Creators. - Artists, content creators, songwriters, producers, and any participant that is involved in the artistic creation of a Record.
Customers. - Listeners of the Track Revenues (Tracks created by the content creators).
Old Business Model of the Music Industry. - model of the music business decades ago, the participants at the top used to be Bir Record Label Companies
The current Business model of the Music Industry. - model of the music business currently, based on streaming services (Distributors).
Copyright Contracts. - Contracts signed by the content creators in order to share the royalties of their creations with the Distributors or the Label Record Companies.
Record. - Considered as all the process that results in a music product which can be used for generating Track Revenues through the Distributors.
Smart Contracts. - In the context of this research, it's simulating a copyright contract for the distribution of Track Revenues.
Data Input. - In the context of the Music Industry, is the Track Revenue.

Implementation variables.

These variables apply just in the Implementation, Data Analysis, and Conclusion chapters.

Asset. - represents the Track Revenue on the overall application.
Traders. - these are the participants of the Music Industry application: distributors, record label companies, content creators, etc.
Agreements. - They represent the smart contracts in the Implementation section.
Emitter/Receiver. - Traders that participate in the distribution of a Track Revenue value. In the context of the implementation, they change roles depending on the position of distribution.
Label. - Represents the record label company
Artist. - represents the content creator.
User. - represents the participant of the Music Industry that uses the application of this research, it can be any trader (participant).

CHAPTER 1

INTRODUCTION

Private and public organizations have been relying on third parties for decades to secure digital asset transactions or data reconciliation across borders. Decentralized solutions such as Blockchain offer new ways to automate processes that depended on trustless third parties for data reconciliation. For example, features between different trading systems depend upon concerns of data transmission, the reliability of audit systems, automation of agreements, scalability, privacy, and security. Our research evaluates the Music Industry. *Blockchain could potentially change the paradigm of the current distribution of Track Revenues (Royalties of Artistic Content) in Music Streaming Services. Distribution of Track Revenues used to facilitate the earnings of Distributors and Label Record companies. Thanks to the approach of this research, it's possible to benefit all the participants in the Music Industry: Distributors, record label companies, content creators, and customers*. Blockchain can help all these actors by automating fast micropayments and smart contracts according to agreements between the participants [1][2][3].

2.1 Old, Current and New Music Industry Business Model

The old business model used to execute in favor of the Record Label Company. The content creator, most of the time, used to sign copyright contracts based on results. If an album or a tour did not achieve the marketing strategy, they had to compensate for the loses with future unpaid labor, undermining the well-being of the artist. For example, Figure 1.1 shows an example of what used to be a typical copyright contract with a Record Label Company; the content creator obtains an advance of 250000 US for the production of a record, with 5 million in revenue, the content creator then remains with -450000 in debt for the following record.

This model still exists but slowly is changing with the digital revolution of Streaming Services (Distributors). Figure 1.2 shows how the current business model of the Music Industry pays the content creators and what Blockchain offers to improve these procedures. The revenue generated from streaming first passes through several participants before finally reaches the content creator.

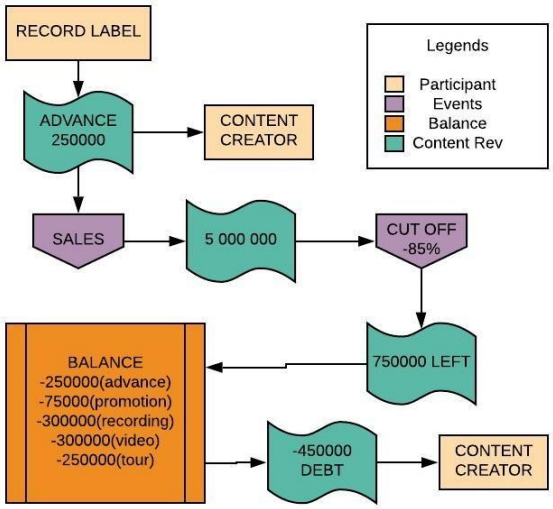


Figure 1.1 The old business model of the Music Industry based copyright contract of records.

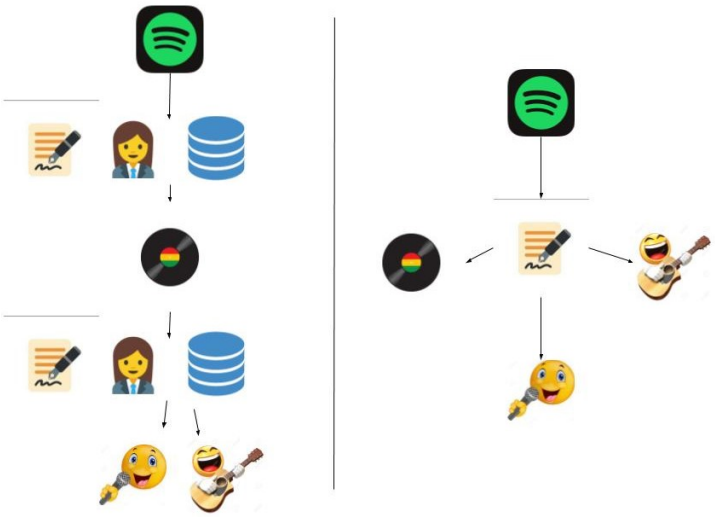


Figure 1.2 Comparison between the current business model (Distributors) and the proposed new workflow using Blockchain-based on Smart Contracts

Nowadays, Distributors are at the top of the hierarchy. Although, the Music Industry does not possess efficient systems of Track Revenue limiting the participation of the content creators: the musicians, songwriters, composers, and producers [2]. Most of these content creators receive late payments or inaccurate amounts of compensation.

Distributors (streaming services) have become one of the most significant sources of revenue for the Music Industry (i.e., Services like Spotify, Youtube, or Pandora). Spotify is currently the current biggest distributor, and its impact on the Music Industry is positive in terms of sales, market fragmentation, and copyright royalty's distribution. Datta, H., Knox, G.,

and Bronnenberg, B. J. [5] suggest that the fragmentation of the market due to distributors like Spotify is relatively favorable for content creators and record label companies. Customers (listeners) tend to follow fewer superstars and expand their attention to a broader set of content creators, which could potentially increase the Track Revenues in complementary goods and live performances. Although, one of the tradeoffs of this new Current Music Industry model trend is the increase in efforts of content creators to stay on top by satisfying the demand. On the other hand, Marshall, L. [6] suggests that distributors could harm even more the Current Music Industry despite the market fragmentation. *Distributors could consolidate long-established power structures*.

Additionally, the current process of royalty's distribution is slow and inefficient. The revenue generated by the distributor's services passes over a network of aggregators and other third parties. Figure 1.3 shows a diagram of how the streaming services pay royalties to the content creators, bypassing several participants in between. The process needs repetitive intermediaries who execute mainly the reconciliation of the databases. This process could take 1 to 2 months to pay all the participants for their contribution. It takes time since the involvement of audit systems with different architectures, producing duplication of data in some instances. Richardson, J. H. [7] argues that the changes in rates of royalties and copyrights emissions are slow by nature and diminish new distributor's innovation efforts. For example, the copyright issuers and license negotiations take considerable large amounts of time to classify the source of the copyright, either as a "musical work" or "sound recording" rights. All changes in distributor's services must pass over paperwork processes, different databases, and various licenses emissions before it's authorized to stream. After that, the payment process time doubles for the artists to get paid [6]. Further, Spotify reported having lost 69$ million in 2011 due to copyright procedure inefficiency [7]. Ultimately, royalty's rates and distribution of copyrights depend upon legislation which constitutes at the same time a government-based monopoly.

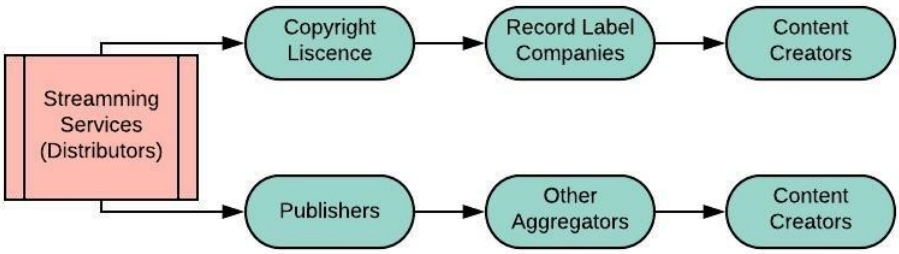


Figure 1.3. Music Streaming Industry Process. (Rethink Music Initiative, 2015) [45]

In effect, the participants on the bottom of the chain (content creators) don't have access to their data, or they lack the authority to do so, disabling them the possibility to make a profit out of this data. The intermediaries gather most of the revenues by creating paper-based agreements between participants to deal with conflicts of interest [1]. Consequently, to satisfy the demand, content creators require more computational tools for data analysis, copyright royalty's immediate distribution, and distributed solutions to plan new strategies. Blockchain technology could potentially simulate the copyright contracts with smart contracts between participants automating the third-party intervention (legal actors) and securing the integrity of the data to avoid fees usually charged audit systems. This research tries to address the following question: *An environment, where all the participants are paid instantaneously for their contribution to the network, can be designed and implemented with blockchain technologies and their integration with legacy software?*

The following sections present our proposal for the design of this new Music Industry model with specific third-party legacy software and current frameworks of blockchain technologies. Therefore, the research organizes as follows:

- Chapter 2 introduces the problem statement, its context, its requirements, its brief architecture explanation, and a brief overview of the data analysis.
- Chapter 3 shows the Literature review, the previous background knowledge on the theory of decentralized systems and blockchain technologies, the underlying architecture of the techniques, and the last academic attempts to compare the technologies' features and performance. Additionally, in this chapter, we present some insights into the opinions of other sectors over Blockchain technologies, some of the history and evolutions of these topics, and the future predictions of the implications among some other sectors.
- Chapter 4 presents the Architecture of the system. This section describes the underlying theoretical machine functioning of both platforms. Additionally, this chapter links the problem statement with the design of the application.
- Chapter 5 shows the Implementation of the proposal. It describes the details of the algorithms, logic, technologies, and frameworks used to address the problem statement. It also describes the reasons for the chosen technologies and the challenges and problems using those technologies in the development cycle.
- Chapter 6 shows the Data Analysis. This chapter discusses the data charts that the experiments threw. Additionally, the payload comparison between the two platforms gives insights over the strengths and weaknesses of both platforms

(Hyperledger and Ethereum). Lastly, this chapter exposes the problems of the implementation that were not conceived and are inherently part of the new frameworks´ development, as these technologies are in their early stages of exposure.

- Chapter 7 discusses the Conclusions and Future Work. Additionally, this section compares the requirements of the problem statement and the outcomes of this research revealed, to determine the success of the study in solving most of the planned goals or the reasons why it failed, and what can be improved in the future.

CHAPTER 2

PROBLEM DEFINITION

Distributors like Spotify represent a possible centralized power structure since the high number of aggregators that must be passed to reach the participants at the beginning of the creative process (Figure 1.2) creates complexity, latency, and low performance. *This research addresses the challenge of creating a nonhierarchical fast distribution of digital assets that integrates Legacy software in two well-known blockchain platforms (Ethereum and Hyperledger), one public, and the other private.*

A digital asset is a piece of content that is digitally stored in a system and that has value to a community of individuals. Over the years, this concept has been changing and nowadays includes photos, videos, documents, etc. In the Current Music Industry, a digital asset could mean an audio file, copyright, an artwork, an artist profile. *Although, this research defines a digital asset as the TRACK REVENUE (contemplated in token currencies or monetary value) generated by the songs and artistic work of the content creators.*

Figure 2.1 shows the Current Music Industry based on streaming services. On the left, the streaming-based approached is hierarchical. In each step, there must be a data reconciliation between different participants. Therefore, the Current Music Industry is inefficient and unreliable. After that, there is an idle scenario where Blockchain serves as a public ledger where the user pays the content creator immediately. This system is ideal since the bridge between the customers, and the content creator is almost nonexistent, and the intermediaries receive their contributions but are not an authority anymore. Although, the Industry is behind the ideal model for different factors, mainly adoption and education. Consequently, there must be an improvement in between the Current Business Model and the Idle Business Model. The proposal then is to achieve a system where the data flow has the same structure, but the data reconciliation is almost instantaneous, paying the content creators immediately what he/she deserves and lowering the fees of intermediaries.

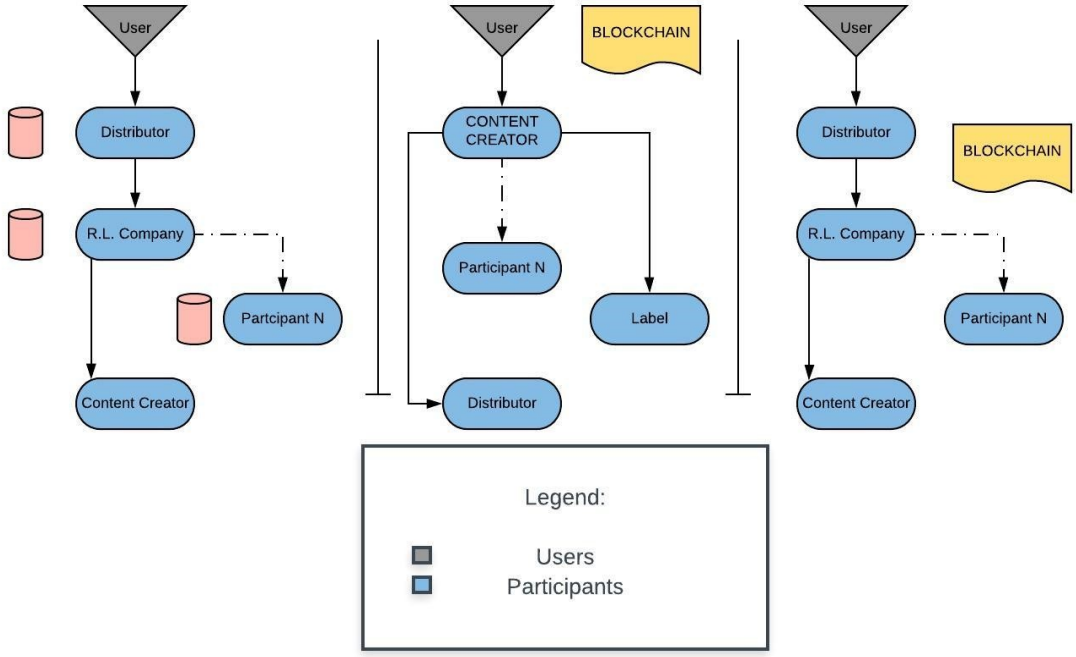


Figure 2.1 Blockchain Approaches for the Idle Music Industry

Figure 2.2 shows the details of the project to solve the inefficient hierarchical data reconciliation system (Distributors based) (Figure 1.2). This picture presents the distribution of the Track Revenue, which represent the *digital asset*, through a system that passes the borders of participants in the Music Industry. The process is straight forward, but the difference is that is powered by Smart Contracts (more information in chapter 3), and Legacy systems that interact with an underlying blockchain environment.

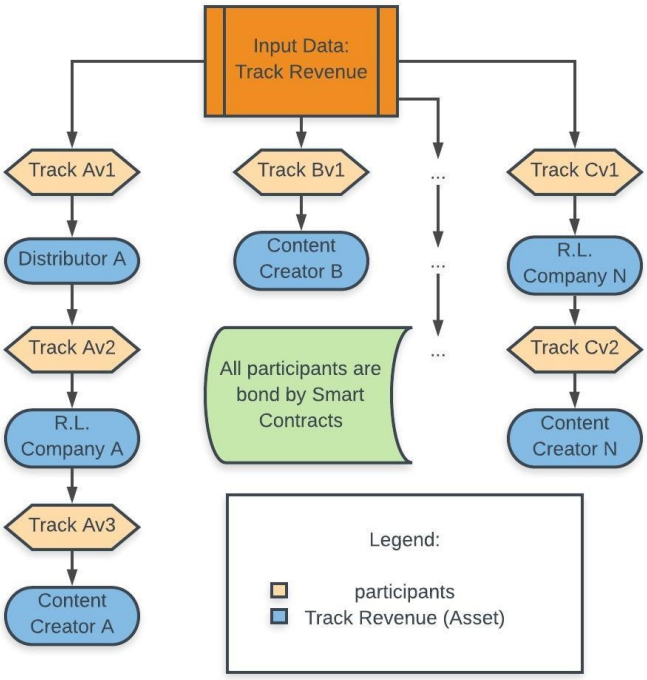


Figure 2.2 Proposed distribution workflow. All participants can be part of the distribution chain if they have Smart Contracts.

Figure 2.2 also represents the possibilities of the flow of compensation payment in a Distributed Ledger. Each branch (like the branches of a tree in Figure 2.2) represents a different *Track Revenue* distribution, and each branch can be shaped by any number of participants with any combination possible as long they have Smart Contracts.

In this design, the *digital assets* (Track Revenue), the Smart Contracts, and the information of the participants can interact with different parties securely and trustfully thanks to a Blockchain environment. Additionally, using a Blockchain as a common ledger of smart contracts, audit processes are met, *and with Smart Contracts, the system automates operations to avoid third party involvements that could delay the efficiency of the data flow*. Consequently, in this environment, the Track Revenue royalties are automatically transferred to the participants when the data input is imported, so all the participants receive their compensation instantaneously thanks to the Smart Contracts. The participants can feel trustful over the network because the underlying technologies use cryptography and consensus protocols so they can dedicate their roles more efficiently.

The implementation of the solution included a *Distribution Algorithm* as the core of the solution for agreement-based systems. The research proposes a workflow with different

Scenarios or cases possible according to the data input. These Scenarios are the guidelines the Smart Contracts must follow. Therefore, each platform uses the same algorithm independently of their programming languages, frameworks, and conditions. The Implementation Section presents two basic Scenarios that model most of the probable behaviors of the data workflow.

The solution to this research involves two of the most well-known blockchain platforms. The first is Hyperledger Project powered by the Linux Foundation and the second is the open-source, public Blockchain Ethereum. Both platforms have strengths and weaknesses according to the requirements of the implementation. Additionally, the system should possess the capacity of integration with Legacy Systems.

The Data Analysis tests the Performance of the system for obtaining conclusions of Scalability in terms of Latency and Throughput. *The Qualitative Argument states that Hyperledger should have a lower Latency and Higher Throughput than Ethereum since Hyperledger is more centralized and private. Therefore, Hyperledger overcomes Ethereum in performance for being a private blockchain.*

Therefore, after analyzing the problem statement, it's crucial to mention that ***this research contributes to the literacy of solutions that accelerate the payment of music track revenues to all the participants in the Current Music Industry based on Streaming Services (Distributors)*** and that the main goals to achieve this contribution can be summarized with the following points:

- Develop a Distribution Algorithm that solves the possible Data Flow Scenarios.
- Integrate blockchain platforms (Ethereum and Hyperledger) with Legacy Systems (Servers, middleware technologies, and front-end technologies).
- Execute experiments of Performance for Scalability conclusions in terms of Latency and Throughput.
- The development of Blockchain platforms (Ethereum and Hyperledger) as a common ledger for the computation of Smart Contracts to solve data reconciliation between different databases. Although, the metadata and media files will not be stored in the blockchain platforms.

CHAPTER 3

LITERATURE REVIEW

This section presents the academic background of previous research made on Blockchain technologies. Some topics that will be covered are the history and definition of Blockchain, a necessary detail of the algorithm that public Blockchains use, the interpretation of smart contracts, the implications of this technology on the economy and the social organizations, the advantages and disadvantages of the technology, the disruption in the music industry and previous attempts to analyze the performance these technologies Blockchains.

3.1 Definition of Blockchain

Blockchain is a distributed structure that contains the information of all transactions ever occurred in the past, securely encrypted, double spending problem-free. These transactions interact in a set of nodes connected via peer to peer networks [35]. This structure doesn't need third parties and therefore achieves levels of security in trustless scenarios.

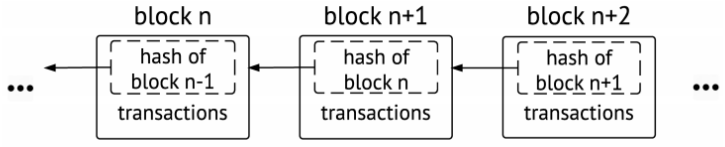


Figure 3.1. Blocks of Transactions. Each block carries a list of transactions and the hash of the previous block [29]

Figure 3.1 shows the process of block production in Blockchain. Each block references the hash of the last block [29]. Therefore, there is a link between blocks, and this link enables all blocks with the permissions to read the world state data.

A blockchain network has a set of nodes (clients) that interact with the blockchain with a private and a public key. Usually, the private key is for signing their transactions and the public key for their identification. These keys work with asymmetric cryptography to be able to be validated. Once that a transaction is approved, it broadcasts to the other peers. The other peers verify the successful transactions, and invalid transactions are discarded. Eventually, the transactions spread all through the network. The validated transactions are

collected and ordered during a time interval and packaged into a timestamped candidate block. Under the design of the first Blockchain Bitcoin (Proof of Work consensus), this process is called Mining (Implementation Section). When the block is completed is broadcasted to the network. The other peers must validate it and then it is referenced by hash with the previous correct block. If everything is completed correctly, the block is added to the chain and apply its transactions to the World View or World State. If it's not the case, the candidate block is discarded. Then the process starts again.

3.1.1 Transaction Tracking and Digital Asset Transmission.

Digital Assets can be transferred from one account to another in Blockchain. Under the principles of the first Blockchain, each account is just a hash value, a unique address. The mechanism that enables the transfer of digital assets is called Unspent Transaction Output (UTXO). Figure 3.2 shows the mechanism of a digital asset transaction.

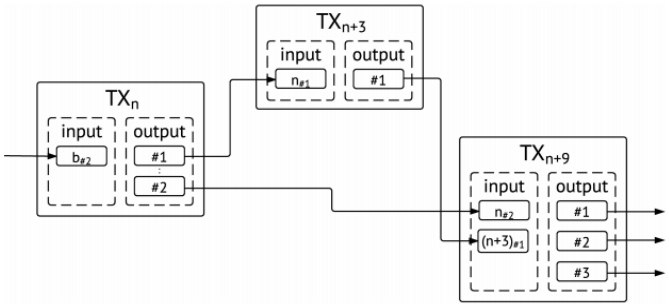


Figure 3.2. Digital Asset Unspent Transaction Output (UTOX). The transaction n spends the second UTXO that transaction b (not pictured) created (b#2), and generates two new outputs (n#1, and n #2), which again are spent by transactions n+3 and n+9 respectively [29].

The ledger adds new rows that represent new values in new accounts and deletes previous transactions that represented former states. Not yet removed rows are called UTXO inputs. The new rows are called UTXO outputs [29].

3.1.2 Smart Contracts

Nick Szabo first introduced the concept of smart contracts as a protocol that executes the terms of a social settlement. It means that hardware could run the contractual clauses of a human agreement under certain conditions. Therefore, because the hardware or any

technological platform is self-executed, the system doesn't need the intervention of an intermediary [28]. Although, Ethereum blockchain was the first attempt to improve the blockchain capabilities from the first Bitcoin version.

A smart contract is like a real contract, a set of rules and agreements on which a transaction or an activity will be executed. Ethereum offered an entire programming language allowing complex scripts to set the rules on which the blockchain will work [9].

Additionally, smart contracts have a unique address, such as the wallets (accounts) in Bitcoin. The nodes constitute a Virtual Machine that runs on every computer. Usually, an intelligent contract should show all the possible outcomes of its execution and should be deterministic. Therefore, the same input will always produce the same output. On the other hand, non-deterministic smart contracts (different random results with the same data) cannot secure consensus, so the security levels increase thanks to the smart contracts [9].

3.2 Generations and History of Blockchain

Blockchain has evolved to become a trend in computer science research. Although, through the last decade, essential improvements over all the tendencies of technology have shaped the way the technology is evolving. The following section presents a small analysis of the evolution of blockchain.

Generation 1 introduced Bitcoin and the concept of Cryptocurrency. With the rise of the popularity of Bitcoin, Internet cash became a feasible idea. The advantages of low transaction fee cost, anonymity, and transparency became attractive to the first adopters. Meanwhile, the problems of latency and scalability started to become apparent. In the **2nd Generation** of Blockchains, Ehtereum and Smart Contracts expanded the application range of Bitcoin by providing an environment of customized programmed sets of instructions powered by the same principles of Bitcoin but subjected to a broader range of solutions. **Generation 3.0** started to present DApps or decentralized applications with a growing number of Legacy software integrations in blockchain networks. Some of the featured combinations are data storage, communication networks, smart contract's communications, and open standard platforms. In the 3rd Generation, Delegated Proof of Stake (DPoS) took traction in the developer's community, and open-source libraries started offering ways to interact with old and new blockchain frameworks [46]. Lastly, **the 4th Generation** is the seamless integration of Blockchain with the Industry. Nowadays, there is a current switch of

the old business models for decentralized solutions. Additionally, enterprises are building private platforms for privacy prevention in blockchain environments. Industries are licensing ambitious projects that might have decentralized features but wouldn't ultimately be identified as blockchains. For example, SKYNET is the world's first IoT chip that could replace the modern CPU with a chip optimized for the Internet of Things and Blockchain.

As a side note, Blockchain reached peaks of academic and non-academic attention in 2017. New businesses have started using this technology in different use cases, especially in the FinTech sector [1]. The demand for legacy software in the development of blockchain increases. Since its first application (Bitcoin) appeared in 2009, Blockchain research has evolved integrating new machine communication protocols, consensus algorithms, encryption protocols, and architectures, to become one of the trends of technology nowadays. According to Singh & Singh [2], Blockchain Technology was explored at an unprecedented speed in 2017. Additionally, Figure 3.4 shows the trends of 2016 in technological innovation according to the Gartner Hype Cycle.

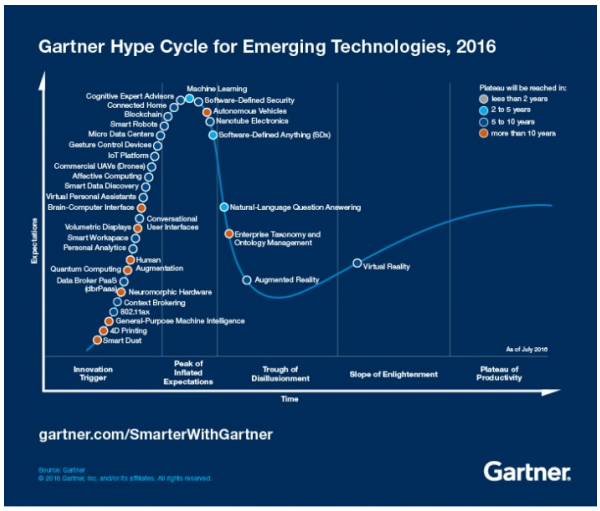


Figure 3.4 Emerging Technologies, 2016, by Gartner Hype [2]

3.3 Bitcoin

Around 2008, the financial services suffered a big crash. The stock market plummeted affecting thousands of people and affected the global economy. Meanwhile, several cryptographers were working on new future solutions for current financial services. Some of those solutions involved cryptography, decentralized consensus, and new protocols of

communications. A group of cryptographers and hackers called by themselves the "Cyberpunks," tried several approaches like "BitGold." Some of those attempts were implemented but didn't reach enough attention from the current market.

After the financial services went down because of economic bubbles, a paper written by an anonymous individual or group of individuals called *Satoshi Nakamoto* was released on forums of cryptographers. This paper was presenting an alternative method of payment that united the strengths of the math, the computer power, algorithms of consensus and an entirely new decentralized architecture that combined all the attempts that the "Cyberpunks" tried and didn't succeed. One of the main goals of its author or authors was written in the abstract at the beginning of the paper:

"A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution" [8] .

Bitcoin pretended to replace protocols of transactions in digital currencies instead of human third parties. It means that with this digital currency technology, no government could determine the interest rate of the operation based on arbitrary factors such as politics (causing inflation), but the transactions would be determined by an algorithm [9].

Bitcoin, as a cryptocurrency, offered no inflation because it's finite (There is 21 million in total). Although some people started arguing Bitcoin could lead to deflation, it can be divided into around two quadrillion units, and this characteristic raised the comparison between Bitcoin and Gold. The fact that Bitcoin is not controlled by banks or governments and is public or incorruptible attracted people's attention to the tech space, economists, researchers, private entities, hackers, and criminals. Although several organizations remained opposed to its use due to its anonymous and illegal use, people were still using it until its highest peak of attention and value in December 2017.

Additionally, tax regulators were afraid that cryptocurrencies could lead to tax evasion. Therefore, several governmental institutions started conversations over the financial regulation of Bitcoin [10].

Crime regulators have argued that Bitcoin can lead to money laundry and fraud. For example, the anonymous website "Silk Road," an online market organized around illicit goods, was shut down by the FBI in 2013. The creator of Silk Road William Ulbricht was

judged in New York, the heart of the financial services to warn the use of bitcoin as an illegal medium of exchange.

According to Baravalle, Lopez, & Lee [11], drug-related items constitute around 80% of the size of the total market of illicit online markets. According to N. Christin [12], approximately 4.5% -9% of the Bitcoin Economy moved through this Silk Road. Moreover, research on Agora, another anonymous online market, showed that 170691.12 bitcoins (about 27 million US Dollars) of merchandise were on sale on the period under examination. Over 30000 products were on sale, and 1233 sellers participated in the market in 29 countries [11]. Ultimately, it is estimated that Deep Web hosts 500 times more content than the Web [13].

Bitcoin start-ups and legitimate Bitcoin businesses have suffered several attacks and hacks. Mt Gox (Tokyo), a Bitcoin exchange company went bankrupt because of an attack on the system in 2014 [13]. Ethereum blockchain also was hacked due to some flaws in the funding of the organization, leading to a split in two versions, leaving behind the system that was compromised [9].

According to Baravalle and Lopez [11], users of bitcoin are driven by anonymity and by political views. Most of its users identify themselves as *libertarians*. Some of them could support ideas like: "*Taking away the power of the money supply from centralist states*…." (The Netherlands, 27 years old) [11].

Ultimately, the community analysis shows the mindset or the incentives of the people that use anonymous blockchains such as Bitcoin. Current research trends suggest that blockchain can have a run impact on areas of global development other than FinTech.

3.4 Use Cases and Applications of Blockchain Technologies

Blockchain is a subcase of Distributed Systems or Non-Centralized Networks or better called Distributed Ledger Technology (DLT). DLTs represents a stack of technologies that let organizations utilize the security, reliability, and smart contracts under an entire programmable platform. Even though Distributed Systems have been developing since decades ago, the first encrypted consensus "Proof of Work" of Bitcoin revolutionized the possible applications this technology could leverage among several industries. Among some of its potential applications already identified are land registration systems to prevent corruption of governments, future integration with the Internet of Things, the track of goods in Supply Chain Industries, Distributed Autonomous Organizations (DAO), cryptocurrencies,

and everything that needs a third trusted party to complete a process. Blockchain can remove it from the equation [2]. This section shows the possible solutions that Blockchain offers to several spaces.

3.4.1 New Decentralized Economies

Bitcoin reached a price value of almost 20000$ per bitcoin in 2017. It has caused unprecedented debate among the most well-known investors, economists, and scientists. Several countries are taking serious attention to the implications of Bitcoin in current currency policies and economic analysis. For example, Vietnam Central Bank is seriously studying the possibility of using Bitcoin. European German Central Bank and Bank of England are investigating the option to issue their cryptocurrency [14]. Sweden Central Bank might use a digital cryptocurrency within the next couple of years to prevent inflation and achieve stability [14].

Additionally, cryptocurrencies lead to think less about physical cash systems as technology evolves, targeting future cashless societies. Some northern countries, such as the Netherlands are studying this future possibility. Tom Lee, an analyst from Bloomberg, says that “Once the market capital of cryptocurrencies reaches about 100 billion dollars, central banks will start buying cryptocurrencies by themselves” [14]. Currently, the market capitalization of the cryptocurrency market is around 278 billion (all cryptocurrencies). It reached 795 billion (all cryptocurrencies) in September 2017 [15].

Although the media, academic literature, and public interest of Bitcoin have only started to emerge since 2009 [16] [17], the debate has created a new wave of interest in Econophysics, defined as the use of paradigms and tools to several theories and statistics to determine economic models [18]. On the other hand, research conducted by Fry John and Cheah Eng-Tuck [19] concluded that Bitcoin has several characteristics of a negative economic bubble (raising concerns about its long-term sustainability). Econophysics of cryptocurrencies is related directly to the external factors, such as hacks and government intervention. Therefore, these researches open an entirely new field of study in future economics for predicting stock markets in real-time. Consequently, Blockchain is opening the possibility to do academic research on global scale economic behaviors.

Negative bubbles are not the only concerns of cryptocurrencies such as Bitcoin. Unprecedented attempts of frauds and hacks are rising within this new industry. For example,

new ICOs (Initial Coin Offerings) are starting to offer promises of entirely new business models, including Blockchain technologies. Investors have now the option to buy and sell value over these start-ups and ICOs directly without the middleman. For example, companies like Binance or CuadrigaCX, are platforms where investors, from a wide range of capital sizes (from professional investors to average internet users), now can trade stocks of ICOs in a live trading platform. These new coming industries are as open and transparent as they can, but there have been already several attempts of frauds or hacks worth millions. The average bank loss of hacking attacks is 1.5$ million. Although, ICOs and crypto exchanges start-ups suffer an average of 2$ billion in damages. The “fishing technique” (social engineering hacking over emails or websites of any contact with the employee of the company) is the most common form of funds theft [9]. Therefore, regulators have started to intervene in the newly forming crypto industry to prevent fraud. The regulation actions move from ignoring ICOs to ban them altogether, depending on the nature of the token and the country. According to Eyal, I., [9], from September to November in 2017, countries like China and Estonia altogether banned ICOs, and countries like Canada, Singapore, Hong Kong, are discussing the regulations to let them operate.

New start-ups bring the possibilities of innovations in data analysis. For example, Binance offers a public API to access in real-time to the cryptocurrency market as a service. Open-source innovations open a new whole of possibilities for businesses and academic research. Moreover, APIs like Binance Analysis open new fields of software development for real-time market analysis and crypto investments. Additionally, developers and entrepreneurs are creating secure platforms to exchange assets digitally. For example, Coinbase and GDAX are part of the same company that operates in California USA and currently is the most know application to buy and sell bitcoin and other cryptocurrencies in the USA.

Despite these attempts for bringing Blockchain to the current business, academic, and non-profit solutions, most people in developing and developed countries don't yet understand the intrinsic value that digital asset representation hold. The main problem resides in understanding the underlying technology and trust in a technology that has been used by hackers and anonymous traders of illicit goods.

Understanding Bitcoin needs an understanding of the mechanisms of money. The mechanism of money distribution has always relied upon trusted third parties that manage its functionality. Money evolved from thousands of years since the use of shells, jewelry, gold and lastly cash and credit cards. Money is an asset that people give value to, as long as it has

some intrinsic functionality, and it's scarce. Although, cash has been subjected in recent years to inflation due to different economic and political environments in different countries. Government and banks are failing in addressing trust in the exchange of national currencies, causing current currency prices to inflate and hyperinflate, such as the case of Venezuela or Zimbabwe. Organizations and individuals depend on a trustful third party to execute any process. Blockchain can achieve a new age of innovation in this area of human interaction.

3.4.2 Social Change

In March 2018, Sierra Leone Presidential elections were supported by Agora, a Swedish blockchain-based foundation that developed a system for transparent voting in one of the poorest countries in the World. Ebola, corrupted elections and GDP losses of $1.4 million since 2014 have affected Sierra Leone development deeply. Blockchain made it possible to offer locals digital tools for transparency. Agora's CEO Leonardo Grammar clarified that the Sierra Leone voters were opened to the experiment and that the company is pointing to repeat the operation in other countries [20].

Digital money is not the only representation of a digital asset; blockchain can also design a digital image of anything that has value, such as land titles, intellectual property, private data, copyrights, etc. Blockchain has unprecedented implications not just over the economy, but also the social and political structures of societies. Don and Alex Tapscott (2016) mention several implications of blockchain technologies in social structures. For example, remittances from developing countries are the highest flows of money in quantity, more elevated than foreign investment and aids [21]. Distributed ledger technology could add billions of individuals to the global economies using exchanges of value from peer to peer since current financial systems exclude them. Existing financial systems think they would not be profitable or be risky in any business transaction. Blockchain can reduce the fees significantly, improve the efficiency of this process, including people in the global economy, and create new opportunities for independent entrepreneurs.

Equally important, corruption over the global financial aids of world organizations is a big issue in countries like Haiti. In 2010 it suffered one of the deadliest natural disasters in history. Foreign governments sent financial aids to help rebuild Haiti from the earthquake, but investigations led to conclude that part of the 500$ millions of Red Cross funds was missed or stolen. Blockchain can improve the delivery of these aids eliminating the

bureaucracy from the equation [21]. Blockchain someday could reduce the corruption of authoritarian governments and unfair public institutions.

3.4.3 Land Registration

Fernando De Soto, a Peruvian economist, suggests that many as five billion people in the world cannot participate in the economy because they don't have strong legal rights over their properties such as land. Blockchain can improve the land registry by giving final proof of immutable records [21].

Land registration is one main component of economic and social growth, although the registration systems are broken in some counties. Fernando De Soto argues that because of the involvement of a third party, the tracking of ownership of properties on the poorest countries, suffers time-wasting, inefficiency, and the corruption of public institutions. Since the users involved in this process of asset exchange cannot rely on the system, millions of people in countries such as Honduras cannot get the benefits of the global economy [21]. Additionally, Lemieux [22] argues that untrusted civil registration could be an obstacle for accessing social protection benefits and could open the door to fraud undermining the developing country's immigration policies and national security and the same for its relationships with other countries [22].

Public notary institutions are an essential part of the legalization of any transaction with property or assets in developing countries. These institutions are responsible for civil registries of births, deaths, marriages, land registration, and repositories of financial transactions. Some states or cities do not have enough resources on providing the legal liabilities to be able to give its citizens options to invest, exchange, sell, or any economic activity. Other related activities of land registration or the real estate industry include claims to citizenship, land, and social protection. Registries are everywhere. As Daniel Novy from Consensys (Ethereum) explains, some of the essential goods such as titles or properties are controlled by governments which because of its authority, can provide authenticity to those records. Although, the problem arises when there are two copies of the same asset. The only way to know which document is authentic is by asking the owner. In Brazil, the amount of the number of real estate registries in Sao Paulo city ascends to 18, and all those registry offices are not unified. Public blockchains such as Ethereum could solve this problem. First, thanks to the immutability, the published record cannot be removed once that is submitted on

the network. Second, the system could be designed as a mobile, user-friendly, and digital stamped (digital signatures) so that way registration and related asset actions could be as easy as sending an email. Third, there is no central point of failure since the architecture is decentralized. Fourth and last, the cryptographic security and the Proof of Work algorithm for consensus prevent cheaters. Ethereum and other public blockchains can transform the real estate industry and social impact for transparency [21].

Ultimately, attempts of implementation of digital records on real estate goods started to be implemented already. Honduras has more than two-thirds of the population living in poverty, and five of ten citizens suffering extreme poverty. In November 2015, the government approached Factom to discuss the implementation of recordkeeping problems associated with its land registration system. Factom is a blockchain-based solution using the Bitcoin blockchain [23].

3.4.4 Blockchains in the Entertainment Industry - Problem Statement Overview

The content creators can benefit significantly from blockchain technologies since the previous business models of the industry undermined them. For example, Napster was a game-changer in the music industry by giving access to the listeners (customers) freely the music of their favorite artists. By violating the copyrights, Napster reached a broader volume of listeners changing the previous business model based on contracts with labels for the distribution. Consequently, online distributors like Apple and Spotify left the content creators at the lowest level of the chain of profit, although reaching the new generation of listeners in smartphones and other devices [24].

Currently, new industries such as music publishing and recording services have emerged as the new intermediaries to bring the music to the ears of the final user. Although, with each intermediary added in the chain, late fees maintain the business model giving the least royalties to the creators of content. Some of those identified intermediaries are labels, publishers, distributors, performance rights organizations, organizations to monitor performance royalties (i.e., American Society of Composers, Authors, and Publishers), producers, venues, concert organizers, promoters, wholesalers, agents, and accounting systems [4]. Moreover, most distributors such as Spotify or Youtube give the creators an even smaller percentage of the revenues (i.e., Spotify fives 0.006$ per stream to the authors). Most

content creators sign contracts of full copyright ownership to the big record label companies to take their music to the public, giving all or most of their profits to the intermediary. Ultimately, the intermediary holds a contract that usually gives them more revenue from using the tracks in other ecosystems, giving little or nothing to the original author.

Nowadays several start-ups such as SingularDTV are creating a new fair industry for content creators, and independent musicians such as Imogen Heap are creating their ecosystems of economies by tokenizing their artworks, all of it thanks to Blockchain technologies. Additionally, Imogen Heap and other content creators and entrepreneurs are analyzing other content creators-based business models never seen previously. For example, she is trying to bring all the metadata of the tracks of her songs to manage her image to her fans. She could use all the metadata to control her contracts with other systems, could be sure that her music will be delivered directly to her fans, and could be confident that her fans could be in contact with her directly. The data the content creators carry could be monetized to manage their resources more efficiently [21].

Blockchain can solve both the efficient payment systems for participants in a music industry ecosystem and create the conditions necessary to develop agreements justly between the participants. Smart contracts could replace significant amounts of investments in paper-based copyright contracts by executing Smart Contracts (agreements) that are triggered by inputs between the customers of a Blockchain application. Ultimately, according to those smart contracts, all the participants could receive their compensation from their services according to the rules that everybody agreed to follow in an instant. New business models and new forms to do creative artworks could become a reality on the internet of value.

3.5 Limitations of Blockchains

Before analyzing the public and private blockchain protocols and architectures, its worth mentioning the limitations and benefits of Blockchains and the reasons why Blockchain is better than other current technologies. Similarly, like any other technology, Blockchain has advantages and disadvantages that academics and developers discover and are worth mentioning. For example, research conducted by Eyal [9] expresses several limitations of Bitcoin based on the classical structure of Nakamoto:

Fairness. - Fairness on blockchain technology is one of the vital incentives users have because they trust that any third party can control it. Although incentives do not always

ensure fairness. Moreover, as miners give more processing power, they receive more significant compensation from the system. Therefore, if a miner has enough resources to achieve the 25% of the mining power, other miners can join these “selfish miners” compromising decentralization fairness. For example, they can form blocks of the highest transaction’s value dynamically, so they get more compensation from it. Consequently, Bitcoin has not yet achieved fairness entirely since the miner's incentives are the rewards [9].

Proof of work overload. -Since each transaction made by the users is encrypted and cannot overlap, the miners require to solve complex cryptographic puzzles. The computational power for these tasks consumes large amounts of physical resources in terms of processing power. Therefore, the costs of power consumption are unacceptable for early adopters [9]. Cost consumption is one of the reasons why the FinTech sector and developers have put effort into trying to solve the efficiency problem with new protocols and architectures. Proof of Work protocol will be analyzed later in this research.

Scalability Challenges and Time of Transaction execution. - When new branches of blocks are created, the rest of the blocks must agree to keep adding new ones, so each block needs the approval of the rest of the miners. The block formation time interval should be longer than the rate of block addition in the network to secure the reliability of the block and prevent forks or blocks that succeed in forming different branches. Initial Nakamoto’s Bitcoin design restricts the time of the propagation of blocks to 10 min, but current high volumes of transactions reach more than one hour sometimes due to the size of the network [9]. Therefore, 10 minutes is unacceptable for early adopters.

Lack of confidentiality and privacy of transactions. - The initial design of Nakamoto’s paper was considered as a general and public ledger log to ensure security and fairness. Although, FinTech industries need private transactions to give users the confidentiality of their financial movement. Therefore, the sensitive history of transaction information should be just available to shareholders, investors of employees [9].

3.6 Benefits of Blockchain Technology

As mentioned previously, early public blockchains presented several features that current system solutions cannot tolerate. Therefore, the private sector has invested efforts to explore the blockchain technologies to address such limitations. FinTech Industries are mostly interested in a technology stack called Distributed Ledger Technology (DLT). The

layers constituting the stack are the following: The system client (balance sheets each account has), a virtual machine (accept transactions and translate them into states), a consensus protocol and network layers to determine how the nodes interact with each other. The layers not necessarily target public networks but rather Private Blockchains.

This section will explore all the possible positive outcomes of using this private blockchain in terms of automation, privacy and security, features of decentralized systems, features that attract business and entrepreneurs. Overall, why use Private Blockchains instead of other current IT solutions?

3.6.1 Audit Systems and Data Transmission Automation

Business model applications rely on the efficiency and trust of the data transmission between different accountability systems. Moreover, not just enterprises rely on these communications between systems, but also public organizations and non-profits. From the most profitable business models such as those implemented by companies like IBM or Microsoft to non-profit organizations and entertainment, rely upon systems that secure the data integrity to transact assets, services, and resources among peers and networks with no points of failure.

Additionally, the development and management of audit systems become more and more complicated due to different protocols of communication, security, and frameworks among different systems complicating the data integrity. Blockchain Technology is a breakthrough solution in this field because it promises automation in all levels of abstraction in one single platform securing integrity. Therefore, audit systems are no longer needed because one single ledger is evaluating the inputs and outputs as each transaction it's being submitted. Moreover, this innovation promises a higher abstraction of audit systems speeding up the end goals of an organization. For example, Hyperledger offers a broader range of possibilities of customization of the system over different layers of the DLT. According to the official goals of Hyperledger Organization, a ledger could operate continuously for 100 years or so with the same features of discoverability, identity resolution, and other vital functions [25] .

Ultimately, its decentralized nature based on smart contracts increases efficiency avoiding bureaucracy and intermediaries. It allows translating the environment where these

transactions are made into a platform of audit completely secure by cryptography, consensus, and legacy protocols.

3.6.1.1 Data Preservation and Integrity of Data

Blockchain can solve an audibility problem that most systems lack when dealing with storage of digital assets, the standards of preservation of information worldwide. This subsection mentions basic concepts of digital preservation to push forward the overall digital music industry with the explored technologies for long-term maintenance and reliability of its logic.

Standards of Data preservation

Some research conducted by Lemieux suggests that it is possible to implement blockchain technologies to store assets that have value to a specific community of individuals by following standards of preservation of digital records. Specifically, measures such as ISO (International Standard Organization) 15,489, ARMA's Generally Accepted Record-Keeping Principles, ISO 14,721, and ISO 16,363. Overall, Lemieux [22] advocates the use of the documentary truth defined as the trustworthiness of a record as a record or the quality of the document concerning what it purports to be, is one of the concepts theorists use for achieving reliability [26]. By these standards, reliability, authenticity, and long-term digital preservation should be obtained to create a legally strong digital asset. Those principles are described as follows:

Reliability

Lemieux [22] thinks that if relying on definitions of reliability on the standards mentioned above, those definitions can be used in the case of conflict. For example, ISO 15,489 states that:

"*A reliable record is one whose contents can be trusted as a full and accurate representation of the transactions, activities or facts to which they attest and can be depended upon in the course of subsequent transactions or activities. [22] (7.2.3 Reliability)*"

Authenticity

The same standards suggest that authenticity must be achieved by preserving the identity and integrity of an asset since the beginning of its recording. On the implementation of such requirements to achieve authenticity, registration processes must be carefully planned to ensure unique identifiers of data.

Long term digital preservation

Records should have long term usability for the users. To achieve this feature of data preservation, standard ISO, 2012a suggests a concept for long term preservation verification:

"Long enough to be concerned with the impacts of changing technologies, including support for new media and data formats, or with a changing user community. Long term may extend indefinitely" [22] (ISO, 2012a, p18)

Although, there are some limitations to the implementation of such concepts using any technology. ISO,2012a [22] prevents that the ability to understand the significance of bits or any representation of computer language is attached to the ability to put that information in a context that has meaning to a specific community. Moreover, the Organization for the Advancement of Structured Information Standards (OASIS) gives a framework for the achievement of the liability of information preservation. This framework states that a record must identify *Provenance* (or source of information), *Context* (How the information relates with other external data), *Reference* (Providing identifiers), *Fixity* (Providing protective shields of the data) and *Access Rights* (Providing terms of access to the users).

3.6.2 Privacy

Organizations need privacy over sensitive data. First blockchain protocols and architectures such as Bitcoin offered public access to the transactions in the network. Since then, enterprises and private organizations have invested efforts to develop permission blockchains to solve their business model requirements and prevent sensitive data to be

public. For example, some organizations won't be willing to share their business strategies with an open market. Instead, they would like to share just the necessary information to complete their transactions efficiently. Additionally, non-profits and public or government institutions need privacy over public data to secure the integrity of its users when required.

3.6.3 Security

According to Singh & Singh [1], cybercrime costs quadrupled from 2013 to 2015 and a large portion of cybercrime is still undetected. Moreover, the Gartner report suggests that the cost of cybercrime is expected to reach 2$ trillion by 2019. Security is a big issue in current systems. Consensus protocols such as Proof of Work (PoW, Bitcoin), execute complex puzzle-solving tasks and in each one of the nodes. The system is backed using Public Key Infrastructure (PKI) which is "asymmetric" cryptography, where one key is for encryption and another for decryption. The standard encryption that PoW uses is SHA-256, published by the U.S. National Institute of Technology of Standards and Technology [21]. The difficulty to find a block solution is correlated with the amount of computational power of the participants. It is a network secured by all the computational power of every single node in the system. To hack it, the attacker would require more than the double of the sum of the participant's computational power. To achieve the same levels of security previously mentioned in current centralized systems, the company must invest in several servers dedicated to run those tasks. In contrast, in a decentralized system, anyone can contribute to the network with their own devices, collaborating, and sharing efforts to achieve the highest levels of security. Consequently, the low costs of decentralized systems would be more attractive to businesses.

3.6.4 Business Model Abstractions and Decentralized Autonomous Organizations (DAO)

Features like role-based, permission-based, and business logic smart contracts help to design autonomous organizations where participants behave according to specific rules of the network. These network’s behaviors are denominated Decentralized Autonomous Organizations (DAO). Moreover, the consensus protocols of blockchains let the business model decisions being achieved by votes or consensus. Consensuses build a trustworthy platform to improve the outcomes of decisions that might impact the future and direction of

an organization. Consensuses let individual or groups of developers, setup the laws over which all the participants within a network interact with each other, providing trust, security, immutability and reputation overall activities executed and overall the history of the organization.

3.6.5 Efficiency

Bitcoin uses Proof of Work protocol, which requires large amounts of processing power. The Bitcoin network power consumption is increasing at an unreasonable rate to ensure security. As the network grows, that amount of waste is unacceptable for FinTech. There are several approaches to broader adoption in FinTech to reduce the amount of computational power by diminishing the participation of all machines as processors and instead of using better consensus algorithms [9] . For example, permission blockchains could be executed in a fraction of the time Bitcoin requires, which is 10 minutes to solve the puzzle between the miners to secure the transaction [27] . Hyperledger and Ethereum reduce that timeframe significantly.

3.6.6 DBMS vs. Blockchains

Why not use DBMS instead of Blockchain? This is a consistent argument on early adopters. In regards to the Financial Sector, one of the main goals of the financial data centers is to reconcile the records among several institutions. Therefore, with Blockchain outsider institutions could be allowed to read data from the distributed ledger and have guaranteed that those transactions are valid against the data held by others. Moreover, Blockchain hashing can be used to create an immutable audit trail, and eliminate the need for most external audits since each asset version has a traceable hash of previous versions.

3.7 Table of the Scope of the Research

Total References used	46
Use of the Paper for the Chapters	Authors
Problem Statement – Music Industry	[4] Passman, D. S. (2015). "*All you need to know about the music business*." Simon and Schuster.
Problem Statement – Music Industry Inefficiency	[45] Rethink Music Initiative. (2015). Fair Music: Transparency and Payment Flows in the Music Industry. *Boston: Berklee Institute of Creative Entrepreneurship*.
Literature Review – Intro to Blockchain	[21] Tapscott, D., & Tapscott, A. (2016). "*Blockchain Revolution:*

	how the technology behind bitcoin is changing money, business, and the world." Penguin.
Literature Review – Intro to Smart Contracts	[29] Christidis, K., & Devetsikiotis, M. (2016). "*Blockchains and smart contracts for the internet of things*." IEEE Access, *4*, 2292-2303.
Literature Review – Overview of Ethereum Clients	[35] Rouhani, S., & Deters, R. "*Performance Analysis of Ethereum Transactions in Private BlockChain*." University of Saskatchewan, Saskatoon.
Architecture of Ethereum	[34] Wood, G. (2014). "*Ethereum: A secure decentralized generalized transaction ledger*. *Ethereum project yellow paper,"*, 151, 1-32.
Performance and Workload Setup	[27] Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017, July). Performance analysis of private blockchain platforms in varying workloads. In the *2017 26th International Conference on Computer Communication and Networks (ICCCN)* (pp. 1-6). IEEE.

Table 3.1 Scope of the Literature Review and Relevant Papers

CHAPTER 4

ARCHITECTURE

In this section, the overall architecture of the system is presented. Its main functionalities and some of the features of the two leading Blockchains implemented, Hyperledger and Ethereum. Additionally, this section explains the consensus and frameworks that fit the necessities of the use case.

First, the levels of abstraction of the system are exposed. Then, this chapter focuses on extensive research on both platforms Hyperledger Fabric and Ethereum. Although, the source code in Ethereum will use the default consensus with a local testing network.

4.1 Layers of Abstraction

This section analyzes each of the layers that intervene in the design, from the underlying Machine Consensus Layer to the top Business Application Layer.

4.1.1 Machine Consensus Layer

This layer describes different setups of the consensus of peers and algorithms for the validation of the world state. Although, the consensuses and underlying architecture of both Hyperledger and Ethereum blockchains will be explained individually in the sections of each Platform. Despite the research over the Machine Communication Layer, the default consensus protocols of each platform were used since the application doesn't need complicated settings on the Machine Communication Layer, but the research shows the possibilities they can offer.

4.1.2 Application Layer

This layer introduces digital asset distribution according to agreements. This project doesn't consider the details of the data input business strategy as a priority. Usually, the data input comes from Distributors to the Labels by Copyright Companies. The application considers data imports from the participants with the tag "Distributors" with random data. Therefore, the project will not focus on how certain tracks revenues (digital asset) have more income than others since nowadays these values are determined by algorithms of streaming services and copyright organizations that intervene in the publishing process.

This project assumes that the input of the tracks is successfully carrying the real value of the revenues of each track either by uploading files or creating an instance of a song on the user interface. The input data is the point of departure for information processing. From this point, the revenues of the tracks flow through several participants by order of smart contracts. It means that if there is more than one smart contract established by two participants. The revenue distributes the value through the first person that proposed the agreement, the network pays the shares of the participants, and then the system takes the following smart contract to execute the next distribution.

In Figure 4.1, the data input is the track revenues. Then, the branch to the left distributes the payment of the Track Revenue between Distributor A1 and Record Label Company A2 according to an agreement between the two of them. This process generates a Track Revenue value A version2. The new version of the Track Revenue represents an update on the Ledger regarding the exchange of the Track Revenue. Finally, the system updates the account balances between the two participants in the Ledger. This process repeats itself as more smart contracts exist, creating different branches as many participants as required.

The number of digital assets (track revenue), smart contracts, and transaction computations within the network increase the expansion of the number of Docker containers automatically in the case of Hyperledger Fabric and the number of virtual machines in the case of Ethereum. The granted access participants can see all the transactions, digital entities, and chaincode versions in a public log in both platforms. In the case of Hyperledger Composer, anybody can see the docker container logs, and in the case of Ethereum, anybody can see the public network transactions.

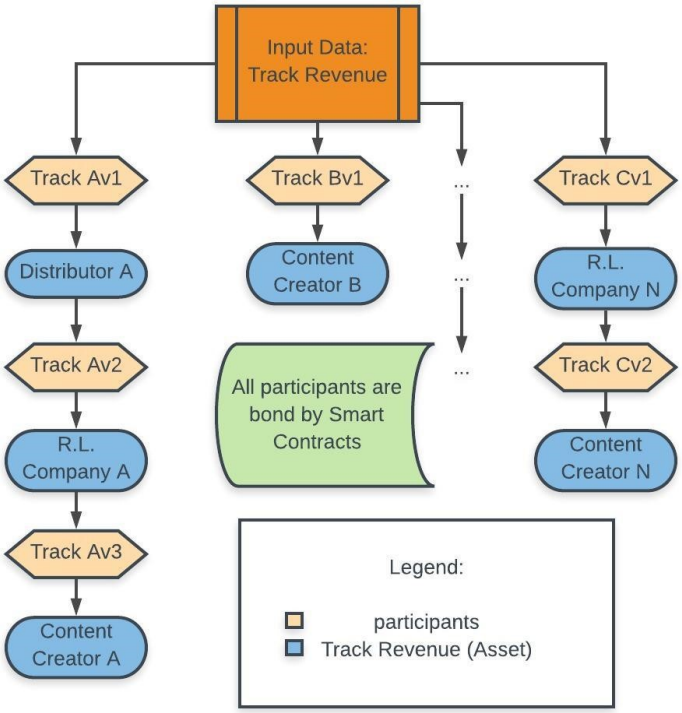


Figure 4.1 Proposed distribution workflow as described in the Problem Statement (Chapter 2)

4.2 Hyperledger Project

In December 2015 Linux Foundation created Hyperledger as a platform for scale enterprise blockchain solutions quickly and with all the support of Linux Community. According to the Hyperledger Project Organization, peer to peer distributed systems promise solutions such as data-sharing networks, cryptocurrencies, storage of value efficiency, decentralized digital communities, financial liquidity with low latency transaction processes, higher levels of data flow in Health Services, improvements on supply chain applications quality, and internet of Things [30]. Overall, Hyperledger is designed to achieve legacy software and modularity by the community. Ultimately, Hyperledger Project has developed several architectural frameworks to work with to address specific solutions. The currently active projects are:

- Hyperledger Sawtooth. -To manage distributed ledgers with Proof of Elapsed Time to improve efficiency.
- Hyperledger Iroha. - A secure framework for businesses to plug to blockchains.
- Hyperledger Burrow. -It interprets smart contracts of Ethereum virtual machines.
- Hyperledger Indy. - It provides libraries or APIs to interact with other blockchains or ledgers.

- *Hyperledger Fabric.* - A platform foundation to develop applications in a modular architecture.

To understand the selection of the components according to the requirements of a specific use case is necessary to analyze the consensus mechanisms that Private Blockchains such as Hyperledger offer.

4.2.1 Consensus Algorithms

Consensus builds a trustworthy platform to improve the outcomes of decisions that might impact the reliability, security, and direction of a participant in the network. The primary function of the consensus is to guarantee the order of the transactions and validate new blocks according to certain rules. It is responsible for plugging specific customized smart contracts and verify the data input that interacts with them. Although, it is essential must understand the nature of public and private consensuses first.

4.2.1.1 Types of Consensuses

There are *two main groups* of consensus algorithms. The permission lottery-based algorithms include algorithms such as Proof of Elapsed Time (PoET) or Proof of Work (PoW, Bitcoin).

Lottery-based algorithms are advantageous because they can scale to a more significant number of nodes and achieve more fairness. Since the winner or winners of the lottery propose a block and transmit it to the rest of the network, there is an equal and fair opportunity of participation (For example in the case of Bitcoin). Although, if two nodes propose a block at the same time, this system creates a fork and just the one winner that has more computational power is validated.

On the other side, permission voting-based algorithms include Redundant Byzantine Fault Tolerance (RBFT) or Paxos. These algorithms offer low latency finality (timeframe to validate a new block) due to the nature of voting consensus. When a majority or several specific nodes validate a transaction or block, consensus exists, and finality occurs. Although because these voters must exchange messages, the time of reaching an agreement also increases. That's why there is a tradeoff between scalability and speed in private blockchains [31].

These algorithms are used according to the necessities of the use case. For example, blockchains applications such as cryptocurrencies are environments where there is no trust. In these scenarios, a lottery-based algorithm such as the PoW could achieve fairness of participation. Instead, in other ecosystems such as businesses need some degree of trust and privacy. In these cases, the voting algorithms could be better suitable.

It is crucial to compare particular parameters such as the *speed of creation of new blocks*, *finality latency,* and *scalability* or level of decentralization to understand the use case of such consensus. Among them, the *finality latency* timeframe is one of the most important.

Finality Latency is the timeframe that takes to validate the first most extended fork of the chain as the new valid block. In PoW, as the solving puzzle difficulty increases, the centralization of miners is needed (mining pool). Therefore, if some mining pool can obtain enough processing power to generate new blocks at a rate faster than the longest chain, it can produce a new longest chain so changing the history of the transactions. Consequently, lottery-based algorithms are not suitable for low *Finality Latency*. Additionally, lottery-based algorithms such as PoW consume large amounts of computational problems that are not acceptable for business models.

Given the previously discussed parameters, it's possible to compare the advantages and disadvantages of using any of the permission consensus algorithms (private blockchains). Table 4.1 shows the comparison of permission consensus approaches and standard PoW.

	Permissioned Lottery-based	Permissioned Voting-based	Standard Proof of Work (Bitcoin)
Speed	GOOD	GOOD	POOR
Scalability	GOOD	MODERATE	GOOD
Finality	MODERATE	GOOD	POOR

Table 4.1 Comparison of Permission Consensus Approaches and Standard PoW [31]

4.2.1.2 Permission Voting Algorithms

Permission voting algorithms have some limitations. Since these are designed for non-fault tolerance, there are some tradeoffs. For example, Byzantine Fault Tolerance (BFT) works if the number of malicious nodes is below the safety threshold of ⅓. Also, the processing time

of messaging to achieve consensus. While there are more nodes in the network, more messages must be sent to reach consensus (voting), and higher reliability is successful. Therefore, even if there are some tradeoffs regarding points of failure, it achieves efficiency. Now, the process of consensus within the Hyperledger Components Environment is detailed.

4.2.1.3 General Process Flow of Hyperledger Project Consensus

In Hyperledger, the consensus is achieved by interacting with different active components within a Hyperledger Instance: the client, the consensus, the smart contracts. The first step in the process is to receive the transactions from the client. Then several ordering services interact with each other. They can vary from more centralized or more distributed. These services are responsible for the encryption, policy, and deterministic ordering of the transactions gathering the transactions in blocks.

The validation occurs when connecting the Consensus Layer with the Smart Contracts Layer. The Smart Contract Layer validates each transaction since it has the business logic. For example, validation of transactions that could result in a double spend, duplication of digital asset, or version control failure. Additionally, the Consensus Layer works with the Machine Communication Layer to communicate to the client and other peers.

Consensus Algorithm	Consensus	Pros	Cons
Kafka in Hyperledger Fabric Ordering Service	Permissioned voting-based	It provides crash default tolerance. Completion happens in a matter of seconds	Kafka is not Bizantine fault-tolerant, preventing the system from reaching agreement in the case of malicious nodes
RBFT in Hyperledger Indy	Election strategy set to a permissioned, voting-based strategy by default.	Provides Byzantine fault tolerance. Finality happens in a matter of seconds	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
Sumeragi in Hyperledger Iroha	Permissioned server reputation system.	Provides Byzantine fault tolerance. Finality happens in a matter of seconds. It can scale to petabytes of data distributed among different clusters	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.

PoET in Hyperledger Sawtooth	Election strategy set to a permissioned, lottery-based strategy by default	Provides scalability and Byzantine fault tolerance	Finality can be delayed due to forks that must be resolved.

Table 4.2 Comparison of Consensus Algorithms used in Hyperledger Frameworks [31]

Table 4.2 shows a comparison between all the consensus available on Hyperledger. Apache Kafka for Hyperledger Fabric, RBFT for Hyperledger Indy, Sumeragi for Hyperledger Iroha, and Proof of Elapsed Time for Hyperledger Sawtooth.

4.2.1.4 Which Consensus to Apply?

Now it's imperative to find the most suitable solution for the application. At the time of this research, some frameworks already have some support from the community, and others still were in development. For example, Hyperledger Iroha that uses Sumeragi looks promising since it achieves BFT and it can scale up to petabytes of data. Although it's still on development. Hyperledger Sawtooth that uses PoET can result in high latency in finality due to its BFT. The Sawtooth community just released v1 in late 2017 and still needs some exploration throughout the tools. Hyperledger Indy with BFT also is still on development. Although, Hyperledger Fabric (HF) is fault-tolerant and still achieves higher speeds [27] Additionally, the HF community released version 1 before Sawtooth. Therefore, the community is more prominent. Moreover, HF has developed several development tools to facilitate the development of specific use cases. Ultimately, HF can plug-in components that accept BFT out of the box consensus due to its modularity, to solve customized requirements. For example, users could plug in BTF protocols such as Practical Byzantine Fault Tolerance, Tendermint, Byzantine Fault Tolerance Smart or Honey Badger [32].

Since the application transacts with a small number of nodes, **Kafka** of Hyperledger Fabric is **enough for achieving efficiency.**

4.2.2 Hyperledger Fabric

Hyperledger Fabric is an enterprise permission distributed ledger that offers modularity and versatility ledger platform for a broad set of industry cases. The architecture accommodates

the diversity of solutions with plug and plays components such as consensus, privacy, and membership services. HF enables the concept of a network of networks. Members of the same network collaborate, but their data remains private to other networks in one single big ledger. HF has features including the use of Docker Containers, client-oriented integration (Node.js), and additional SDKs for other languages such as Java, Javascript, due Go programming language is at its core.

4.2.2.1 Hyperledger Fabric Architecture

According to the Hyperledger Architecture Working Group, a cross-project forum for architects and technologists from the Hyperledger community, there are several abstract components to follow a modular philosophy. These components can be modified and plugged with other external parts of other blockchains as required. Some of the possible components are described as follows:

- **Consensus Layer** to generate agreements for generating the next block
- **Smart Contract Layer.** It is a layer for developing the custom logic of the transactions.
- **Communication Layer**.- It is a layer for developing communication protocols between peers.
- **Data Store Abstraction Layer.-** It is a layer that integrates with other storage modules.
- **Crypto Abstraction Layer**.- It is a layer to develop the algorithms for cryptography.
- **Identity Services Layer.-** This layer manages the initial settings such as registration of identities, specific permissions for user, and identity security.
- **Policy Services Layer.-** This layer establishes policies for specific decisions.
- **APIs Layer**.- It integrates Hyperledger with other applications.
- **Interoperation Layer.-** It communicates between different blockchain instances.

4.2.2.2 Hyperledger Fabric Consensus Mechanism

Hyperledger Fabric consensus breaks into 3 phases according to the Hyperledger Architecture Group [31]. The *endorsement phase* which is driven by policy, the *ordering phase* that accepts an incoming order transaction to be committed to the ledger afterward, and the *validation phase* which takes a block of requested transactions and checks the correctness of the results. Figure 4.2 shows the transaction flow according to these phases Project [31].

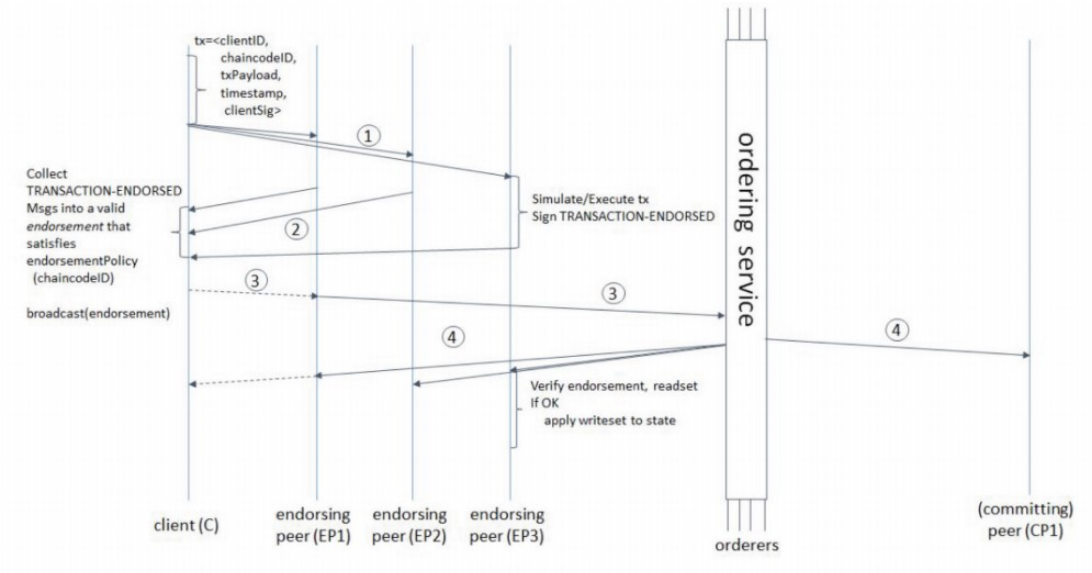


Figure 4.2 Transaction Flow in Hyperledger Fabric [31].

The consensus is achieved as follows. (1) When a client creates a transaction, a message request invokes a chain code function. (2) The transactions are executed by the endorsing peers simulating against the real current world state, and no updates are made to the ledger at this point yet. These procedures established by the endorsing peers are called endorsement signatures. (3) Then the client verifies those signatures by matching sets according to endorsement policies. (4) If these conditions are met, the client creates a sealed envelope and broadcast it to the ordering service. Although the ordering service only gathers data shells but not read the contents. (5) Then, it creates signed chain blocks and delivers it to the rest of the peers. Back to the peers, the endorsement policies validate the blocks and read a set of versioned keys at the time of simulating a transaction. (6) If all the conditions are met, the transaction proposal on the envelopes is marked as valid or invalid otherwise. Finally, an event is triggered to notify the client that the transaction has been appended to the blockchain, making it immutable [33].

Additionally, the ordering service API allows incorporating other algorithms such as the BFT agreement. Currently, several algorithms are supported to be plugged in on the ordering services. Among some of them, BFT smart, Simplified Byzantine Fault Tolerance (SBFT), Honey Badger of BFT, etc. As mentioned before, Hyperledger Fabric V1 doesn't come with an external pluggable algorithm by default. It currently includes two of the following options as default:

1.- A centralized non-replicated ordering service that does not execute any specific protocol and it's used mostly for testing (The choice for the application).

2.- An ordering service using Apache Kafka cluster to prevent crash faults. It uses a few hardware resources, but it is vulnerable to a single point of failure (cheats of peers) [31].

4.2.2.3 Docker Containers

Hyperledger Fabric uses docker technology, which is container-based virtualization, which doesn't require each guest (node on the blockchain) to run the entire operative system. The containers are more efficient than VMs because additional resources are for each OS are not needed. Indeed, the instances of these containers are smaller, faster to create, migrate, and more than one case can be deployed on the same hardware. Additionally, Docker Containers are written in go programming language.

There are several benefits to using docker. For example, putting all the application dependencies in an instance of a container. Also, it is fast and lightweight in comparison with Virtual Machines. Moreover, it let us use all the resources appropriately limiting the memory, CPU, network, and disk according to the requirements. Container architecture is also better suitable for microservices, which are applications with a single function. Ultimately, Open Container Project is supported by companies like VMWare, Amazon Web Services, HP, IBM, Microsoft, Google, EMC, Red Hat [2] .

4.2.2.4 Hyperledger Composer

Due to the necessities of more natural and faster development, Linux Foundation created Hyperledger Composer (HC) for developers. It offers a wide variety of features for the quick development of blockchain solutions. These features include easy Hyperledger Fabric network instance installation, web services (REST API) easy integration, query frameworks, chain code development.

4.3 Ethereum Blockchain

Ethereum is the second most known blockchain and cryptocurrency. According to coinmarketcap.com, Ethereum capitalization is around 59B US$, and the price of ether is 593$[15]. Vitalik Buterin proposed the kernel of Ethereum in 2013. Since its foundation, Ethereum has aimed to respect the social contract, using the internet as a decentralized value transfer, shared across the world, and virtually free to use to everybody. Also, the network

would aim for providing the end developer an integrated end-to-end system to build software in a trustful object messaging framework [34].

4.3.1 Ethereum Philosophy and Concepts

Despite the necessary adjustments in public policy for real use case scenarios, Ethereum core developers have guided the technology to follow consensus programming language ends. According to Gavin Wood, one of the founders of Ethereum and Ethcore, *"The incorruptibility of judgment could be achieved by a disinterested algorithmic interpreter."* Moreover, he warns transparency or the capacity to ensure that rules and protocols (that are followed in human interaction) never happen in human-based systems because the information is often lacking, and everyday old prejudices are present [34]. Additionally, Wood exposes the fundamentals of the Ethereum Network Architecture [34]:

World State. - Ethereum is a transaction-based state machine. A *genesis state* is incrementally being executed until it reaches a *final state*. The *state* might include information such as account balances, reputations, trust arrangements, data about details of the physical world. The World State is a point in time of the network, which is the accepted and validated version of itself. It is a mapping between addresses and account states encrypted in 256-bit hashes. It allows the previous *states* to compare with current or prior versions to validate some events. The World State prevents invalid states where, for example, a decrease in balance is executed without the correspondent increase in other.

Ethereum Virtual Machine. - The environment, on which miners execute transactions and smart contracts, is called the Ethereum Virtual Machine (EVM). It behaves like a single processor computer where every node is part of the network and contributes its processing power. This global virtual machine executes on a single transaction in a moment. Therefore, it still has the same problems as concurrent systems. The smart contract could be written well-formed and safe, but an untrusted external call could lead to unexpected results caused by foreign transactions. Additionally, the Blockchain interaction enables the possibility of Cross-function Race Condition, which is an attack that uses two different functions or contracts to share the same state [35].

Ethereum Clients. - To become an Ethereum participant or a node, the participant (Ethereum client) must download the history of transactions of the blockchain. There is plenty of clients written in different programming languages, each with advantages and disadvantages. Rouhani et al., [35] researched to determine the speed of two of the most popular Ethereum

Clients, Parity, and Geth. The study concluded that Parity 89.8 percent faster than Geth in terms of transaction processing. Figure 4.3 shows a diagram of the comparison between the two clients. Although, this application used Ganache for the implementation in the Ethereum Platform (See more details in the Implementation Section Chapter 5).

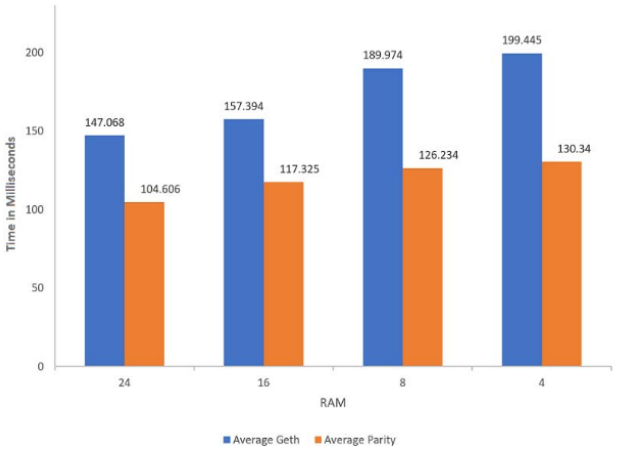


Figure 4.3 Average Time for each transaction on a client with a different amount of RAM [35].

Value over the network (Ether). – Wood [34] mentions that Ethereum was built to incentivize computation. Therefore, a method for transmitting value was developed, the Ether cryptocurrency. The cryptocurrency can divide itself into several parts. Figure 4.4 shows all the sub-denominations the Ether can be grouped:

International System	Usual name	Effigy
10^{-18} – attoether	wei	Wei Dai
10^{-15} – femtoether	kwei or ada	Ada Lovelace
10^{-12} – picoether	mwei or babbage	Charles Babbage
10^{-9} – nanoether	gwei or shannon	Claude Shannon
10^{-6} – microether	szabo or micro	Nick Szabo
10^{-3} – milliether	finney or milli	Harold Finney
1 – ether	ether	
10^{3} – kiloether	kether, grand or einstein	Albert Einstein
10^{6} – megaether	mether	
10^{9} – gigaether	gether	
10^{12} – teraether	tether	

Figure 4.4. Ether token system [34].

Additionally, Ether is an incentive for the miners but also to develop quality code since an inefficient code will cost more Ether. Remix solidity testing environment was used for the

first contract deployments. Ultimately, Ether was issued at a rate of 5 Eth per block on a block completion time of 12 seconds [36].

Transactions. -There are two types of transactions; some that result in a message and some others that occur in the creation of new accounts. An *account state* represents wallets, smart contracts, and any other data representation. When a transaction is submitted, a fee is charged to the senders of *account states* to avoid network abuse or the *Turin Completeness Test* for security purposes.

Gas. -. The *gas* is the fee for Ethereum blockchain transactions. The *gas* is taken from the sender and remains invalid until the transaction is verified and accepted. The cost of the transaction in the *gas limit* is predefined by how much the system executes code. Otherwise, the network would reply to an 'Out of Gas' statement.

The *gas limit* is the maximum number of units of gas measured in Wei (1 Ether = 10^18 wei) willing to spend in the transaction. It's a threshold set up by the user account. The *gas limit* for standard transactions is, on average, 21000 Wei.

The *gas price* is the cost per unit of gas that the transaction needs for mandatory for being executed. The value of it varies according to the performance of the miners. For example, in low traffic times, it can go down to 2 Wei, and in Token Creation Periods, it can go up to 50 Wei. If the transaction is unsuccessful, the difference between the gas limit minus the gas price is sent back to the sender.

Accounts. - Accounts are required to submit transactions. In Ethereum there are two types of accounts: The *Externally Owned Accounts* (EOA), which are used by the users to send transactions, and the *Contract Accounts*, to store the information about a specific contract. Each account within Ethereum has a 20-byte address with the following main four attributes:

- *Nonce:* Counter to verify that an account is created once and only once.
- *Ether balance:* Accounts Ether value
- *Contract code: a* container for logic (Solidity)
- *Storage:* Empty by default, but is for holding data of the logic or memory

EOA and Contract Accounts are indexed by the address. Although, the main difference is that EOAs have no Ether balance, no logic, and use private keys. Meanwhile, Contract accounts

have Ether but are executed by logic. Solidity can write and read to the internal storage and send additional messages or create other contracts (In Solidity, a Smart Contract is such as a Bean in Java). It's crucial to mention that if the system just has EOAs, the system becomes such as another altcoin of Bitcoin, losing the value of the programmable logic [36].

Blocks and Receipts. -In Bitcoin's Proof of Work, the transactions are saved in blocks in the chain since the creation of the genesis. On the other hand, in Ethereum, the blocks collect relevant pieces of information such as the Hash, the Beneficiary Miners, the Number of Previous Blocks, the Gast limit used, the Gas Price, the Timestamp, and other properties. When a transaction is executed, a Transaction Receipt is also generated, containing execution information concerns.

Decentralized Applications DApps. - Distributed Applications (DApps) are applications (typically a web application that runs in the browser for Ethereum) that interact directly with Smart Contracts on the blockchain. A traditional web application would have a web client that makes API requests to a backend server and persists. Data would be stored on a database wholly owned by the application or in the cloud. On the other hand, in a DApp, the web application reads data directly from the blockchain and writes data via transactions back to the blockchain. More information about the implementation of the DApp in Implementation Chapter 5.

4.3.2 Process of Ethereum Virtual Machine Mining

Figure 4.5 shows the process of block completion in the Ethereum Virtual Machine [36]. The main difference is the way how the blocks. The method of mining Ethereum is as follows:

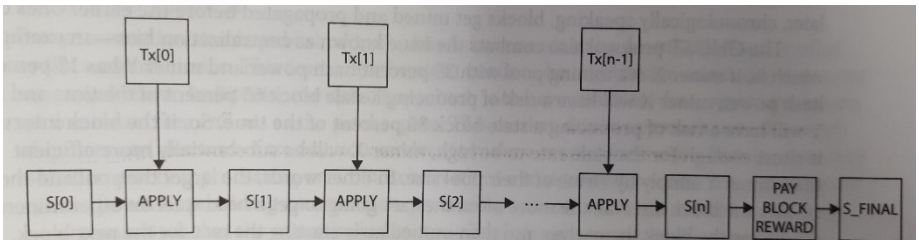


Figure 4.5: Ethereum Blockchain transaction list and state [36]

1. The system checks and validates the following:
   - Prior Block reference exists and it's valid.

- The timestamp of the current block is later than a prior block and 15 minutes later
- The block number, difficulty, transaction root, uncle root, and Gas Limit are valid.

2. Now the system set the previous block index in S[0](S=World State)
3. The system executes the transaction list over the appropriate state. If a call returns an error or the gas consumed exceeds the Gas Limit, the state returns an error.
4. S_Final (Final and current world state) becomes S[n] when the reward is paid to the miner.
5. The system verifies the hash tree of S_FINAL is equal to the final state root provided by the block header.

4.3.3 Ether Mining

As an educational practice, a miner to contribute to the network with computational power and get rewards in Ether was set up, so Ether could be available to deploy the contracts.

Mining cryptocurrencies is an activity that involves both electronic and computer science to integrate decentralized servers and allocation and transportation of value expressed in cryptographic wallets. Mining cryptocurrencies, especially bitcoin, has become a full-time job for some independent people and companies. Companies like Bitmain or Genesis Mining have some of the biggest bitcoin mining farms in all the world. Due to the PoW protocol, bitcoin miners consume significant amounts of energy.

These computers use powerful mainframes of computer power to solve cryptographic puzzles. Companies like Bitmain have developed their hardware to reach the maximum capacity of mining. Being a miner is not that simple as it seems. If a person wants to mine with mainstream hardware, their probabilities of puzzle-solving reward share are very low. Although, nowadays, several companies have joined computational power to offer shares to individual miners in exchange for some fee of reward share. These subnetworks are called mining pools, which let the average user join their network and work together to achieve better computational power. Usually, these mining pools also allow the user, add several “workers” (individual processors, GPUs, or any other device that has computational power) to get the share to a specific cryptocurrency wallet. The user of a mining pool can choose to assign the rewards directly to a cryptocurrency wallet or to hold it in a point of access to be withdrawn afterward. Usually, most of these pools offer a dashboard to watch the performance of the worker, the estimated rewards made to be paid, and other characteristics of the pool.

The hash rate is the measure over a probability of solving the puzzle. For example, a GPU Nvidia 8GB has an average of 27Mh/s. When the rounds of problem-solving are complete, all the individuals connected to the mining pool get a fair share of the reward.

The software was written commonly in C++ to interact with the GPU to set up an Ethereum miner. This software usually consists of bash commands that run either Linux, Mac or Windows. A crypto wallet, the URL of the pool, and the number of threads executed by the machine must be typed to initiate the task of the miner. Once the job starts, the user must take care of the maintenance of the devices, add more or so. As more time the miner works, its components get warmer, so a cooling system and a power supply protection is recommended.

Figure 4.6 Ethereum GPU Miner

CHAPTER 5

IMPLEMENTATION

This section describes the implementation of the proposal in the Problem Statement Section Chapter 2 and the Architecture Section Chapter 4. This section also explains all the tools for development that were taken into consideration for the completion of the proposal.

5.1 Definition of the Environment Variables

This section will define some variables that are applicable just to the design of the architecture of the solution and graphics of the workflow (more information in the Terminology section). The application's proposal distributes an *Asset* that has a numeric value among participants. The *Asset* represents a Track Revenue value, a tokenized song revenue, or any representation of content that has value on the network. The participants are called *Traders.* They represent distributors such as Spotify, a record label company such as Sonya content creator, or a customer. Third, the *Traders* can create *Agreements (smart contracts)* of the shares of an *Asset* for their services. An *Agreement* represents a copyright contract which is currently issued by content publishers, for example, the Digital Millennium Copyright Act (DMCA) or the Digital Performance Right in Sound Recordings Act (DPRA) [3]. The application defines an *Agreement* when a *Receiver,* an *Emitter*, and the *Percentage* of the *Receiver*, are specified. *Receiver* and *Emitter* are just names assigned to a *Trader* when participating in the creation of an *Agreement*.

The distribution of *Asset's* values starts when a *Trader* imports an *Asset* with a numeric value. The network evaluates possible Scenarios based on conditions and distributes the *Asset* among the *Trader's* balances. To illustrate the behavior of the system, Figure 5.1 shows the example of a basic Scenario of distribution with 4 *Traders*.

Trader 1: Distributor,

Trader 2: Label,

Trader 3: Artist A,

Trader 4: Artist B.

In this example, these Traders have created the following agreements:

Agreement 1: Distributor->Label (80% to receiver),

Agreement 2: Label->Artist A (40%),

Agreement 3: Label->Artist B (40%).

The process starts when the trader ‘Distributor’ imports Asset. For example, let's take $100. The Distributor obtains the entire *Asset* value. Then, according to *Agreement* 1, Distributor gives 80% of its import ($80) to Label. The Distributor remains with 20% ($20). After that, according to *Agreement* 2 and *Agreement* 3, Artist A and Artist B obtain 40% off Label import ($32). The Label remains with 20% ($16).

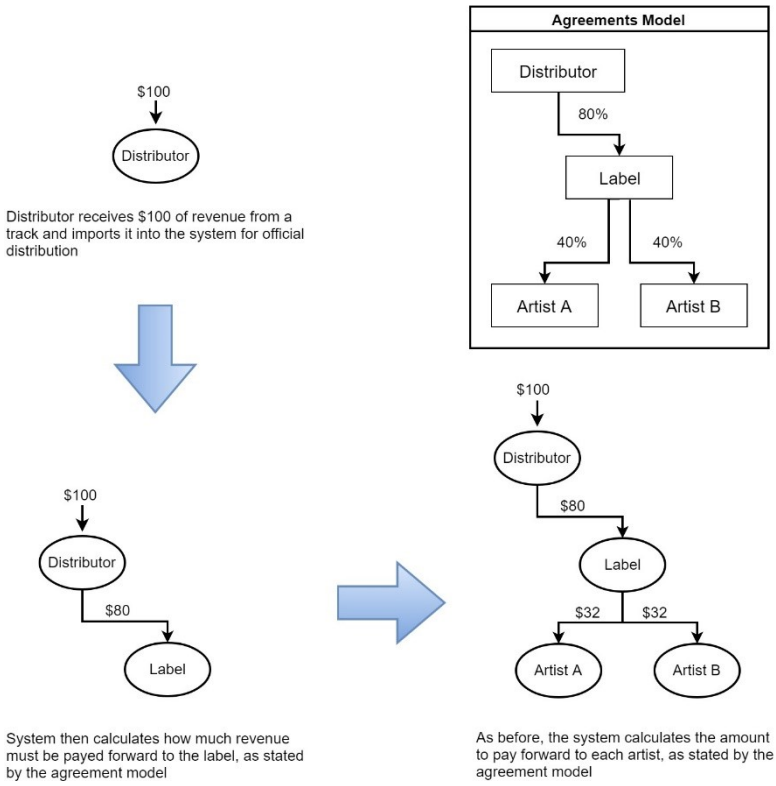


Figure 5.1 Example of distribution with agreements

5.2 Workflow of Data

Some steps of the workflow are achieved by interacting with the front end. Although, its crucial to understand the data workflow from the source to acknowledge the expected t input and output. Figure 5.2 shows the *General Diagram* of the application’s workflow of the *Asset* import down to the withdrawal. The *Asset* imports affect the balance of the users instantaneously. Then the user can remove the asset values and ultimately request real payments.

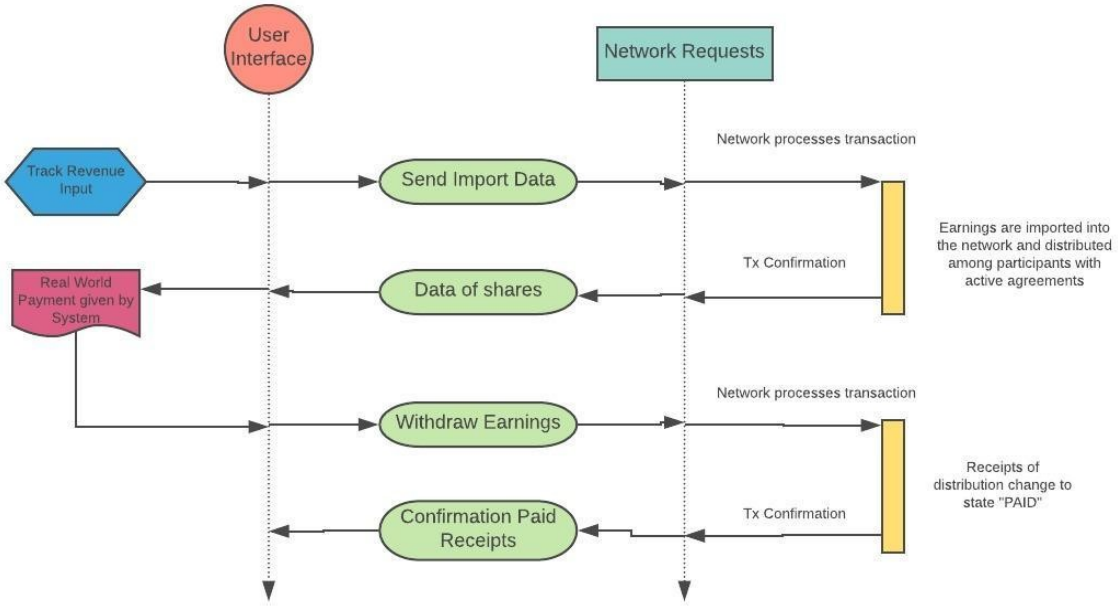


Figure 5.2 General Diagram of Data Flow

5.2.1 Balances of Users

In the application, there is a *disabled balance* and an *enable balance*. Figure 5.3 shows the logic of *disabled balance* and *enabled balance.* The yellow hexagons represent a cryptographic value of the Asset, and the green boxes represent a Receipt transaction of those values.

The disabled balance is earned when a user with the Distributor imports an *Asset,* and the system finds that this *Asset* doesn't have any Agreements associated with other Traders. Therefore, the system gives the user the entire amount of the import of the *Asset* with the tag *disabled balance*. This *disabled balance* can be distributed later when an *Agreement* is created with other *Traders*. In Figure 5.3, *Trader* A imports $5 as revenue into the system, but there are not any agreements between *Trader* A and any other. After that, the system adds the quantity to the *disabled balance* (or “On Hold Balance”-More information in the following section) of *Trader* A.

The enabled balance is the value earned by a normal distribution where an *Asset* that has one or more agreements in the chain of payments. Each participant receives a share of the payments according to the Agreement. In Figure 5.3, *Trader* A imports 5$ as revenue into the system but the system finds an agreement between *Trader* A and *Trader* B. Consequently,

according to the *Agreement Trader* A must give 80% of the revenue ($5) to *Trader* B ($4), and *Trader* A will remain with the 20% ($1).

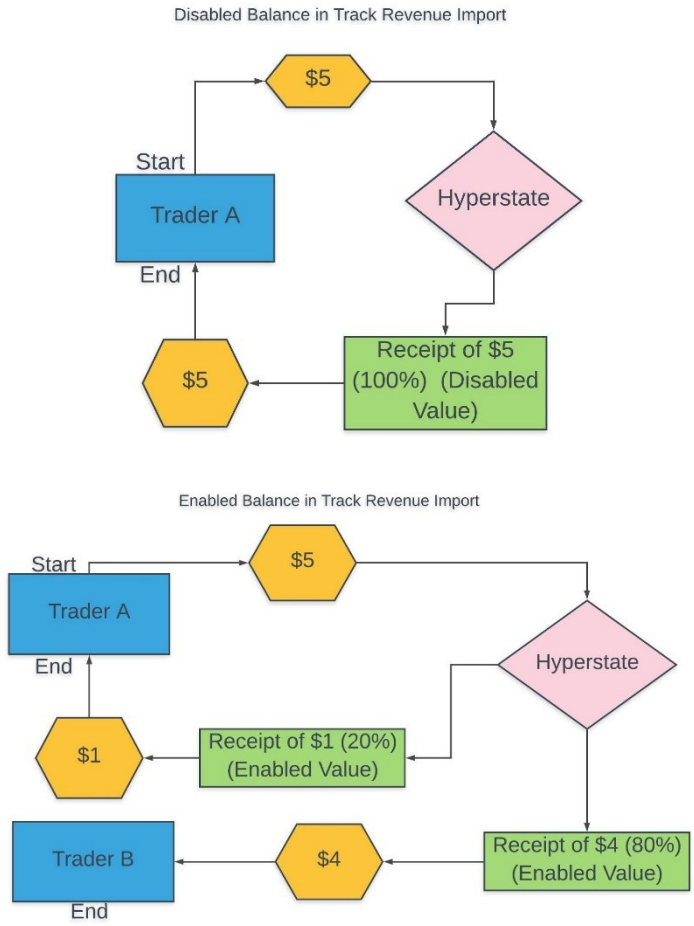


Figure 5.3 Logic of the Balance Enabled and Balance Disabled in Track Revenues

5.3 Distribution Algorithm

As mentioned in the previous section, the system uses two kinds of balances, the *Enabled Balance* and the *Disabled Balance.* Both constitute the *world state.* Figure 5.4 shows the Distribution Algorithm Workflow. It's assumed that the user (the *Uploader Trader*) selected one or more tracks and has imported a new Asset value.

There are two main sub-processes within the Distribution Algorithm: The *First Distribution* (Light Blue big box in Figure 5.4), and the *Recursive Distribution* (Light Pink big box in Figure 2). The input for the *First Distribution* process is the *Trader* Id (*Uploader Id*) that imported that *Asset*, the *Asset* information, the *datetime*.

5.3.1 First Distribution Process

Figure 5.4 shows an overview of the Distribution Algorithm. This first subprocess has the following procedure:

1. The system obtains the information of the *Asset* from the Ledger. Information such as the *Emitter Id* (*Uploader Id*), the *Previous Receiver Id* (At the beginning is 'None' since there aren't previous receivers), *Previous Agreement Id* (At the beginning is 'None' since there aren't prior Agreements evaluated), and the *Amount* (At the beginning is the total *Asset* value).

2.- The system sends the last information to the process *Evaluate Receivers* process. Figure 5.5 shows the *Evaluate Receivers* process. In this process, the system executes a query to find possible agreements between the *Emitter Id* and any other *Trader*.

Case 1: If there are not *Agreements*, then the network returns an empty *Receiver List*.

Case 2: If there were one or more *Agreements*, then for each *Agreement*, some relevant information is stored in the *Receiver List*:

-The *Percentage* of *Receiver's* share
-The *Emitter Id (Uploader Id* first iteration and then according to new inputs*)*
-The *Receiver Id*
-The *Agreement Id*
-The *Amount* computed destined for the *Receiver* [(*Amount Input)* * (*Individual Receiver Percentage)*].
-The *datetime*

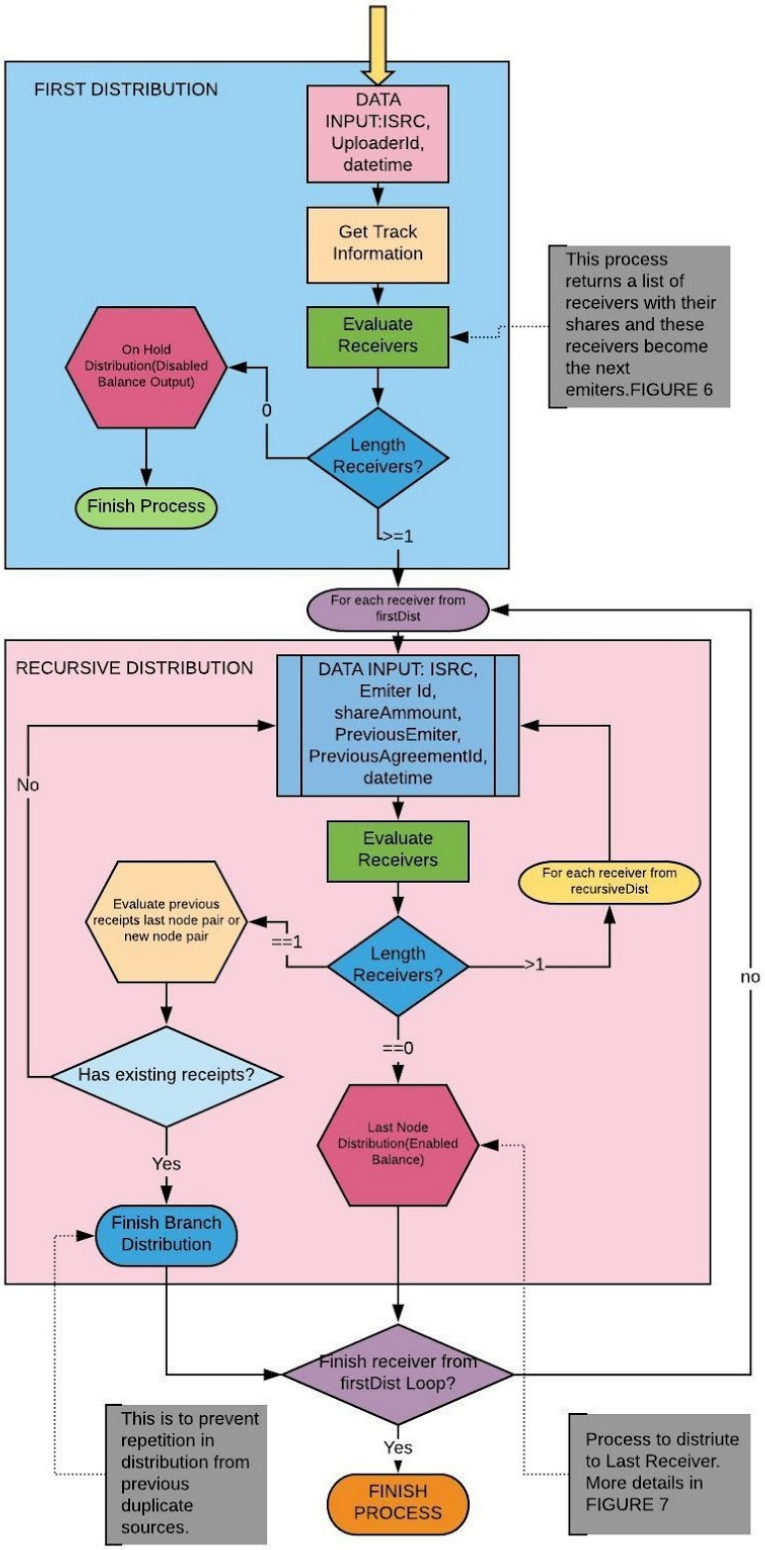


Figure 5.4: Distribution Algorithm Workflow

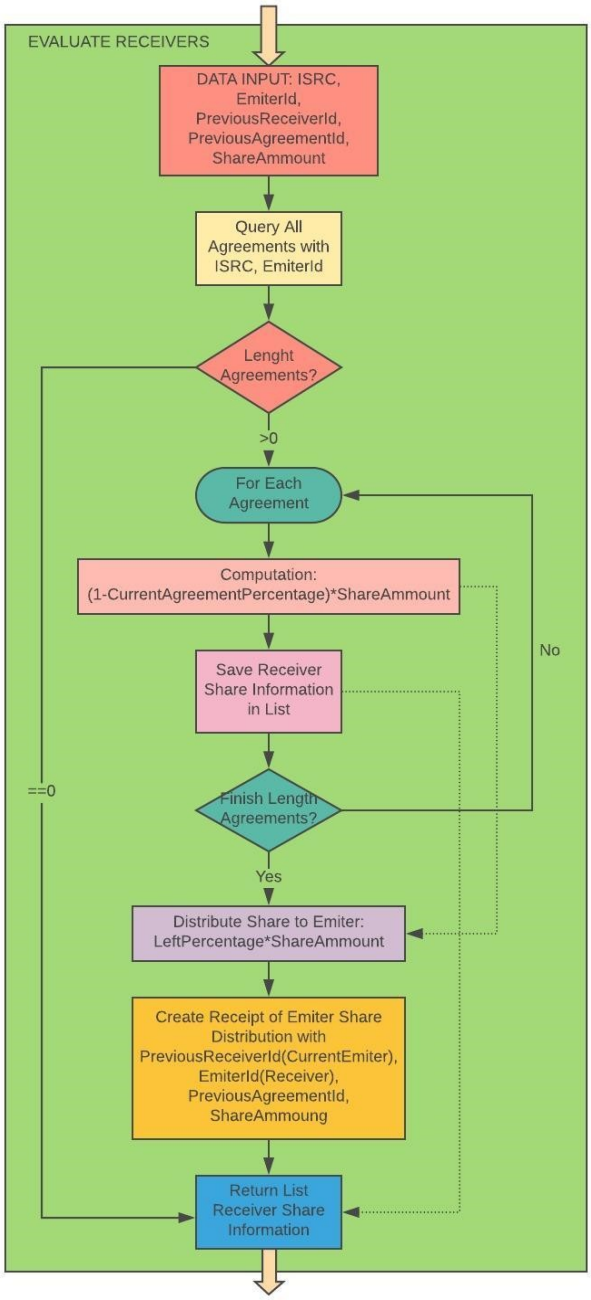


Figure 5.5: Evaluate Receivers Process

As the final step of this first iteration, a computation for the *Emitter* payment (*Uploader* at the beginning) is executed by multiplying what is left of the percentage of the share of the *Asset* with the current iterative amount. Finally, a *Receipt* is created to acknowledge the network of a change in the *Balance* of the *Emitter* account.

3. The system evaluates the length of the *Receiver List* as a result of the *Evaluate Receivers* process in case 2.

Case 1: If the *Receiver List* is empty (length 0), the process *On Hold Distribution* starts. In this process, there are not *Agreements* associated with the *Uploader (Unique case)*. Consequently, the *Uploader* that updated the *Asset* value automatically gets the entire amount but with a *hold* flag (an amount that can be distributed later when *Agreements* are recognized). Ultimately, a *Receipt* is created to acknowledge the change of the balance of the *Trader* (*Uploader Id*) with a flag "*on hold."*

Case 2: If the *Receiver List* length is 1 or more significant, the *First Distribution* process finishes, and the *Recursive Distribution* process starts.

5.3.2 Recursive Distribution Process

The pink container in Figure 5.6 shows the workflow of the *Recursive Distribution* process. The *Receiver List* is the data input with some extra information:

-*Emitter Id* (*Current Emitter in the loop)*
-*Previous Emitter Id* (At the beginning is the *Uploader* but later depends on the recursive input)
-*Previous Agreement Id* (Of the prior process *Evaluate Receivers*)
-*Amount* (From the last receiver in first *Evaluate Receivers* call)
-*datetime* (to discriminate other inputs or threads).

After that, once the system had these parameters, there are three possible outcomes with the length of the *Receiver List*:

Case 1: If the *Receiver List* is empty, the process enters the *Last Node Distribution* Process *(*Figure 5.6*)*. In this process, the *Last Receiver*(node) doesn't have any *Receivers* left to distribute. Additionally, the network makes the last query to ensure the prediction that the node is the last one on the branch of distribution. Consequently, the system should find an *Agreement* between the *Current Emitter* and the hypothetical *Last Receiver*. Finally, the *Percentage* of a share of the *Agreement* is taken and multiplied with the *Amount* input of the iteration, adding this value to the *Balance* of that *Last Receiver*. Finally, a *Receipt* for the *Last Receiver* is created, and the system passes to the next iteration of the *Receiver List*.

Case 2: If the *Receiver List*'s length is "equal" to 1, a validation process called *Evaluate Receipt* process (Figure 5.7) is executed. *Evaluate Receipt* process is control of duplication for the distribution between two nodes. Additionally, it uses the *datetime* to discriminate the distribution flows.

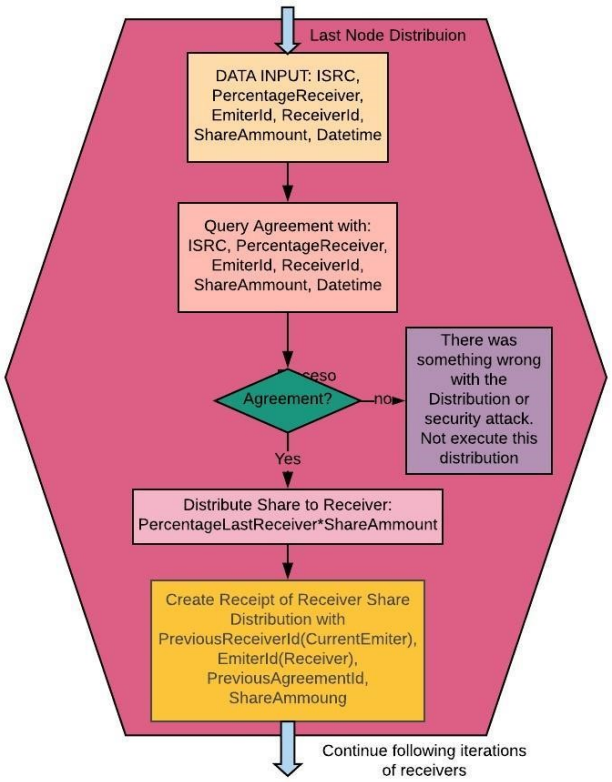


Figure 5.6: Last Node Distribution

Figure 5.7 shows the procedure to *Evaluate Receipt* process. Each node represents a trader in the network. The problem resides when the *Current Receiver* is getting an *Amount* from more than one *Emitter*. Duplications were avoided by analyzing the relationships between *Traders* with metadata such as previous *Receipts* with the same *Emitter*, *datetime,* and *Asset Id*. Therefore, the branches between the *Previous Emitter* (node A in Figure 5.7), the *Current Emitter* (node B or C in Figure 5.7), and the *Final Receiver* (node D in Figure 5.7) are evaluated. The procedure consists in a *Previous Receipt Evaluation* (*Emitter*(A) -> *Receiver* (B or C) in Figure 5.7) and a *Next Receipt Evaluation* (*Emitter* (B or C) -> *Receiver*(D) in Figure 5.7). Both branches are the same but with different paths. After the procedure, there are two possibilities:

(1) If there were previous *Receipts,* it means that somehow the network already paid that branch part (A-B-D or A-C-D in Figure 5.7) and the current process of *Recursive Distribution* "finishes==break."
(2) If there are not *Receipts,* the *Recursive Distribution* function executes again.

Case 3: If the *Receiver List*'s length is more significant than 1, then for each item in the *Receiver List*, the *Recursive Distribution* is self-called with the following change of input parameters:

-*Emitter Id* (*Current Receiver* becomes the following *Emitter* in the loop*)*
-*Previous Emitter Id* (Is the current *Emitter.* Is a parameter needed in the following *Evaluate Receipts* process)
-*Previous Agreement Id* (current *Agreement Id* resulted from *Evaluate Receivers* process)
-*Amount* (current *Share Amount* from *Evaluate Receivers* call)
-The same *datetime* (For the following *Evaluate Receipts* processes)

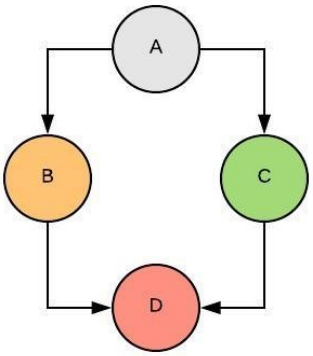


Figure 5.7: Receipt Evaluation-Branch Evaluation Process

5.4. Limitations of the Algorithm

This section shows that the algorithm has some limitations in specific possible scenarios. There are two *Scenarios* suitable to be successful in the distribution and one *Scenario* that enters in an infinite loop and will need further research to improve it.

5.4.1 Scenario 1: *Two sources of revenue for one node. Equal length branches of distribution.*

Figure 5.8 shows the Scenario 1with the following configuration:

Agreement 1: *Trader* 1 -> *Trader* 2 25%
Agreement 2: *Trader* 1 -> *Trader* 3 25%
Agreement 3: *Trader* 2 -> *Trader* 4 50%
Agreement 4: *Trader* 3 -> *Trader* 4 50%

Agreement 5: *Trader* 4 -> *Trader* 5 50%

The challenge in this scenario is to make sure that the distribution from (*Trader* 2 or *Trader* 3) -> *Trader* 4, executes correctly. It means that the system must ensure that 25% of *Trader* 1 is destined to *Trader* 2 and 25% to *Trader* 3 by avoiding duplication from other sources. This problem solves by using the *Evaluate Receipt Process* explained previously. The mechanism validates the existence of previous receipts in the upper node relationships (*Previous Receipt Evaluation* process) between (*Trader* 2->*Trader* 4) or (*Trader* 3->*Trader* 4) and the lower node relationships (*Next Receipt Evaluation* process). Consequently, *Scenario 1* successfully distribute the *Asset* value among the *Traders*.

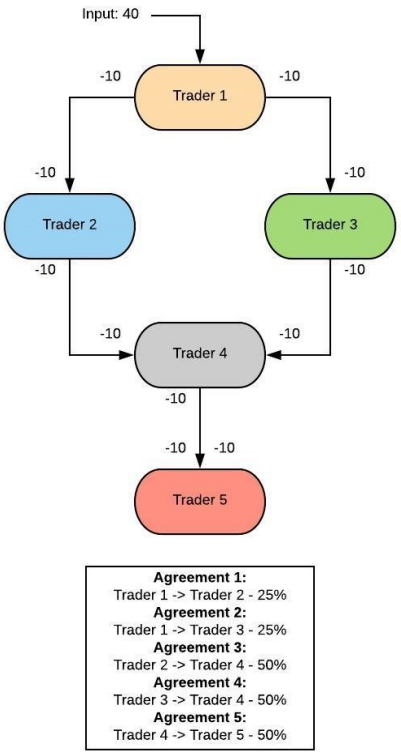


Figure 5.8: Scenario 1

5.4.2 Scenario 2: Two sources of revenue for one node. Different lengths of branches of distribution.

Figure 7 shows *Scenario* 2 with the following agreements between 5 Traders:

Agreement 1: *Trader* 1 -> *Trader* 2 25%

Agreement 2: *Trader* 1 -> *Trader* 3 25%

Agreement 3: *Trader* 2 -> *Trader* 4 50%

Agreement 4: *Trader* 3 -> *Trader* 5 50%

Agreement 5: *Trader* 4 -> *Trader* 6 50%

In this *Scenario*, the algorithm should automatically look for possible distributions when the last nodes receive the distribution. For example, the left branch in figure 5.9: (*Trader 1>Trader 2->Trader 4->Trader 6*) is longer than the right branch (*Trader 1>Trader 3->Trader 5*). The system secures that both branches (left and right), are being distributed as different branches. Additionally, the implementation mechanism uses one single thread for each branch saving information in memory information for the next branches. Therefore, Scenario 2 does not have any issue in succeeding.

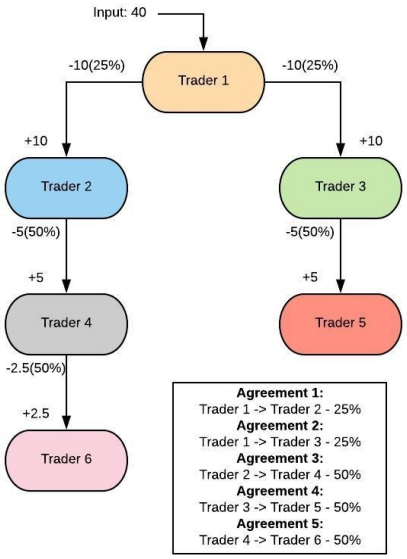


Figure 5.9: Scenario 2

5.4.3 Scenario 3: Infinite loop. Failing to evaluate previous and following branches in a node.

Figure 5.10 shows Scenario 3 with the following agreements between 4 Traders:

Agreement 1: *Trader* 1 -> *Trader* 2 50%

Agreement 2: *Trader* 2 -> *Trader* 3 50%

Agreement 3: *Trader* 3 -> *Trader* 4 25%

Agreement 4: *Trader* 3 -> *Trader* 1 25%

The problem that arises when the network tries to solve the second distribution between Trader 3 and Trader 4 and Trader1, it ends in an infinite loop. The first distribution in the first branch (*Trader 1>Trader 2->Trader 3->Trader 4*) is executed without problems, although the second branch (*Trader 1->Trader 2->Trader 3->Trader1*) causes a problem. Since in Trader 3 there is more than 1 Receiver (case 3 in *Recursive Distribution* Process), the system doesn't stop the distribution and continues if Trader 1 doesn't have to execute the *Last Node Distribution*. Therefore, the distribution enters an infinite loop.

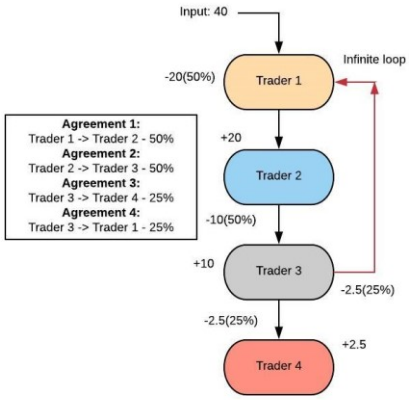


Figure 5.10: Scenario 3

The following section describes all the software development processes of the application: all the software components, third-party software, APIs, and technologies used overall in the system described in previous sections.

5.5 Hyperledger Fabric

The following HF frameworks accelerated development. Additionally, the architecture designed for performance experiments of the HF platform in the application was created to separate modules in case of the necessity of reusing them on future research. For example, the HF Network is separated from the client (HC), and both are separated from the testing server.

5.5.1 Architecture of HF Software Components

In a real case scenario, the application would work with the following set of components. The **front end** uses the Ionic Framework, and an API for communicating with the **back-end** which holds the HF network, the Client (HC), and the middleware backend server written in NodeJs.

Figure 5.11 shows the architecture for components of HF. The front end has a middleware component that communicates with the backend through HTTP requests. The backend (cloud-based) accepts the requests of the front-end managing the endpoints with a NodeJs express server package. The endpoint methods use the data input and process it to the blockchain network with HC.

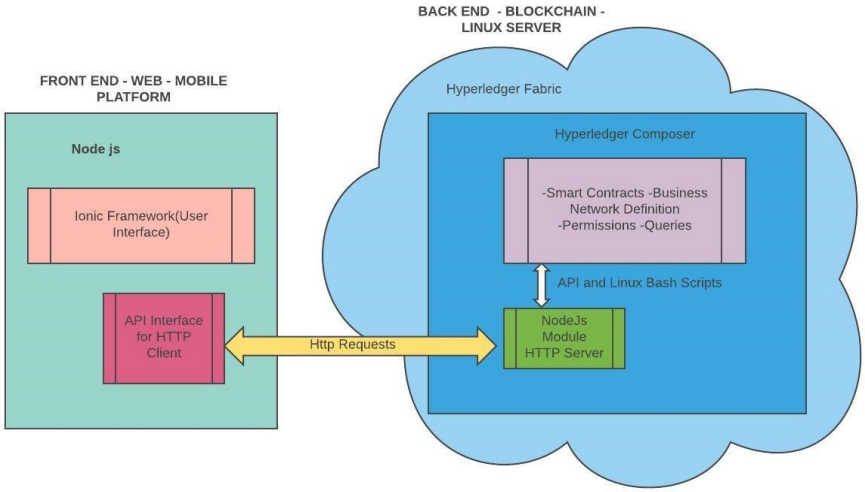


Figure 5.11 Architecture of Hyperledger Fabric Solution

It is important to mention that the front-end (Ionic framework) in Hypereldger was written with the same endpoint methods developed in Ethereum. Therefore, there no duplication of the front-end component in Ethereum.

5.5.2 Hyperledger Fabric Components

This section mentions the Client (HC) configurations and the NodeJs Server interaction. The front-end implementation will be detailed in later sections since the front end is the same for both platforms.

Hyperledger Composer (HC). - The HC Framework was used to achieve the definition of the network digital entities without worrying about specific settings of the Machine Communication Layer. SOLO configuration was chosen since the application is focused on the design of the algorithm of distribution.

In HC, the *docker-composer.ylm* file in fabric version hlfv12 shows the possible configurations allowed in HC. This file could be modified to add several peers. The default configuration of HC SOLO is described in Figure 5.12. Additionally, the SOLO configuration plus the application data model have the following considerations:

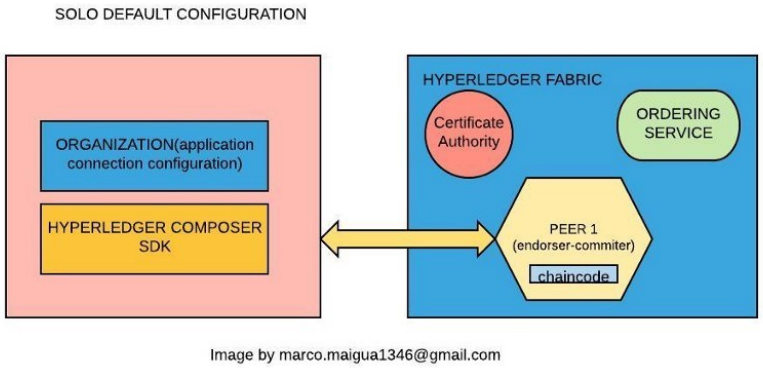


Figure 5.12 SOLO Configuration for Hyperledger Composer

- 1 Single organization (Hyperledger Composer Application) and one single client connection configuration (channel).
- 1 unique ordering service with SOLO configuration. It means that Kafka the system doesn't use BFT.
- 1 peer with endorsing and committing roles. This single peer holds the chaincode, interacts with the ordering service, and finally validates the transaction. Further, the peer is configured to use CouchDB as the state database. It means that the docker that represents the peer uses CouchDB.
- 1 single Certificate Authority. This file extends the capabilities of the permissions.acl file which gives access to the data to other HF networks (more information in the HF Data Modeling).

NodeJs Server. - NodeJs is a backend library of Javascript. It is designed to build scalable applications and does asynchronous processing on a single thread to provide more performance and scalability for applications that handle millions of concurrent requests [39].

The server component developed in NodeJs is responsible for accepting the Http requests from the front end and passing the information to the Node Js SDK of the HC framework to interact directly with the HF (runtime in docker containers that do hash validation). Moreover, the component handles the route of URLs for the endpoint computations

and the response parsing for the front end. Figure 5.13 shows more in detail the back-end component.

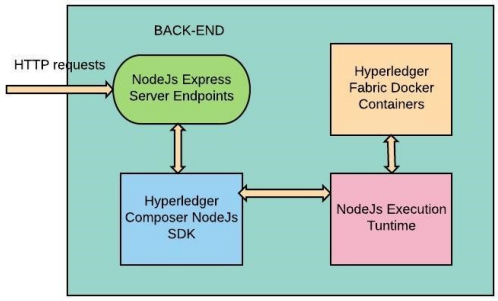


Figure 5.13 Software Implementation Architecture of the Back End

Ultimately, a very similar component was developed for Ethereum, with the same endpoints, the same NodeJS package, and the same separation of files. The difference is that Instead of the Hyperledger Composer SDK, web3 was used for interacting with the Solidity contracts, and instead the NodeJs Execution time, it has an ABI (Application Binary Interface) for communicating with the opcodes of the Ethereum Virtual Machine.

5.5.3 HF Data Modeling

HC possesses a .cto file that represents the *Business Network Definition (BNA)* which defined all the digital entities of the application. For example, objects like *Track, Trader, Agreement*, and the properties of each can be specified in this file. The file has a syntax like JSON objects. This file is essential since it is needed if other Hyperledger Composer Applications (HCAs) want to connect with the business network. The very first line of the file needs the name and Id of the *Business Network Definition*. For example, if other HCA wants to access a Track, it will look for org.organization.project.Track#TrackId. Additionally, this Id is useful in queries over the Ledger. There are .qry files that read queries to dictate individual access permissions to other HCAs. Although, queries were developed within the NodeJs calling functions assuming the network is open to other HCAs. Figure 5.14 shows an example of how the digital application entities look in the .cto file (BNA):

```
participant Organization identified by organizationId {
  o String organizationId
  o String email
  o String pwd
  o String name
  o OrganizationType organizationType// This user type
  o String traderId optional //this is the trader representation that belongs to an specific user
}
/**These are just tags, could be useful for future roles in the front end.
So far just the ADMIN is different from other Organization Types
 */
enum OrganizationType{
  o ADMIN
  o DISTRIBUTOR
  o LABEL
  o ARTIST
}
participant Trader identified by traderId{
  o String traderId
  o String name
  o String email
  o Double balance default=0.00//summarize of current Enabled Balance- Disabled Balance
  o TraderType traderType
  o String tokenAccountId //Currentlythe Trader just have one Token Account and is the same Id as the TraderId
}

enum TraderType {
  o DISTRIBUTOR
  o LABEL
  o ARTIST
  o ADMIN
}
asset Track identified by isrc {
  o String isrc //Main identifier
  o String title
  o Double revenueTotal default=0.00
  o String vendorIdentifier
  o String label
  o String author
  o OwnerType ownerType
  o String uploaderId optional// the participant that created a new track
}
```

Figure 5.14 BNA .cto File for the definition of the objects of the network.

Additionally, in HC Linux bash scripts run several processing instructions to initiate the network. These bash scripts contain all the instructions of the HC initial setup to save the time of development. Since each time the *BNA* .cto file was changed or updated, HF needed a reset. Each update created different versions of the hash that contains a new version of the chaincode. Consequently, the process of populating data on the network was tedious and time-consuming so bash scripts were necessary for setting up HF and fill data instantly. Figure 5.15 shows the bash script used to install each version of the chaincode each time there was an update.

```
sudo ../../fabric-dev-servers/startFabric.sh
sudo ../../fabric-dev-servers/createPeerAdminCard.sh

#With this following command we create the bna file
sudo composer archive create -t dir -n ../

# The source command can be used to load any functions file into the current shell script or a command prom
# Gets hyperstate version and latest bna file
source ./getLatestBnaAndVer.sh

echo "Using hyperstate version: $hyperstateVer "
echo "Using hyperstate bna file: $hyperstateBNA "

#With the following install the HC runtime with the specific name of the application
sudo composer network install --card PeerAdmin@hlfv1 --archiveFile $hyperstateBNA
#We deploy the business network with an administrator, the association with the bna card and the output name
sudo composer network start --networkName hyperstate --networkVersion $hyperstateVer --networkAdmin admin --
#To import the card
sudo composer card import --file hyperstate.card
#We test that we can communicate with the fabric
sudo composer network ping --card admin@hyperstate

cd ../middleware

sudo node server.js
```

Figure 5.15 Bash script for installing the BNA file and setup the network.

5.6 Ethereum

5.6.1 Architecture of Software Components

This architecture is very similar to HF. Figure 5.16 shows the architecture for Ethereum. Like HC, the front-end middleware sends HTTP requests to a NodeJs express server package. After that, the endpoints are handled by the Web3(green box) library for interacting with the EVM or any testing network. Web3 finds the solidity contracts that should be deployed by this point and computes what is necessary. Truffle framework holds the NodeJs Server, the Solidity contracts, and the configurations for the connections with any Ethereum network.

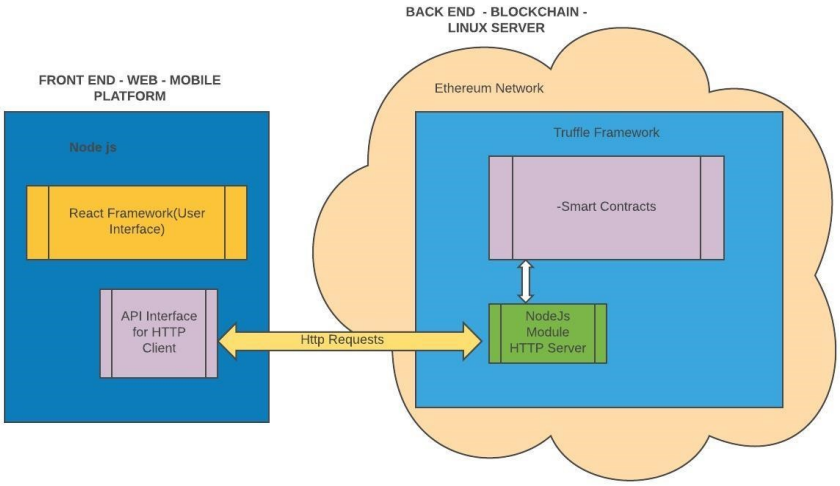


Figure 5.16 Architecture of Ethereum Solution

Now some third-party software and APIs used before obtaining the data for the analysis are going to be described. These tools helped to accelerate the development and to design the Ethereum implementation:

Truffle. -It is a development environment, a testing framework, and an *Asset* pipeline for using Ethereum virtual machine to help developers automate contract testing, migrate contracts, and interact with the clients by commands or scripts. Additionally, Truffle commands were used to test APIs quickly. For example, Truffle commands can help to compile and deploying solidity contracts in selected testing networks such as Ganache or in real networks such as Ropstein in the case to have enough gas resources.

Ganache Network Client. – It is a tool for testing Smart Contracts in a blockchain application before its deployment a real network to avoid unnecessary expenditure of gas resources. Ganache behaves very similarly to an actual client since it requests Gas values.

Although Ganache initialization gives Ethereum testing accounts with simulated Ether, and public logs to see the live transactions.

INFURA. - INFURA is a scalable, standards-based, globally distributed cluster and API endpoint for Ethereum, IPFS (Interplanetary File System), and other blockchain projects [36]. It's focused on Transport Layer Security to obtain access to a remote Ethereum node with a generated Key. Since downloading any node of ETV such as Geth is costly in terms of space and processing for the experiments, INFURA was used to not worry about the impact over the performance of the machine in the data analysis. Ultimately, the contracts of the application were deployed in Ropsten Ethereum network for testing the production stage of the research. Although, for the data analysis, the Ganache network was used for obtaining the results since Hyperledger Fabric simulates a local machine for running their nodes. Therefore, Ganache and Hyperledger Fabric instance run locally in the same machine with the same characteristics. More information in the Data Analysis section.

Web3.- It is a collection of APIs that let the developer interact with Ethereum nodes using HTTP, WebSockets, or IPC connections. As mentioned previously, INFURA permits the link to a remote node in Ethereum Virtual Machine. Usually, the application talks to the EVM node with JSON RPC calls [36].

The steps required for the deployment of contracts pass through the compilation of the solidity code into an ABI (Application Binary Interface) usually written in JSON. After that, the ABI could communicate with the EVM by translating the methods specified in the JSON interface to bytecodes that the EVM understands. Finally, the bytecodes are stored in some addresses within the EVM. Figure 5.17 shows the step by step process of deploying contracts in the EVM [36].

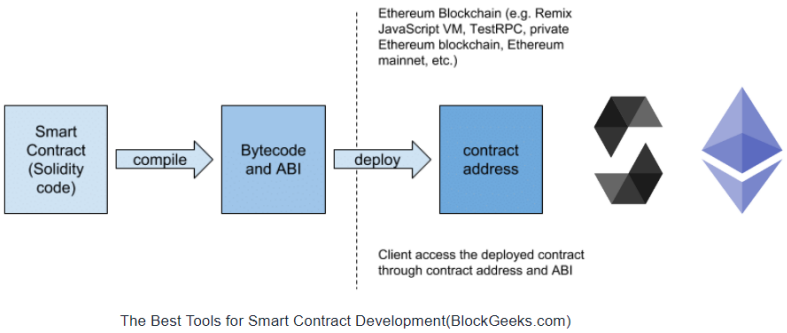


Figure 5.17 Process of using Smart Contracts in the EVM (BlockGeeks.com)

The following procedure for deploying smart contracts and interacting with the EVM was followed. Figure 5.18 shows the unique approach to implement Smart Contracts in the EVM

using Web3. Truffle-contract NodeJs package transforms the solidity contract to a JSON representation. Then, the Smart Contract is deployed with *truffle compile*, and then *truffle migrates* commands. Consequently, the artifact (JSON representation of the contract) was selected, and an instance of the contract ABI (Application Binary Interface) was created. Ultimately, an instance of the contract deployed in the network is created, and finally, it's possible to interact with the methods of the contract. Figure 5.19 shows the procedure in the NodeJs environment.

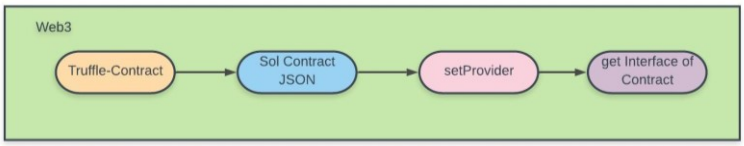


Figure 5.18 Process of deployment the application with Web3

```
const contractTruffle = require('truffle-contract');
const Web3 = require('web3');
const web3Provider = new Web3(new Web3.providers.HttpProvider('http://localhost:7545')); //I am usi
const contract_artifact = require('../build/contracts/Tracks.json'); //The JSON was creating with t
const Contract = contractTruffle(contract_artifact); //we create the contract abi

 Contract.setProvider(web3Provider.currentProvider);
 const contract_interface = await Contract.deployed();

 const accounts = web3Provider.eth.accounts;

 await contract_interface.method(input, {from:accounts[1],gasLimit:'6721975'}) //first method
 const result = await trackInterface.getObjectMethod.call(1004); //second method

console.log('ADDRESS');
console.log(trackInterface.address);
```

5.19 Process of deployment of the application in NodeJs using Web3

Additionally, Web3 offers some features that are useful for the development of DApps. For example, *events* let applications know that certain conditions were met, and some actions must be taken. For example, if an agreement in a transaction needs the signature of two users, and if one user proposed the agreement, then the other user must sign. Figure 5.20 shows how a listener is applied to a specific event declared in the contract.

```
1  var event = myContractInstance.MyEvent();
2  //watch for the changes
3  event.watch(fuinction(error, result){
4              if(!error)
5                console.log(result);
6                //execute business logic
7              });
```

Figure 5.20 Code Snippet of Web3 Events

NodeJs Server. – This component behaves similarly to the Node Js server in HF by accepting Http requests from the front-end. Although, it interacts with the Ethereum network client. As mentioned previously, Web3 and Truffle deploy the contracts through JSON RPC calls to INFURA, and finally, they arrive at the EVM. Figure 5.21 shows the architecture of the back-end in the platform of Ethereum.

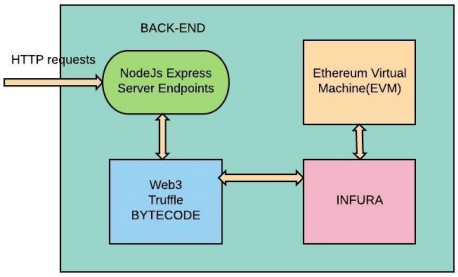


Figure 5.21 Software Implementation Architecture of the Back End

5.6.2 Ethereum Data Modeling – Solidity

This section explains further how the Solidity Smart Contracts were written and how this project dealt with the limitation that solidity showed since the data modeling depended on the solidity structure to approach the HF implementation for the Data Analysis.

Solidity files (contracts) are responsible for dictating the classes that represent each entity and its methods. The Smart Contracts are later deployed in any Ethereum network. The developers are responsible for connecting the DApp with the address of the contract and access the methods to interact with it. The following are some considerations that had to be addressed in the development of a successful flow.

Ethereum Accounts Properties. - Each account within Ethereum has 20 bytes of address's hash, which is a digital object with four main attributes:

Nonce: This is a counter that is used to ensure that each transaction is processed once and once only.

Ether Balance: The amount of Ether

Contract Code: Container for the Logic (Solidity-Optional).

Storage: It's a space for storing data on the account (empty by default).

External Owned Accounts (EOA) and Contract Accounts. - Both accounts are indexed by the address. Although, the main difference is that EOAs have no Ether balance, no logic, and are managed by private keys. Meanwhile, *Contract Accounts* have Ether but are

controlled by Solidity logic. It can write and read to the internal storage and send additional messages or create other contracts. Something important to remember is that *if the system has EOAs, the system becomes an altcoin of Bitcoin, losing the value of the programmable logic since the account is only for storing math computations of value.*

Ether incentives, Gas Price, and Gas Limit. - Ether is an incentive for the miners but also to develop quality code since the inefficient code will cost more Ether. The cost of the execution of the contracts is determined by the multiplication between the Gas Limit (approximate calculation of costs-computational resources costs is called opcodes/instructions), and the Gas Price (In Ether). Usually, it is recommended that developers should analyze the boundaries and the current price of transactions when developing contracts. Additionally, the Gas Limit helps the user know how much willing to spend in the transaction cost. The system sets up a limit that can take from the user, preventing Denial of Service (DoS) attacks.

Contract Best Practices. - Solidity contracts have some similarities with Object-Oriented Programming Languages such as Java. Moreover, in Solidity, a contract is very similar to a Bean in Java. Figure 5.22 shows a common way to write a Java Bean like Solidity Contract.

```
pragma solidity ^0.5.0;

contract CalleeTest{
    uint public integerTest=0;
    string public anotherMessage ="HOLA";

    struct StructTest{
        uint id;
        string name;
    }

    mapping(uint=>StructTest) public structs;

    function getString() public returns(string memory){
        return (anotherMessage);
    }
    function setString(string memory stringInput)public returns(string memory){
        anotherMessage = stringInput;
    }
    function getInteger() public returns(uint){
        return integerTest;
    }
    function setInteger(uint id) public returns(uint){
        integerTest= id;
        return integerTest;
    }
     function getStruct(uint id) public payable returns(uint, string memory) {
        return (structs[id].id, structs[id].name);
    }
    function setStruct(uint id, string memory name) public{
        structs[id]=StructTest(id,name);
    }

    event stringLogs(string);
    event intLogs(uint);
}
```

5.22 Java Bean Like Solidity Contract structure

Figure 5.22 shows how getters and setters are written as in any Java Bean. There are different types of variables where new values can be set up. For example, mapping a struct is

the way to create Lists. Any item of the list of Structs can be called by defining the id of the item. Additionally, there are event logs that can be used for listening to specific actions in the blockchain or just as in my case to follow the code debugging. Ultimately, since EVM tries to save processing is common to find the STACK TOO BIG ERROR. This error means that the number of declared variables in a function was exceeded.

Interaction with other Contracts. - In Ethereum, *the contracts are stored in different Contract Account Addresses*. Therefore, there are some considerations when calling functions or variables from other Smart Contracts. The following won't work (Figure 5.23).

```
pragma solidity ^0.5.0;
import './CalleeTest.sol';

contract CallerTest{
    CalleeTest public calleeTestInterface;

    function getTest2()public{
        calleeTestInterface = new CalleeTest();

        calleeTestInterface.setInteger(2);
        uint test2 = calleeTestInterface.getInteger();

        calleeTestInterface.setString("MESSAGE INPUT");
        string memory stringTest2=calleeTestInterface.getString();

        calleeTestInterface.setStruct(44,"this is a testnet");
        (uint id, string memory name)=calleeTestInterface.getStruct(44);

        emit intLogs(id);
        emit stringLogs(name);
    }

    event stringLogs(string);
    event intLogs(uint);
}
```

Figure 5.23 Wrong attempt for importing and calling another Smart Contract.

When a new instance of the Contract in another file is created just by importing it, it is only a reference to a piece of code but nothing else. It's possible to *create a temporary instance that can be called, for setting and getting values. Although, it's not possible to obtain any persisted data from that contract since the address of the Smart Contract that is called to must be specified*. The following code reflects the best way to find a Smart Contract and interact with its methods (Figure 5.24).

```
pragma solidity ^0.5.0;

contract CalleeTest{
    function getString() public returns(string memory){}
    function setString(string memory stringInput)public returns(string memory){}
    function getInteger() public returns(uint){}
    function setInteger(uint id) public returns(uint){}
    function getStruct(uint id) public payable returns(uint, string memory){}
    function setStruct(uint id, string memory name) public{}
}

contract CallerAddressTest{
    CalleeTest public calleeTestInterface;

    function getTest2(address calleAddress)public{
        calleeTestInterface = CalleeTest(calleAddress); //this CREATES A NEW EMPTY CONTRACT INSTANC

        calleeTestInterface.setInteger(2);
        uint test2 = calleeTestInterface.getInteger();

        calleeTestInterface.setString("MESSAGE INPUT");
        string memory stringTest2=calleeTestInterface.getString();

        calleeTestInterface.setStruct(44,"this is a testnet");
        (uint id, string memory name)=calleeTestInterface.getStruct(11);

        emit intLogs(id);
        emit stringLogs(name);
    }

    event stringLogs(string);
    event intLogs(uint);
}
```

Figure 5.24 Proper way to import and interact with other contracts.

In Figure 5.24, a procedure called ‘delegated calls’ creates proxies between the addresses by targeting the contract address. It’s feasible to interact with different contracts in different addresses without any problem.

Immutability of Deployed Contracts. – Once a contract is deployed, there is no way to update it in the same EOA account. If the Smart Contract needs modification, the only way to do it is to deploy a contract in another account with the new updates or call a proxy contract with new methods.

5.7 Web Application-Front End

A GUI that serves the blockchain solution was necessary. Therefore, it’s possible to understand which the necessities of the final user are. Ionic Framework can compile in all platforms: IOs, Android, and web applications. This framework was developed over Angular2.js, which uses Javascript programming language mask typescript. Typescript is useful for developers because it gives them tools to work in an object-oriented environment. Additionally, it offers material design components such as templates, scrollable lists, animations, a cycle of pages, and the use of essential features such as a camera or geolocation, among others.

The application was created based on the design of a product that is destined to be in production in the future. Some of these functionalities were developed for fulfilling the user experience of the final user. The functionalities and features the application have are:

Login, Signing, Updates, and Lists of Users. - A login page where the user can either create a new account or log in as an existing user was necessary. The user, when creating a new profile, can choose the type of *Trader* it is, as mentioned previously, it can be Distributor, Label or Artist. When the user enters the application, it can choose the functionalities the application offers, such as creating agreements accepting other *Traders*, analyzing the lists of Tracks available, the records of other *Traders*, the public log of *Transactions* that had a relationship with agreement-based distributions, the profile of the user (where the user can update the information), etc. Although, these *Traders* were created randomly for the Data Analysis Section.

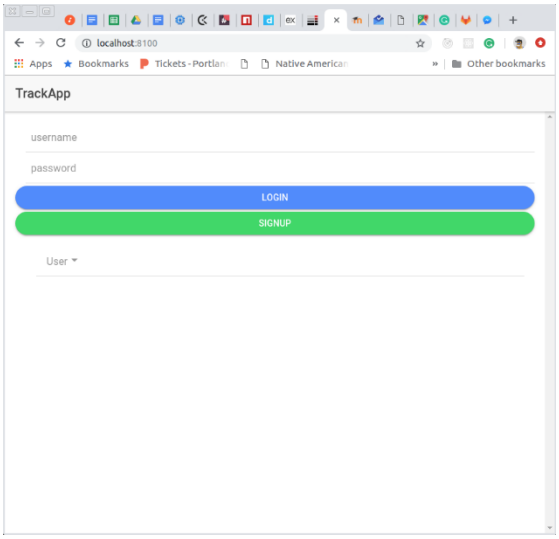


Figure 5.25 Login Page

Creation, Updates, Imports, and Lists of Tracks (Digital Asset) – The Track Revenue is the digital *Asset*. The Ledger has the lists of all the tracks, which properties of each should be visible to all the participants in the network. Although in the simulated Music Industry Model, just the *Traders* with the tag 'Distributor' should have permission to import the new Track Revenue (*Asset*) to the network (track revenue value usually determined by the number of streams the distributors have. After that, the cycle of the Distribution Algorithm explained previously begins.

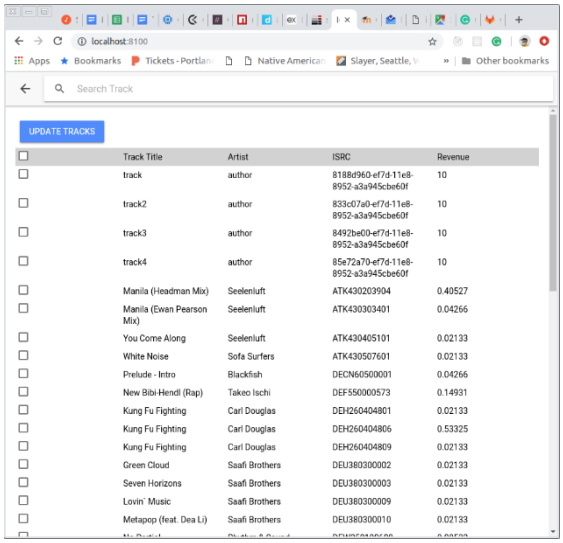


Figure 5.26 List of Tracks Page

Creation, Updates, and Lists of Agreement Creation - The application offers a page to create an individual agreement between two participants. Then, the user types the service fee that the *Receiver* obtains from the *Emitter (User)*. This feature can be applied once for a single *Asset* or several *Assets*. The user can create an *Agreement* once for all these selected tracks with any of the *Traders* in the Ledger. Additionally, the application can show all the Agreements that exist in an *Asset* in particular.

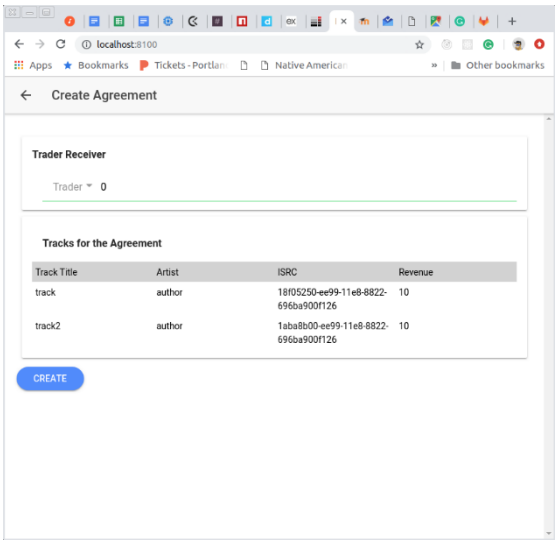
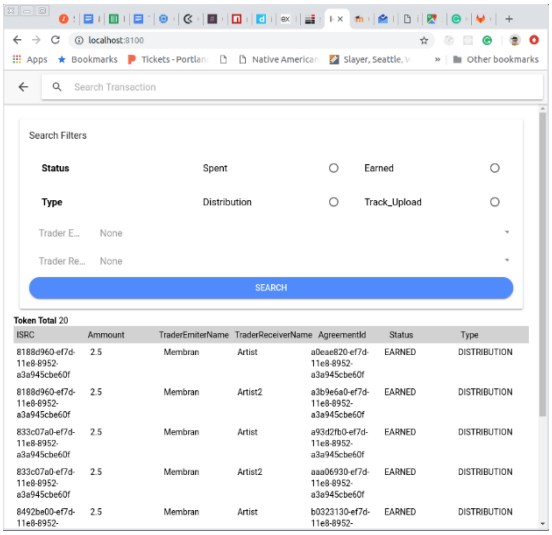


Figure 5.27 Creation of Agreement Page.

List of Transactions (Payment Receipts) - A *Transaction* gives a *Payment Receipt* from the *Asset* (Track Revenue) Distribution. The application offers a page where all the *Traders(*users*)* can see all the previous *Transactions* with other *Traders*. Each Transaction has relevant information of the transaction such as the *Emitter* name or the *Receiver* name, the *DateTime* of the *Receipt*, the A*mount,* and some extra information as explained previously.



Figures 5.28 List of Transactions (Receipts) Page

5.9 Development Tools

To achieve the state of the art of the development and from previous development experience, the following concepts help to organize the source code organized, and ready for being used by other developers in future endeavors. Some of these tools of software development are widely accepted in the corporate world and academic research. Such tools include MVC architecture, SCRUM agile project management tools, and UML graphics.

Material Design. - The design of the front end was made minimalistic. The only consideration over the interface for the final user was that it used combined colors according to the rules of Material Design, which are widely used in the industry for Mobile Development in the corporate world.

MVC. - The architecture is divided into three components: a *model* that expresses the domain knowledge, the *view* that presents the user interface, and *control* that manages the updates to views. Overall, the paradigm solves the following challenges:

- The same information should be shown in different formats in different views
- The changes in a view should be reflected in the remaining ones
- The changes in the user interface should be easy to make
- The central functionality should be independent of the interface to allow multiple interfaces to coexist.

Figure 5.29 picture can illustrate the relationship between the parts of an MVC architecture [40].

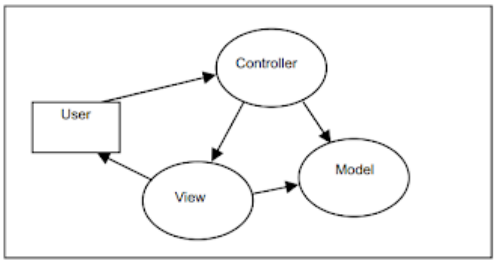


Figure 5.29 MVC Relationship [40]

UML Diagrams. - It stands up for Unified Modeling Language. These diagrams are diagrammatic representations of software components, activities, and functionalities processes (called workflows as well). These diagrams are used to represent a software product before the implementation of the source code to represent the final solution and give feedback to other developers. These diagrams help in understanding the data workflows in the Distribution Algorithm section.

SCRUM. -Agile management has recently been widely adopted by the IT sector and software development industry due to the challenges of demand for complexity and uncertainty of the continuous increasing features of a project. It's a series of values, principles, methods mainly the capacity to remain flexible within an environment of dynamic and adaptable change. It's a lightweight process for using iterative and incremental practices [41]. Ultimately, SCRUM was used for tracking the features and bugs and handle continuous changes due to the requirements of the proposal.

This Section accomplished two of the goals proposed in the Problem Statement section, the development of a Distribution Algorithm that solves the possible Data Flow Scenarios, and the Integration of blockchain platforms (Ethereum and Hyperledger) with Legacy Systems (Servers, middleware technologies, and front end technologies).

CHAPTER 6

EXPERIMENTS AND EVALUATIONS

In this section, the Data Analysis model the experiments of Pongnumkul et al. in the research paper "*Performance Analysis of Private Blockchain Platforms in Varying Workloads*" [27] to expose the performance of the payload of the two platforms Ethereum and Hyperledger for later comparisons.

According to Pongnumkul et al., [27], *Latency* and *Throughput* are the main problems and limitations regarding the Tech Industry requirements. The parameters *Execution Time* is defined as the total amount of time to execute a task (specifically in blockchain, the time it took to validate the transaction), *Average Latency* is defined as the difference between the deployment time and the completion time, and the *Throughput* is defined as the number of transactions successfully executed per second. A comparison between the two blockchains with several operations up to 10000 was set up to withdraw conclusions about its performance [27] .

6.1 Settings of the Reference Experiment

The experiment of Pongnumkul et al. [27] used most of the underlying features of each blockchain, but the consensus protocols were avoided to not interfere with the performance directly. For example, Ethereum uses virtual machines to execute Smart Contracts, and it offers an open-source software to configure the network. On the other hand, Hyperledger fabric uses the Docker container technology to enable smart contracts or mostly known as "chaincode." Both platforms had to be implemented differently.

The infrastructure used for comparing the works of Pongnumkul et al.[27], was built on an Amazon AWS EC2 with the Intel E5-1650 8 core CPU, 15GB RAM, 128GB SSD hard drive and running Ubuntu 16.04. Hyperledger Fabric network was used as the Hyperledger Framework, and the Geth Ethereum network was used as the Ethereum framework. Additionally, they did not use any consensus protocol besides the default fabric setting in the case of Hyperledger [27].

The experiment developed a cash transfer application with essential functions of evaluation, such as *issue money*, *transfer money,* and *creation of the account*. The time of

execution was measured with several sets of numbers of the transaction, for example, 10, 100, 100, and 10000. The function snippets of the transactions defer in each platform, but the interactions between the client and the blockchain were the same. Ultimately, they used HTTP requests with a Node.js application (Server) [27] . Table 6.1 shows some of the differences in the settings of the experiment.

Blockchain	Ethereum	Hyperledger Fabric
Peer to peer protocol	Ethereum Virtual Machines	Docker Containers
Transaction Denomination	Smart Transactions	Go based ChainCode
Client and Communication	Node.js application with HTTP requests	
Queries	Web3.js - JSON RPC APIs	Restful APIs
Measurements	Latency and Throughput	
Up to 10000 trx with no consensus protocols.		

Figure 6.1 Performance Experiment Settings of the Research Reference [27]

The result of Pongnumkul et al. [27] showed that Hyperledger had lower latency and higher throughput than Ethereum. Latency plays a crucial role in money transfer applications and broader adoption of the market. Figures 6.2-6.5 show that the latency of Hyperledger is lower and the throughput higher by a considerable amount. The calculations tested methods like create_account, receive, and issue money. In latency, for example, with 10000 Tx, it takes almost 500 seconds to complete 10000 Tx in Ethereum (unacceptable for early adopters), whereas in Hyperledger takes just around 34 seconds. In the case of throughput, 20.6Tx/sec can be executed when 10000 Tx are sent, but Hyperledger can handle 159Tx/sec. Therefore, the researchers concluded the preference over Hyperledger [27].

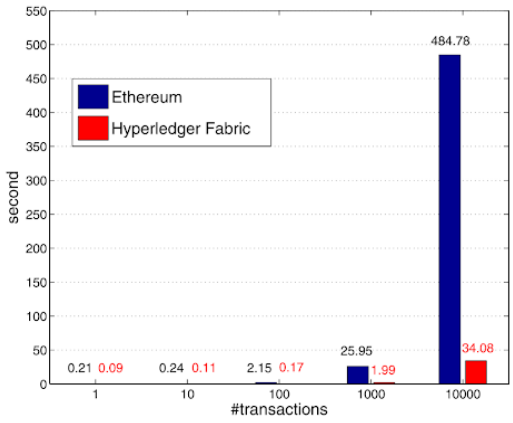


Fig 6.2 Comparison of average latency between Ethereum and Hyperledger [27]

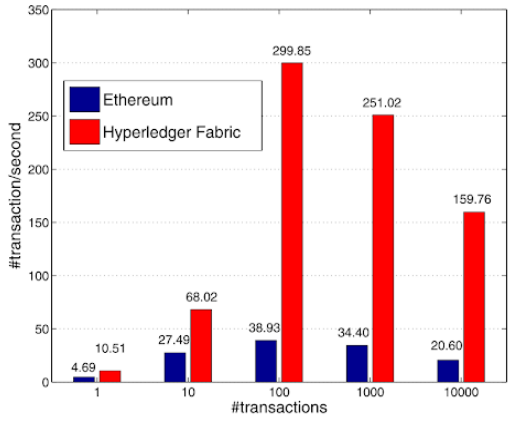


Fig 6.3 Comparison of average throughput between Ethereum and Hyperledger [27]

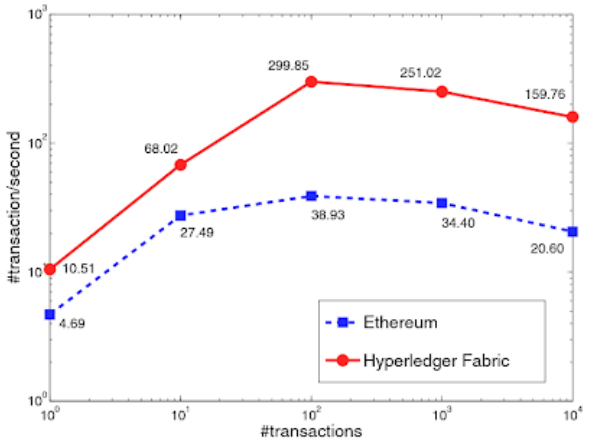


Fig 6.4 Average throughput of Ethereum and Hyperledger with varying number of transactions of TransferMoney function [27]

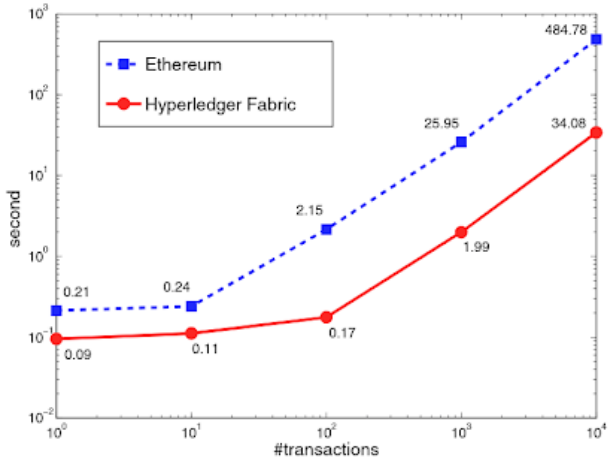


Fig 6.5 Average Throughput of Ethereum and Hyperledger with varying number of transactions of TransferMoney function [27]

Although, Pongnumkul et al. [27] mention the limitations of their settings in the experiment. Especially, the importance of the difference in consensus protocols which affect the performance directly. For example, the nature of the Proof of Work is much slower than the mechanism in Hyperledger Fabric Byzantine Fault Tolerance. It means that the experiment is limited to the Smart Contract Infrastructure layer. Consensus Protocols performance was analyzed in other ways [27].

6.2 The approach of this research

Ethereum and Hyperledger are the platforms chosen to address the problem of audit and data transmissions, as well as the issue of privacy and security. Each platform has its limitations and strengths. Each technology can be adjusted to the requirements. Although, a performance experiment was necessary to analyze the capabilities and constraints over the use case. *The hypothesis of the research states that Hyperledger can adjust a better modular architecture than Ethereum and can handle a more significant number of successful transactions in a shorter period based on the conclusions of Pongnumkul et al. [27].*

The methodology of this research for the evaluation of platform performance follows a *Qualitative Analysis* and *Quantitative Analysis* [42].

For the *Qualitative Analysis,* some assumptions are detailed. These subjective hypotheses are assumed based on the literature review, previous attempts of experimentation, and intuition of the promise of Blockchain technologies. Such assumptions could be the expectancy of the advantages and disadvantages of both platforms and components due its architecture, the features expected to be experienced from the theory in both platforms, or the expectation of the testing environment dependency (machinery to the be used for the experiment, either a physical machine or virtual machine in the cloud).

On the other hand, for the *Quantitative Analysis,* the data from the experiments are interpreted in graphs and tables. JMeter Software was used for the extraction of the data simulation.

6.3 Qualitative Analysis

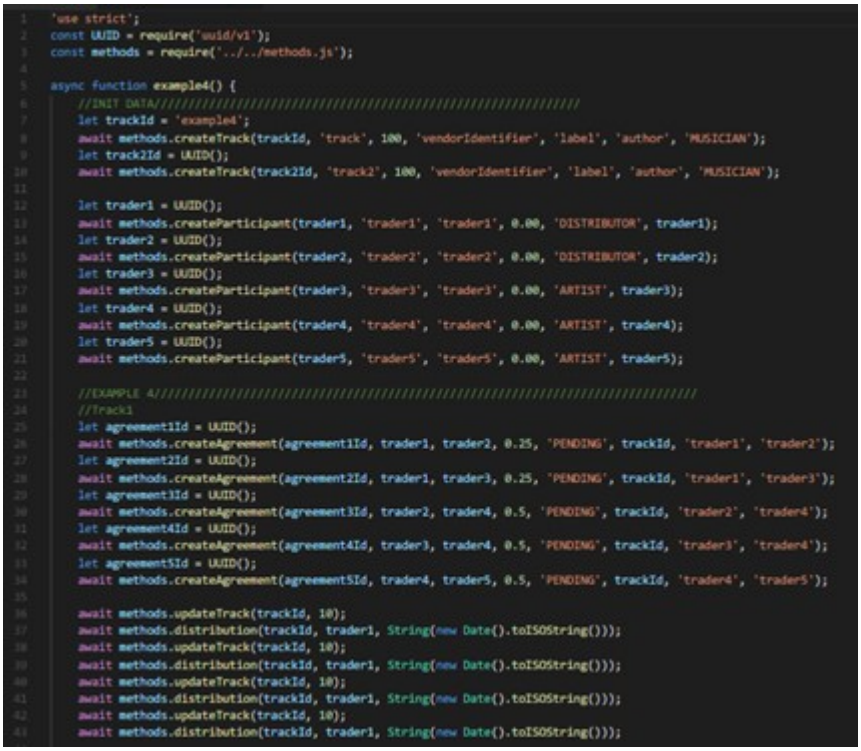
The hypotheses of this research were inspired by the experiments of Pongnumkul et al. [27]. Nevertheless, in the Architecture Section, the Distribution Algorithm was presented as the main subject of this research. Although the analysis expects a similar hypothesis as Pongnumkul et al. [27] (higher performance in Hyperledger Fabric), the implications give a broader scope of conclusions, especially real-life simulation simulations. Consequently, these are some of the hypotheses of this research:

- Hyperledger Fabric can achieve higher Throughput and lower Latency compared with Ethereum when workloads are varied up to 500 blockchain transactions in Hyperledger Fabric and up to 160 in Ethereum. It's assumed that the nature of the implementation directly impacts the performance since the design of the application differs significantly in both platforms. The research tried to model both platforms as similar as possible, but the nature of the technologies forced to change some patterns of design.
- Differences between these two platforms in execution time and average latency become more significant as the number of transactions grows.
- Hyperledger Fabric would handle more concurrent transactions with fewer glitches of errors than Hyperledger Fabric, such as in Pongnumkul et al. [27].

6.4 Data Analysis Implementation Architecture

The architecture of the experiment includes specific components designed to test cases of the Distribution Algorithm. For both platforms, a new component *Testing Endpoint* for incoming testing requests was developed.

This component was responsible for sending random requests to populate the network with the relevant data needed for the experiments. Additionally, the objects had random identifiers, but the creation of *Traders*, *Assets,* and *Agreements* emulates the Scenarios described in the previous Implementation Chapter. Moreover, routines that adjusted to the Scenarios explained previously in the Distribution Algorithm were written. For example, Figure 6.6 shows the bash scripts of Scenario 1 using Hyperledger Fabric for the Data Analysis.



```
'use strict';
const UUID = require('uuid/v1');
const methods = require('../../methods.js');

async function example4() {
    //INIT DATA//////////////////////////////////////////////////////////
    let trackId = 'example4';
    await methods.createTrack(trackId, 'track', 100, 'vendorIdentifier', 'label', 'author', 'MUSICIAN');
    let track2Id = UUID();
    await methods.createTrack(track2Id, 'track2', 100, 'vendorIdentifier', 'label', 'author', 'MUSICIAN');

    let trader1 = UUID();
    await methods.createParticipant(trader1, 'trader1', 'trader1', 0.00, 'DISTRIBUTOR', trader1);
    let trader2 = UUID();
    await methods.createParticipant(trader2, 'trader2', 'trader2', 0.00, 'DISTRIBUTOR', trader2);
    let trader3 = UUID();
    await methods.createParticipant(trader3, 'trader3', 'trader3', 0.00, 'ARTIST', trader3);
    let trader4 = UUID();
    await methods.createParticipant(trader4, 'trader4', 'trader4', 0.00, 'ARTIST', trader4);
    let trader5 = UUID();
    await methods.createParticipant(trader5, 'trader5', 'trader5', 0.00, 'ARTIST', trader5);

    //EXAMPLE A//////////////////////////////////////////////////////////////////////////
    //Track1
    let agreement1Id = UUID();
    await methods.createAgreement(agreement1Id, trader1, trader2, 0.25, 'PENDING', trackId, 'trader1', 'trader2');
    let agreement2Id = UUID();
    await methods.createAgreement(agreement2Id, trader1, trader3, 0.25, 'PENDING', trackId, 'trader1', 'trader3');
    let agreement3Id = UUID();
    await methods.createAgreement(agreement3Id, trader2, trader4, 0.5, 'PENDING', trackId, 'trader2', 'trader4');
    let agreement4Id = UUID();
    await methods.createAgreement(agreement4Id, trader3, trader4, 0.5, 'PENDING', trackId, 'trader3', 'trader4');
    let agreement5Id = UUID();
    await methods.createAgreement(agreement5Id, trader4, trader5, 0.5, 'PENDING', trackId, 'trader4', 'trader5');

    await methods.updateTrack(trackId, 10);
    await methods.distribution(trackId, trader1, String(new Date().toISOString()));
    await methods.updateTrack(trackId, 10);
    await methods.distribution(trackId, trader1, String(new Date().toISOString()));
    await methods.updateTrack(trackId, 10);
    await methods.distribution(trackId, trader1, String(new Date().toISOString()));
    await methods.updateTrack(trackId, 10);
    await methods.distribution(trackId, trader1, String(new Date().toISOString()));
```

Figure 6.6 Data Analysis of Scenario 1 Source Code.

This component (for each platform Ethereum and Hyperledger) listens to HTTP requests from JMeter software with different sets. This component accepts the HTTP requests to trigger the execution of random simulations for the examples of the Distribution Algorithm (more information

in the next section). Figure 6.7 shows how the JMeter Request activates the routine with the Testing Endpoint component.

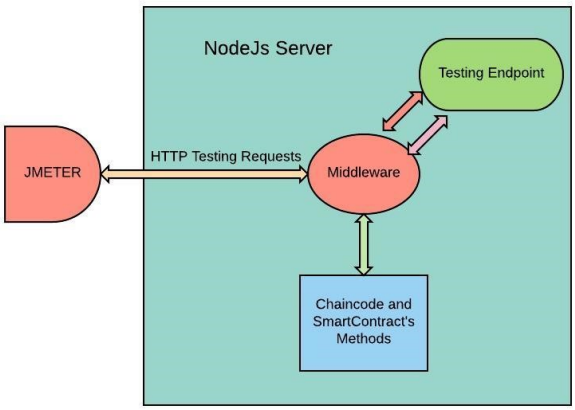


Figure 6.7 Architecture of the testing environment

Despite the use of this component in both platforms, the flow of data is slightly different. Subcomponents of abstraction that facilitated the debugging phase were developed since they could be tested separately. It's necessary mentioning that more layers could have added latency to some degree, but it was essential for the tracking of errors and design patterns. Figure 6.8 shows the data flow of Hyperledger Fabric that starts in the JMeter request and finishes in the response back to JMeter. Figure 6.9 shows the data flow of Ethereum using Ganache Client.

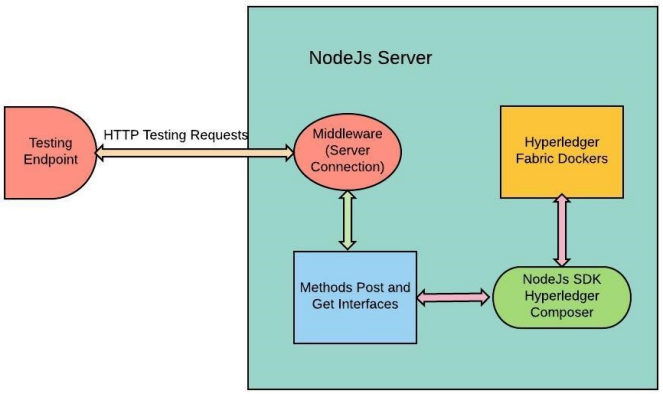


Figure 6.8 Data Flow of the Data Analysis in Hyperledger Fabric

To understand the data flow of the testing routines in more detail is recommendable to look at Figure 5.11 (Architecture of Hyperledger Fabric Solution), Figure 5.13 (SOLO configuration for Hyperledger Composer), Figure 5.14 (Software Implementation Architecture of the Back End).

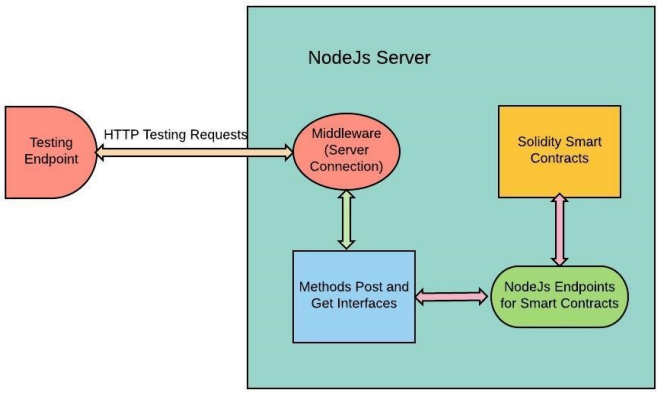


Figure 6.9 Data Flow of the Data Analysis in Ethereum- Ganache testing network.

Additionally, Figure 5.17 (Architecture of Ethereum Solution), Figure 5.19 (JSON RPC calls), and Figure 5.21 (Process of deploying the application with Web3) can be taken as a reference of the data flow and how some components chance in this section.

Nevertheless, it is worth mentioning the necessity of using a testing network such as Ganache instead of a testing network such as Ropstein. Ropstein testing network is not centralized and has other users using the network. Therefore, there is an excess accumulation of computation that would not be convenient for the comparison. Therefore, Ganache in Ethereum is what mostly assimilates to Hyperledger Fabric. Both have peers that produce hashes for executing transactions in decentralized code, but both are in one single computer.

Ultimately, since NodeJs SDKs for Hyperledger composer is one single thread programming language, a package called *child_process* to assign the HTTP incoming request to different ‘processes’ (NodeJs threads) in the Server was necessary since there were problems of concurrent incoming HTTP requests giving write/read maximum listeners in Hyperledger Fabric and JSON RPC connection hang-ups.

6.5 Quantitative Analysis

This section compares the works of Pongnumkul et al. [27] and takes the JMeter results to interpret them in each platform. Pongnumkul et al. [27] set several transactions to evaluate the Latency and Throughput of both platforms. For example, they took the following sets of transactions to plot the Latency and Throughput: 1,10,100,1000,10000 (ref). In contrast, the network sends several transactions per client or HTTP request in JMeter. For example, in Scenario 1 of the Distribution Algorithm using Hyperledger Fabric platform, the number of transactions per client is 109 (1 Distribution of asset using Scenario 1 = 1 HTTP request). Therefore, in this example, the sets change to 109, 218, 327, 436, 545. Consequently, the sets change according to the number of transactions in each Scenario of the Distribution Algorithm and in each platform. Nevertheless, similarly to Pongnumkul et al. [27] experiments, the experiments tried to load more transactions. The network, in theory, can handle or tolerate without errors. The machine used for these experiments was an 8 CPUs and RAM 32Gb running Ubuntu 18.04, trying to imitate the equipment used in the analyses of Pongnumkul et al. [27], which were an 8 CPUs machine with 15GB RAM running Ubuntu 16.04.

Additionally, the experiments are simulating real case traffic. The tests emulate real users that could create entities and interact with each other creating several transactions. Among those methods being followed, the Distribution Algorithm consumes the highest number of transactions. The transactions developed in both platforms include methods like “calling to the registry”, “making ping connections with the ledger”, “updating entities,” “creating entities.” Ultimately, the Distribution Scenarios discussed in the Implementation Section, belong to the same Examples 1,2 and 3 of the Data Analysis. Table 6.1 shows the methods and number of transactions included in the experiment using Hyperledger Fabric, and Table 6.2 shows the number of transactions involved in the development using the Ethereum Ganache Client.

Hyperledger Composer	create_asset()	create_trader()	create_agreement()	distribution()	Total Transactions
Example 1 (Scenario 1)	3	15	45	41	109
Example 2 (Scenario 2)	3	18	45	41	107
Example 3 (Scenario 3)	3	12	36	41	92

Table 6.1: Table of the number of methods that connect with Hyperledger Fabric core.

Hyperledger Composer	create_asset()	create_trader()	create_agreement()	distribution()	Total Transactions
Example 1 (Scenario 1)	1	5	5	10	21
Example 2 (Scenario 2)	1	6	5	10	22
Example 3 (Scenario 3)	1	4	4	10	19

Table 6.2 Table of the number of methods that connect with Ethereum Ganache Client

As discussed in the Problem Statement section, the main parameters of Evaluation are mainly the Execution Time and Latency. For the analysis, the following parameters are necessary to understand the results in Table 6.4 and Table 6.5:

Number of Requests per Thread. - JMeter was configured to have seven thread groups that represent the samples (thread group = sample). Each number thread represents a distribution. Therefore, for example, the 5th sample has five distributions.

Number of Transactions per Thread. - Table 6.2 and Table 6.3 shows the number of transactions per distribution in each example (different simulations of distributions). In each thread group, there is an increasing number of HTTP requests (Each HTTP request represents one single distribution completed). For example, in the sample 5th, there are 5 x (Number of Transactions per distribution). Therefore, in Scenario 1, using Hyperledger Fabric, in the 5th sample, there are 5 x (109 Tx/thread or Tx/distribution).

Execution Time. - It is defined as the amount of time that the blockchain takes to complete a request and confirm the transactions of the ledger. JMeter calculates the *execution time* of one Http request by making the *deployment time(t1)*, the time when the HTTP was requested, the *completion time(t2)*, as the time when a response is detected in JMeter and differentiating (t2-t1) to obtain the e*xecution time* of each HTTP within a thread group. For example, Scenario 1 using Hyperledger Fabric (Table 6.2) has 109 transactions to be completed and considered a successful distribution. Therefore, the *execution time* is the time the system takes to achieve those 109 transactions and get the response back.

Additionally, to obtain the *total execution time* from the 2nd sample and above, the *execution times* of each HTTP request are summarized within each thread group to calculate the whole execution time in each sample.

Throughput. - In the experiments, it is defined as successful HTTP requests per second. The *average throughput* is the calculated throughput over the *execution time. T*his process is given by JMeter. Although to obtain the *average throughput* in terms of *Number of Transactions* (connections to the blockchain either Hyperledger Fabric or Ethereum Ganache Client), the information of the analysis must be processed. For example, taking again Scenario 1 using Hyperledger Fabric, 109 Tx/distribution has a throughput of 0.38 (Distribution/seconds). Therefore, the following calculation is considered:

$$\frac{109Tx}{1\ Distribution}\ x\frac{0.38\ Distribution}{seconds} = 109\frac{Tx}{seconds}$$

The same multiplication to obtain the Throughput in terms of the number of transactions in each sample is applied.

Percentage of Error. - JMeter detects when a response in a thread group has thrown an error. Therefore, each HTTP request that is not successful counts as a part of the percentage of error at the end of all the experiments. For example, is in the sample 7^{th} (7 HTTP requests), there were two responses with errors, then the percentage of failure is 2/7 ~ 0.28 or 28%.

6.5.1 Assumptions of the experiment in Hyperledger Fabric

There are some assumptions to simulate correctly real traffic and do not have crushes in the system due to test plan errors:

- The default configuration of Hyperledger Fabric is the SOLO configuration (Architecture Section). It means that there is not high processing in the consensus algorithm.
- A successful simulation is defined as the creation of an asset, some traders, some agreements between traders, one distribution
- There are random inputs for the following methods: create_trader(), and create_asset() with random asset values. The method create_agreement() chooses any of the random traders and assets created previously. Random traders and tracks were simulated due to the security protection of each platform to prevent cheating. For example, in Hyperledger Composer, when several JMeter clients try to change the value of an asset with the Distribution

Algorithm, the network considered it as a threat to the system, throws an error, and close the connection to prevent an attack. Hyperledger Composer throws an MVCC error, which is a protection to prevent duplication by watching the *read* and *write* method's behavior within the Hyperledger Fabric environment.

- Each example of the Distribution Algorithm has a different number of transactions per request. Therefore, the scale on the figures in the axis x for Latency and Throughput varies depending on the example.
- Several thread groups in JMeter were configured to calculate the average transactions per second.

6.5.2 Hyperledger Fabric Results

Table 6.3 shows the results that JMeter throws from the testing planning. This section discusses some of the insights of such data outputs. Overall HF did not show error percentages in the simulations of request. Therefore, it can be assumed that all the transactions had a successful response, and the distributions were executed correctly among all the participants.

Hyperledger Fabric JMeter Results						
Example 1 - Scenario 1						
#Request/thread (#Distributions)	# Tx/thread	Total Execution Time(sec)	Average Execution Time(sec)	AverageThroughput (distribution/second)	Throughput (#Tx/sec)	error%
1	109	86.18	86.18	0.3831192661	41.76	0
2	218	176.835	88.4175	0.3735412844	81.432	0
3	327	278.35	92.78333333	0.3559266055	116.388	0
4	436	404.322	101.0805	0.3266422018	142.416	0
5	545	531.205	106.241	0.3108550459	169.416	0
6	654	709.926	118.321	0.2789724771	182.448	0
7	763	840.635	120.0907143	0.2750249017	209.844	0
Example 2 - Scenario 2						
#Request/thread (#Distributions)	# Tx/thread	Total Execution Time(sec)	Average Execution Time(sec)	AverageThroughput (distribution/second)	Throughput (#Tx/sec)	error%
1	107	85.56	85.56	0.3933084112	42.084	0
2	214	178.086	89.043	0.3778317757	80.856	0
3	321	277.225	92.40833333	0.3640373832	116.856	0
4	428	394.59	98.6475	0.3409906542	145.944	0
5	535	537.598	107.5196	0.3128971963	167.4	0
6	642	718.195	119.6991667	0.2809345794	180.36	0
7	749	840.352	120.0502857	0.2802616822	209.916	0

Table 6.3 Results of JMeter in Hyperledger Fabric

Analysis Execution Time. -Figure 6.10 shows the *total execution time* in Scenario 1 and Scenario 2 (blue and red lines) using HF with HC. Figure 6.10 shows an increase in the *total execution time* as the number of transactions per sample increases. Each Scenario has very similar steps in the axis x(samples) since they have approximately the same number of transactions per sample (Table 6.2). Therefore, the same behavior is expected in both curves. The same incremental response is shown in the works of Pongnumkul et al. [27].

When the highest number of transactions, 763 Txs in Scenario 1, or 5 distributions (thread group) in Scenario 1, the average percentage CPU consumption during the experiments was more than 90%, with a response time of 840 seconds or 14 minutes (very similar for Scenario 2). The response time of 840 seconds was the limit of experimentation in both scenarios for Hyperledger Fabric. After this threshold, the machine cannot handle more processing.

Analysis of Average Throughput. - Figure 6.11 shows the average throughput for both Scenarios using HF with HC. Like the *total execution time,* the *average throughput* increases as the number of incoming transactions increase. Although, in contrast with the works of Pongnumkul et al. [27], the point of inflection where the *average throughput* slowly decreases after a specific number of transactions cannot be found. The problem is that the environment reaches its top capacity after 763 Txs in scenario 1. The system cannot handle more concurrent transactions, but until this point, the bottleneck opens linearly.

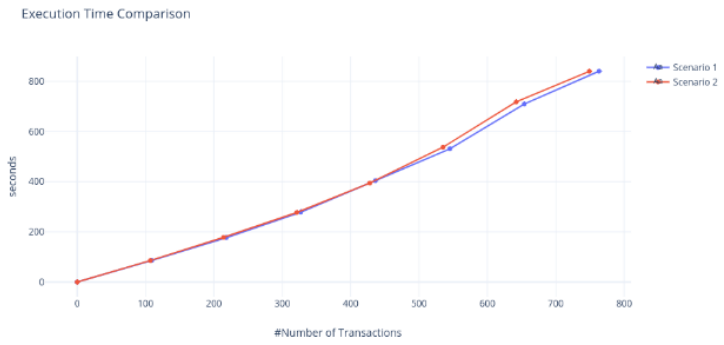


Fig 6.10 Total Execution Time for both Scenarios using Hyperledger Fabric

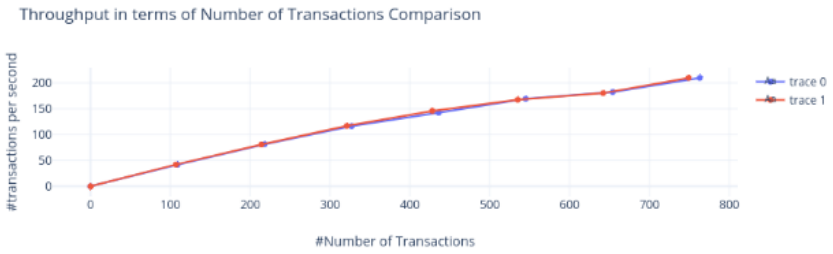


Fig 6.11 Average Throughput for both Scenarios using Hyperledger Fabric

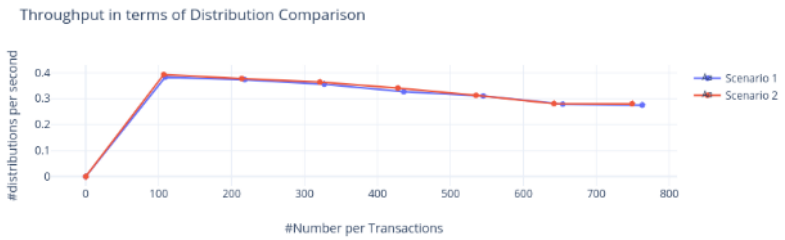


Fig 6.12 Throughput in terms of Distribution per second for both Scenarios using Hyperledger Fabric

Additionally, Figure 6.12 shows the *average throughput* of both scenarios in terms of distribution per second using HF with HC. For example, when 109 Txs in Scenario 1 are being executed, 0.38 distributions per second are possible in the bottleneck. On the other hand, when 763 Txs are executed in Scenario 1, the bottleneck decreases to 0.27 distribution per second as concurrent transactions are being performed in the blockchain. In comparison with Pongnumkul et al. [27], this is a similar behavior because the bottleneck becomes shorter as concurrent transactions in each time are executed.

6.5.3 Assumptions for Ethereum Environment

Since the algorithm implementation was first executed in Hyperledger Fabric, the implementation in the Smart Contracts of Ethereum had to be adjusted to the technologies that this platform uses. Therefore, the entire code had to be partially rewritten. These changes were relevant in the comparison of both platforms. For example, in Table 6.3, the number of connections with the blockchain that each method has in Ethereum is considerably lower than the number of connections in Hyperledger Fabric. The difference is a consequence of the Hyperledger Composer Framework usage that adds layers of necessary steps to execute transactions in Fabric. Since Web3 connects directly with the Solidity Smart Contracts, there are fewer steps to complete the transactions.

Although the downside of the shorter path of communication in the Ethereum scenario is the complexity with which the system had to deal in the development cycle. The best example is the modification of the smart contracts that could serve the purposes of the completion of the

distribution algorithm. The identification of objects with hashes as addresses of each object had to be changed within the Contract (More information in the Implementation Section). Therefore, a way to send different unique addresses as input parameters from JMeter or (in the case this research) needed to be found, target random addresses from the Ganache Client autogenerated blockchain. Additionally, some validations within the contracts required to be eliminated to avoid blockchain blockers.

6.5.4 Ethereum Ganache Client Results

The Execution time and Throughput in Ethereum differ in range significantly in comparison with HF, especially the Throughput (Table 6.4). The time of response, therefore, the latency is higher despite the decrease in the number of connections the Scenario has in comparison with HF. Consequently, the throughput is lower too. Additionally, there are some instances of HTTP requests that suffered errors in the response, meaning that the distributions were not completed correctly. Moreover, a standard error called "Error of RPC connection timeout - Invalid JSON RPC" appeared regularly. Whenever this problem arose, the Ganache Client stopped and could accept any incoming requests anymore. Therefore, some of the HTTP requests are not valid as to be considered in the analysis.

Ethereum Ganache Client JMeter Results						
Example 1 - Scenario 1						
#Request/thread (#Distributions)	# Tx/thread	Total Execution Time(sec)	Average Execution Time(sec)	AverageThroughput (distribution/second)	Throughput (#Tx/sec)	error%
1	21	30.366	30.366	0.0235730854	0.4950347934	0
2	42	70.252	35.126	0.0403244972	1.693628882	0
3	63	125.119	41.706	0.0502071668	3.163051508	0
4	84	199.747	49.936	0.056027419	4.706303196	0
5	105	301.472	60.294	0.0593401569	6.230716475	0
6	126	606.418	101069	0.0424907436	5.353833694	0.8571428571
7	147	767.263	109609	0.1062053806	15.61219095	0.2142867143
Example 2 - Scenario 2						
#Request/thread (#Distributions)	# Tx/thread	Total Execution Time(sec)	Average Execution Time(sec)	AverageThroughput (distribution/second)	Throughput (#Tx/sec)	error%
1	22	33.157	33.157	0.030159544	0.663509968	0
2	44	75.13	37.565	0.0527843758	2.322512535	0
3	66	135.415	45.138	0.0656067531	4.330045705	0
4	88	262.261	65.565	0.03331612	2.93181856	0.25
5	110	450.815	90.163	0.0416479251	4.581271761	0.6
6	132	720.799	120.133	0.0499438132	6.592583342	1
7	154	741.05	105.864	0.0640468457	9.863214238	1

Table 6.4 Ganache Client Ethereum JMeter Results

Analysis Execution Time. -Figure 6.13 shows the *total execution time* in Scenario 1 and Scenario 2(blue and red lines) using the Ethereum Ganache client. Figure 6.13 shows an increase in the *total execution time* as the number of transactions per sample increases. Each Scenario has very similar steps in the axis x(samples) since they have approximately the same number of transactions per sample (Table 6.3). Therefore, the same behavior in both curves was expected. The same incremental response is shown in the works of Pongnumkul et al. [27]. Although, the difference in error percentage in comparison with HF was clear. Table 6.5 shows clearly that Scenario 1 starts having response errors in samples 6th and sample 7th. Additionally, Scenario 2 starts having response errors that rise to 100% in samples 4th-7th. Therefore, some samples ultimately cannot be considered for the comparison since the system crushed.

On the other hand, when the highest number of transactions, 154 Txs in Scenario 1, the average percentage CPU consumption during the experiments was stable and without glitches, with a response time of 741 seconds or around 12 minutes (very similar for Scenario 2). In comparison with HF, Ganache doesn't crush the machine since the number of transactions is significantly lower than HF.

Analysis of Average Throughput. - Figure 6.14 shows the average throughput for both Scenarios using Ethereum Ganache client. Like the *total execution time,* the *average throughput* increases as the number of incoming transactions increase. Although, in contrast with the works of Pongnumkul et al. [27], the point of inflection where the *average throughput* slowly decreases after a specific number of transactions could not be found. The problem in contrast with the case HF is that the information after the 4th sample is not reliable since the system starts having issues with RPC connections, as explained previously.

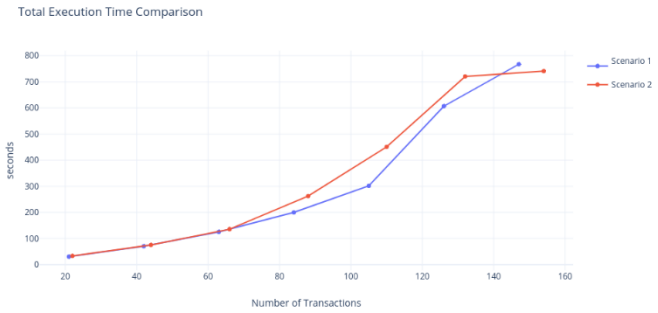


Fig 6.13 Total Execution Time for both Scenarios using Ethereum Ganache

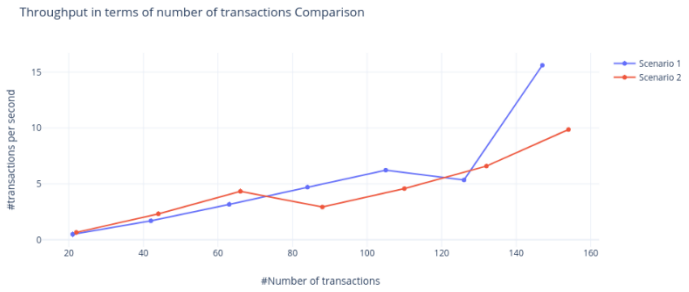


Fig 6.14 Throughput in terms of number of Tx for both Scenarios using Ethereum Ganache

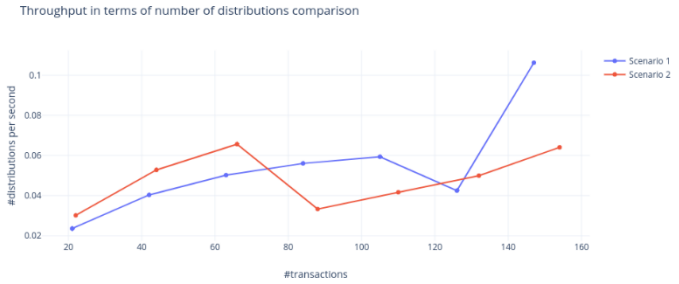


Fig 6.15 Throughput in terms of Distribution per second for both Scenarios using Ethereum Ganache

Additionally, Figure 6.15 shows the *average throughput* of both scenarios in terms of distribution per second using the Ethereum Ganache Client. For example, when 21 Txs in Scenario 1 are being executed, 0.49 distributions per second are possible in the bottleneck. On the other hand, when 105 Txs are completed in Scenario 1, the bottleneck increases to 6.23 distribution per second as concurrent transactions are being executed in the blockchain. This behavior differs from HF since there is an increase in the performance of Ganache Client for a period, and then the system crashes after 105 Txs sent at the same time.

6.6 Limitations of the Performance Analysis

Since Pongnumkul et al. [27] experiment included Golang Hyperledger Fabric Chaincode instead of the NodeJs SDK of Hyperledger Composer, there was an increase in the execution time. For example, when executing 100 transactions, Pongnumkul et al. [27] obtained 2.59 seconds to complete the request. In Scenario 1, it lasted around 80 seconds. Although, the Chaincode varies and could possess a more complex logic, it is possible that HC NodeJs increases the *average execution time* and the *total execution time* since it adds additional layers of abstractions to the

network. Therefore, Golang programming is more suitable if better performance is desired, even if it doesn't offer other features of abstraction as HC. Moreover, using Golang could increase the accuracy of the comparison between HF and Ethereum.

Additionally, one of the advantages that HC offers is the Multiversion Concurrency Control Model (MVCC). This feature adds a security layer on top of the application since these features do not let an entity's property to be modified unless it is within a specified period, preventing suspicious incoming transaction requests. If two transactions are requested to be validated in a period that is shorter than the world state update, one of those transactions' claims will fail and will throw an MVCC_READ_CONFLICT. The error can be avoided by changing the default settings. Although, by abstracting HF, the focus can be shifted towards the development of the application with friendly classes for faster progress in exchange for downsides in the performance.

Consequently, it is essential to notice that the Ethereum attempts to adjust better to the works of Pongnumkul et al. [27] than the attempts in HF. Despite the lack of features in Solidity Contracts, the performance behavior modified to the practices of Pongnumkul et al. [27] in terms of the accuracy of the Smart Contracts. Although, Ethereum Ganache client shows low reliance on the research model since after the 4th sample, there are problems within the system. It's assumed that this behavior could be since Ganache is operating as a single node or that the way the requests are being queued is not correct for the modeling of a real case scenario.

6.7 Discussions and Conclusions of the Data Analysis

Figure 6.16, 6.17, 6,18 show the three parameters compared previously between HF and Ethereum in both Scenarios. The range of results differs, especially in the Throughput figures.

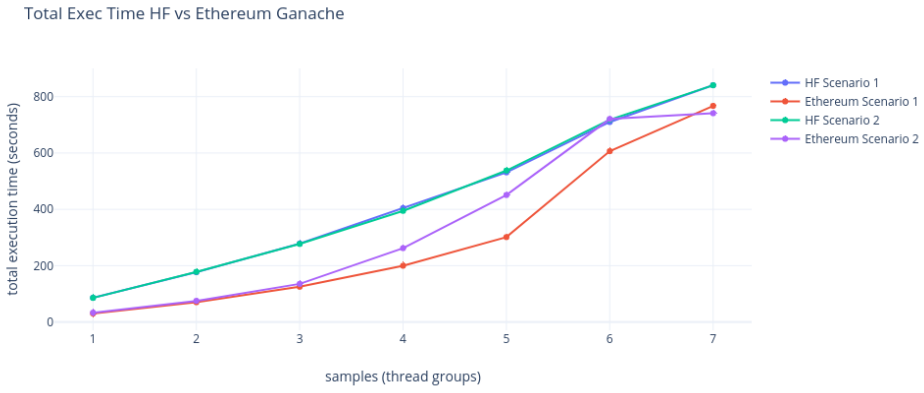


Figure 6.16 Total Execution Time of HF vs. Ethereum in both Scenarios

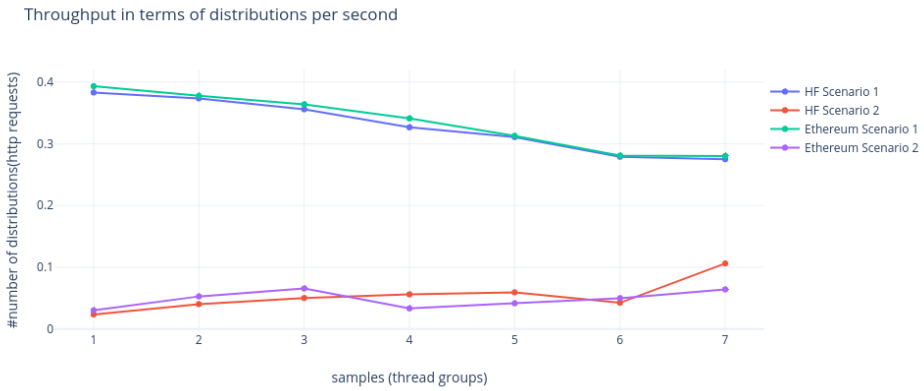


Figure 6.17 Throughput in terms of distributions of HF vs. Ethereum in both Scenarios.

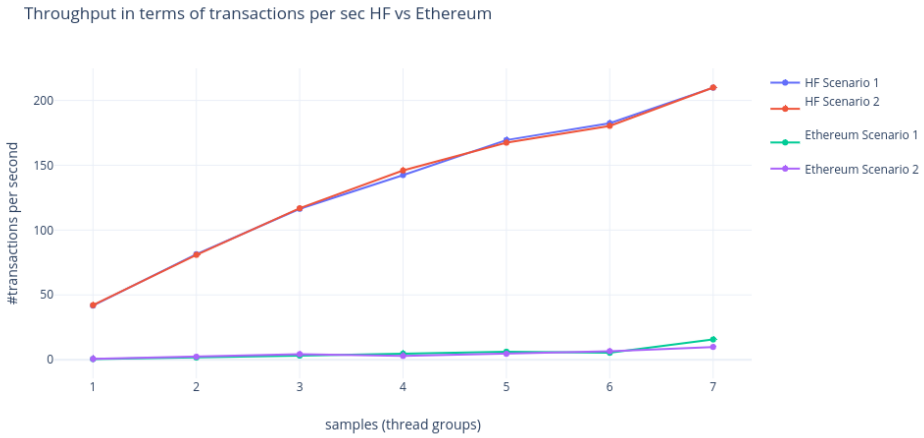


Figure 6.18 Throughput in terms of transactions per second of HF vs. Ethereum in both Scenarios

Scenario 3 represents a case in the distribution algorithm that enters an infinite processing loop as explained in the Implementation section. This behavior limits the application scope of the algorithm. Therefore, Scenario 3 was not part of the analysis, and the algorithm must find improvement in future work for Scenario 3.

Since Scenario 1 and Scenario 2 have a similar number of transactions a linear curve in both scenarios was expected. Although, the way those transactions are executed in the NodeJs SDK for HF and Ethereum Ganache Client varies and therefore the comparison suffered some lack of information.

Additionally, the average throughput in terms of distributions per second in Scenario 1 or Scenario 2 decreases as the number of incoming transactions are executed in HF and increases slightly in Ethereum.

Ultimately, the expectations of the results exposed in Section 6.1 turned to be accurate since Hyperledger Fabric achieved higher Throughput and lower Latency, the increment of several transactions differentiated the range of outcomes in which both platforms operated, and Hyperledger Fabric using Hyperledger Composer handled concurrent transactions without glitches and error percentages. Therefore, this research accomplished one of the main goals of the Problem Statement section, which is to execute the experiments for Performance in terms of Latency and Throughput.

CHAPTER 7

CONCLUSIONS, CONTRIBUTION, AND FUTURE WORK

This section mentions the conclusions of the current research in reconciliation with the Literature Review and where the ongoing study of blockchain technology is.

7.1 Summary of the Implementation and the Data Analysis Conclusions and Future Work

The implications over non-hierarchical decentralized incoming Music Industry are positive***. This research contributes to the literacy of solutions that accelerate the payment of music track revenues to all the participants in the Current Music Industry based on Streaming Services (Distributors)***. More detailed conclusions and future work can include:

Conclusions:

- The performance experiment leads to a positive overview of a single machine computational capacity of around 800Tx/s in Hyperledger Fabric and 6 Txs/second in Ethereum Ganache Client.
- The conclusions of the experiments of this research contribute to the results of the performance of both blockchains in the specific case of the application scalability (Music Industry Streaming Services – based participant payment improvement). This contribution differs from the works of Pongnumkul et al. [27] since their smart contracts were basic smart contract-based currency transactions.
- It's important to acknowledge that since just one local machine runs each platform, these quantities are subjected to be higher in bigger networks. Moreover, when more dedicated servers in a decentralized network add computational resources, the efficiency is foreseen to increase considerably. These extra resources are needed for developing in production, especially since streaming services such as the Spotify stream around 750000 streams/minute [1].
- Ionic Framework showed issues in the dependencies and is slowly losing track of adoption. Therefore, is not recommended for projects like this one. React is recommended.

- Blockchain platforms (Ethereum and Hyperledger) as a common ledger for the computation of Smart Contracts solve the necessity of data reconciliation between different databases. Although, the metadata and media files are not stored in the blockchain platforms. The platforms just store basic information of the variables such as the id of the Assets and the Participant ids.

Future Work Goals:

- It's planned to add the extra Latency analysis by building a component that reads the transaction confirmation within the NodeJS SDK Chaincode since the Latency computed by JMeter adds a layer of the other timeframe in the analysis
- The works of Pongnumkul et al. [27], mention that the consensus protocols would significantly affect the results. Therefore more robust consensus protocols will be added to test the distribution algorithm in the future.
- The deployment of Smart Contracts in the Ropstein network was brief experimentation of the impact this application has on a real Ethereum network. It was not possible to compare this pre-production cycle with Hyperledger Fabric since it has not yet any pre-production environments or frameworks, were other people interact with, such as in Ropstein of Ethereum.
- Development of a UI that uses newer frameworks such as React.
- Add Audit process to focus on the security of Smart Contracts and Middleware integration. The system of this research still needs improvement in the security aspect.
- The data regarding the economical advantages and challenges of the implementation of these solutions in the Current Business model was difficult to acquire at the time of this research. There are some assumptions that are taking place. For example, new companies that develop these blockchain solutions will offer these services to the current participants of the Music Industry, and the decision of these participants will determine if these solutions create decentralized networks.

Overall, the project tried to emulate the efficiency of a blockchain solution in a specific industry. **But how far can this research go? What other applications mentioned in the Literature Review can be model with the algorithms and components?** Some of the examples are:

-The Real State Industry due to the standards of preservation discussed previously in the Literature Review Section[43].

-The Financial Sector due to the distribution algorithm that record receipts for Audit Systems

-Proof of ownership in the Entertainment Industry in general since the copyrights can be validated and manipulated by Smart Contracts.

7.2 Where is current research on blockchain technology?

At the beginning of this research in 2017, Bitcoin's price rise seemed to be unstoppable, and the media was paying attention to this phenomenon until it crashed at the end of the year. Since then, the price has slowly decreased, and the focus started to fade away. Therefore, it is important to acknowledge where the blockchain research is to imagine the scope of the importance of the study. As mentioned in the Literature Review, Blockchain is one of the fastest fields of study in future years to come. Moreover, some trends show where Blockchain research is heading. A study conducted by Yli-Huumo et al.[44] (2016), showed the current research on Blockchain to help researchers to identify the gaps and challenges. The research mentions that 80% of the papers blockchain-related are Bitcoin-based and 20% in other networks and blockchain applications. Additionally, this study reveals that the current main technical challenges are **Throughput, Latency**, size and bandwidth, security, wasted resources, usability, versioning and forks, privacy. This research is similar in the sense that the project covers the Throughput and Latency challenges of two big players of the scene: Hyperledger and Ethereum.

[44] examined 41 filtrated original papers from 121 initially from scientific databases. These papers went through a systematic mapping represented by specific categorizations. The conclusions over the source, the geography, publication type, and publication year, are shown in Figures 7.1 - 7.5.

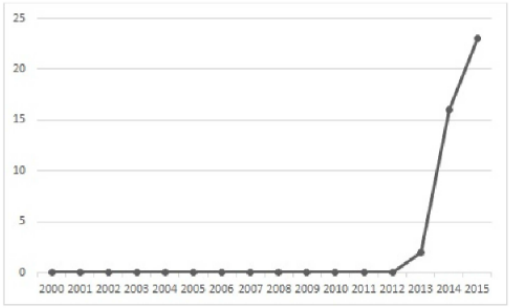


Fig 4. Publication year of the selected primary papers.

doi:10.1371/journal.pone.0163477.g004

Figure 7.1 Publication year of the selected primary papers [44]

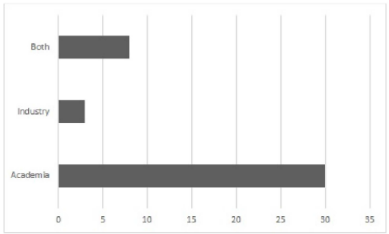


Fig 5. Source of the selected primary papers.

doi:10.1371/journal.pone.0163477.g005

Figure 7.2 Source of the selected primary papers [44]

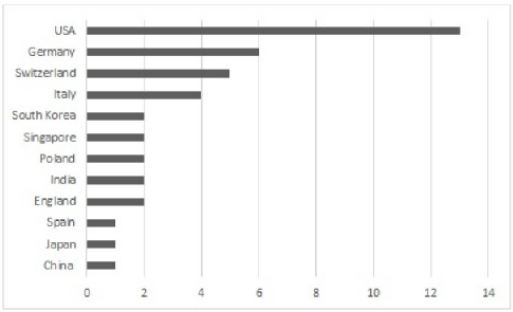


Fig 6. Geographic distribution of the selected primary papers.

doi:10.1371/journal.pone.0163477.g006

Figure 7.3 Geographic distribution of the selected primary papers [44]

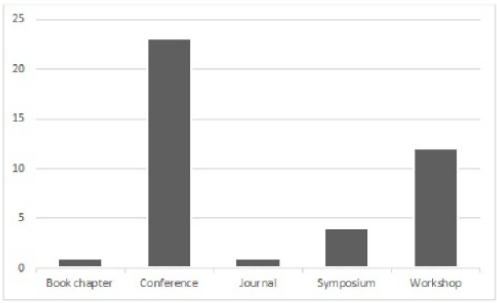


Fig 7. Publication type.

doi:10.1371/journal.pone.0163477.g007

Figure 7.4 Publication type [44]

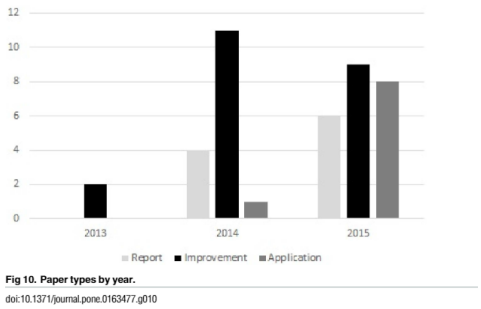


Fig 10. Paper types by year.

doi:10.1371/journal.pone.0163477.g010

Figure 7.5 Paper types per year [44]

Additionally, Figure 7.5 shows the classification of these papers by types: report, improvement, or application. The current project could be classified as a report of efficiency and application. Also, [44] concluded the categorization of research diagram in Figure 7.6. One of the insights of this diagram shows that there were not enough papers on the topics of the technical challenges on Latency, size and bandwidth, Throughput, versioning, hard forks, and multiple chains until 2016. The current project hopefully could be added for future references on the considerations of throughput and latency performance on the platforms studied.

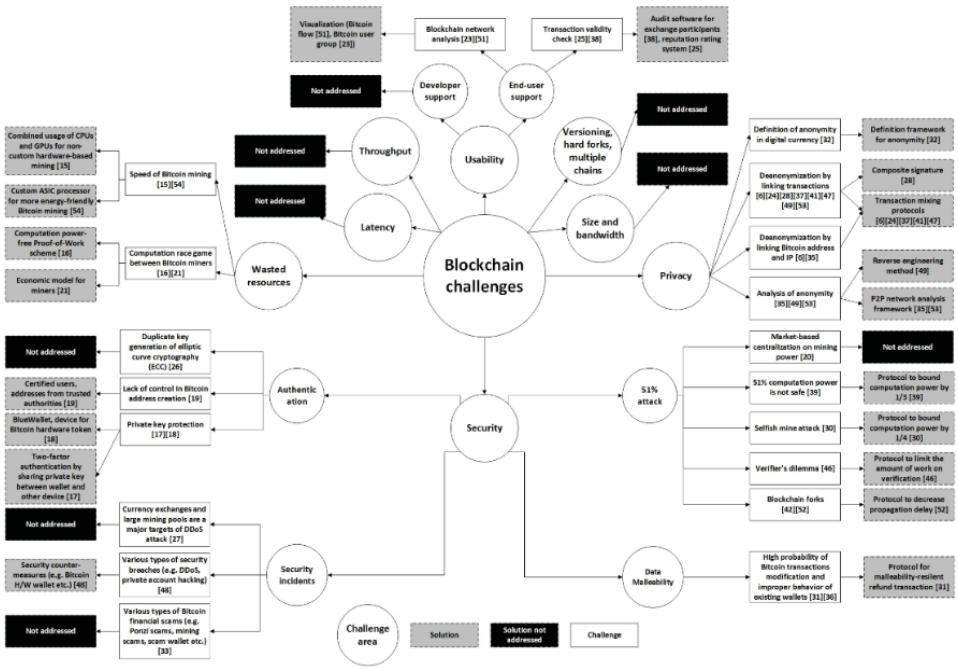


Figure 7.6: Summary of the identified challenges and solutions of blockchain [44]

Ultimately, the research of [44] mentions that smart contracts are still a challenge with the current blockchain research field. From the time of the publication of the paper, works like the one of [44], started to present other blockchain implementations on use cases previously unknown. This works showed a solution to the blockchain-based digital content distribution system. The idea is meant to be given to creators, content owners, and digital content

stakeholders and is aiming to provide tools to improve this field so it can be shared with those participants.

Personal Overview

Regarding the personal experience expansion, the mindset of decentralized systems is mind switching. The idea of source code decentralization was challenging at the beginning of the research. The first challenge was to use higher abstraction frameworks that used blockchain as underlying networks such as Hyperledger Fabric with presets that configured the system automatically. After that, the change to the developing Smart Contracts in Ethereum cleared the gaps of knowledge at the beginning of the research since the way the Smart Contracts written in Solidity are entirely different from the paradigm of Object-Oriented Programming Languages such as Java or Javascript. The fact that the objects must be modeled as data located in different hashes and have restrictions due to the capabilities of the Ethereum Virtual Machine opens the possibility to understand the nature of decentralized networks. Networks where not only the databases and business logic are located but the code itself is located in different anonymous places, forming a giant machine that works with anyone connected to it securing privacy and security. These features open the door for the future. A future where the data is not controlled by single points of failure, but a future where the information is safe and private by the collaboration of all the participants in a global network. It might be the rise of a new way of structuration the Internet.

REFERENCES

[1] De Filippi, P. (2015). Blockchain-based Crowdfunding: what impact on artistic production and art consumption? Observatório Itaú Cultural, (19).

[2] Singh, S., & Singh, N. (2016). "Blockchain: Future of financial and cybersecurity." In Proceedings of the 2016 2nd International Conference on Contemporary Computing and Informatics, IC3I 2016.

[3] LeRoy Carr III, Anthony Newtson, James Joshi, "Towards Modernizing the Future of American Voting," *Collaboration and Internet Computing (CIC) 2018 IEEE 4th International Conference on*, pp. 130-135, 2018.

[4] Passman, D. S. (2015). "*All you need to know about the music business*." Simon and Schuster.

[5] Datta, H., Knox, G., & Bronnenberg, B. J. (2017). Changing their tune: How consumers' adoption of online streaming affects music consumption and discovery. Marketing Science, 37(1), 5-21.

[6] Marshall, L. (2015). 'Let's keep music special. F—Spotify': on-demand streaming and the controversy over artist royalties. Creative Industries Journal, 8(2), 177-189.

[7] Richardson, J. H. (2014). The Spotify paradox: How the creation of a compulsory license scheme for streaming on-demand music platforms can save the music industry. UCLA Entertainment Law Review, 22(1).

[8] Nakamoto, S. (n.d.). "*Bitcoin: A Peer-to-Peer Electronic Cash System*."

[9] Eyal, I. (2017). Blockchain technology: Transforming libertarian cryptocurrency dreams to finance and banking realities. *Computer*, *50*(9), 38-49.

[10] NG, D. C. M., & GRIFFIN, P. R. (2018). The wider impact of a national cryptocurrency. *Global Policy*, 1.

[11] Baravalle, A., Lopez, M. S., & Lee, S. W. (2016, December). Mining the dark web: drugs and fake IDs. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*(pp. 350-356). IEEE.

[12] Christin, N. (2013, May). Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web* (pp. 213-224). ACM.

[13] Bohr, J., & Bashir, M. (2014, July). Who uses bitcoin? An exploration of the bitcoin community. In *2014 Twelfth Annual International Conference on Privacy, Security, and Trust* (pp. 94-101). IEEE.

[14] Gnan, E., & Masciandaro, D. (2018). Do we need a central bank digital currency? Economics, technology, and institutions. Vienna: SUERF-The European Money and Finance Forum.

[15] Cryptocurrency Market. Retrieved from www.coinmarketcap.com.

[16] Frisby, D." *Bitcoin: The future of money?*". London: Unbound, 2014.

[17] Vigna, P & Casey, M. "*The age of cryptocurrency: How Bitcoin and digital currency are changing the global economic order*." New York: St. Martin's Press, 2015.

[18] Matenga, R.N, & Stanley. H. E." *An introduction to econophysics*". New York: Cambridge University Press, 1999.

[19] Fry, J., & Cheah, E.T. (2016). Negative bubbles and shocks in cryptocurrency markets. International Review of Financial Analysis, 47, 343-352.

[20] Kazeem Yomi. "*The World's First Blockchain Supported Elections Just Happened in Sierra Leone*." Quartz Africa, March 13, 2018.

[21] Tapscott, D., & Tapscott, A. (2016). "*Blockchain Revolution: how the technology behind bitcoin is changing money, business, and the world."* Penguin.

[22] Lemieux, V. L. (2016). Trusting records: is Blockchain technology the answer?. *Records Management Journal*, *26*(2), 110-139.

[23] Wild, J., Arnold, M., & Stafford, P. (2015). Technology: Banks seek the key to the blockchain. *Financial Times*, *1*, 2015.

[24] Sutherland, M. (2018). *"Stream If You Want to Go Faster."* Music Week, 22–24. Retrieved from http://cyber.usask.ca/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=mah&AN=133383040&site=ehost-live

[25] Hyperledger Project. (2016). "*Overview of Hyperledger Introduction to the Linux Foundation's Hyperledger Project, (June)*." Retrieved from http://www.redwoodmednet.org/projects/events/20160718/docs/rwmn_20160718_behlendorf.pdf

[26] Duranti, L., & Rogers, C. (2012). Trust in digital records: An increasingly cloudy legal area. *Computer Law & Security Review*, *28*(5), 522-531.

[27] Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017, July). Performance analysis of private blockchain platforms in varying workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)* (pp. 1-6). IEEE.

[28] Szabo, N. (1997). The idea of smart contracts. *Available: http://szabo.best.vwh.net/smart.contracts.html*

[29] Christidis, K., & Devetsikiotis, M. (2016). "*Blockchains and smart contracts for the internet of things*." IEEE Access, *4*, 2292-2303.

[30] Hyperledger Organization. (n.d.). "*About Hyperledger*." Retrieved October 17, 2017, from http://hyperledger.org/about

[31] Hyperledger Architecture Working Group. "*Hyperledger Architecture Volume 1: Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus*". Hyperledger Org, 2017.

[32] Friebe Tjark. "*Trust your Competitor? How you can do that with Hyperledger Fabric Blockchain*". Medium, 2017.

[33] Sousa, J., Bessani, A., & Vukolić, M. (2017). "*A Byzantine fault-tolerant ordering service for the hyper ledger fabric blockchain platform*." arXiv preprint arXiv:1709.06921.

[34] Wood, G. (2014). "*Ethereum: A secure decentralized generalized transaction ledger. Ethereum project yellow paper,"*, 151, 1-32.

[35] Rouhani, S., & Deters, R. "*Performance Analysis of Ethereum Transactions in Private BlockChain*." University of Saskatchewan, Saskatoon.

[36] Allen, Paul R, Bambara, Joseph J. *"Blockchain: a Practical Guide to Developing Business, Law, and Technology Solutions."* Mc Graw Hill, 2018

[37] Kaufmann, Aviv, Dolan, Kerry. "*Price Comparison: Google Cloud Platform vs. Amazon Web Services*." ESG Lab Analytics. June 2015

[38] Google Cloud 2017 Review. "*Google Infrastructure Security Design Overview."* Retrieved from https://cloud.google.com/security/infrastructure/design/resources/google_infrastructure_whitepaper_fa.pdf

[39] Satheesh Mithun, D'mello Bruno, Krol Jason. *"Web development with MongoDB and NodeJs*." PACKT publishing. Second Edition, 2015.

[40] Selfa, D. M., Carrillo, M., & Boone, M. D. R. (2006, February). A database and web application based on MVC architecture. In *Electronics, Communications, and Computers, 2006. CONIELECOMP 2006. 16th International Conference on* (pp. 48-48). IEEE.

[41] Hu, Z. G., Yuan, Q., & Zhang, X. (2009, July). Research on agile project management with scrum method. In *Services Science, Management and Engineering, 2009. SSME'09. IITA International Conference on* (pp. 26-29). IEEE.

[42] Schutt, Russell K. "*Investigating the Social World: The Process and Practice of Research*." Ninth Edition. University of Massachusetts, Boston, USA. SAGE Publications, Inc, 2019.

[43] Spielman, A. (2016). Blockchain: digitally rebuilding the real estate industry (Doctoral dissertation, Massachusetts Institute of Technology).

[44] Yli-Huumo, J., Ko, D., Choi, S., Park, S., & Smolander, K. (2016). "*Where is current research on blockchain technology?" —*A systematic review. *PloS one*, *11*(10), e0163477.

[45] Rethink Music Initiative. (2015). Fair Music: Transparency and Payment Flows in the Music Industry. *Boston: Berklee Institute of Creative Entrepreneurship*.

[46] Abhishek S, Promaya B, Arumendra S. *"A systematic review on Evolution of Blockchain Generations."* ITEE journal. December 2018. ISSN: - 2306-708x